Є.О. ЗАЙЦЕВ

ОСНОВИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

КИЇВ–2017

УДК 004.4
ББК 32.973я7
ISBN 978-

Рецензенти:

М.В. Мислович, д. техн. наук, проф., завідувач відділу теоретичної електротехніки Інституту електродинаміки НАН України, професор кафедри автоматизації технологічних процесів Національного університету будівництва і архітектури, лауреат державної премії України в галузі науки і техніки, м. Київ

Б.А. Кромпляс, к. техн. наук, старший науковий співробітник, директор ТОВ "Енергоефективні технології – ХХІ", м. Київ

, к. техн. наук, доцент, доцент кафедри програмної інженерії та інформаційних систем Київського національного торговельно-економічного університету, м.Київ

Рекомендовано до друку

Основи програмної інженерії: навчальний посібник
/ Є. О. Зайцев – К.: КНТЕУ, 2017. – с.

У навчальному посібнику наведено визначення програмної інженерії, її дисциплін та областей ядра знань відповідно до стандарту ISO/IEC TR 19759:2015 Software Engineering – Guide to the software engineering body of knowledge (SWEBOK V3) та з урахуванням Computer Engineering Curricula 2016. Визначено положення життєвого циклу програмного забезпечення. Розглядається розробка вимог, планування й аналіз ризиків при проектуванні й створенні програмних продуктів і систем на їх основі, а також питання пов’язані із створенням та веденням програмної документації.

Для студентів вищих навчальних закладів, а також спеціалістів, зайнятих проектуванням, впровадженням, забезпеченням роботи програмних продуктів і систем на їх основі.

УДК 004.4
ББК 32.973я7
ISBN 978-

ЗМІСТ

РОЗДІЛ 1
ВСТУП

При вивченні цієї теми важливо пам'ятати, що у загальному випадку *програмна інженерія* – це наука про принципи і методології, що використовуються при розробці і супроводі програмного забезпечення, пов'язана із всіма аспектами виробництва програмного забезпечення від початкових стадій створення специфікацій до підтримки системи після здачі її в експлуатацію та дослідження цих аспектів, тобто застосування принципів інженерії до всіх процесів життєвого циклу програмного забезпечення.

Програмна інженерія – це приваблива з точки зору інвестицій, галузь економіки будь – якої розвиненої держави, яка піклується про своє майбутнє. В галузі реалізуються великі та малі проекти, що потребують не тільки кваліфікованих розробників, а і кваліфікованих менеджерів

Зміст розділу

1.1 Програмна інженерія

1.2 Основні поняття програмної інженерії

1.3 Історичні умови виникнення та розвитоку програмної інженерії, як галузі знань

1.4 Основні характеристики та фази розвитку програмної інженерії

1.5 Методи та інструменти програмної інженерії

1.1 ПРОГРАМНА ІНЖЕНЕРІЯ

Термін «програмна інженерія вперше був озвучений в жовтні 1968 року на конференції підкомітету НАТО з науки і техніки (м. Гарміш, Німеччина). Були присутні 50 професійних розробників ПЗ з 11 країн. На конференції розглядалися проблеми проектування, розробки, поширення і підтримки програм. Незважаючи на розвиток програмної інженерії, на сьогоднішній день немає єдиного визначення поняття програмної інженерії. Відповідно до прийнятих стандартів та керівництв (див. додаток А) програмна інженерія – це:

– встановлення і використання обґрунтованих інженерних принципів для економного отримання програмного забезпечення, яке є надійним і реально працює (Фрідріх Бауер);

– форма інженерії, яка застосовує принципи комп'ютерних наук і математики для рентабельного розв'язання проблем програмного забезпечення (CMU/SEI-90-TR-003);

– застосування систематичного, дисциплінованого, вимірюваного підходу до розробки, використання й супроводу програмного забезпечення (IEEE, 1990);

– дисципліна, метою якої є створення якісного програмного забезпечення, яке завершується вчасно, не перевищує виділених коштів і задовольняє висунуті вимоги (Стівен Шах);

– система методів, способів і дисциплін з планування, розробки, експлуатації та супроводу програмного забезпечення, призначених для промислового виробництва (SWEBOK);

– розділ комп'ютерної науки, який:

– вивчає методи і засоби побудови комп'ютерних

програм;

– відображає закономірності розвитку та узагальнює накопичений досвід програмування;

– оперує об’єктами (модулями, компонентами, програмними аспектами тощо) та визначає автоматизовані операції щодо їхнього застосування;

– виробляє правила, порядок інженерної діяльності і керування технологічним процесом побудови з простих об’єктів нових, більш складних, цільових об’єктів (програмного забезпечення, програмних систем, сімейств систем, програмних проектів тощо), а також методи вимірювання й оцінювання готового продукту.

– теоретичні, формальні методи та відповідні засоби побудови складних програмних об’єктів.

– інженерна дисципліна, пов’язана зі всіма аспектами виробництва програмного забезпечення, від початкових стадій створення специфікації до підтримки системи після здачі в експлуатацію.

На відміну від математичної або інших фундаментальних наук, метою яких є отримання нових знань для розв’язання відповідних задач, метою програмної інженерії є застосування знань для розробки складних програмних об’єктів, де знання – це уособлення загальної теорії побудови програм для комп’ютерів, орієнтованої на виготовлення продукту, впровадження якого принесе певну користь користувачеві.

Програмна інженерія, як інженерна дисципліна пов’язана з практичною діяльністю інженерів. Інженери – це ті фахівці, які виконують практичну роботу й добиваються практичних результатів.

Для виконання завдання інженери застосовують відповідні теорії, методи і засоби, але вони застосовують їх вибірково і завжди намагаються знайти рішення, навіть у тих

випадках, коли теорій або методів, відповідних даному завданню, ще не існує. У цьому випадку інженер шукає метод або засіб для розв'язання задачі, застосовує його і несе відповідальність за результат – адже метод або засіб ще не перевірені. Набір таких інженерних методів або способів, можливо, теоретично не обґрунтованих, але таких, що отримали неодноразове підтвердження на практиці, відіграє велику практичну роль.

Інженери працюють в умовах обмежених ресурсів (рис.4.11): часових, фінансових і організаційних (обладнання, техніка, люди). Іншими словами, програмний продукт має бути створений у встановлені терміни, в рамках виділених коштів, обладнання і людей.

Програмна інженерія займається не тільки технічними питаннями виробництва програмного забезпечення (специфікація вимог, проектування, кодування тощо), але і управлінням програмними проектами, включаючи питання планування, фінансування, управління колективом і так далі. Крім того, завданням програмної інженерії є розробка засобів, методів і теорій для підтримки процесу виробництва програм.

Програмні інженери застосовують систематичні й організовані підходи до роботи для досягнення максимальної ефективності та якості програмного забезпечення. Їх завдання полягає в адаптації існуючих методів і підходів для розв'язання конкретної проблеми.

Отже, *програмна інженерія* – це наука про принципи і методології, що використовуються при розробці і супроводі програмного забезпечення, що вивчає застосування системного, вимірюваного підходу до розробки, використання та супроводу програмного забезпечення, та дослідження цих підходів, тобто застосування принципів інженерії до всіх процесів життєвого циклу програмного забезпечення.

1.2 ОСНОВНІ ПОНЯТТЯ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

До основних понять програмної інженерії відносяться дані і їх структури (прості і складні), функції і композиції, базові об’єкти (модуль, компонент, каркас, контейнер, компонент повторного використання та тощо) і цільові об’єкти (програмне забезпечення, програмна система, сімейство систем, програмний проект, складне ПЗ тощо).

Розробка простих об’єктів – це елементарні дії з їх формального опису, а розроблення цільових об’єктів – застосування інженерних методів, включаючи керування строками і вартістю виробництва.

Цільовими об’єктвами програмної інженерії є:

Програмна (прикладна) система (Application) – комплекс інтегрованих програм і засобів, що реалізують набір взаємопов’язаних функцій деякої предметної області в заданому середовищі. У комплекс можуть входити:

– прикладні системи (наприклад, програми розрахунку зарплати, обліку матеріалів на складі тощо);

– загальносистемні програмні засоби (наприклад, транслятор, редактор, СКБД тощо);

– спеціалізовані ПЗ для реалізації функцій захисту інформації, забезпечення безпеки функціонування та ін.

Програмне забезпечення **(software)** – сукупність програм системи обробки інформації і програмних документів, необхідних для експлуатації цих програм.

Розрізняють системне програмне забезпечення (зокрема, операційна система, транслятори, редактори, графічний інтерфейс користувача) та прикладне програмне забезпечення, що використовується для виконання конкретних завдань, наприклад, статистичне програмне забезпечення.

Виконання програмного забезпечення комп'ютером

полягає у маніпулюванні інформацією та керуванні апаратними компонентами комп'ютера. Наприклад, типовим для персональних комп'ютерів є відтворення інформації на екран та отримання її з клавіатури.

Програмне забезпечення (software) та апаратне забезпечення (hardware) – це два комплементарні компоненти комп'ютера, причому межа між ними нечітка, тобто деякі фрагменти програмного забезпечення на практиці реалізуються суто апаратурою мікросхем комп'ютера, а програмне забезпечення, в свою чергу, здатне виконувати (емулювати) функції електронної апаратури.

Іншими словами програмне забезпечення – це комплекс програм, які забезпечують ефективне управління компонентами обчислювальної системи, такими як процесор, оперативна пам'ять, канали введення-виведення, мережеве обладнання, виступаючи як «міжшарової інтерфейс» з одного боку якого апаратура, а з іншого додатка користувача.

На відміну від прикладного програмного забезпечення, системне не вирішує конкретні прикладні задачі, а лише забезпечує роботу інших програм, управляє апаратними ресурсами обчислювальної системи і т.д. Системне ПЗ керує ресурсами комп'ютерної системи і дозволяє користувачам програмувати в більш виразних мовами, ніж машинних мову комп'ютера.

Склад системного ПЗ мало залежить від характеру вирішуваних завдань користувача.

До системного програмного забезпечення відносяться:

– операційні системи;
– інтерфейсні оболонки для взаємодії користувача з програмним забезпеченням;
– системи управління файлами;
– системи програмування;
– утиліти.

Системне програмне забезпечення призначене для:

– створення операційного середовища функціонування інших програм (іншими словами, для організації виконання програм);
– автоматизації розробки (створення) нових програм;
– забезпечення надійної та ефективної роботи самого комп'ютера й обчислювальної мережі;
– проведення діагностики і профілактики апаратури комп'ютера й обчислювальних мереж;
– виконання допоміжних технологічних процесів (копіювання, архівування, відновлення файлів програм і баз даних і т.д.).

Типова структура ПЗ складається з:

1. *Прикладного рівня.* Програмне забезпечення цього рівня являє собою комплекс прикладних програм, за допомогою яких виконуються конкретні завдання (від виробничих до творчих, розважальних та навчальних). Між прикладним та системним програмним забезпеченням існує тісний взаємозв'язок. Універсальність обчислювальної системи, доступність прикладних програм і широта функціональних можливостей комп'ютера безпосередньо залежать від типу наявної операційної системи, системних засобів, що містяться у її ядрі й взаємодії комплексу людина-програма-обладнання.
2. *Службового рівня.* Програми цього рівня взаємодіють як із програмами базового рівня, так і з програмами системного рівня. Призначення службових програм (утиліт) полягає у автоматизації робіт по перевірці та налаштуванню комп'ютерної системи, а також для покращення функцій системних програм. Деякі службові програми (програми обслуговування) відразу додають до складу

операційної системи, доповнюючи її ядро, але більшість є зовнішніми програмами і розширюють функції операційної системи. Тобто, у розробці службових програм відслідковуються два напрямки: інтеграція з операційною системою та автономне функціонування.

3. *Системного рівня.* Системний рівень - є перехідним. Програми цього рівня забезпечують взаємодію інших програм комп'ютера з програмами базового рівня і безпосередньо з апаратним забезпеченням. Від програм цього рівня залежать експлуатаційні показники всієї обчислювальної системи. При під'єднанні до комп'ютера нового обладнання, на системному рівні повинна бути встановлена програма, що забезпечує для решти програм взаємозв'язок із цим пристроєм. Конкретні програми, призначені для взаємодії з конкретними пристроями, називають драйверами.

4. *Базового рівня.* Цей рівень є найнижчим рівнем програмного забезпечення. Відповідає за взаємодію з базовими апаратними засобами. Базове програмне забезпечення міститься у складі базового апаратного забезпечення і зберігається у спеціальних мікросхемах постійного запам'ятовуючого пристрою (ПЗП), утворюючи базову систему введення-виведення BIOS. Програми та дані записуються у ПЗП на етапі виробництва і не можуть бути змінені в процесі експлуатації.

Програми системного рівня відповідають за взаємодію з користувачем. Завдяки йому є можливість вводити дані у обчислювальну систему, керувати її роботою й отримувати результат у зручній формі. Це засоби забезпечення користувацького інтерфейсу, від них залежить зручність та

продуктивність роботи з комп'ютером. Сукупність програмного забезпечення системного рівня утворює ядро операційної системи комп'ютера. Наявність ядра операційної системи - є першою умовою для можливості практичної роботи користувача з обчислювальною системою. Ядро операційної системи виконує такі функції: керування пам'яттю, процесами введення-виведення, файловою системою, організація взаємодії та диспетчеризація процесів, облік використання ресурсів, оброблення команд і т.д.

Сімейство систем (Systems family) – сукупність програмних систем із загальним (незмінним для всіх членів сімейства) і керованим (змінним) набором характеристик, що задовольняють визначені потреби прикладної області (домену). Спосіб виготовлення – інженерія домену (Domain Engineering) або конвеєрне виробництво однотипних ПП за єдиною схемою на основі спеціально розроблених базових членів сімейства й інших готових програмних ресурсів (assets) за допомогою базового процесу або автоматизованої лінійки продукту (Product line).

Програмний проект – унікальний і інтегрований комплекс взаємозалежних заходів, орієнтованих на досягнення цілей і задач об’єкта розробки за визначеними вимогами до строків, бюджету та характеристик очікуваних результатів діяльності від нього.

Складні програмні об'єкти – сукупність взаємопов’язаних цільових об'єктів різних типів, які виконують необхідні функції в складній системі, подані як самостійно розроблені прості та цільові об’єкти або вибрані з репозитарію готових ресурсів.

Отже, основні поняття програмної інженерії пов’язані з діями з реалізації цільових об’єктів програмної інженерії

1.3 ІСТОРИЧНІ УМОВИ ВИНИКНЕННЯ ТА РОЗВИТОКУ ПРОГРАМНОЇ ІНЖЕНЕРІЇ, ЯК ГАЛУЗІ ЗНАНЬ

Перші комп'ютери, що працювали під управлінням програм з'явилися у 40 – 50-х рр. XX ст. та незважаючи на відносно недовгий період програмна інженерія пройшла досить бурхливий розвиток. Етапи розвитку програмної інженерії можна виділяти по-різному. Кожен етап пов'язаний з появою (або усвідомленням) чергової проблеми і знаходженням шляхів і способів вирішення цієї проблеми.

Етапи розвитку беруть свій початок з появою комп'ютерів. Поява програмованих комп'ютерів спричинила до розділення розробників програм на два типи – прикладних і системних програмістів.

До першого типу (прикладні програмісти) увійшли фахівці з прикладних галузей (доменів) – математики, фізики, економіки, освіти, технологій. Вони писали програми мовами високого рівня (Cobol, Fortran) для вирішення тих завдань, що виникають у галузях. Їх діяльність називалася прикладним програмуванням.

До другого типу (системні програмісти) увійшли фахівці, від яких не вимагалося знань доменів, оскільки вони займалися автоматизацією процесів розробки програм. Системні програмісти зазвичай писали програми в машинному коді або мовою АСЕМБЛЕР. Їх діяльність називалася системним програмуванням. Сукупність прикладних і системних програм називається програмним забезпеченням.

Перший етап пов'язаний із 50-ми роками коли потужність комп'ютерів була невеликою, а програмування для них велося, в основному, в машинному коді. На цьому етапі вирішувалися, головним чином, науково-технічні задачі

(розрахунки за формулами). Використовувалася інтуїтивна технологія програмування: майже відразу приступали до складання програми за завданням, при цьому часто завдання кілька разів змінювалося, мінімальна документація оформлялася вже після того, як програма починала працювати. Проте, саме в цей період народилася фундаментальна для технології програмування концепція модульного програмування (для подолання труднощів програмування в машинному коді). З'явилися перші мови програмування високого рівня, з яких лише мова ФОРТРАН пробилася для використання в наступні десятиліття.

У 60-і роки відбувся бурхливий розвиток, який привів до широкого використання мов програмування високого рівня, роль яких в технології програмування явно перебільшувалася. Надія на те, що ці мови вирішать всі проблеми при розробки великих програм, не виправдалася. В результаті підвищення потужності комп'ютерів і накопичення досвіду програмування на мовах високого рівня швидко росла складність вирішуваних на комп'ютерах задач, внаслідок чого виявилася обмеженість мов, що проігнорували модульну організацію програм. І лише ФОРТРАН, що зберіг можливість модульного програмування, пройшов у наступні десятиліття. Крім того, стало зрозуміло, що важливим є не лише те, якою мовою ми програмуємо, але і те, як ми програмуємо. Це було вже початком серйозних роздумів над методологією і технологією програмування.

У 70-і роки набули широкого поширення інформаційні системи і бази даних. Цьому сприяла дуже важлива подія, що відбулася в середині 70-х років: вартість зберігання одного біта інформації на комп'ютерних носіях стала меншою, ніж на традиційних. Інтенсивно розвивалася технологія програмування, що дало змогу дати обґрунтування і можливість для широкого впровадження низхідної розробки і

структурного програмування, відбувся розвиток абстрактних типів даних і модульного програмування, дослідження проблем забезпечення надійності і мобільності ПЗ, створення методики управління колективною розробкою ПЗ, поява інструментальних програмних засобів підтримки технології програмування. Саме в ці роки стало ясно, що із зростанням складності вирішуваних за допомогою комп'ютерів задач неймовірно зростає вартість розробки програм. Причому, якщо вартість апаратури зростає помірними темпами (а інколи і спадає), то з вартістю розробки програм нічого зробити не вдається. Отже, на перший план вийшло питання про те, як оптимізувати процес розробки ПЗ.

80-і роки характеризуються широким впровадженням ПК у всі сфери людської діяльності і тим самим створенням різноманітного контингенту користувачів ПЗ. Це привело до бурхливого розвитку інтерфейсів користувача і створення чіткої концепції якості ПЗ. З'являються мови програмування (наприклад, Ада), що враховують вимоги технології програмування. Розвиваються методи і мови специфікації ПЗ. Виходить на передові позиції об'єктний підхід до розробки ПЗ. Створюються різні інструментальні середовища розробки і супроводу ПЗ. Розвивається концепція комп'ютерних мереж.

90-і роки знаменні широким обхватом людського суспільства міжнародною комп'ютерною мережею, ПК почали підключатися до неї як термінали. Це поставило ряд проблем регулювання доступу до комп'ютерно-мережевої інформації (як технологічного, так юридичного і етичного характеру). Гостро встала проблема захисту комп'ютерної інформації і повідомлень, що передаються в мережі. Стали бурхливо розвиватися комп'ютерна технологія (CASE-технологія) розробки ПЗ і пов'язані з нею формальні методи специфікації програм.

Наведений екскурс в історію підкреслює розвиток

програмної інженерії, що виростає з програмування. Одначе були й інші технологічні віяння, що зіграли помітну роль у становленні галузі. Найбільш сильним виявився вплив змін цінового балансу між апаратним і програмним забезпеченням комп'ютерів. Також, кардинальні зміни в галузі створення програмного забезпечення (ПЗ) були обумовлені й швидким зростанням ринкового програмного продукту – тієї частини розроблених програм, яка отримувалася користувачем у вигляді готових до експлуатації пакетів програм різного призначення. Не дивлячись на те, що значна частина створюваного програмного забезпечення не доводиться до комерційного використання, тобто не виходить за межі фірми-розробника, вона представляє велику цінність для подальших розробок і для накопичення досвіду і знань. Вже до початку 80-х років тільки в США було створено ПЗ на сотні мільярдів доларів.

Впровадження комп'ютерних технологій в різноманітні сфери людської діяльності привело до виникнення і бурхливого розвитку нової галузі суспільного виробництва – промисловості обробки даних, сумарний обсяг продажів продукції в якій швидко залишив позаду всі традиційні галузі промисловості. Перерозподіл кількості працюючих у сфері матеріального виробництва привів до того, що в найрозвиненіших країнах більше половини працівників виявилися зайнятою обробкою інформації.

Таким чином, ми можемо очікувати продовження зростання значення програмної інженерії з цілого ряду причин. По-перше, світові витрати на програмне забезпечення постійно ростуть. Наприклад, у 1985 р. вони складали 140 млрд. доларів, а у 2000 р. перевищували 800 млрд. Уже один цей факт гарантує, що програмна інженерія ростиме як окрема галузь комп'ютерних знань. По-друге, програмне забезпечення чимраз ширше проникає в наше суспільство: дедалі більше програм використовується для управління

найважливішими функціями самих різних машин, таких як літаки і медичні прилади, а також для підтримки глобальних процесів, подібних електронній комерції. Цей факт гарантує зростаючий інтерес суспільства до надійного програмного забезпечення, до розширення законодавчої основи для відповідних стандартів, вимог і процедур сертифікації.

Отже, історичними умовами виникнення та розвитку програмної інженерії була і є потреба в реалізації систем, що можуть вирішувати задачі із різних сфер людської діяльності

1.4 ОСНОВНІ ХАРАКТЕРИСТИКИ ТА ФАЗИ РОЗВИТКУ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Програмна інженерія як інженерна галузь характеризується діяльністю, заснованою на таких принципах:

- ефективність – результати одержують заданими ресурсами, і вони відповідають заданим вимогам і стандартам;
- практичність – результати мають конкретних замовників;
- фундаментальність – результати одержують на основі знань фундаментальних наук;
- наслідуваність (повторне використовування) – результати одержують на основі накопиченого досвіду, виключаючи діяльність «з нуля»;
- відчутність – результати є відчутними продуктами, які можна застосовувати, руйнувати і досліджувати за допомогою емпіричних методів пізнання;
- супроводжуваність – результати, знаходячись в експлуатації, обов’язково супроводжуються (обслуговуються).

У процесі розвитку людства з’явилося багато інженерних галузей, але їх становлення проходило один і той самий шлях, який Сидоров М.О. розподіляє на три фази (рис. 1.1).

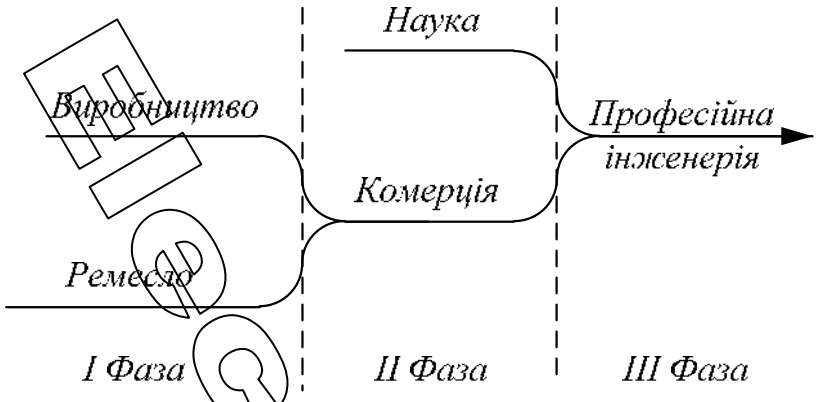


Рис. 1.1 – Фази розвитку інженерної галузі

Кожна фаза відрізняється виконавцями, ресурсами, методами реалізації і використання продуктів галузі (табл. 1.1):

– *фаза I:*
  – виконавці – віртуози і талановиті одинаки;
  – ресурси – інтуіція і власний досвід;
  – методи – випадкова передача досвіду, екстравагантне застосування матеріалів;
  – використання – виробництво для себе;
– *фаза II:*
  – виконавці – майстерні виробники;
  – ресурси – окремі інструменти;
  – методи – механічний тренінг, облік економічних чинників у виборі матеріалів;
  – використання – виробництво для продажу, утворення ринку;
– *фаза III:*
  – виконавці – освічені професіонали;
  – ресурси – машини і комплекси, які використовуються в технологіях;
  – методи – теоретичні і емпіричні, передача знань шляхом диференційованого навчання, супровід;
  – використання – сегментація ринку.

Таблиця 1.1 – Характеристики фаз розвитку

Фаза	Характеристики			
	Виконавці	Ресурси	Методи	Використання
Фаза I	Віртуози і талановиті одинаки	Інтуїція, груба сила і власний досвід	Випадкова передача досвіду, екстравагантне застосування матеріалів	Виробництво для себе
Фаза II	Майстерні виробники	Окремі інструменти	Механічний тренінг, облік економічних чинників у виборі матеріалів	Виробництво для продажу, утворення ринку
Фаза III	Освічені професіонали	Машини і комплекси, які використо-вуються в технологіях	Теоретичні та емпіричні, передача знань шляхом диференційованого навчання, супровід	Сегментація ринку

Характеристика вказаних фаз в табл. 1.1. для інженерії програмного забезпечення наводиться в табл. 1.2.

Таблиця 1.2 – Характеристики фаз розвитку

Характеристика	Фаза (початок		
	II (з 1960 р.)	п (з 1970 р.)	III (з 1980 р.)
Особливості програмування	Програмування «абияк»	Програмування «в малому»	Програмування «у великому»
Підготовка кадрів	Практично відсутня	Прикладна математика	Комп'ютерні науки
Ресурси	Асемблери, машинні коди	Транслятори, лінкери, завантажувачі, програмні системи	Середовища розробки програм
Технології	Відсутні	*HIPO, формалізовані технічні завдання	Системи *PSL, *SREM, *SADT
Економіка	Відсутня	Інтуїтивна	Системи *SCEP. *SLIM
Ринок	Відсутній, замовні програми	Виробництво для продажу	Сегментація ринку

*HIPO (Hierarchical-Input-Processing-Output) – технологія розробки програмного забезпечення на базі ідеї структурного програмування.
*PSL (Problem Statement Language) – мова постановки задач.
*SREM (Software Requirements Engineering Methodology) – методологія розробки технічних вимог до програмного забезпечення.
*SADT (Structured Analysis and Design Technique) – структурний аналіз і проектування.
*SCEP (Simple Certificate Enrolment Protocol) – протокол реєстрації сертифікатів.
*SLIM (ervice Level Improvement Method) – метод поліпшення рівня обслуговування.

Отже, характеристики та фази розвитку програмної інженерії тісно пов'язані з умови виникнення та розвитку програмної інженерії

1.5 МЕТОДИ ТА ІНСТРУМЕНТИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Методи програмної інженерії забезпечують проектування, реалізацію і виконання програмного забезпечення. Вони накладають деякі обмеження на процеси програмної інженерії у зв'язку з особливостями застосування нотацій і процедур, а також забезпечують оцінку і перевірку процесів і продуктів, що реалізуються при створенні програмного забезпечення.

Методи інженерії ПЗ – це евристичні методи (heuristic methods), формальні методи (formal methods) і методи прототипування (prototyping methods).

Евристичні методи містять у собі: структурні методи, засновані на функціональній парадигмі; методи, орієнтовані на структури даних, якими маніпулює ПЗ; об'єктно-орієнтовані методи, що розглядають предметну область як колекцію об'єктів; методи, орієнтовані на конкретну область застосування, наприклад, на системи реального часу, безпеки та ін. Формальні методи засновані на формальних специфікаціях, аналізі, доведенні і верифікації програм. Специфікація записується мовою, синтаксис і семантика якої

визначені формально і засновані на математичних концепціях (алгебрі, теорії множин, логіці). Розрізняються наступні категорії формальних методів:

– мови і нотації специфікації (specification languages and notations), орієнтовані на модель, властивості і поведінку;
– уточнення специфікації (refinement specification) шляхом трансформації в кінцевий результат, близький до кінцевого програмного продукту, що виконується;
– методи верифікації/доведення (verification/proving properties), що використовують твердження (теореми), перед- і постумови, формально описуються і застосовуються для встановлення правильності специфікації програм.

Методи прототипування (Prototyping Methods) засновані на використанні прототипу ПЗ для моделювання на ньому завдань нової системи і базуються на:

– стилях прототипування, що уособлюють тривалість використання прототипів, наприклад, стиль створення тимчасово використовуваних прототипів (throw away),
– моделях еволюційного прототипування
– перетворення прототипу в кінцевий продукт і розроблення специфікацій, відповідно до якої він виконується;
– техніках оцінки/дослідження (evaluation) результатів прототипування.

Інструменти забезпечують програмну підтримку окремих методів інженерії програмного забезпечення для автоматизованого виконання задач пов’язаних з процесами життєвого циклу програмного забезпечення і містять у собі множину інструментів, що охоплюють усі процеси життєвого

циклу програмного забезпечення. До інструментів програмної інженерії відносяться:

1. Інструменти роботи з вимогами (Software Requirements Tools) – це:

– інструменти розробки (Requirement Development) і керування вимогами (Requirement Management), орієнтовані на аналіз, збирання, специфікування і перевірку вимог;

– інструменти трасування вимог (Requirement traceability tools) є невід'ємною частиною роботи з вимогами, їх функціональний зміст залежить від складності проектів і рівня зрілості процесів.

2. Інструменти проектування (Software Design Tools) – це інструменти для створення ПЗ із застосуванням базових нотацій (структурної SADT/IDEF, моделювання UML і т.п.).

3. Інструменти конструювання ПЗ (Software Construction Tools) – це інструменти для трансляції і об'єднання програм. До них належать:

– редактори програм (program editors) і програми редагування загального призначення;

– компілятори і генератори коду (compilers and code generators) як самостійні засоби об'єднання програмних компонентів в інтегрованому середовищі для одержання вихідного продукту з використанням препроцесорів, складальників, завантажників і ін.;

– інтерпретатори (interpreters), які забезпечують контрольоване виконання програм за їх описом. Намітилася тенденція злиття інтерпретаторів і компіляторів (наприклад, Java, в .NET);

– програми призначені для перевірки правильності опису вихідних програм і усунення помилок (debuggers);

– інтегроване середовище розробки (IDE – integrated

development environment) та бібліотеки компонентів (libraries components), що є утворюють середовище виконання процесу розроблення ПЗ;

– програмні платформи (Java, J2EE і Microsoft .NET) і платформи для розподілених обчислень (CORBA і WebServices, тощо).

4. Інструменти тестування (Software Testing Tools) – це:

– генератори тестів (test generators), що допомагають у розробці сценаріїв тестування;

– засоби виконання тестів (test execution frameworks), які забезпечують виконання тестових сценаріїв і відслідковують поведінку об'єктів тестування;

– інструменти оцінки тестів (test evaluation tools), які підтримують оцінювання результатів виконання тестів і ступеня відповідності поведінки тестованого об'єкта очікуваній поведінки;

– засоби керування тестами (test management tools), які забезпечують інженерне керування процесом тестування ПЗ;

– інструменти аналізу продуктивності (performance analysis tools), кількісної її оцінки та оцінки поводження програм у процесі виконання.

5. Інструменти супроводу (Software Maintenance Tools) містять у собі:

– інструменти полегшення розуміння (comprehension tools) програм, наприклад, різні засоби візуалізації;

– інструменти реінженерії (reengineering tools) підтримують діяльність з перетворення програм і зворотної інженерії (reverse engineering) для відновлення (артефактів, специфікації, архітектури) застарілого ПЗ або генерації нового продукту.

6. Інструменти конфігураційного керування (Software Configuration Management Tools) – це:

– інструменти відстеження (tracking) дефектів; – інструменти керування версіями;

– інструменти керування складанням, випуском версії (конфігурації) продукту та його інсталяції.

7. Інструменти керування інженерною діяльністю (Software Engineering Management Tools) діляться на:

– інструменти планування і відстеження ходу проектів, кількісної оцінки зусиль і вартості робіт у проекті (наприклад, Microsoft Project 2003);

– інструменти керування ризиками, які використовуються для ідентифікації, моніторингу ризиків і оцінки нанесеного ушкодження;

– інструменти кількісної оцінки властивостей ПЗ шляхом вимірювань і розрахунків остаточного значення надійності і якості.

8. Інструменти підтримки процесів (Software Engineering Process Tools) розділені на:

– інструменти моделювання та опису моделей ПЗ (наприклад, UML і його інструменти);

– інструменти керування програмними проектами (наприклад, Microsoft Project);

– інструменти керування конфігурацією для підтримки версій і всіх артефактів проекту.

9. Інструменти забезпечення якості (Software Quality Tools) діляться на дві категорій:

– інструменти інспектування для підтримки перегляду (review) і аудиту;

– інструменти статичного аналізу артефактів, даних, потоків робіт і перевірки їх властивостей на відповідність показникам.

Додаткові аспекти інструментального забезпечення

(Miscellaneous Tool Issues) стосуються:

– техніки інтеграції інструментів (платформ, представлень, процесів, даних) для їх природного сполучення в інтегрованому середовищі;
– мета інструментів для генерації інших інструментів для програмного забезпечення;
– оцінки інструментів при їх еволюції.

Отже, методи та засоби програмної інженерії – це засоби, що націлені на створення ефективного програмного забезпечення найбільш оптимальним шляхом

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення програмної інженерії?
2. Охарактеризуйте програмну інженерію, як інженерну діяльність.
3. Охарактеризуйте програмну інженерію, як наукову діяльність.
4. Дайте визначення програмній (прикладній) системі
5. Дайте визначення програмному забезпеченню (software)
6. Які види програмного забезпечення існують.
7. Охарактеризуйте основні типи програмного забезпечення.
8. В чому відмінності між основними типами програмного забезпечення
9. Дайте визначення сімейства систем
10. Дайте визначення програмному проекту
11. Дайте визначення складним програмним об'єктам

РОЗДІЛ 2

ОГЛЯД ОСВІТНЬОГО СТАНДАРТУ З ПРОГРАМНОЇ ІНЖЕНЕРІЇ – SWEBOK V3

При вивченні цієї теми важливо пам’ятати, що області знань приведені в стандарті ISO/IEC TR 19759:2015 Software Engineering – Guide to the software engineering body of knowledge ***(SWEBOK V3)*** описують загальноприйняті знання про програмне забезпечення, засоби його створення, тестування, розробки та інші знання з базових та суміжних областей знань, що знаходять використання на всіх етапах життєвого циклу програмного забезпечення.

Зміст розділу

2.10 Моделі та методи програмної інженерії (Software Engineering Models and Methods).
2.11 Якість програмного забезпечення (Software Quality).
2.12 Професійна практика програмної інженерії (Software Engineering Professional Practice).
2.13 Економічні аспекти програмної інженерії (Software Engineering Economics).
2.14 Основи обчислювальних технологій програмної інженерії (Computing Foundations).
2.15 Математичні засади програмної інженерії (Mathematical Foundations).
2.16 Основи інженерної діяльності в програмній інженерії (Engineering Foundations)

2.1 ЦІЛІ ТА ЗМІСТ СТАНДАРТУ SWEBOK V3

Стандарт ISO/IEC TR 19759:2015 Software Engineering – Guide to the Software Engineering Body of Knowledge (SWEBOK) в галузі програмної інженерії є одним із базових науково-технічних документів, структура і зміст якого базуються на професійній думці провідних закордонних та вітчизняних спеціалістів в галузі програмної інженерії. Області знань, що зібрані та структуровані у SWEBOK узгоджуються із сучасними стандартизованими процесами життєвих циклів програмного забезпечення відповідно до стандарту ISO/IEC 12207:2008 Systems and software engineering – Software life cycle processes.

Опис областей знань з інженерії програмного забезпечення у SWEBOK побудовано за ієрархічним принципом, який дозволяє отримати визначення й систематизацію тих аспектів діяльності, які є підґрунтям професійної підготовки інженера-програміста. Така ієрархічна побудова звичайно нараховує два-три рівня деталізації, прийнятих для ідентифікації тих чи інших загальновизнаних аспектів програмної інженерії. Ієрархічний принцип прийнятий у SWEBOK деталізовано тільки до того рівня, який потрібен для розуміння загальновизнаних аспектів програмної інженерії й дає можливість пошуку джерел інформації з необхідної галузі знань інженером-програмістом.

Області знань, які розглядаються в SWEBOK були створені для вирішення наступних задач:

- просування єдиного уявлення про програмну інженерію на основі консенсуса при створенні структури областей знань SWEBOK;

- визначення місця і межі програмної інженерії по відношенню до суміжних галузей знань;

- визначення базового змісту й характеристики дисципліни програмної інженерії;

- структурування областей знань з програмної інженерії;

- створення бази для розробки навчальних планів і матеріалів, ліцензування та індивідуальної сертифікації в галузі програмної інженерії (Software Engineering 2016 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering).

Базова версія SWEBOK описувала лише 10 областей знань із галузі програмної інженерії, починаючи з третьої версії SWEBOK (SWEBOK Guide V3.0) перелік областей знань було істотно розширено до 15 (рис. 2.1), а області знань, що перейшли до нової версії були істотно доповнені та зміні відповідно до сучасних тенденцій в галузі інженерії програмного забезпечення.

Також, SWEBOK містить інформацію з суміжних областей, які тісно пов'язані із інженерією програмного забезпечення, а саме:

1. Комп'ютерні технології (Computer Engineering).
2. Комп'ютерні науки (Computer Science).
3. Менеджмент (General Management).
4. Математика (Mathematics).
5. Управління проектами (Project Management).
6. Управління якістю (Quality Management).
7. Системотехніка (Systems Engineering).

Області знань визначають необхідний мінімум знань інженера-програміста. Області знань не представляють етапи реалізації та процеси підтримки еволюції програмного проекту, а тільки надають знання з можливих і допустимих методологій виконання аналізу та оцінки вимог та рішень, що використовуються при розробці програмного продукту.

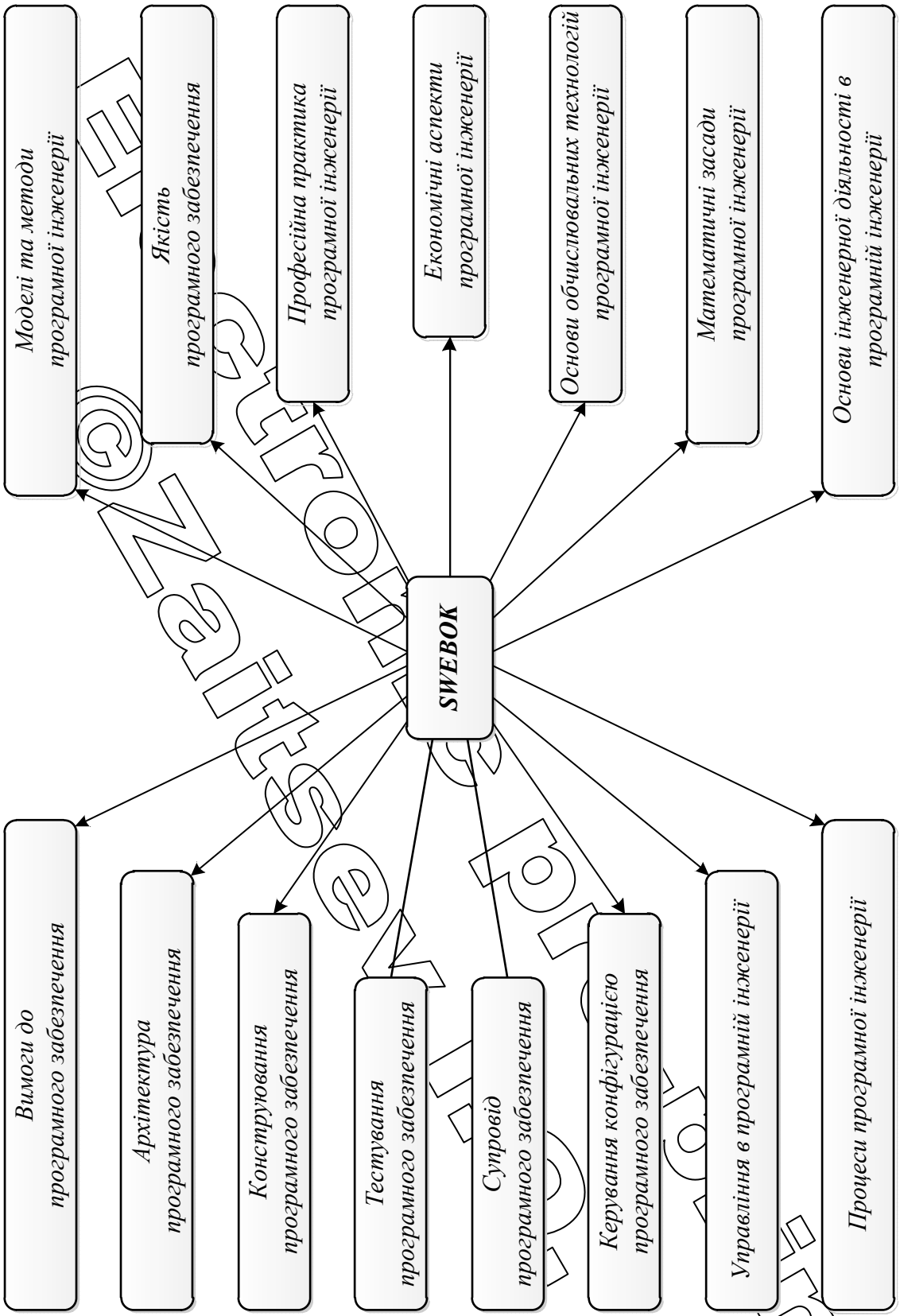


Рис. 2.1 – Структура освітнього стандарту SWEBOK V3

2.2 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE REQUIREMENTS)

"Вимоги до програмного забезпечення (Software Requirements)" - це розділ програмної інженерії присвячений вивченню процесів пов'язаних із збором та управлінням вимогами до програмного забезпечення, які визначають повною мірою сукупність всіх властивостей, які повинно мати програмне забезпечення, а також адекватному визначенню функцій, умов і обмежень прийнятих для оптимальної роботи програмного забезпечення (обсягів даних, технічного забезпечення і середовища його виконання).

Вимоги виражають потреби людей (замовників, користувачів, розробників), зацікавлених у створенні ПЗ. Замовник і розробник спільно формулюють вимоги, аналізують, переглядають, визначають необхідні обмеження і умови, а також описують їх. Розрізняють вимоги до продукту і до процесу, а також функціональні, не функціональні і системні вимоги. Вимоги до продукту і до процесу визначають умови виконання і режими роботи ПЗ в операційному середовищі, обмеження на структуру і пам'ять комп'ютерів та принципи взаємодії програм.

Розділ програмної інженерії "Вимоги до програмного забезпечення (Software Requirements)" складається із 8 основних тем (рис 2.2), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються методів та засобів збору вимог до програмного забезпечення, їх специфікації, документуванню, аналізу, виявлення суперечностей та неповноти вимог (детально розглядається в розділі 4).

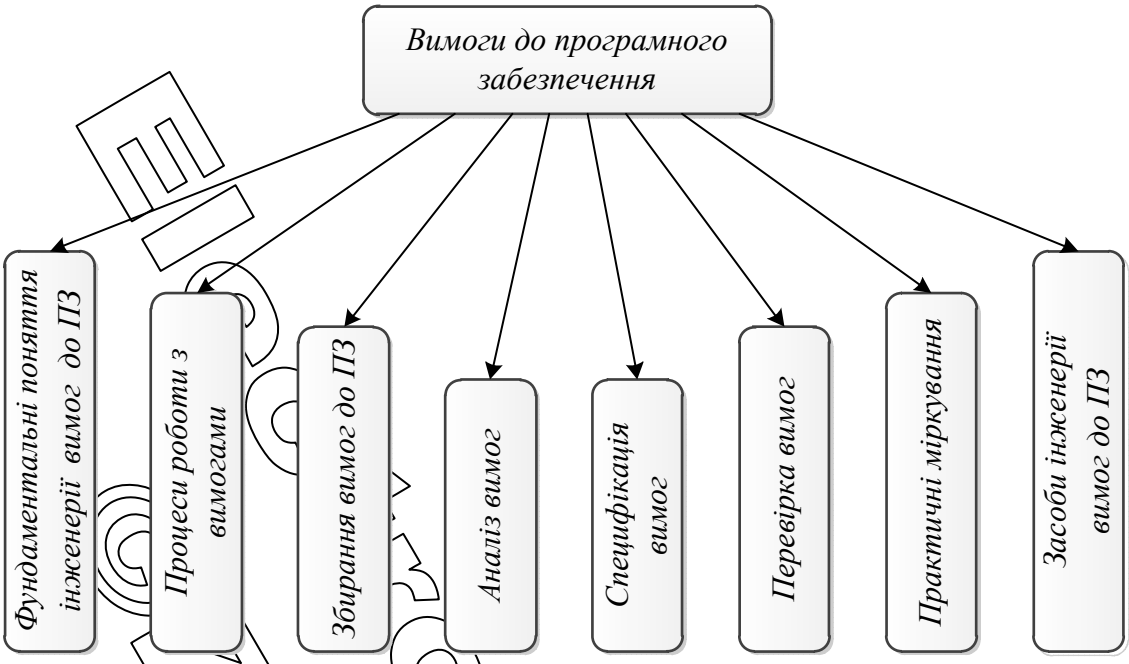


Рис. 2.2 – Склад розділу
"Вимоги до програмного забезпечення
(Software Requirements)"

Фундаментальні поняття інженерії вимог до програмного забезпечення (Software Requirements Fundamentals) – поняття та визначення, що використовуються в процесах пов’язаних з вимогами до програмних продуктів, описі основних типів вимог і їх відмінностях для програмного продукту, програмного забезпечення та процесів, що супроводжують всі етапи життєвого циклу програмного продукту.

Процес роботи з вимогами (Requirements Process). Описує процеси, що стосуються питань роботи з вимогами, і певною мірою "зшиває" в єдине ціле решту тем, що присвячені вимогам до програмного забезпечення та надає розуміння того, що таке процеси роботи з вимогами.

Збирання вимог (Requirements Elicitation) – є початковим і невід'ємним етапом процесу розробки програмного забезпечення.

Висвітлює питання збору вимог як з точки зору організації

процесу, так і визначення джерел, звідки надходять вимоги. Це перша стадія побудови бачення автоматизованої проблемної області. Ідентифікація зацікавлених осіб, їх взаємодії, виконуваних ними бізнес-процесів - все це є ключовими питаннями, без чіткої й однозначної відповіді на які навіть неможливо уявити успішність завершення програмного проекту.

Один з ключових принципів програмної інженерії полягає в забезпеченні взаємодії між користувачами та інженерами. Перш ніж починається розробка програмного забезпечення, саме фахівці "за вимогами" - аналітики перекидають той самий "місток" між замовниками та виконавцями, який задає той рівень комунікацій і взаєморозуміння між ними, який необхідний для вирішення завдань проекту.

Він полягає у визначенні набору функцій, які необхідно реалізувати в продукті. Збір вимог реалізується:

– в спілкуванні з замовником;

– за допомогою мозкових штурмів розробників;

– аналізу мети і задач проекту, що формулює замовник майбутньої системи, і які

повинні бути реалізовані розробниками;

– спілкуванні з колективом розробників, які виконують реалізацію функцій системи.

Результатом є формування набору вимог до системи, іменованої у вітчизняній практиці технічного завдання.

Аналіз вимог (Requirements Analysis) починається після обговорення проблематики проекту та присвячена процесам аналізу вимог, тобто трансформації інформації, отриманої від користувачів (та інших зацікавлених осіб) у чіткі і однозначні певні вимоги, передані інженерам для реалізації в програмному коді. Аналіз вимог включає:

• виявлення та вирішення конфліктів між вимогами;

• визначення меж завдання, розв'язуваної створюваним

програмним забезпеченням; в загальному випадку - визначення "scope" (або "bounds"), меж та змісту програмного проекту;

• деталізацію системних вимог для встановлення програмних вимог;

Тобто діяльність з виявлення, аналізу, специфікації та перевірки вимог має бути пристосована та враховувати різні типи проектів та обмежень, що в них існують. Аналіз вимог є критичним для успішної розробки проекту.[1] Вимоги мають бути задокументованими, вимірними, тестовними, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи. Вимоги можуть бути архітектурними, структурними, поведінковими, функціональними, та не функціональними.

Специфікація вимог (Requirements Specification) – це формалізований опис функціональних, нефункціональних і технічних вимог, вимог до характеристик якості, до структури програмного забезпечення та принципів взаємодії з його компонентами.

Для складних програмних систем(див. розділ 5), існує цілий комплекс специфікацій, документів, які є результатом аналізу вимог, їх моделювання та архітектурного проектування. Ці документи систематично аналізуються, в них вносяться зміни, вони переглядаються і затверджуються. Найчастіше, для опису комплексних проектів (в частині вимог) використовуються три основні документи (специфікації):

• Визначення системи (system definition)
• Системні вимоги (system requirements)
• Програмні вимоги (software requirements)

Перевірка вимог (Requirements Validation) – це перевірка вимог для переконання, що вони визначають саме дану систему. Мета полягає в тому, щоб визначити будь-які

проблеми, перш ніж вимоги до програмного забезпечення будуть реалізовані. Документи, що містять вимоги можуть підлягати процедурам валідації та верифікації. Вимоги можуть бути перевірені, щоб забезпечити розуміння вимогами інженера програмного забезпечення; важливо також перевірити, чи документ вимог відповідає стандартам компанії, і що він є зрозумілим, послідовним та повним. У випадках, коли стандартні або термінологічні документи компанії не відповідають загальноприйнятим стандартам, узгодження та доповнення до документів повинні бути узгоджені між собою.

Практичні міркування (Practical Considerations) – комплекс процесів управління вимогами, що охоплює весь життєвий цикл програмного забезпечення, а також опис послідовності дій потрібних для роботи з вимогами на всіх етапах життєвого циклу програмного забезпечення. Управління змінами та супровід, підтримка актуальності вимог і їх реалізації - ключ до успішних процесів програмної інженерії. Управління змінами, концепцією, баченням продукту не може бути хаотичним - історія індустрії однозначно це доказує. Тому ставлення до управління вимогами як до постійно діючого бізнес-процесу - абсолютно обґрунтований підхід, що вимагає застосування певних практик. В іншому випадку, розробники практично гарантовано зіткнуться із вкрай негативними наслідками.

Засоби інженерії вимог до програмного забезпечення (Software Requirements Tools) – засоби, що використовуються для вирішення задач моделювання та керування вимогами. Засоби керування вимогами – це насамперед програмна документація за допомогою якої можливо відстеження та управління змінами в проекті. Відстеження та управління змінами є практично можливими, якщо вони підтримуються та документуються.

2.3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE DESIGN)

"Архітектура програмного забезпечення (Software Design)" - це розділ програмної інженерії присвячений процесам визначення архітектури, набору компонентів, їх інтерфейсів, інших характеристик програмної системи, а також кінцевого складу програмного продукту. Архітектура повинна реалізовуватися з точки зору найкращої відповідності вимогам до програмної системи, що створюється відповідно до принципу "форма відповідає функції".

Дослідження архітектури ПЗ намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію.

Проектування архітектури ПЗ — це процес розроблення, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури системи може проводитися різними методами (стандартизованим, об'єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об'єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо).

Розділ програмної інженерії "Вимоги до програмного

забезпечення (Software Requirements)" складається із 8 основних тем (рис 2.3), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються архітектури програмного забезпечення.

Базові питання проектування архітектури ПЗ (Software Design Fundamentals) – це методологія проектування архітектури за допомогою різних методів (об'єктного, компонентного й ін.), процеси ЖЦ (стандарт ISO/IEC 12207) і техніки – декомпозиція, абстракція, інкапсуляція й ін., що є основою для розуміння ролі та обсягу архітектури в розробці програмного забезпечення. На початкових стадіях проектування предметна область декомпозується на окремі об'єкти (при об'єктно-орієнтованому проектуванні) або на компоненти (при компонентному проектуванні). Для подання архітектури програмного забезпечення вибираються відповідні артефакти (нотації, діаграми, блок-схеми і методи).

Базові питання проектування (Key Issues in Software Design) – це декомпозиція програм на функціональні компоненти для незалежного і одночасного їхнього виконання, розподіл компонентів у середовищі функціонування і їх взаємодія між собою, забезпечення якості і живучості системи й ін.

Структура та архітектура програмного забезпечення (Software Structure and Architecture) – набір структур, необхідних для розуміння системи, які містять елементи програмного забезпечення, алгоритми взаємодії між елементами. Проте в середині 1990-х років архітектура програмного забезпечення почала виникати як більш широка дисципліна, яка включала вивчення програмних структур та архітектури більш загальним способом. Це породило ряд цікавих понять про розробку програмного забезпечення на різних рівнях абстракції. Останнє, може бути використано для розробки сімейства програм.

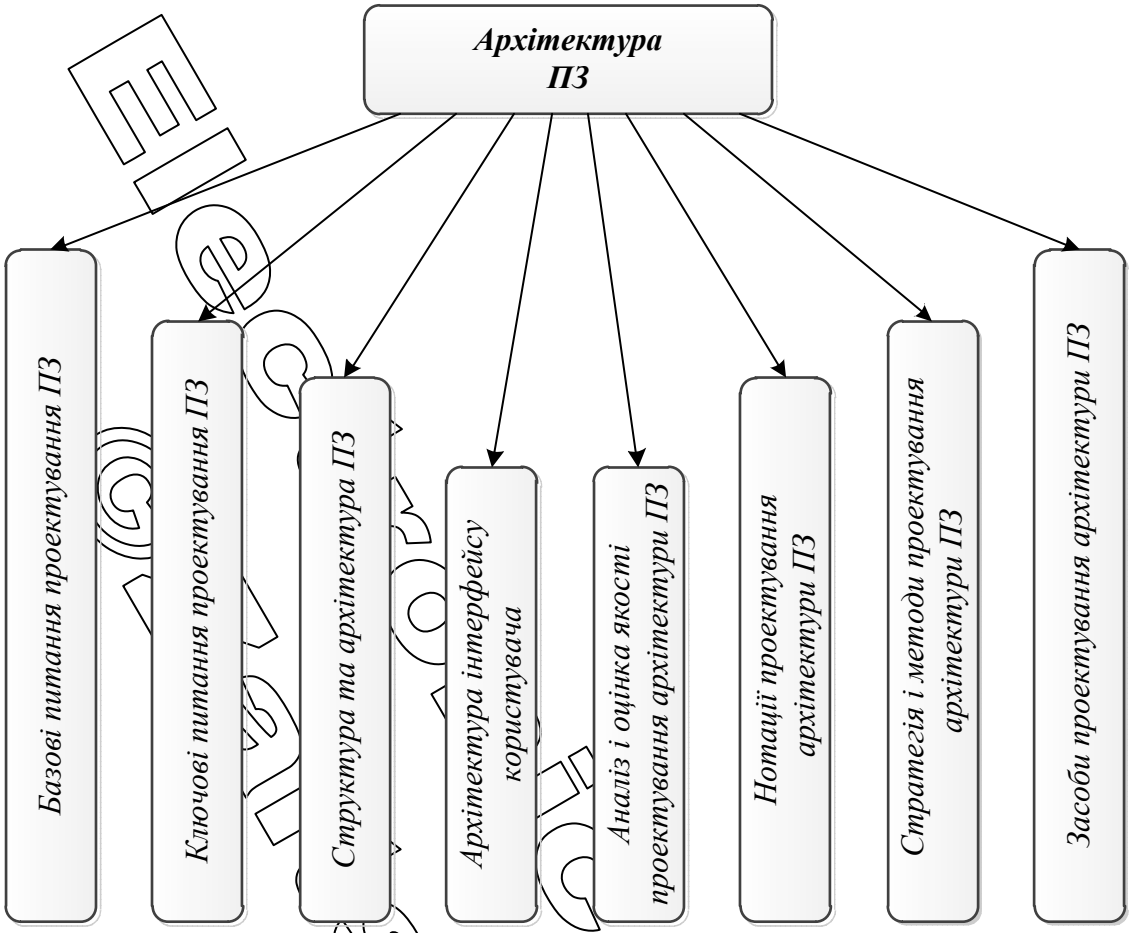


Рис.2.3 – Склад розділу
"Архітектура програмного забезпечення
(Software Design)"

Архітектура інтерфейсу користувача(User Interface Design) є важливою частиною процесу розробки архітектури програмного забезпечення та програмного забезпечення в цілому. Дизайн інтерфейсу користувача повинен гарантувати, що взаємодія між людиною та апаратно-програмним забезпеченням забезпечує ефективне функціонування та управління машиною.

Аналіз і оцінка якості проектування ПЗ (Software Design Quality Analysis and Evaluation) – це заходи щодо аналізу сформульованих у вимогах атрибутів якості, функцій, структури ПЗ, з перевірки якості результатів проектування за

допомогою метрик (функціональних, структурних і ін.) і методів моделювання і прототипування.

Нотації проектування (Software Design Notations) дозволяють представити опис об'єкта (елемента) ПЗ і його структуру, а також поведінку системи за цим об’єктом. Існує два типи нотацій: структурна, поведінкова, та множина їх різних представлень.

Структурні нотації – це структурне, блок-схемне або текстове подання аспектів проектування структури ПЗ з об'єктів, компонентів, їх інтерфейсів і взаємозв'язків. До нотацій відносять формальні мови специфікацій і проектування: ADL (Architecture Description Language), UML (Unified Modeling Language), ERD (Entity–Relation Diagrams), IDL (Interface Description Language) тощо. Нотації містять у собі мовний опис архітектури й інтерфейсу, діаграм класів і об'єктів, діаграм сутність-зв'язок, конфігурації компонентів, схем розгортання, а також структурні діаграми, що задають у наочному вигляді оператори циклу, розгалуження, вибору і послідовності.

Поведінкові нотації відбивають динамічний аспект роботи системи та її компонентів. Ними можуть бути діаграми потоків даних (Data Flow), діяльності (Activity), кооперації (Colloboration), послідовності (Sequence), таблиці прийняття рішень (Decision Tables), передумови і постумови (Pre-Post Conditions), формальні мови специфікації (Z, VDM, RAISE) і проектування.

Стратегія і методи проектування ПЗ (Software Design Strategies and Methods) – різні загальні стратегії та методи, які допоможуть керувати процесом проектування архітектури програмного забезпечення. На відміну від загальних стратегій, методи є більш конкретними, оскільки вони, як правило, забезпечують набір позначень, які будуть використовуватися з методом, опис процесу, що буде використовуватися під час

реалізації методу. До стратегій відносять: проектування вгору, вниз, абстрагування, використання каркасів і ін. Методи є функціонально-орієнтовані, структурні, які базуються на структурному аналізі, структурних картах, діаграмах потоків даних й ін. Вони орієнтовані на ідентифікацію функцій і їх уточнення знизу-вгору, після цього уточнюються діаграми потоків даних і проводиться опис процесів.

В об’єктно-орієнтованому проектуванні ключову роль відіграє спадкування, поліморфізм й інкапсуляція, а також абстрактні структури даних і відображення об'єктів. Підходи, орієнтовані на структури даних, базуються на методі Джексона і використовуються для подання вхідних і вихідних даних структурними діаграмами.

Метод UML призначений для опису сценаріїв роботи проекту у наочному діаграмному вигляді. Компонентне проектування грунтується на використанні готових компонентів (reuse) з визначеними інтерфейсами і їх інтеграції в конфігурацію, як основи розгортання компонентної системи для її функціонування в операційному середовищі. Формальні методи опису програм грунтуються на специфікаціях, аксіомах, описах деяких попередніх умов, твердженнях і постумовах, що визначають заключну умову одержання програмою правильного результату. Специфікація функцій і даних, якими ці функції оперують, а також умови і твердження – основа доведення правильності програми.

Засоби проектування архітектури програмного забезпечення (Software design tools) – це інструменти для підтримки створення артефактів архітектури програмного забезпечення під час розробки. Вони підтримують частину чи цілі види діяльності з:

• переведення вимог до дизайнерського бачення;

• підтримки функціональних компонентів та їх інтерфейсу;

• здійснення удосконалення та розділення евристики програмного забезпечення;

- надання рекомендації щодо оцінки якості архітектури та програмного забезпечення.

2.4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE CONSTRUCTION)

"Конструювання програмного забезпечення (Software Construction)" - це розділ програмної інженерії присвячений вивченню процесів створення ПЗ з конструкцій (блоків, операторів, функцій) і його перевірка методами верифікації і тестування. До інструментів конструювання ПЗ віднесені мови конструювання, програмні методи й інструментальні системи (компілятори, СКБД, генератори звітів, системи керування версіями, конфігурацією, тестуванням й ін.). До формальних засобів опису процесу конструювання ПЗ, взаємозв'язків між людиною і комп'ютером з урахуванням середовища оточення віднесені структурні діаграми Джексона.

Розділ програмної інженерії "Конструювання програмного забезпечення (Software Construction)" складається із 5 основних тем (рис 2.4), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються методів та засобів конструювання програмного забезпечення.

Основи конструювання програмного забезпечення (Software Construction Fundamentals) – поняття та визначення, що використовуються при конструюванні програмного забезпечення на включають знання з наступних розділів: мінімізації складності програмного забезпечення; очікування змін(еволюція); конструювання програмного

забезпечення з можливістю перевірки; повторне використання елементів програмного забезпечення; стандарти у конструюванні Перші чотири концепції застосовуються не тільки до конструювання, але і проектування, і лежать в основі сучасних методологій управління життєвим циклом програмних засобів.

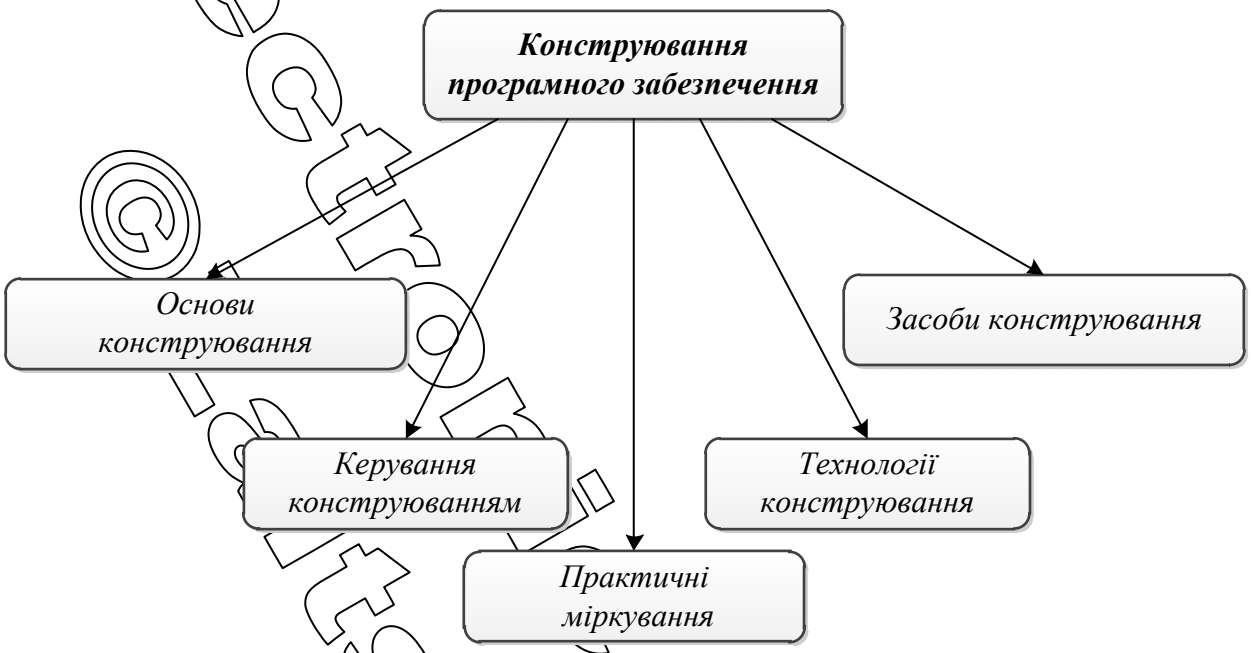


Рис.2.4 – Склад розділу
"Конструювання програмного забезпечення
(Software Construction)"

Керування конструюванням (Managing Construction) – це керування процесом конструювання ПЗ, планування, оцінка виконання плану і розроблення заходів щодо внесення змін.

Планування – це визначення порядку операцій, термінів і рівня виконання заданих умов у процесі конструкторської діяльності за моделлю ЖЦ, що містить у собі задачі і дії зі створення, перевірки й оцінки показників якості. Виконавці розподіляються за процесами і виконують відповідні задачі з реалізації проміжного і кінцевого продукту. Остаточний результат виміряється за обсягом коду, ступенем повторного використання, кількістю помилок і дефектів, а також оцінюванням показників якості ПЗ.

Внесення змін пов'язане з помилками, виявленими при перевірці і тестуванні, проводиться з метою збереження функціональної цілісності системи. У випадку виявлення помилок на процесі супроводження приймається рішення про внесення змін або заміну коду у цілому.

Практичні міркування (Practical Considerations) – комплекс процесів пов'язаних з конструюванням ПЗ, а також діяльність, в рамках якої ПЗ конструюється з обмеженнями відповідно до вимог та реального світу(іноді - хаотичними). Наближаючись до обмежень реального світу конструювання програмного забезпечення ведеться на основі експерементально отриманих даних.

Технології конструювання (Construction Technologies) – система інженерних принципів для створення економічної програмного забезпечення, яке надійно і ефективно працює в реальних системах.

Засоби конструювання (Software Construction Tools) – це інструменти для трансляції і об'єднання програм. Інструменти конструювання програмного забезпечення забезпечують автоматизовану або автоматичну підтримку методів, що використовуються при створенні або підтримці програмного забезпечення:

- редактори програм (program editors) і програми редагування загального призначення;
- компілятори і генератори коду (compilers and code generators) як самостійні засоби об'єднання програмних компонентів в інтегрованому середовищі для одержання вихідного продукту з використанням препроцесорів, складальників, завантажників і ін.;
- інтерпретатори (interpreters), які забезпечують контрольоване виконання програм за їх описом. Намітилася тенденція злиття інтерпретаторів і

компіляторів (наприклад, Java, в .NET);

– відлагоджувачі (debuggers), призначені для перевірки правильності опису вихідних програм і усунення помилок;

– інтегроване середовище розробки (IDE – integrated development environment) та бібліотеки компонентів (libraries components), що є утворюють середовище виконання процесу розроблення ПС;

– програмні платформи (Java, J2EE і Microsoft .NET) і платформи для розподілених обчислень (CORBA і WebServices, тощо).

Ці інструменти використовуються для виробництва і трансляції програмного коду зрозумілого для машинного виконання.

Засоби конструювання можна розділити.

а) Редактори коду (program editors). Ці інструменти використовуються для створення і модифікації «вихідного коду» програм. Це можуть бути:

– редактори "загального призначення" (що упродовж багатьох років спостерігається в UNIX і UNIX-подібних середовищах: vi/Vim, SciTE, Notepad++)

– спеціалізовані редактори з підтримкою специфіки цільової мови програмування (що являється, у більшості випадків, прерогативою інтегрованих середовищ розробки – IDE).

б) Компілятори і генератори коду (compilers and code generators). Під транслятором (translator) зазвичай розуміють спеціальну програму, яка переводить текст програми в послідовність машинних команд. Транслятори мов високого рівня, таких як C, C++, Pascal і інших, називають зазвичай компіляторами (compiler). Традиційно, компілятори були не інтерактивними (командними) трансляторами початкового коду. Проте, існує тенденція інтеграції компіляторів і

редакторів в інтегровані середовища програмування. До цього класу також відносяться:

в) Препроцесори.

г) Лінкувальники/завантажувачі. Вже на самому початку розвитку методів програмування став застосовуватися простий і ефективний прийом виділення часто використовуваних алгоритмів в самостійні програми, що дістали назву стандартних підпрограм. Прикладом можуть служити підпрограми обчислення елементарних функцій(синус, косинус та ін.), а також процедури обміну із зовнішніми облаштуваннями комп'ютера. Одного разу складені і такі, що відкомпілювалися, вони надалі можуть застосовуватися програмістами у своїх завданнях шляхом під'єднування їх до розробленого коду основного алгоритму. У систему засобів програмування входить програма, що називається редактор зв'язків (компоновщик, лінкувальник), яка забезпечує пошук допоміжних підпрограм в спеціальних бібліотеках програм і їх приєднання до основної програми користувача. Результатом роботи редактора зв'язків є повністю готовий до виконання двійковий код програми, що називається завантажувальним модулем.

д) Генератори коду (за виключенням, можливо, об'єктно-орієнтованих засобів проектування, що підтримують зв'язок з вихідним кодом і мають тенденцію бути тісно інтегрованими з новим поколінням IDE). В деяких випадках генератори самостійно створюють програмний код, що виконує певні стандартні дії. Прикладом таких засобів може служити, наприклад, Microsoft Visual Studio, яка автоматично створює і заповнює полями клас "Форма" при створенні нами нового вікна і наповненні його різними компонентами.

е) Інтерпретатори (interpreters). Ці інструменти забезпечують виконання програм за допомогою емуляції. Вони можуть підтримувати дії з конструювання програмного

забезпечення, надаючи для виконання програм оточення, що більше контрольоване і піддається спостереженню, чим це зазвичай здатна зробити та або інша операційна система. Хочеться відмітити певне " злиття", якщо так можна виразитися, між компіляторами і інтерпретаторами. Яскравим тому свідченням є використання так званої just-in-time компіляції – компіляції "на льоту", коли проміжний програмний код, у міру виконання або з випередженням (наприклад, в процесі запуску/завантаження програми) перетворюється в набір інструкцій, що виконуються безпосередньо засобами операційної системи, але під контролем середовища виконання, в першу чергу, з точки зору безпеки. Такого роду підхід став родоначальником ряду сучасних програмних платформ, наприклад, Java і .NET. На цьому фоні можливо об'єднати інтерпретатори з компіляторами і генераторами коду, як засоби безпосередньої підготовки(трансляції) початкового коду до виконання.

ж) Налагоджувачі (debuggers). У систему засобів програмування входять також програми, що полегшують відладку (пошук помилок). При усьому різноманітті реалізацій відладчиків їх основні можливості полягають в так званому трасуванні роботи програми. Трасування – це відстежування (ведення протоколу) роботи програми. В процесі трасування програміст може простежити порядок виконання операторів, а також динаміку зміни значень змінних програми. Ці інструменти було прийнято виділити в самостійну категорію, оскільки вони підтримують процес конструювання програмного забезпечення, але, в той же час, «функціонально» відрізняються від редакторів і компіляторів.

2.5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE TESTING)

"Тестування програмного забезпечення (Software testing)" - це розділ програмної інженерії присвячений процесам перевірки готової програми в статиці (перегляди, інспекції, налагодження вихідного коду) і в динаміці (прогін на наборі тестових даних) з метою перевірки різних шляхів виконання програми і порівняння отриманих результатів із заздалегідь заданими.

Існує дві форми перевірки коду – модульна й інтеграційна. Спочатку використовують стандарти (IEEE 829:1996 і IEEE 1008:1987) з перевірки і тестування модулів. Потім проводиться інтеграційне тестування модулів системи і їх інтерфейсів у динаміці виконання. Під час різних видів перевірок збираються дані про помилки, дефекти, відмови тощо і оформляється відповідна документація (таблиці типів помилок, частоти і часу виявлення відмов і ін.). Зібрані дані використовуються при оцінюванні характеристик якості готового програмного забезпечення, наприклад, надійності.

Дана область знань SWEBOK визначає методи перевірки правильності програмного забезпечення: верифікація, валідація, тестування. Наводяться типи, рівні і техніки тестування програмного забезпечення, методи планування процесу тестування, розроблення тестових наборів даних для прогону програмного забезпечення в режимі випробування модулів або системи в цілому і наступною оцінкою результатів тестування.

Розділ програмної інженерії "Тестування програмного забезпечення (Software testing)" складається із 6 основних тем (рис 2.5), які розглядаються як сукупність логічно пов'язаних

тем, що безпосередньо стосуються методів та засобів тестування програмного забезпечення.

Основи тестування (Software Testing Fundamentals) – це базові терміни, ключові проблеми і їхній зв'язок з іншими областями знань. Тестування визначається як процесс перевірки правильності програми в динаміці її виконання за тестовими даними. При тестуванні виявляються недоліки: відмови (faults) і дефекти (defects) як причини порушення роботи програми, збої (failures) як небажані ситуації, помилки (errors) як наслідки збоїв і ін. Базове поняття тестування – тест, що виконується в заданих умовах і за наборами даних. Тестування вважається успішним, якщо знайдено дефект або помилка, і вони відразу усуваються. Ступінь тестованості визначається критерієм покриття системи тестами, перевірки всіх можливих шляхів виконання програм і імовірності припущення стосовно того, що може з'явитися збій або помилкова ситуація в системі.

Рис.2.5 – Склад розділу
"Тестування програмного забезпечення (Software testing)"

Рівні тестування (Test Levels) – призначені для проведення тестування програмного забезпечення на різних стадіях його розробки та обслуговування. Рівень тестування визначає те, над чим виконуються тести: над окремим модулем, групою модулів або системою, в цілому. Досягнення певної мети тестування при тестуванні програмного забезпечення не повинно бути метою тестування як такового: це лише спосіб поліпшення шансів знайти недоліки в програмному коді

При цьому рівні тестування можуть відрізнятися за ціллю або метою та підрозділяються на 4 рівні:

- компонентне або модульне тестування (Component testing or Unit testing)
- інтеграційне тестування (Integration testing)
- системне тестування (System testing)
- приймальне тестування (Acceptance testing)

а) Компонентне тестування перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо (модулі програми, функції і т.д.).

Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити чи за підтримки середовищ розробки, таких як фреймовки (каркаси) для модульного тестування або інструменти для дебагу. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі багів (Bug Tracking System). Один з найбільш ефективних підходів до компонентного (модульного) тестування – це підготовка автоматизованих тестів до початку основного кодування ПЗ. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). При цьому підході створюються і інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір поки всі тести не будуть

успішними.

б) Інтеграційне тестування призначено для перевірки зв'язку між компонентами, а також взаємодії з різними частинами системи (операційної системи, обладнанням небудь зв'язку між різними системами). Рівні інтеграційного тестування:

– компонентний інтеграційний рівень (Component Integration testing) перевіряє взаємодія між різними системами після проведення компонентного тестування.

– системний інтеграційний рівень (System Integration testing) перевіряє взаємодію між різними системами після проведення системного тестування.

в) системне тестування – основним завданням є перевірка як функціональних так і не функціональних вимог у системі в цілому. При цьому виявляються дефекти, такі як невірне використання ресурсів системи, непередбачені комбінації даних рівня користувача, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання і т.д. Для мінімізації ризиків, пов'язаних з особливостями поведінки системи в тому чи іншому середовищі, під час тестування рекомендується використовувати оточення максимально наближене до того, на яке буде встановлений продукт після релізу. Існує два підходи до системного тестування:

Для кожної із початкових вимог (requirements based) до програмного забезпечення пишуться тестові випадки (test cases), що дозволяє перевірити виконання даної вимоги.

На базі випадків використання (use case based) – На основі уявлення про способи використання продукту створюються випадки використання системи (Use Cases). По конкретному випадку використання можна визначити один або більше

сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які повинні бути протестовані.

г) приймальне тестування проводиться з метою: визначення чи задовольняє система приймальні критерії там винесення рішення замовником або іншою уповноваженою особою приймається програма чи ні.

Приймальне тестування виконується відповідно до Плану приймальних Робіт. Рішення про проведення приймального тестування приймається, коли: продукт досяг необхідного рівня якості та замовник ознайомлений з Планом приймальних Робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні і т.д.

д) методи приймального тестування:

– тестування замовником самостійно. Це ризиковано в тому плані що у замовника може не бути творчих ресурсів, а завантаження по поточним завданням може розтягти процес приймання.
– аудит-тестування, що проводиться третьою стороною. Наймається спеціалізована компанія на тестуванні або підписується договір з конкурентом постачальника на надання послуг аудиту. Оптимально.
– спільне тестування за сценаріями із замовником. Постачальник допомагає готувати пакет матеріалів для приймального тестування, готує команду замовника до методичного приймального тестування, контролює хід приймального тестування і терміни його виконання. Присутність інженера з тестування з боку виконавця допоможе краще зафіксувати розбіжності, зауваження та виявлені дефекти.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або реліз програми. Незважаючи на те, що приймання знаходиться в кінці етапу (а в невеликих проектах і в кінці проекту) – готуватися до неї потрібно заздалегідь і перший прогін потрібно робити трохи раніше – щоб визначитися з повнотою і якістю робочого набору артефактів приймання, привнити до нього замовника, заздалегідь виявити можливі проблеми в приймальних тестах або в продукті.

Техніки тестування (Test Techniques) основне призначення виявлення найбільш можливої кількості помилок в програмному забезпеченні. Для виявлення помилок в програмному забезпеченні розроблено багато технік, але основна ціль у всіх методів однакова – це визначення вхідних даних, які дадуть репрезентативну поведінку програми. Техніки тестування базуються на певних теоретичних і практичних положеннях щодо проектування (компонентного, об'єктно-орієнтованого, сервісного і т.п.), а також на таких даних як:

- інформація про структуру ПЗ або системи в документації («біла скринька»);
- підбір тестових наборів даних для перевірки правильності роботи компонентів і системи в цілому без знання їх структури («чорна скринька»);
- аналіз граничних значень, таблиць прийняття рішень, потоків даних, статистики відмов і ін.;
- блок-схеми побудови програм і складання наборів тестів для покриття системи цими тестами;
- виявлені і зафіксовані в таблицях системи дефекти, перед- і посту мови виконання, структурні характеристики системи (кількість модулів, обсяг даних тощо).

Також в залежності від доступу розробника тестів до

вихідного коду програми, що тестується розрізняють «тестування (по стратегії) білого ящика» і «тестування (по стратегії) чорного ящика».

При використанні моделі білого ящика (також кажуть - прозорого ящика), при тестуванні розробник тесту має доступ до вихідного коду програм і може писати код, який пов'язаний з бібліотеками програмного забезпечення, що тестується. Це типово для компонентного тестування, при якому тестуються лише окремі частини системи. Воно забезпечує те, що компоненти конструкції працездатні і стійкі, до певної міри. При тестуванні білого ящика використовуються метрики покриття коду або мутаційне тестування.

Тестувальник при використанні моделі чорного ящика при тестуванні має доступ до програми тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншого комп'ютера або іншому процесу підключитися до системи для тестування. Наприклад, тестує компонент може віртуально натискати клавіші або кнопки миші в програмі, що тестується за допомогою механізму взаємодії процесів, з упевненістю в тому, чи всі йде правильно, що ці події викликають той же відгук, що й реальні натискання клавіш і кнопок миші. Як правило, тестування чорного ящика ведеться з використанням специфікацій або інших документів, що описують вимоги до системи. Зазвичай в даному виді тестування критерій покриття складається з покриття структури вхідних даних, покриття вимог і покриття моделі (в тестуванні на основі моделей).

При використанні моделі сірого ящика при тестуванні розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібно.

Вимірювання результатів тестування (Test-Related Measures) вимірювання результатів тестування ПЗ й оцінки якості використовуються метрики. Вимір як частина планування і розробки тестів базується на розмірі програм, їх структурі і кількості виявлених помилок і дефектів.

Метрики тестування – це вимірювання процесу планування, проектування і тестування, а також результатів тестування на основі таксономії відмов і дефектів, покриття границь тестування, перевірки потоків даних і ін.

Процес тестування документується і, відповідно до стандарту IEEE 829.1995, містить у собі опис тестових документів, їх зв'язку між собою і з задачами тестування. Без документації на процес тестування неможливо провести сертифікацію продукту за моделями зрілості, зокрема, моделлю CMM. Після завершення тестування оцінюється вартість і ризики ПЗ, викликані збоями або недостатньо надійною роботою системи. Вартість тестування – одне з обмежень, на основі якого приймається рішення про його припинення або продовження.

Вимірювання зазвичай вважається основним для аналізу якості. Вимірювання також може використовуватися для оптимізації планування та виконання випробувань. Управління тестуванням може використовувати кілька різних заходів процесу для контролю прогресу

Процес тестування (Test Process) – це процес, що охоплює всі можливі активності при тестуванні програмного забезпечення. Цей процес починається з тестового планування, розробки тестових сценаріїв, підготовки до виконання та оцінки статусу для закриття тесту та в межах основного процесу тестування ділиться на наступні основні етапи:

– планування процесу тестування (складання планів, тестів, наборів даних) і оцінювання показників

якості готового продукту;

– проведення тестування компонентів повторного використання і патернів як основних об'єктів складання программного забезпечення;
– генерація необхідних тестових сценаріїв, що відповідають середовищу виконання програмного забезпечення;
– верифікація правильності реалізації системи і валідація реалізації вимог до програмного забезпечення;
– збирання даних про відмови, помилки і виявлені непередбачені ситуації при виконанні програмного продукту;
– підготовка звітів за результатами тестування й оцінка характеристик системи.

Отже, концепції, стратегії, техніки і методи вимірювання, що використовуються при тестуванні об'єднані в єдиний процес тестування як діяльності щодо забезпечення якості програмного забезпечення. Процес тестування підтримує роботи з тестування та визначає "правила гри" для членів команди тестування - від планування тестів до оцінки їх результатів. Хоча, в більшості сучасних методів розробки, зокрема, гнучких (agile) підходів, тестування виходить на передній план і є однією з базових практик, багатостороннє тестування і, тим більше, прогнозування на основі отриманих результатів, часто підмінюється окремими роботами в цій області, що не дозволяють домогтися необхідних параметрів якості. Тільки в тому випадку, якщо тестування розглядати як один з важливих процесів всієї діяльності по створенню і підтримці програмного забезпечення, можна домогтися оцінки вартості відповідних робіт і, врешті-решт, дотримати ті обмеження, які визначені для проекту.

Засоби тестування (Software Testing Tools) це:

- генератори тестів (test generators), що допомагають у розробці сценаріїв тестування;
- засоби виконання тестів (test execution frameworks), які забезпечують виконання тестових сценаріїв і відслідковують поведінку об'єктів тестування;
- інструменти оцінки тестів (test evaluation tools), які підтримують оцінювання результатів виконання тестів і ступеня відповідності поведінки тестованого об'єкта очікуваній поведінки;
- засоби керування тестами (test management tools), які забезпечують інженерне керування процесом тестування програмного забезпечення;
- інструменти аналізу продуктивності (performance analysis tools), кількісної її оцінки та оцінки поводження програм у процесі виконання.

2.6 СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE MAINTENANCE)

"Супровід програмного забезпечення (Software Maintenance)" - це розділ програмної інженерії присвячений вивченню сукупності дій із забезпечення роботи програмного забезпечення, внесення змін при виявленні помилок, адаптації ПЗ до нового середовища функціонування, а також підвищення продуктивності або поліпшення деяких характеристик ПЗ. Супровід розглядається з точки зору задоволення вимог споживача у готовому ПЗ, коректності його виконання, процесів навчання й оперативного обліку його процесу. Супровід відповідно до стандартів ISO/IEC 12207 і ISO/IEC 14764 проводиться з метою виконання і модифікації програмного продукту в процесі експлуатації за

умови збереження його цілісності.

Розділ програмної інженерії "Супровід програмного забезпечення (Software Maintenance)" складається із п'яти різних тем (рис 2.6), які необхідно розглядати як сукупність логічно пов'язаних тем, що безпосередньо стосуються супроводу програмного забезпечення на всіх етапах його життєвого циклу.

Основні концепції супроводу програмного забезпечення (Software Maintenance Fundamentals) – це базові визначення і термінологія, підходи до еволюції програмного забезпечення та його супроводу, до оцінки вартості супроводу та тощо. До основних концепцій, що використовуються при супроводі програмного забезпечення можна віднести життєві цикли програмного забезпечення (стандарт ISO/IEC 12207) та ведення програмно документації. Метою супроводу є підтримка використання готової програмної системи, фіксація помилок, що виникають при виконанні, дослідження причин їх виникнення, аналізі необхідності модифікації системи з метою усунення помилок, оцінка вартості робіт із проведення модифікацій функцій і програмної системи в цілому, а також розгляд та аналіз методів подолання комплексу проблеми, що виникають при ускладнені програмного продукту за умови великої кількості змін.

Ключові питання супроводу ПЗ (Key Issue in Software Maintenance) – це управлінські, вимірювальні і вартісні. Суть управлінських питань – контроль ПЗ при модифікації й удосконалюванні функцій і недопущення зниження продуктивності системи. Питання вимірювання пов'язане з оцінкою характеристик системи після її модифікації, а також повторного тестування для оцінки показників якості. Вартісні питання пов'язані з оцінкою витрат на супровід залежно від його типу, кваліфікації персоналу, платформи й ін.

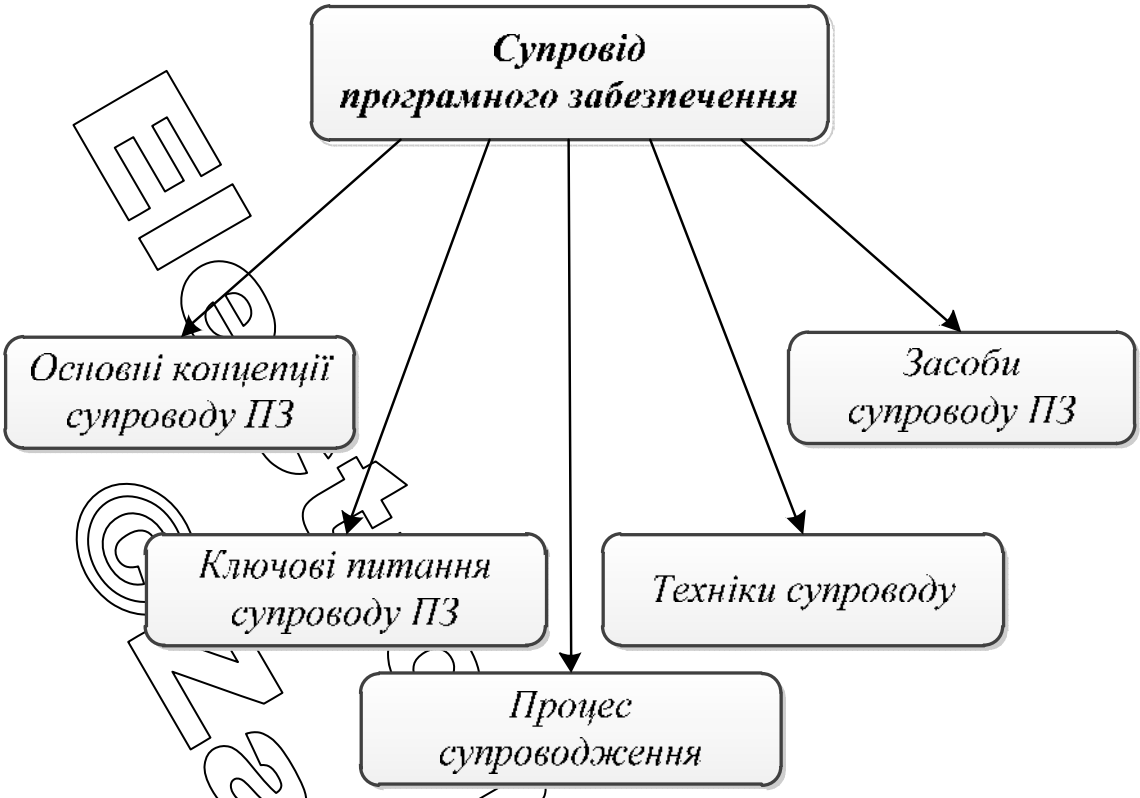


Рис. 2.6 – Склад розділу
"Супровід програмного забезпечення
(Software Maintenance)"

Процес супроводження (Process Maintenance) – моделі процесу супроводу і планування діяльності людей, що проводять запуск ПЗ, перевірку правильності його виконання і внесення в нього змін. Цей процес згідно з стандартом ISO/IEC 14764 проводиться шляхом:

– коригування, тобто зміни продукту для усунення виявлених помилок або нереалізованих задач;
– адаптації, тобто настроювання продукту в умовах експлуатації, що змінилися, або в новому середовищі виконання;
– поліпшення, тобто еволюційної зміни продукту для підвищення продуктивності або рівня супроводу;
– перевірки ПЗ, пошуку і виправлення помилок при експлуатації системи.

Техніки супроводу (Techniques for Maintenance) – супровід еволюції програмного забезпечення. Внаслідок змін система стає більш складною і погано керованою. У зв'язку з цим виникає проблема зменшення її складності. До технологій еволюції ПЗ відносять реінженерію, реверсну інженерію і рефакторинг.

Реінженерія – це удосконалення застарілого ПЗ шляхом його реорганізації або реструктуризації, а також перепрограмування окремих елементів або настроювання параметрів на іншу платформу, середовище виконання зі збереженням зручності його супроводу.

Реверсна інженерія полягає у відновленні специфікації (графів викликів, потоків даних і ін.) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонеитів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу. Найчастіше реверсна інженерія застосовується після того, як у код ПЗ було внесено багато змін і воно стало некерованим або змінилася платформа комп'ютера.

Рефакторинг – реорганізація коду для поліпшення характеристик і показників якості об’єктно-орієнтованих і компонентних програм без зміни їх поведінки. Цей процес реалізується шляхом поступової зміни окремих операцій над текстами, інтерфейсами, середовищем програмування і виконання ПЗ, а також настроювання або внесення змін в інструментальні засоби підтримки ПЗ. Якщо при зміні зберігається формат існуючої системи, то рефакторинг – один з варіантів реверсної інженерії.

Засоби супроводу програмного забезпечення (Software Maintenance Tools) – інструменти, що використовуються в обслуговуванні програмного забезпечення та за його модифікації. Приклади використання інструментів є:

– програвачі, які вибирають лише модифіковані

частини програми;

- статичні аналізатори, які дозволяють отримати загальний перегляд програми та її зміст;
- динамічні аналізатори, які дозволяють простежити шлях виконання програми;
- аналізатори потоку даних, які дозволяють відслідковувати всі можливі потоки даних програми;
- перехресні посилання, дозволяють генерувати індекси компонентів програми;
- аналізатори залежностей, які допомагають супроводжуючим аналізувати та розуміти взаємозв'язок між компонентами програми.

Інструменти зворотного інжинірингу допомагають в процесах створення специфікацій і опису дизайну, які потім можуть бути використані при створенні нового продукту зі старого. Інструменти супроводу також використовуються при тестуванні програмного забезпечення, керування конфігурацією програмного забезпечення та в процесах створення документації програмного забезпечення.

2.7 КЕРУВАННЯ КОНФІГУРАЦІЄЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE CONFIGURATION MANAGEMENT)

"Керування конфігурацією програмного забезпечення (Software Configuration Management)" - це розділ програмної інженерії присвячений вивченню процесів пов'язаних із конфігурації системи в різних точках часу з метою систематичного контролю змін конфігурації та збереження цілісності та простежуваності конфігурації протягом всього життєвого циклу системи. Система може бути визначена як

комбінація взаємодіючих елементів, організованих для досягнення однієї або більше заявлених цілей. Керування конфігурацією програмного забезпечення (SCM) включає процес підтримки життєвого циклу програмного забезпечення, що забезпечує управління проектами, розробку та технічне обслуговування, діяльність з забезпечення якості, а також клієнтів та користувачів кінцевого продукту.

Конфігурація системи - це функціональні та фізичні характеристики апаратного або програмного забезпечення, викладені в технічній документації або реалізовані у програмному продукті або це сукупність конкретних версій апаратного забезпечення, прошивки або програмних засобів, об'єднаних відповідно до конкретних процедур для виконання певних цілей.

Конфігурація ПЗ складається з набору функціональних і технічних характеристик ПЗ, заданих у технічній документації і реалізованих у готовому продукті. Це сполучення різних елементів продукту з заданими процедурами збирання компонентів і настроювання на середовище. Вхідними елементами конфігурації є графік розробки, проектна документація, вихідний виконуваний код, бібліотека компонентів, інструкції з установки і розгортання системи.

Поняття керування конфігурацією застосовуються до всіх елементів, що підлягають контролю, хоча існують деякі відмінності в застосуванні між обладнанням СМ та програмним забезпеченням СМ. SCM тісно пов'язана з діяльністю із забезпечення якості програмного забезпечення (SQA). Як визначено в області знань про якість програмного забезпечення (KA), процеси SQA забезпечують впевненість, що програмні продукти та процеси в життєвому циклі проекту відповідають їхніми визначеними вимогами шляхом планування, прийняття та виконання певних заходів для забезпечення адекватної довіри до якості будучи вбудованим в

програмне забезпечення. Дії СКМ допомагають досягти ці цілі у сфері СВО. У деяких контекстах проекту певні вимоги SQA вимагають певних заходів SCM.

Розділ програмної інженерії *"Керування конфігурацією програмного забезпечення (Software Configuration Management)"* складається із 7 основних тем (рис 2.7), які розглядаються як сукупність логічно пов'язаних тем, безпосередньо пов’язаних з керування конфігурацією програмного забезпечення протягом всього життєвого циклу.

Управління та планування процесу SCM (Management of the SCM Process) призначений для контролю еволюції та цілісності програмного продукту, визначаючи його елементи; елементи управління та контролю змін; перевірку, документування та звітування про конфігурацію програмного забезпечення. Управління та планування призначене для полегшення розробки та ведення змін впровадженних заходів в программному проекті.

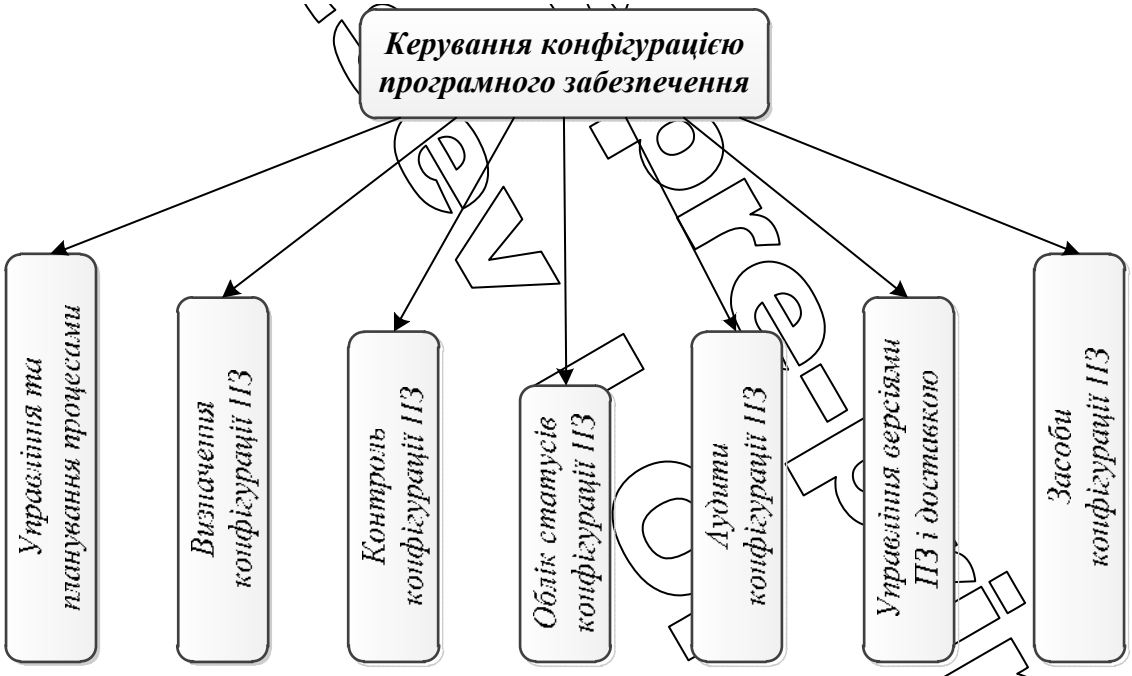


Рис.2.7 – Склад розділу
"Керування конфігурацією програмного забезпечення
(Software Configuration Management)"

Визначення конфігурації програмного забезпечення (Software Configuration Identification) призначений для ідентифікації конфігурації програмного забезпечення й визначення його елементів, що підлягають контролю, встановлення схем ідентифікації елементів програмного забезпечення та його версій, а також методів та засобів використання інструментів, які застосовуються в управлінні елементами керування.

Контроль конфігурації програмного забезпечення (Software Configuration Control) – це роботи з координації, затвердження або відкидання реалізованих змін в елементах конфігурації після ідентифікації, а також з аналізу вхідних компонентів конфігурації. Контроль охоплює процеси визначення того, які зміни слід внести, як здійснити підтримку впровадженних змін, а також які відхілення від вимог проекту є можливими, а також вибору алгоритмів відмов від реалізації певних вимог проекту, за умови їх зміни.

Облік статусів конфігурації програмного забезпечення (Software Configuration Status Accounting) – це елемент керування конфігурацією, що складається з запису та звітності інформації, необхідної для ефективного управління конфігурацією. Основне призначення ведення обліку стану та внесених змін в конфігурацію програмного забезпечення.

Аудит конфігурації програмного забезпечення (Software Configuration Status Accounting) – це незалежна експертиза робочого програмного продукту або набору робочих програмних продуктів для їх оцінки на відповідність вимогам, специфікаціям, стандартам, контрактним угодам або іншим критеріям. Тобто при аудиті визначають ступінь задоволення конфігурації функціональним і фізичним (апаратним) очікуваним характеристикам системи.

Управління версіями програмного забезпечення і доставкою (Software Release Management and Delivery) – це відстеження наявної версії компонентів конфігурації; складання компонентів; створення нових версій системи на основі існуючих шляхом внесення змін у конфігурацію; узгодження версії продукту з вимогами і проведеними змінами на процесах ЖЦ; забезпечення оперативного доступу до інформації про елементи конфігурації і системи, до яких вони належать; розповсюдження елементів конфігурації програмного забезпечення поза межами проекту з розробки програмного забезпечення; внутрішні версії, а також розповсюдження версій програмного забезпечення серед клієнтів.

Засоби конфігурації програмного забезпечення (Software Configuration Management Tools) – це:

– інструменти відстеження (tracking) дефектів;
– інструменти керування версіями;
– інструменти керування складанням, випуском версії (конфігурації)продукту та його інсталяції.

Інструменти конфігурації з точки зору обсягу, на якому вони надають підтримку поділяють на три класи:

а) індивідуальну підтримку;
б) підтримку, пов'язану з проектами;
в) підтримку компанії по всій технології.

Індивідуальні засоби підтримки є належними та зазвичай достатніми для невеликих організацій або груп розробників вони включають:

- інструменти керування версіями та призначені для відстеження, документування та зберігання окремих елементів конфігурації, таких як вихідний код та зовнішня документація;
- інструменти для створення засобів обробки складають та пов'язують між собою елементи

програмного забезпечення в єдину виконувану версію програмного забезпечення;

- інструменти керування змінами в основному підтримують контроль запитів на зміну та повідомлення про події (наприклад, зміни статусу запиту на зміну, досягнуті етапи).

Інструменти підтримки, пов'язані з проектами та використовуються для управління робочим середовищем розробниками. Такі інструменти є придатними для середніх і великих організацій з варіантами їх програмних продуктів та паралельним розвитком, але не мають сертифікаційних вимог.

Інструменти підтримки компанії в цілому, як правило, можуть автоматизувати частини загальнопоширих процесів, забезпечуючи підтримку управлінь робочими процесами, ролі та відповідальності. Такі засоби доповнюють проектну підтримку, підтримуючи більш формальні процеси розвитку, включаючи вимоги до сертифікації програмного забезпечення.

2.8 УПРАВЛІННЯ В ПРОГРАМНІЙ ІНЖЕНЕРІЇ (SOFTWARE ENGINEERING MANAGEMENT)

"Управління в програмній інженерії (Software Engineering Management)" - це розділ програмної інженерії присвячений вивченню методів та засобів, що використовуються при управлінні роботами в команді розробників програмного забезпечення у процесі виконання програмного проекту, визначенні критеріїв ефективності роботи цієї команди й оцінки процесів й продуктів програмного проекту з використанням загальних методів планування і контролю робіт.

Як будь-яке управління, управління в програмній інженерії полягає у плануванні, координації, контролі, вимірі

й обліку виконаних робіт у процесі розроблення програмного проекту, а також у координації людських, фінансових і технічних ресурсів. Тобто, управління в програмній інженерії так само, як і будь-яким іншим комплексним процесом схожі, але й мають свої відмінності, які в якійсь мірі, ускладнюють досягнення необхідного рівня ефективності управління. Серед таких ускладнюючих факторів:

- Сприйняття клієнтів (споживачів, замовників) таке, що часто відсутня з їх боку розуміння складності, породженої самою природою програмної інженерії, зокрема пов'язаної з впливом змінюються вимог.
- Практично неминуче зміниться або поява нових клієнтських вимог, в наслідок функціонування процесів програмної інженерії.
- В результаті, програмні системи часто будуються з застосуванням ітеративних процесів, замість послідовного виконання і закриття робіт (завдань).
- Рівень новизни і складності програмних систем часто вкрай високий (і не дозволяє в повній мірі застосовувати вже існуючі напрацювання та досвід, прим. Автора).
- Застосовувані технології мають високу швидкість зміни, поновлення і старіння

Що стосується програмної інженерії, управлінська діяльність в цій галузі відбувається на трьох рівнях:

- Організаційне управління і управління інфраструктурою
- Управління проектами
- Планування та контроль програм кількісної оцінки.

Розділ програмної інженерії *"Управління в програмній інженерії (Software Engineering Management)"* складається із 7 основних тем (рис 2.8), які розглядаються як сукупність

логічно пов'язаних тем, безпосередньо пов’язаних з управлінням роботами команди розробників програмного забезпечення.

Ініціювання та визначення змісту (Initiation and Scope Definition) – набор дій, пов'язаних з ефективним визначенням вимог до програмного забезпечення з використанням різних методів збору вимог, а також оцінки ймовірних можливостей реалізації та використання програмного проекту з різних точок зору. Якщо проект визнано можливим, то наступним завданням є специфіка процедур перевірки та зміни вимог (див. розділ 2.2).

Рис 2.8 – Склад розділу
"Управління в програмній інженерії
(Software Engineering Management)"

Планування програмного проекту (Software Project Planning) пов’язано з планування організації, моніторингу і контролю усіх аспектів проекту. Процес планування є ітеративним і базується на змісті вимог до програмного

проекту. При цьому, варто зазначити, що планування проекту може деталізуватися у вигляді структурної декомпозиції робіт з прив’язкою до очікуваних результатів і їх характеристик. Такі характеристики, як правило, пов'язані з питаннями якості та іншими встановленими атрибутами вимог до програмного продукту, дотриманням термінів виконання робіт, ресурсами тощо. Відповідно до планування ресурси розподіляються за завданнями таким чином, щоб забезпечити оптимальну продуктивність (на персональному, командному та організаційному рівні), використання коштів (інфраструктури, інструментів та тощо) і обладнання, а також суворого дотримання термінів реалізації проекту. Планування, також включає управління ризиками, визначенні допоміжних процесів і обов'язків в частині управління планом проекту, його оцінкою і порядком перегляду різних аспектів проекту.

Виконання програмного проекту (Software Project Enactment) пов'язаний з виконанням плану проекту за рахунок виконання процесів, представлених в план проекту. Дотримання плану протягом виконання проекту пов'язано з очікуваннями, що дотримання плану призводить до успішного задоволення вимог зацікавлених осіб і досягненню цілей щодо реалізації програмного проекту. Основою для успішного виконання проекту є управлінська діяльність з ведення оцінки та вимірювань, моніторингу, контролю і звітності в процесі реалізації програмного проекту.

Огляд і оцінка (Review and Evaluation) пов’язана з досягненні встановлених цілей і задоволення вимог зацікавлених осіб при реалізації програмного проекту. Для оцінки ефективності застосовуються різні методи, інструменти і техніки. Сам процес оцінки є систематичним, а процедури - періодичними. Оцінка може проводитися в запланованих точках під час реалізації програмного проекту у відповідності з планом проекту або у критичних точках у

відповідності до результатів аналізу ризиків. Також, аналогічно, може проводиться оцінка (assessment) ефективності процесів, роботи персоналу, інструментів і методів, що використані в роботах, проведених за певних проміжок часу або по завершенню певного планового етапу при реалізації програмного проекту. Етапи на яких проводиться огляд та оцінка залежать від прийнятих управлінських технік, методологій, організаційних принципів, метрик та тощо.

Завершення (Closure) є останнім етапом реалізації будь-якого проекту в тому числі і програмного. Завершення або закриття програмного проекту (дане поняття не стосується поняття припинення робіт над проектом, так як припинення робіт може відбуватися на будь-якому етапі в залежності від прийнятих рішень, щодо доцільності продовження робіт над проектом) відбувається коли всі плани і процеси успішно виконані і завершені відповідно до прийнятих очікувань. На цій стадії до результатів проекту застосовуються критерії оцінки його успішності та якості та виконуються дії з архівування проектних даних, аналізу результатів, отриманих в процесі роботи над проектом (post mortem analysis), що можуть бути використані для поліпшення процесів (process improvement) планування інших програмних проектів.

Вимірювання в програмної інженерії (Software Engineering Measurement) при управлінні процесами пов'язаними із реалізацією або підтримкою програмного продукту є важливими. Ефективні вимірювання, які можливо використовувати при прийнятті управлінських рішень стають одним аспектів організаційної зрілості проектної команди.

Засоби управління інженерною діяльністю (Software Engineering Management Tools) – інструменти, що використовуються в обслуговуванні та модифікації програмного забезпечення. Інструменти управління

програмним розділяються на автоматизовані та ручні. При цьому, як автоматизовані так і ручні інструменти управління можуть використовуватися в інтегрованих комплексах управління процесами у продовж усього проекту, для планування, збору та реєстрації, моніторингу, контролю, діагностики та звітування про хід робот над програмних проектом та надання інформації про кінцевий продукт й його характеристики. Інструменти управління можна розділити на наступні категорії:

– інструменти планування і відстеження ходу проектів;
– інструменти управління ризиками (див. розділ 3);
– інструменти комунікації;
– інструменти вимірювання.

Інструменти планування та відстеження ходу проекту використовуються для кількісної і якісної оцінки зусиль та витрат, що були понесені при реалізації програмного проекту. Інструменти планування також включають в себе автоматизовані інструменти планування, які аналізують завдання в рамках структури розбиття на типи робіт, їх оцінювання тривалість, пріоритету, ресурсів виділених для виконання кожного етапу відповідно, наприклад до діаграми Ганта. При цьому основне призначення інструментів відстеження – відстеження основних етапів проекту, планування зустрічей, планування ітераційних циклів, демонстрації програмного продукту та/або елементів програмного продукту.

Інструменти управління ризиками можуть використовуватися для відстеження ідентифікації, оцінки та моніторингу ризиків. Ці інструменти включають використання підходів, таких як моделювання чи аналіз на основі дерева рішень для оцінки впливу вірогідності подій на ризик перевитрати ресурсів при реалізації програмного проекту.

Результати моделювання (наприклад методом Монте-Карло) можуть бути використані для отримання ймовірнісних розподілів використання ресурсів, графіків та ризиків при реалізації програмної системи шляхом об'єднання множинних розподілів імовірнісних вхідних даних алгоритмічним способом.

Інструменти комунікації або засоби зв'язку призначені допомогти у забезпеченні отримання своєчасної та послідовної інформації зацікавленими сторонами, залучених до реалізації проекту.

Інструменти вимірювання – інструменти кількісної оцінки властивостей (надійності і якості) програмного продукту, які в багатьох випадках базуються на прийнятих метриках або електронних таблицях, що розроблені членами проектної групи.

2.9 ПРОЦЕСИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ (SOFTWARE ENGINEERING PROCESS)

"Процеси програмної інженерії (Software Engineering Process)" - це розділ програмної інженерії присвячений вивченню інженерних процесів, що складаються з сукупності взаємопов'язаних дій, які здійснюється при реалізації робот на всіх життєвих циклах програмного проекту. Процеси інженерії програмного забезпечення пов'язані з роботою діяльністю, яку проводять інженери програмного забезпечення для розробки, обслуговування та експлуатації програмного забезпечення, таких як виявлення й аналіз вимог, проектування, архітектура, конструювання, тестування, управління конфігураціями та іншими процесами розробки програмного забезпечення. "Програмний процес" означає роботу, а не сам процес

виконання дій з реалізації життєвих етапів програмного забезпечення.

Програмні процеси включають в себе: управління, моделі та метод розробки та забезпечення якості програмного забезпечення.

Розділ програмної інженерії *Процеси програмної інженерії (Software Engineering Process)"* складається із 5 основних тем (рис 2.9), які розглядаються як сукупність логічно пов'язаних тем, безпосередньо пов’язаних з процесами програмної інженерії.

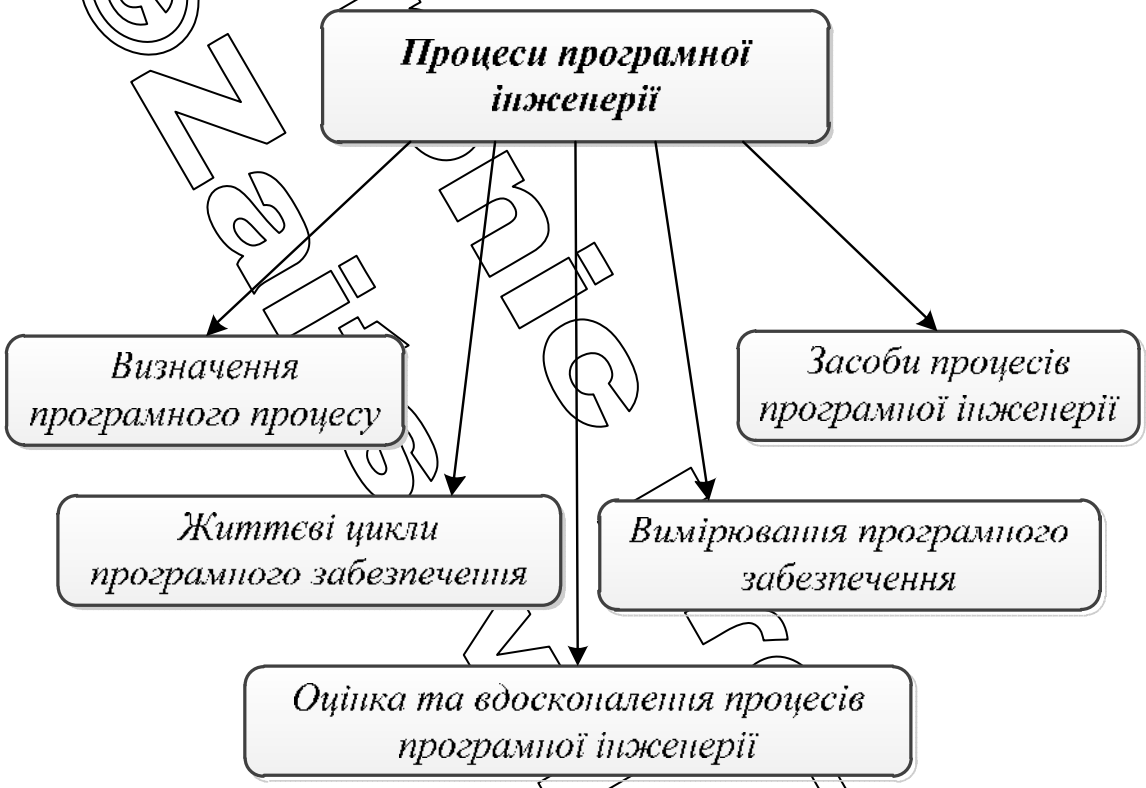


Рис 2.9 – Склад розділу
"Процеси програмної інженерії (Software Engineering Process)"

Визначення програмного процесу (Software Process Definition) може бути процедурою, рекомендацією або стандартом та стосується процесу управління програмним процесами пов’язаними із забезпечення реалізації програмного продукту та забезпечення інфраструктури на всіх

життєвих етапах. програмний процес являє собою сукупність взаємопов'язаних дій та завдань, які перетворюють вхідні робочі продукти на вихідні робочі продукти. Вхідний процес може являти собою ініціюючу подію або вихід іншого процесу. Критерії входження в процес повинні бути задоволені перед початком. Усі зазначені умови повинні бути виконані до успішного завершення процесу, включаючи критерії прийняття вихідного робочого продукту чи робочої продукції.

Основною метою процесу є підвищення якості одержуваного продукту, поліпшення різних аспектів програмної інженерії, автоматизація процесів і ін.

Процеси життєвого циклу програмного забезпечення чітко визначаються з різних причин, зокрема, з метою підвищення якості одержуваного продукту, поліпшення комунікацій і поліпшення розуміння різних аспектів програмної інженерії окремими фахівцями, підтримки вдосконалення процесів, підтримки управління процесами, забезпечення автоматизації процесів і т.п. Використовувані типи описів процесів, часто, залежать (як мінімум, частково) від цілей визначення процесів. Також необхідно відзначити, що проектний і організаційний контексти допомагають визначити найбільш підходящі визначення процесів. Важливими чинниками при визначенні процесу є природа робіт (наприклад, розробка або супровід), прикладна область (application domain), модель життєвого циклу і зрілість самої організації.

Програмний процес може включати підпроцеси. Наприклад, перевірка вимог програмного забезпечення - це процес, який використовується для визначення того, чи будуть вимоги створювати адекватну основу для розробки програмного забезпечення. Програмні процеси повинні бути обрані, адаптовані та застосовані відповідно до кожного проекту та кожного організаційного контексту. Ідеальний

процес або сукупності процесів не існує.

Життєві цикли програмного забезпечення (Software Life Cycles) визначають послідовність виконання і взаємозв'язку процесів, дій і завдань відповідно до вибраного життєвого циклу програмної системи(див. розділ 3.3). Вибір модель життєвого циклу залежить від специфіки, масштабу і складності проекту і специфіки умов, в яких система створюється і функціонує.

Життєвий цикл програмного забезпечення - це період часу з моменту прийняття рішення про необхідність створення програмної системи до моменту її повного вилучення з експлуатації. Життєвий цикл розробки програмного забезпечення включає в себе програмні процеси, що використовуються для визначення та перетворення програмних вимог в готовий програмний продукт.

Моделі життєвого циклу, як правило, підкреслюють основні процеси програмного забезпечення в рамках моделі та їх тимчасові та логічні взаємозалежності та відносини. Найбільш характерні моделі життєвого циклу прийнято поділяти на наступні види:

– каскадна модель;
– V-подібна модель;
– модель швидкого еволюційного прототипування;
– модель швидкого розроблення (Rapid Application Development);
– спіральна модель;
– модель фази-функції
– інкрементна модель та інші.

Детальні визначення процесів, що реалізуються в програмному забезпеченні в залежності від обраної моделі життєвого циклу можуть надаватися безпосередньо або за допомогою спеціальних документів(див. тему 5).

Оцінка та вдосконалення процесів програмної інженерії (Software Process Assessment and Improvement) стосується моделей оцінки програмного забезпечення, методів оцінки програмного забезпечення, моделей вдосконалення програмного забезпечення та оцінок процесів.

Оцінки програмного забезпечення процесу використовуються для оцінки форми та змісту процесу реалізації програмного продукту, який може бути визначений стандартизованим набором критеріїв. Оцінки можливостей, як правило, здійснюються покупцем. Результати використовуються як індикатор того, чи програмні процеси, що використовуються постачальником (або потенційним постачальником), є прийнятними для покупця. Оцінки на продуктивність, як правило, виконуються в межах організації для визначення програмних процесів, які потребують вдосконалення, або для визначення того, чи відповідає процес (або процеси) критеріям на заданому рівні можливостей процесу або розробки програмного продукту.

Аудит процесу відрізняється від оцінки процесу. Оцінки виконуються для визначення рівнів здатності або зрілості та визначення програмних процесів, які потребують вдосконалення. Аудит, як правило проводиться для перевірки дотримання політики та стандартів. Аудити забезпечує видимість управління фактичними операціями, що проводяться при реалізації програмного проекту, з тим, щоб можна було прийняти точні та суттєві рішення щодо питань, які впливають на проект, його розвиток, технічне обслуговування та тощо.

Фактори успіху в оцінці та вдосконаленні процесів програмної інженерії в організаціях з розробки та підтримки програмного забезпечення включають спонсорство, планування, навчання, стажування, зобов'язання команди, управління очікуваннями, а також пілотні проекти та

експерименти з інструментами, що використовуються в інженерії програмного забезпечення.

Вимірювання програмного забезпечення (Software Measurement) стосується програмного процесу та вимірювання продукту, якості результатів вимірювань, моделей програмного забезпечення та методів вимірювання програмного забезпечення. Перш ніж реалізувати новий процес або змінити поточний процес, необхідно отримати результати вимірювання поточної ситуації для забезпечення базової лінії для порівняння поточної ситуації з новою ситуацією.

Засоби процесів програмної інженерії (Software Engineering Process Tools) використовуються для визначення, впровадження та управління окремими програмними процесами та моделями життєвого циклу програмного забезпечення.

Розрізняють наступні інструменти процесів програмної інженерії:

- Інструменти моделювання, що дозволяють, зокрема, описати і модель процесів, як таку.
- Інструменти управління процесами. Надають підтримку для управління програмною інженерією.
- Інструменти конфігураційного управління, що підтримують роботу з актуальними версіями усього комплексу артефактів проекту і, що не менш важливо, що дозволяють задати поведінкові характеристики (у спрощеному розумінні – потік робіт, workflow) і атрибути цих артефактів у формі елементів конфігурацій.
- Ролеві платформи розробки програмного забезпечення, що охоплюють усі стадії життєвого циклу і, на сьогодні є розвитком інтегрованих

засобів розробки і CASE-інструментів у напрямі підтримки "суміжної" функціональності – управління вимогами, робіт по конфігураційному управлінню з підтримкою управління змінами, тестування і оцінки якості.

Перші три види інструменти дозволяють описати вживані процеси програмної інженерії. Четвертий клас – "супер-інтегровані середовища розробки", що називаються сьогодні ролевими платформами розробки, забезпечують підтримку заданих процесів, описаних, наприклад, у вигляді відповідних правил на рівні глибоко інтегрованих в такі середовища інструментів конфігураційного управління. Приклади: MS Team Foundation Server. Team Foundation Server(сокр. TFS) – продукт корпорації Microsoft, що є комплексним рішенням, що об'єднує в собі систему управління версіями, збір даних, побудову звітів, відстежування статусів і змін за проектом і призначене для спільної роботи над проектами по розробці програмного забезпечення. Цей продукт доступний як у вигляді окремого застосування, так і у вигляді серверної платформи для Visual Studio Team System (VSTS).

2.10 МОДЕЛІ ТА МЕТОДИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ (SOFTWARE ENGINEERING MODELS AND METHODS)

"Моделі та методи програмної інженерії (Software Engineering Models And Methods)" - це розділ програмної інженерії присвячений вивченню структури програмного забезпечення з метою реалізувати програмний проект, як систематичний та повторюваний процес, що орієнтований на

успішне завершення. Використання методів забезпечє підхід до систематичної специфікації, проектування, побудови, тестування та перевірки програмного забезпечення, як кінцевого продукту. Тому, основою створення якісного програмного продукту є використання моделей і методів при його реалізації, які дозволяють:

- наочно продемонструвати бажану структуру і поведінку програмної системи;
- здійснювати візуалізації та реалізувати бажані сценарії управління архітектурою програмної системи;
- забезпечити краще розуміння створюваної програмної системи, що найчастіше призводить до її спрощення й забезпечує можливість повторного використання;
- мінімізувати ризики при реалізації програмної системи, що дозволяє реалізувати більш якісне програмне забезпечення.

Моделі та методи розробки програмного забезпечення відрізняються широкими можливостями, починаючи від вирішення однієї фази життєвого циклу програмного забезпечення для забезпечення повного життєвого циклу програмного забезпечення.

Розділ програмної інженерії "Моделі та методи програмної інженерії (Software Engineering Models And Methods)" складається із 4 основних тем (рис 2.10), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються методів та моделей програмної інженерії та їх використання в програмній інженерії .

Моделювання (Modeling) є усталеною і загальноприйнятою інженерною методикою, яка допомагає інженерам з розробки програмного забезпечення зрозуміти вимоги предметної області та отримати інформацію для

зацікавлених сторін про очікувані характеристики та можливості програмного забезпечення. Зацікавлені сторони - це ті особи або сторони, які мають інтерес до програмного забезпечення (наприклад, користувач, покупець, постачальник, архітектор, сертифікаційний орган, оцінювач, розробник, інженер програмного забезпечення та, можливо, інші).

Метою моделювання ПЗ є розуміння процесів, що мають місце в програмній системі, яка розроблюється та її складових частинах(модулях, об'єктах тощо). Моделювання дозволяє вирішити чотири основні завдання:

1. Візуалізувати систему в її поточному або бажаному для замовника стані.
2. Описати структуру або поведінку системи.
3. Отримати шаблон, що дозволяє сконструювати систему.
4. Документувати прийняті рішення, використовуючи отримані моделі.

Чим більша і складніша система, тим більшого значення набуває моделювання при її розробленні. Без моделі складну систему неможливо сприйняти як одне ціле (навіть програмний еквівалент собачої будки суттєво виграє від застосування моделювання). Людське сприйняття складних сутностей є обмеженим. Моделювання системи чи об’єкта дозволяє звузити проблему, зосередивши увагу в кожен момент тільки на певних її аспектах (принципу «розділяй і володарюй»). Складне завдання завжди легше вирішити, якщо розділити його на менші. Моделювання підсилює можливості людського інтелекту. Правильно обрана модель дозволяє реалізовувати проекти використовуючі вищі рівні абстракції, що дозволяє отримувати більш якісні програмні продукти.

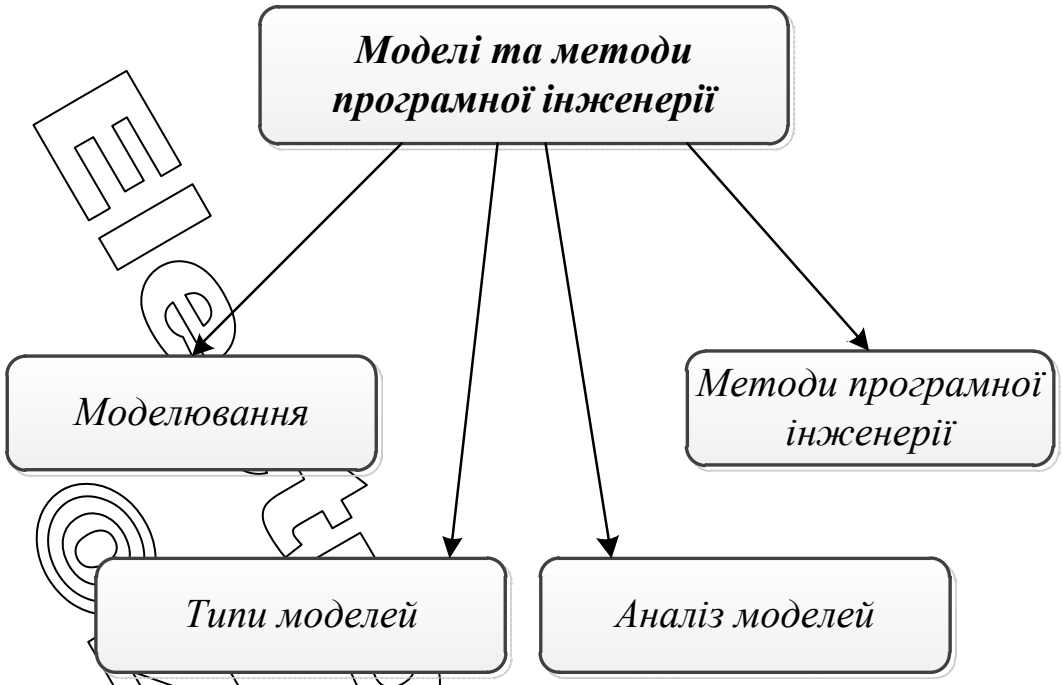


Рис.2.10 – Склад розділу
"Моделі та методи програмної інженерії
(Software Engineering Models And Methods)"

Типи моделей (Types of Models) розрізняються в залежності від цілей проведення моделювання. Типова модель базується на об'єднанні елементів в одну систему підмоделей. При цьому кожна підмодель є частковою характеристикою і створена для певної цілі. При використанні моделювання при реалізації програмного проекту зазвичай використовують три типи моделей:

- інформаційні моделі;
- поведінкові моделі;
- структурна модель.

Інформаційна модель – модель об'єкта, представлена у вигляді інформації, що описує істотні для даного розгляду параметри та змінні величини об'єкта, зв'язки між ними, входи і виходи об'єкта і дозволяє шляхом подачі на модель вхідних величин моделювати можливі стани об'єкта.

Поведінкова модель – модель суть, якої полягає в тому, що особа, яка приймає рішення не має повної, достовірної

інформації про певні сценарії роботи програмного забезпечення і приймає рішення приймає на основі поведінки програмного забезпечення при моделюванні.

Основні характеристики поведінкової моделі полягають в тому, що особа, яка приймає рішення:

– не має повної інформації щодо ситуації прийняття рішення;
– не має повної інформації щодо всіх можливих альтернатив;
– не здатна або не схильна (або і те, і інше) передбачити наслідки реалізації кожної можливої альтернативи.

Структурна модель – модель, що відображає фізичний або логічний склад програмного забезпечення включаючи його різні складові частини. Структурне моделювання встановлює певну межу між програмним забезпеченням, що реалізується або моделюється та середовищем, в якому воно повинне працювати.

Аналіз моделей (Analysis of Models) надає інженеру програмного забезпечення можливість вивчення, пояснення та розуміння структури, функцій, можливостей використання програмного забезпечення. Аналіз побудованих моделей необхідний для забезпечення повноти, узгодженості та коректності для забезпечення адекватності моделювання та якості отриманих результатів при їх використанні зацікавленими сторонами.

Методи програмної інженерії (Software Engineering Methods) призначені для забезпечення організованого та системного підходу до розробки програмного забезпечення та дозволяють суттєво вплинути на успіх програмного проекту.

2.11 ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SOFTWARE QUALITY)

"Якість програмного забезпечення (Software Quality)" - це розділ програмної інженерії присвячений вивченню набору якостей продукту (сервісу або програм), що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника. Поняття якості має різні інтерпретації залежно від конкретної програмної системи і вимог до неї. Крім того, у різних джерелах таксономія (класифікація) характеристик у моделі якості розрізняється.

Моделі мають різну кількість рівнів і повністю або частково збігаються щодо набору характеристик якості. Наприклад, модель якості МакКолла на найвищому рівні має три характеристики: функціональність, модифікованість і переносність, а на нижчих рівнях моделі – 11 підхарактеристик якості і 18 критеріїв (атрибутів) якості.

Стандарт ISO 9126:2001 регламентує зовнішні і внутрішні характеристикиякості. Перші відображають вимоги до функціонування програмного продукту. Для кількісного встановлення критеріїв якості, за якими буде здійснюватися перевірка і підтвердження відповідності ПЗ заданим вимогам, визначаються відповідні зовнішні вимірювані властивості (зовнішні атрибути) ПЗ, метрики (наприклад, час виконання окремих компонентів), діапазони зміни значень і моделі їх оцінки.

Внутрішні характеристики якості і внутрішні атрибути програмного забезпечення використовуються для складання плану досягнення необхідних зовнішніх характеристик якості продукту. Для квантифікації внутрішніх характеристик якості застосовують внутрішні метрики, як інструмент перевірки відповідності проміжних продуктів внутрішнім вимогам до

якості, які формулюються на процесах, що передують тестуванню.

Зовнішні і внутрішні характеристики якості відображають властивості програмного забезпечення (працюючого або не працюючого), а також погляд замовника і розробника на властивості та функції програмного забезпечення. Безпосереднього кінцевого користувача програмного забезпечення цікавить експлуатаційна якість програмного забезпечення – сукупний ефект від досягнення характеристик якості, що вимірюється строком результату, а не властивістю самого ПЗ. Це поняття ширше, ніж будь-яка окрема характеристика (наприклад, зручність використання або надійність).

Якість може підвищуватися за рахунок постійного поліпшення використовуваного продукту виявленням, усуненням дефектів у ПЗ і їх запобіганням.

Розділ програмної інженерії "Якість програмного забезпечення (Software Quality)" складається із 4 основних тем (рис 2.11), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються оцінки якості, процесів та інструментів, що використовуються для реалізації та підтримки якості програмного забезпечення.

Основи якості програмного забезпечення (Software Quality Fundamentals) визначає необхідні характеристики якості програмного забезпечення. Досягнувши домовленості про те, що становить якість для всіх зацікавлених сторін і чітко передаючи цю угоду інженерам-програмістам, потрібно, щоб багато аспектів якості були формально визначені та обговорені. Розробник повинен розуміти якісні концепції, характеристики, цінності та їх застосування для розроблювального або технічного обслуговування програмного забезпечення. Також вимоги до програмного забезпечення впливають на методи вимірювання та критерії

прийняття для оцінки ступеня за якого програмне забезпечення та відповідна документація досягають бажаного рівня якості.

Процеси управління якістю програмного забезпечення (Software Quality Processes) застосовується до всіх аспектів процесів, продуктів і ресурсів та визначає процеси, відповідальних за процес, а також вимоги до процесів, методи та засоби вимірювання процесів і аналізу й оцінювання результатів,а також можливі канали зворотного зв'язку між усіма життєвими циклами програмного забезпечення. Процеси управління якістю містять багато дій. Деякі з них дозволяють безпосередньо знаходити дефекти, в той час, як інші допомагають визначити де саме є важливим провести більш детальні дослідження, після чого, проводяться роботи по безпосередньому виявленню помилок. Багато дій також можуть вестися з метою досягнення і тих і інших цілей. Планування якості програмного забезпечення включає:

- визначення необхідного продукту в термінах характеристик якості;
- планування процесів для отримання необхідного продукту.

Ці процеси відрізняються від процесів управління якістю програмного забезпечення, як таких, що, в свою чергу, спрямовані на оцінку планованих показників якості, а не на реальну реалізацію цих планів. Процеси управління якістю повинні адресуватися питань, наскільки добре продукт буде задовольняти потребам замовника і вимогам зацікавлених осіб, володіти цінністю для замовника і заінтересованих осіб і якістю, необхідним для відповідності сформульованим вимогам до програмного забезпечення.

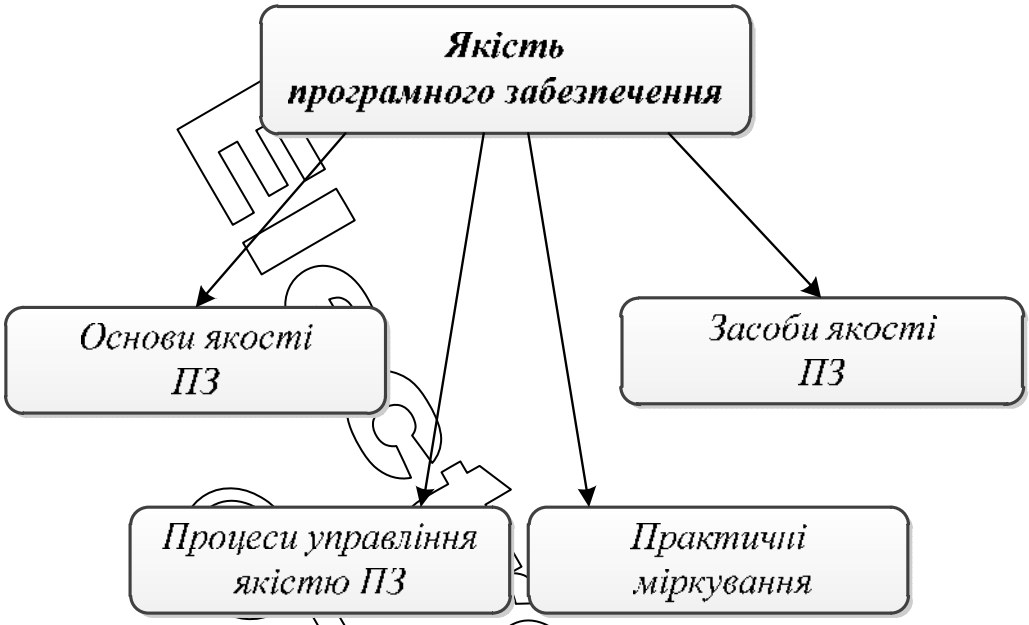


Рис.2.11 – Склад розділу "Якість програмного забезпечення (Software Quality)"

Практичні міркування (Practical Considerations) – комплекс процесів пов'язаних з оцінкою якості програмного забезпечення, з урахуванням факторів впливу на якість програмного забезпечення та їх аналізу, оцінюванням дефектів та помилок в програмному забезпеченні, а також з вимірюванням якості програмного забезпечення.

Засоби якості програмного забезпечення (Software Quality Tools) – інструменти, що включають статичні та динамічні інструменти аналізу. Інструменти статичного аналізу вводять вихідний код, виконують синтаксичний та семантичний аналіз без виконання коду та представляють результати для користувачів. Існує велика різноманітність в глибині, ретельності та масштабах інструментів статичного аналізу, які можуть застосовуватися до артефактів, включаючи моделі, на додаток до вихідного коду. Розрізняють наступні інструменти статичного аналізу:

– інструменти, що полегшують та частково автоматизують огляди та перевірки документів та

кодів (дозволяють оцінити маршрути роботи різних учасників, щоб частково автоматизувати та контролювати процес перегляду результатів роботи; дозволяють користувачам виявляти та оповіщати розробників про дефекти під час перевірок та оглядів для подальшого виправлення);

– деякі інструменти допомагають організаціям проводити аналіз ризику безпеки програмного забезпечення(забезпечують автоматичну підтримку для аварійного режиму, аналізу ефектів (FMEA) та аналізу дерева несправностей (FTA));
– інструменти, що підтримують відстеження програмних проблем, забезпечують введення аномалій, виявлених під час тестування програмного забезпечення та подальшого аналізу, розгашування та вирішення;
– інструменти, що аналізують дані, отримані в середовищах програмного забезпечення та тестових середовищах програмного забезпечення, і створюють візуальні відображення кількісних даних у вигляді графіків та таблиць.

2.12 ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ (SOFTWARE ENGINEERING PROFESSIONAL PRACTICE)

"Професійна практика програмної інженерії (Software Engineering Professional Practice)" - це розділ програмної інженерії присвячений вивченню основних знань, навичок і відносин, якими повинні володіти розробники програмного забезпечення.

Поширення програмного забезпечення та його використання у суспільному та особистому житті має глибокий вплив на наш добробут і соціальну гармонію. Розробники програмного забезпечення мають вирішувати унікальні технічні проблеми при розробці програмного забезпечення для забезпечення відповідних особливостей, надійності і якості функціонування. Саме тому термін "професійна практика" відноситься до способу роботи над проектом з дотриманням певних стандартів або критеріїв, як в виконанні робіт над проектом, так і в процесах, які пов’язані з життєвим циклом кінцевого продукту з дотриманням стандартів і критеріїв якості прийнятих в програмній інженерії. Тобто розробник програмного забезпечення у своїй діяльності має дотримуватися загально прийнятих методик, стандартів і рекомендацій, які створені професійним суспільством (ACM, CS IEEE, BCS, IFIP).

Стандарти і критерії якості, які використовуються в "професійній практиці" включають технічні та нетехнічні розділи, які використовуються в процесі створення програмного забезпечення.

Розділ програмної інженерії "Професійна практика програмної інженерії" складається із трьох основних тем (рис 2.12), які розглядаються як сукупність логічно пов'язаних тем, що безпосередньо стосуються професійної практики програмної інженерії.

Професіоналізм (Professionalism). Для розробника відбувається шляхом дотримання кодексів етики та професійної поведінки, стандартів та практик, які встановлюються професійним співтовариством розробників. Професійне співтовариство часто представлено одним або кількома професійними групами, які розроблюють та публікують кодекси етики та професійної поведінки, а також критерії допуску до спільноти. Ці критерії є основою для

діяльності з акредитації та ліцензування та можуть бути використані як захід для визначення інженерної компетенції або некомпетентності розробника.

Робота у команді розробників ПЗ та психологічні аспекти професійної практики програмної інженерії (Group Dynamics and Psychology). Робота на програмним проектом за звичай – це результат злагодженої роботи команди. Розробники повинні мати можливість взаємодіяти з іншими членами проектної групи та сторонніми розробниками при необхідності та можливості, щоб функціональні можливості відповідали потребам та очікуванням замовника. Знання групової динаміки та психології є об'єктом взаємодії з клієнтами, колегами, постачальниками та підлеглими з вирішення проблем пов'язаних із створення, просуванням та підтримкою програмного забезпечення.

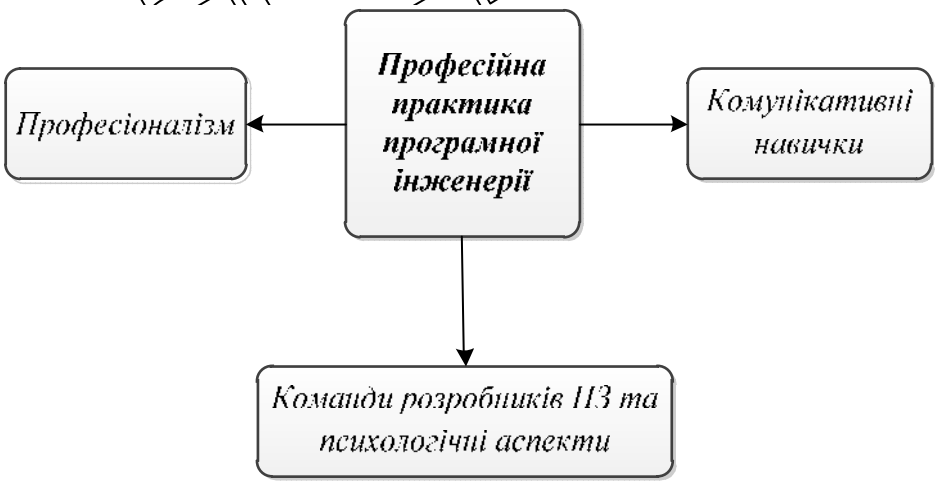


Рис.2.12 – Склад розділу
"Професійна практика програмної інженерії
(Software Engineering Professional Practice)"

Комунікативні навички (Communication Skills). При реалізації програмного проекту є важливо, щоб інженер-програміст вмів спілкувався, як усно, так і письмово. Успішне виконання програмних вимог та в заданий термін залежить від чіткого розуміння між розробником програмного забезпеченням з одного боку та клієнтами, керівниками,

колегами та постачальниками з іншого. Успішна реалізація програмного продукту багато в чому залежить від здатності досліджувати, розуміти та підсумовувати усну та письмову інформацію надану в деяких випадках з різних джерел. Приймання продукту клієнтом та безпечне використання продукту залежать від відповідної підготовки програмної документації. Отже, успіх кар'єри розробника програмного забезпечення залежить від здатності ефективно та своєчасно вести усну та письмову комунікацію з усіма задіяними в розробці сторонами.

2.13 ЕКОНОМІЧНІ АСПЕКТИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ (SOFTWARE ENGINEERING ECONOMICS)

"Економічні аспекти програмної інженерії (Software Engineering Economics)" - це розділ програмної інженерії пов’язаний з економічними процесами, які пов’язують економічні аспекти програмної інженерії, її основні терміни, поняття і загальну практику розробки програмного продукту насамперед виходячи із цілей пов’язаних з бізнес-перспективами.

Успіх програмного продукту, рішень чи послуги пов’язаних з використанням програмного продукту безпосередньо залежить від якості управління бізнес-процесами на всіх етапах життєвого циклу програмного продукту. Тим не менш, у багатьох компаніях і організаціях, які пов’язані з розробкою програмного забезпечення, виконання послуг з підтримки еволюції та функціонування програмного забезпечення, залишаються невизначними. Така невизначеність в свою чергу впливає на ризики, які

визначають ділові відносини між замовниками та розробниками, а також на економічні процеси, які їх пов'язують. В цьому разі якість як економічних відносин так і ділових безпосередньо впливає на повноту та якість розробки програмного продукту.

В загальному визначенні "економічні аспекти програмної інженерії" - це вивчення цінностей, витрат, ресурсів та їхніх відношень відповідно до контексту ситуації на всіх етапах життєвого циклу програмного продукту. Тобто економічні аспекти пов’язані з розподіленням ресурсів з ціллю забезпечення виготовлення та підтримки програмного продукту. Адже, розробка програмного продукту завжди пов’язана з бізнес-цілями організації, яка є замовником розробки програмного продукту або його використовує при досягненні поставлених цілей.

Аналіз економічних факторів на всіх етапах життєвого циклу програмного продукту дає можливість вивчити атрибути програмного забезпечення та процеси пов’язані із життєвими циклом програмного продукту на системній основі. Такий економічний аналіз використовуються при аналізі і прийнятті рішень в рамках організації процесів життєвого циклу програмного продукту, інтегрованих в процеси виробництва та продажу в галузі програмної інженерії.

Використання апарату економічного аналізу при прийнятті рішень є важливими, тому що розробники програмного продукту мають можливість оцінити рішення не тільки з точки зору вирішення суто технічної задачі, а й з точки зору бізнесу. Адже багато інженерних пропозиції і рішень стосовно дилеми розбити або купити готовий той чи інший модуль або програму, повинні мати глибоке внутрішнє економічне підґрунтя.

Розділ програмної інженерії "Економічні аспекти

програмної інженерії (Software Engineering Economics)" складається із п'яти різних тем (рис 2.13), які необхідно розглядати як сукупність логічно пов'язаних тем, що безпосередньо стосуються економічних аспектів інженерії програмного продукту.

Економічні основи створення ПП (Software Engineering Economics Fundamentals) в базисі теорії програмної інженерії пов'язані з прийняття рішень в бізнес-контексті узгодження технічних рішень з бізнес-цілями організації замовника або користувача програмного продукту відповідно до структури приведеної на рис. 2.14. Прийняття таких рішень на сам перед базується на забезпеченні економічних аспектів, які виникають на всіх життєвих циклах програмного забезпечення (пропозиції, грошового потоку, тимчасової вартості грошей, термінів, інфляції, знецінення, заміни), прийняття рішень непов'язаних з отриманням прибутку (аналіз витрат і вигод, аналіз оптимізації створення програмного продукту), оцінки економічного ризику і невизначеності (методи оцінки прийняття рішень в умовах ризику і невизначеності), а також врахування атрибутів, пов'язаних з прийняттям рішень (значення і вимір ваги атрибутів, використання компенсаційних і не компенсаційних методів врахування атрибутів та критеріїв).

Економічні аспекти життєвого циклу (Life Cycle Economics) програмного продукту призначені для економічного аналізу, оцінки та управління програмним продуктом на всіх стадіях життєвого циклу відповідно до бізнес-цілей організації замовника або кінцевого користувача та включають необхідні для цього методи, процеси та інструменти, які дозволяють дотримуватися поставлених цілей залишаючись в рамках бюджету.

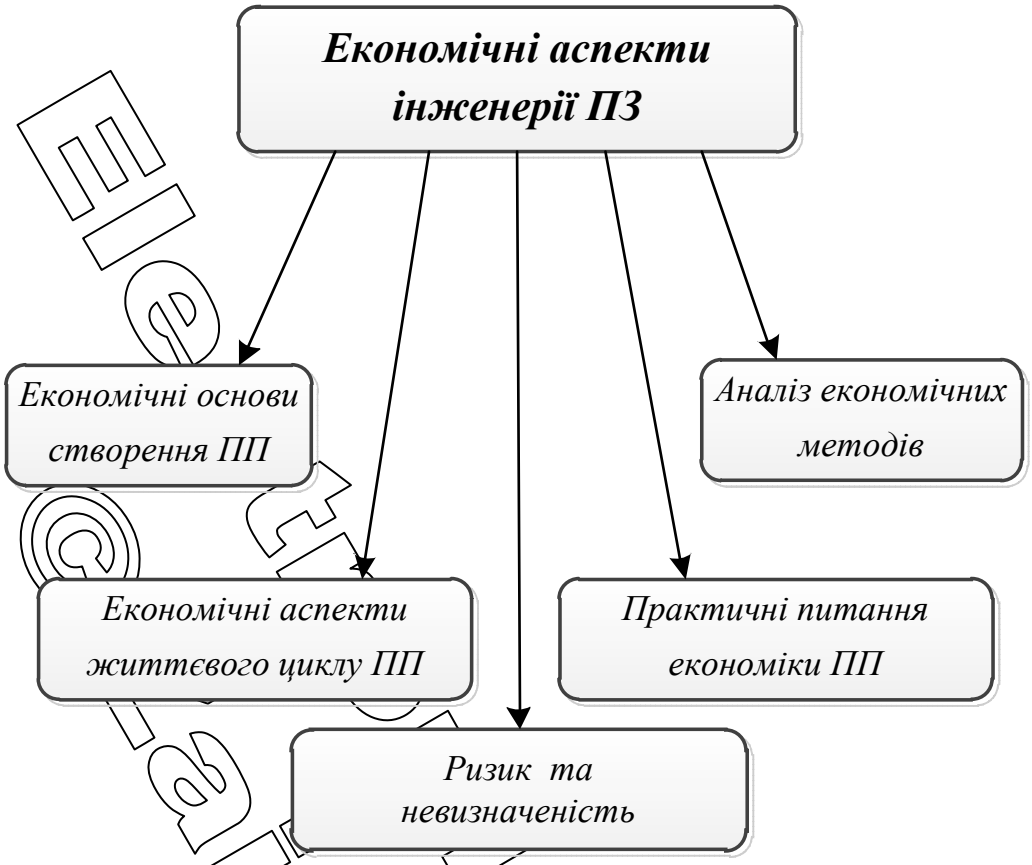


Рис. 2.13 – Склад розділу
"Економічні аспекти програмної інженерії
(Software Engineering Economics)"

Ризик та невизначеність (Risk and Uncertainty) визначаються причинно-наслідковими зв'язками економічних ризиків, що виникають внаслідок взаємозв'язків економічних суб'єктів (виконавця та замовника або користувача) та обумовлених здебільшого хаотичним, недальновидною поведінкою суб'єктів або вірогідністю виникнення випадкових подій чи впевненості в їх настанні, що впливають на економічні ризики та несуть невизначеність на будь-якій стадії життєвого циклу програмного продукту.

Методи економічного аналізу (Economic Analysis Methods), що використовуються в програмній інженерії базуються на математичних та логічних інструментах моделювання та аналізу, що дозволяють при створенні та підтримці програмного продукту аналізувати та прогнозувати

мінімально допустиму норму прибутковості, витрати і доходи на кожному етапі життєвого циклу ПЗ та тощо.

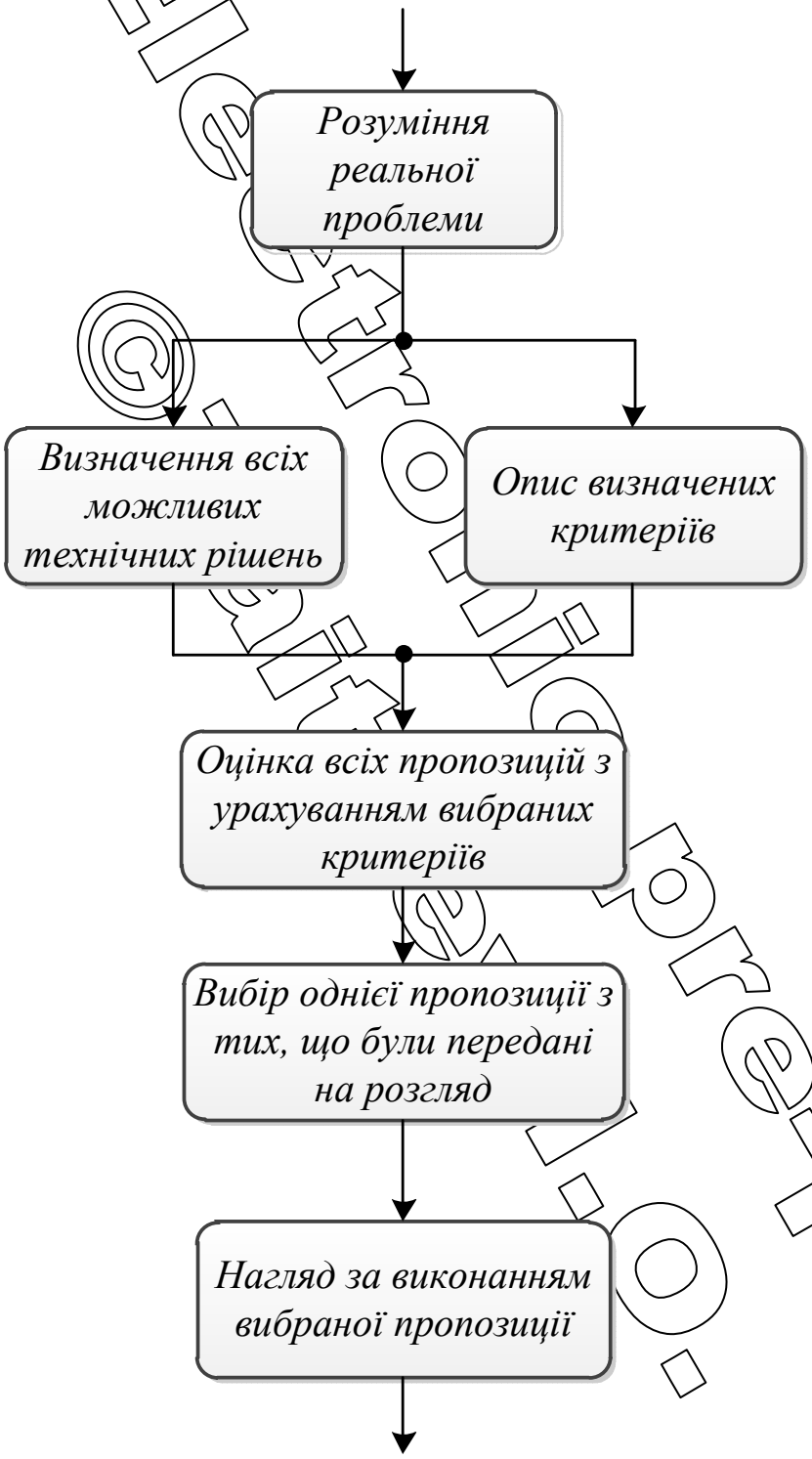


Рис. 2.14 – Етапи прийняття рішень при виборі оптимальної пропозиції

Практичні питання (Practical Considerations) застосування економічних аспектів в програмній інженерії базуються на вивченні, аналізі набутого досвіду та практичному застосуванні критеріїв "достатності" в реальних програмних проектах для оцінки різних механізмів пошуку та контролю якості їх виконання. Ці критерії залежать від бізнес-цілей і пріоритетів та можливих альтернатив, а також від вагових коефіцієнтів таких критеріїв як базовий рейтинг програмних вимоги, метрик якості програмного забезпечення, кінцевої або проміжної вартості програмної продукції.

2.14 ОСНОВИ ОБЧИСЛЮВАЛЬНИХ ТЕХНОЛОГІЇ ПРОГРАМНОЇ ІНЖЕНЕРІЇ (COMPUTING FOUNDATIONS)

"Основи обчислювальних технології програмної інженерії (Computing Foundations)" – це розділ програмної інженерії яка охоплює вивчення основних принципів створення та еволюції не тільки програмного забезпечення та операційного середовища, а також і апаратного забезпечення. Адже програмне забезпечення не може існувати у вакуумі або працювати без комп'ютера (апаратного забезпечення), ядро в такому середовищі комп'ютер і його різні компоненти. Знання про комп'ютер і його основні принципи роботи і є основою, на якій базується розробка сучасного програмного забезпечення. Таким чином, всі розробники програмного забезпечення повинні бути добре освідченні в галузі знань "Обчислювальних технологій".

Галузь знань "Обчислювальні технології *(Computing Foundations)*" розділяють на сімнадцять різних тем (рис 2.15), які необхідно розглядати як сукупність логічно пов'язаних тем безпосередньо пов’язаних з розробкою програмного

забезпечення в цілому і конструювання програмного забезпечення зокрема.

Методи вирішення проблем(Problem Solving Techniques) призначені для аналізу, розуміння та пошуку методів вирішення проблеми в незалежності від попереднього досвіду. Методологія вирішення проблеми складається з наступних основних етапів:

- опис проблеми;
- формулювання проблеми;
- аналіз проблеми;
- обрання методів вирішення проблеми;
- створення ПЗ для вирішення проблеми.

Окрім основних виділяють і допоміжні етапи серед яких можна окремо виділити етап запровадження та етап зворотного зв'язку, хоча фактична кількість етапів визначається відповідно до проблеми, що вирішується.

1) Опис проблеми відноситься до першого етапу на якому основна увага приділяється первинному розумовому аналізу, що проводиться з цілю вирішити або почати процес вирішення проблеми. Перший етап дозволяє описати проблему загалом і є дуже важливим. Адже формулювання самої проблеми є не менше складним етапом, а ніж її вирішення, яка в свою чергу напряму залежить від повноти й коректності формулювання проблеми, вимог, обмежень та розуміння бажаного кінцевого результату.

Існує багато способів початку визначення та опису проблеми, причому кожен спосіб використовує різні інструменти і процеси. Найбільшого поширення набули два способи вирішення проблем. Згідно з першим, проблемою вважається ситуація, коли поставлені цілі недосяжні. Відповідно до другого, проблемою вважають потенційну можливість.

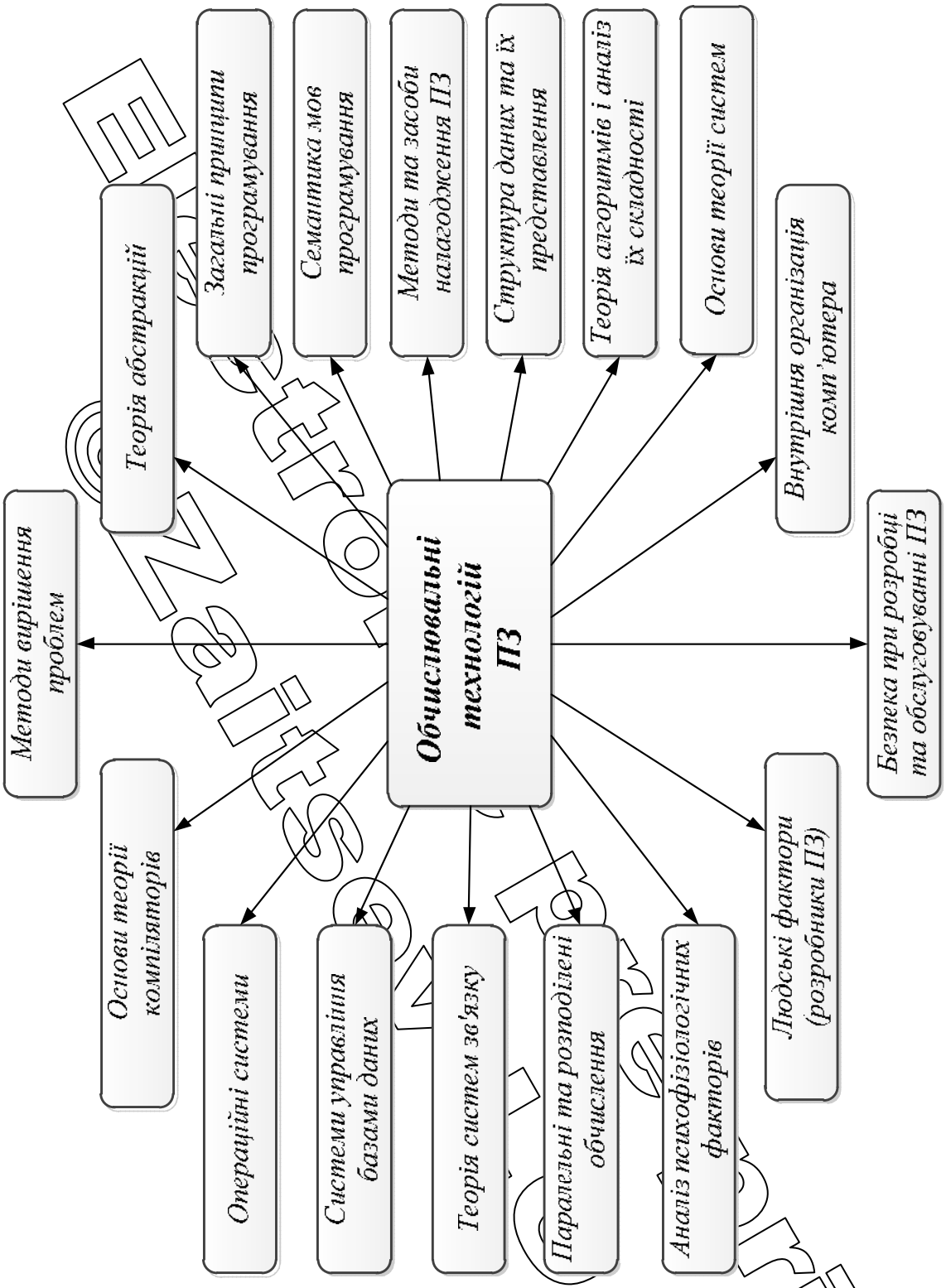


Рис. 2.15 – Склад розділу
"Основи обчислювальних технологій програмної інженерії (Computing Foundations)"

Наприклад, активний пошук способів підвищення ефективності програмного забезпечення, навіть якщо програмне забезпечення повністю задовольняє потреби користувачів (покращання функціональності або швидкодії).

З метою з'ясування причин виникнення проблеми слід зібрати та проаналізувати потрібну внутрішню та зовнішню інформацію, яку можна отримати на основі формальних і неформальних методів. Така інформація ділиться на:

– *ревалентна інформація* – це інформація, що є основою для рішення, що містить точні та відповідні данні про проблему.
– *недоречна інформація* – це інформація, що не впливає на рішення користувачів при оцінці ними попередніх, теперішніх або майбутніх можливостей програмного продукту.

Важливо усвідомити відмінності між ревалентною та недоречною інформацією й уміти відокремити їх.

В процесі опису проблеми можуть виникають ситуації коли багато з можливих рішень є нереалістичними, що насамперед пов’язано з обмеженістю ресурсів. Тому важливо на наступному етапі, після визначення проблеми, сформулювати обмеження, що виникають при вирішенні проблеми. Деякими загальними обмеженнями є: неадекватність засобів; недостатня чисельність працівників, котрі мають потрібну кваліфікацію та досвід; неспроможність закупити ресурси за прийнятними цінами; потреба в технології, яку ще не розроблено або занадто дорога; гостра конкуренція; закони й етичні міркування; зменшення повноважень.

2) Формулювання проблеми. Після того, як проблема описана переходять до формулювання проблеми. На етапі формулювання проблеми відбувається постановка завдання, аналіз обмежень та опис бажаного кінцевого результату

вирішення проблеми.

Хоча універсального способу формулювання проблеми не існує, загалом проблема повинна бути визначена таким чином, щоб полегшити розробку рішень. Загальні методи при формулюванні проблеми враховують також фактори, розміщені за межами організації (наприклад закони) при цьому аналізуючи нинішній і бажаний стан, при цьому в разі необхідності залучаючи незалежних експертів. Також при аналізі обмежень встановлюються критерії, які є рекомендаціями з оцінювання рішень.

3) Аналіз проблеми. Після того, як завершено формулювання проблеми переходять до аналізу проблеми. Даний етап полягає в аналізі проблеми, що допомагає структурувати пошук рішення проблеми. Розділяють чотири типу аналізу, які розділяються на:

- аналіз проблеми, в якому насамперед визначаються найбільш термінові або важливі фактори;
- аналіз проблеми, в якому насамперед визначається причина проблеми;
- аналіз проблеми, в якому насамперед визначаються дії, які необхідні для усунення проблеми або усунення причини її виникнення;
- аналіз проблеми, в якому насамперед визначаються необхідні дії для усунення будь-яких повторів проблеми або появи нових проблем, що потребують початкового визначення.

4) Обрання методів вирішення проблеми. Після того, як завершено аналіз проблеми переходять до обрання методів вирішення проблеми. Даний етап полягає в структуруванні та обранні "кращої" стратегії (методу) вирішення проблеми ("кращий" може вибиратися з максимізації досягнення - швидкодії, дешевизни, більш придатних для використання факторів, різних можливостей, функціональності, дизайну

тощо). Пошук оптимального методу має базуватися на точній інформації про проблему.

5) Створення програмного забезпечення для вирішення проблеми. Після того, як завершено обрання методів вирішення проблеми переходять до наступного етапу. Даний етап полягає в реалізації обраного методу вирішення проблеми у вигляді програмного забезпечення (обраною мовою програмування), на основі розробленого алгоритму.

Перше завдання у вирішенні проблеми за допомогою апаратно-програмних засобів комп'ютера створити відповідний до обраного методу алгоритм та відповідну структуру даних для її вирішення. Наступним кроком є реалізація програмного забезпечення у відповідності до алгоритму.

Тобто, вирішення проблеми за допомогою обчислювальної техніки можна розглядати як поступовий процес трансформації проблеми від перетворення опису проблеми до її вирішення.

Загалом таке перетворення може бути розбите на три етапи:

а) Розробка алгоритмів відповідно до опису проблеми.

б) Застосування алгоритмів до вирішення проблеми.

в) Створення програмного забезпечення відповідно до алгоритмів вирішення проблеми.

Перетворення опису проблеми в алгоритм і алгоритми в програмне забезпечення слід виконувати за правилом"покрокового уточнення» (Систематичного розкладання), яке починається з опису проблеми, запису її у вигляді завдання, і далі рекурсивного розкладання завдання на кілька простих підзадач до того моменту поки завдання не набуде простого рішення. Існує три основних способи розкладання: послідовні, умовні і ітераційні.

Теорія абстракцій (Abstraction) в програмній інженерії використовується, як одна із технік, що допомагає програмісту при реалізації програмних проектів. Теорія абстракцій базується на вирішенні поставлених задач шляхом узагальнення процесів, що описують об’єкти або явища. Таке узагальнення проводиться до масштабів базової концепції, проблеми або опису спостережуваного явища, що дозволяє розробнику зосередитися на розумінні процесів в вигляді "великої картини" при реалізації програмного забезпечення. Ступінь абстракції визначається в кожному програмному проекті індивідуально в залежності від вибраного рівня абстракції для відображення основних залежностей між процесами програмного забезпечення. Рівень абстракції є способом приховування деталей реалізації певного набору функціональних можливостей. В програмному забезпеченні використовують рівні абстракції, які включають семирівневу модель OSI (рівні абстракцій, які використовуються для опису протоколів передачі даних в комп'ютерних мережах) та інше. Абстракція проводитися для досягнення одного з наступних рівнів:

- формальної абстракції - виділення таких властивостей об’єктів або явищ, які поодинці і незалежно від нього не існують (форма або колір).
- змістовної абстракції - виділення тих властивостей об’єктів або явищ, які поодинці мають відносну самостійність (клітина організму).

За типами абстракції поділяють:

- примітивна чуттєва абстракція - не враховує одні властивості об’єкта або явища, виділяючи інші його властивості або якості (виділення форми об’єкт, без розгляду його кольору або навпаки);
- узагальнююча абстракція - дає узагальнену картину об’єкта або явища, яка виділяє загальні властивості

досліджуваних об'єктів або явищ;

- ідеалізація - заміщення реального емпіричного явища ідеалізованої схемою, вільною від реальних недоліків;
- ізолююча абстракція - тісним чином пов'язана з виділенням певних рис або властивостей, оскільки при цьому виділяється той зміст, на якому зосереджується увага.
- конструктівізація - відволікання від невизначеності меж реальних об'єкта або явища, їх «огрубіння».

Однією з найважливіших навичок програміста є обрання відповідного типа та рівня абстракції при аналізі процесу. Через абстракції програміст отримує інструментарій, який дозволяє розглядати проблеми та знаходити можливі шляхи її вирішення з більш високого рівня концептуального розуміння, зокрема при розробці структур даних, аналізі потоків і т.д.

Загальні принципи програмування (Programming Fundamentals) базуються на методології створення програмного забезпечення з урахуванням можливостей обраної апаратної платформи та операційного середовища, що призначене для вирішення визначених проблем. Взагалі процес програмування (створення програм) є сукупністю процесів проектування, написання коду програми, його налагодження та підтримки еволюційних змін на стадіях життєвого циклу програмного забезпечення. Тому програмісту при написанні коду програми потрібно знати не лише мову на якій відбувається написання коду(програмування), а також володіти предметною областю в якій вирішується проблема, мати досвід роботи із структурами даних, розробки алгоритмів, аналізу вимог до програмного забезпечення, методами програмування та конструювання програмного забезпечення. Такі знання накопичуються на власному досвіді та вивченні помилок сторонніх розробників.

Загальні принципи, які слід використовувати при розробці програмного забезпечення наступні:

• частотний принцип - заснований на виділенні в алгоритмах і в оброблюваних структурах дій і даних за частотою використання. Для дій, які часто зустрічаються при роботі програмного забезпечення, забезпечуються умови їх швидкого виконання. До даних, до яких відбувається часте звертання, забезпечується найбільш швидкий доступ. «Часті» операції намагаються робити більш короткими;

• принцип модульності. Під модулем у загальному випадку розуміють функціональний елемент даної системи, що має оформлення, яке закінчене і виконане в межах вимог до системи, а також засоби сполучення з подібними елементами або елементами більш високого рівня даної або іншої системи;

• принцип функціональної вибірковості є логічним продовженням частотного і модульного принципів і використовується при проектуванні програмного забезпечення, обсяг якого істотно перевершує наявний обсяг оперативної пам'яті. У програмному забезпеченні виділяється деяка частина важливих модулів, які постійно повинні бути в стані готовності для ефективної організації обчислювального процесу. Цю частину програмного забезпечення називають ядром. При формуванні складу ядра потрібно досягати компромісу між розміром ядра та необхідними ресурсами для його безперебійної роботи. Так як до складу ядра входять керуючі модулі та найчастіше модулі, що часто використовуються. Модулі, що входять до складу ядра, постійно зберігаються в оперативній пам'яті;

• принцип генерування визначає спосіб представлення вихідного програмного забезпечення, який дозволяє здійснювати налаштування конфігурації технічних засобів та функціональності програмного забезпечення відповідно до

умов роботи користувача;

• принцип функціональної надмірності враховує можливість проведення однієї і тієї ж роботи (функції) різними засобами. Особливо важливо дотримання принципу при розробці інтерфейсу для врахування психологічних відмінностей у сприйнятті інформації різними користувачами;

• принцип «замовчування» застосовується для полегшення організації зв'язків з системою як на стадії генерації, так і при роботі з уже готовим програмним забезпеченням. Принцип заснований на зберіганні в системі деяких базових описів структур, модулів, конфігурацій обладнання та даних, що визначають умови роботи з програмним забезпеченням;

• загальносистемні принципи - рекомендується застосовувати наступні загальносистемні принципи:

а) принцип включення передбачає, що вимоги до створення, функціонування та еволюції програмного забезпечення визначаються з найбільшою повнотою, що включає в себе програмний продукт на будь-якому етапі свого життєвого циклу;

б) принцип системної єдності полягає в тому, що на всіх стадіях створення, функціонування та розвитку програмного забезпечення його цілісність буде забезпечуватися зв'язками між підсистемами, а також функціонуванням підсистеми управління;

в) принцип розвитку передбачає в програмному забезпеченні можливість його нарощування, вдосконалення компонентів і зв'язків між ними;

г) принцип комплексності полягає в тому, що програмне забезпечення забезпечує зв'язність обробки інформації, як окремих елементів, так і для всього обсягу даних в цілому на всіх стадіях обробки;

д) принцип інформаційної єдності, тобто у всіх підсистемах, засобах забезпечення і компонентах програмного забезпечення використовуються єдині терміни, символи, умовні позначення та способи подання;

е) принцип сумісності - мова, символи, коди і засоби забезпечення програмного забезпечення узгоджені, забезпечують спільне функціонування всіх його підсистем і зберігають відкритою структуру системи в цілому;

ж) принцип інваріантності зумовлює, що підсистеми і компоненти програмного забезпечення інваріантні до оброблюваної інформації, тобто є універсальними або типовими.

Отже, додержання принципів програмування стосується всіх стадії розробки та реалізації програмного продукту і включає в себе аналіз вимог до програми, вибір алгоритму, структури даних та системи програмування; написання (кодування) програми і підготовка даних; налагодження і тестування програми; розробки документації до програмного продукту.

Семантика мов програмування (Programming Language Basics) базується на принципах та методах, що досліджують способи передачі інформації, властивості знаків та знакових систем. У програмній інженерії "семантика" – це інтерпретація (смислове значення) абстрактного синтаксису, виражена у термінах математично строгої формальної моделі. Тобто, це інтерпретація допустимих конструкцій мов програмування.

Точний опис семантики необхідний для доведення правильності (верифікації) програм, тобто доведення, що ця програма видає результат у відповідності з постановкою задачі. На практиці замість верифікації найчастіше

застосовують тестування, але ніяке тестування не може довести правильність програми. Воно може тільки виявити помилки. Точний опис семантики необхідний також для створення компіляторів з даної мови програмування.

В програмній інженерії виділяють семантику (зміст), синтаксис (форма або структура) та прагматику (відношення між знаковими системами і користувачами), як основні розділи семіотики в програмуванні.

Семантику мов програмування можна класифікувати наступним чином:

Операційна семантика використовується для синтаксичних понять мови. У ній функції розглядаються як текстуальні правильно побудовані визначення в математичному розумінні цього терміна.

Аксіоматична (деваційна) семантикам - семантику визначають як набір аксіом або правил виведення, який можна використовувати для виведення результатів виконання цієї конструкції.

Денотаціонная семантика виразам в програмі ставилять у відповідність математичні об'єкти.

Інтерпретаційна семантика - опис операційної семантики конструкцій в термінах мов програмування низького рівня (мова асемблера, машинний код).

Трансляційна семантика - семантики визначається, як конструкцій в термінах мов програмування високого рівня.

Трансформаційна семантика - семантика є основою метапрограмування.

Тобто процес використання комп'ютерів для вирішення проблем включає в себе створення програмного забезпечення, написання коду у вигляді семантично організованої конструкції, яку використовує комп'ютер для поетапного виконання програми. Іншими словами, при написанні програмного забезпечення використовуються семантичні

можливості мови програмування для опису проблеми або розробки алгоритму вирішення проблеми.

Методи та засоби налагодження програмного забезпечення (Debugging Tools and Techniques). Налагодження програми є методичним процесом пошуку і зменшення числа помилок, дефектів та відмов у роботі програмного забезпечення. Помилки програм можуть бути синтаксичні, логічні або помилки в даних, що використовуються в програмі.

Мета налагодження з'ясувати, чому програма не працює або працює некоректно. Як виняток, налагодження може не застосовуватися для дуже простих програм, але й для них рекомендовано проводити налагодження по базовим точкам, тобто перевірку правильності виконання поставленого завдання.

Процес налагодження включає в себе безліч заходів і буває статичним, динамічним або на основі аналізу причин, що призвели до збоїв та неполадок. Статичний виконується на основі перевірки коду, в той час як при динамічний засновано на відстеженні результатів тестування. Усі три методи використовуються на різних стадіях розробки програми.

До засобів статичного тестування належать засоби розрахунку тривалості виконання модулів та їхніх характеристик. Вони дозволяють отримувати середні значення і розподіли розрахунків тривалості аналітично, без виконання програми на машині. Внаслідок розрахунків виявляються компоненти програми, які потребують багато часу для виконання і перевірки виконуваності програмного забезпечення в реальному часі. Ці дані дозволяють знайти деякі помилки порушення надійності функціонування через невідповідність тривалості виконання програмного забезпечення потребам реальної системи.

Засоби динамічного налагодження та тестування можна

розподілити на два типи. Перший безпосередньо забезпечує виконання програм відповідно до тестових завдань, другий тип – допоміжні засоби, які обчислюють результати виконаного тестування і проводять необхідні коригування програм. Під час обробки завдань і тестів провадиться вибір результатів відповідно до завдання. Під час обробки тестів проводиться селекція результатів тестування відповідно до операторів налагодженого завдання, а також їх зіставлення з еталонними значеннями. Деякі результати зберігаються для дальшого їх використання. Результати виконання програм у режимі тестування використовуються розробниками для формулювання висновків про напрями подальшої перевірки правильності програм.

Відповідно до стандарту ANSI/IEEE 829(див. додаток А), результати налагодження і тестування програм на всіх етапах життєвого циклу зберігаються в спеціальних документах:

- загальний опис завдань, призначення та зміст програмної системи, а також опис її функцій відповідно до вимог замовника;
- опис технології розробки системи;
- опис планів тестування різних об'єктів, необхідних ресурсів, відповідних спеціалістів для проведення тестування та технологічних засобів;
- специфікацію тестів, контрольних прикладів, критеріїв та обмежень оцінки результатів програмного продукту, а також процесу тестування;
- облік тестування, звіт про аномальні події, відмови та дефекти з підсумковими результатами тестування компонентів і системи в цілому.

Отже, у при процесі налагодження використовується багато спеціальних засобів, які направлені на поліпшення якісних показників створюваної системи.

Структура даних та їх представлення (Data Structure and Representation). Робота програмного забезпечення пов'язана з використанням даних. Для ефективної роботи данні повинні бути організовані в структури - бази даних. Існує багато типів баз даних, які використовуються при організації процесів в програмному забезпеченні, причому кожен тип бази даних створений і підходить для організації роботи з певними типами програмних модулів.

База даних - це пойменована сукупність зв'язаних структурованих даних, організованих за певними правилами, що передбачають загальні принципи опису, зберігання і маніпулювання для обраної моделі бази даних, незалежно від прикладних задач. Тобто база даних узагальнення і розширення структури даних, як правило зовнішні по відношенню до програмного забезпечення, що її використовує.

В процесі історичного розвитку баз даних знайшли використання наступні моделі структур даних: ієрархічна, мережева та реляційна. При чому бази даних можуть бути побудовані на основі однієї моделі або їх комбінації, в яких використовуються наступні типи даних:

Прості змінні - описують структури, що складаються з одного елемента, тому вони характеризується одним (скалярний) значенням. Ім'я простий змінної характеризує номер комірки (одного або декількох), де зберігається її значення.

Масиви - змінні з індексами описують структури, що складаються з обмеженого безлічі компонент, впорядкованих у відповідності зі значеннями індексів. Число індексів визначає розмірність (одномірні, двовимірні і т.д.). Індекс забезпечує прямий доступ до будь-якого елементу масиву. Елементами масиву можуть бути як прості, так і структуровані дані.

Рядки - впорядковані, обмежені послідовності символів деякого алфавіту.

Записи - структура даних, що складається з фіксованого числа компонент, званих полями, кожна з яких може мати свій тип. Записи дозволяють в зручній формі подавати відомості, таблиці, картотеки, каталоги і т.д.

Списки - ланцюжка записів. Основні операції зі списками: перегляд записів, включити нову запис і виключити запис зі списку. Списки дозволяють створювати об'єкти зі складною змінною структурою.

Таблиці - набір записів, з кожною з яких пов'язане ім'я, зване ключем. Пошук потрібного запису в таблиці проводиться за її ключа. Основні операції з таблицями: знайти запис, включити нову запис і виключити запис з таблиці.

Черги - структури даних організовані за принципом «першим прийшов - першим пішов». Це динамічні структури, число елементів яких може змінюватися в процесі обробки. Обробка елементів черзі ведеться послідовно один за іншим. Додавання нових елементів проводиться в кінець черги. Основні операції з елементами черги: читання, обробка, запис в чергу, видалення з черги

Стеки - структури даних організовані за принципом «останнім прийшов - першим пішов». Приклади: стопка книг, пістолетна скоба, магазин автомата, черга в магазині. Тому ця пам'ять називається магазинної.

Посилання - поля адреси пам'яті, змістом якого є іншого поля пам'яті.

Графи - математичні моделі системи зв'язків між об'єктами. Граф складається з вершин (вузлів) і ребер (гілок) з'єднують вузли розташовані на різних рівнях.

Дерева - зв'язний граф, у якому немає циклів. При вирішенні багатьох прикладних задач буває зручно представляти набори об'єктів у вигляді дерев. Наприклад,

подання двійкових кодів.

Теорія алгоритмів і аналіз їх складності (Algorithms and Complexity) – підрозділ теоретичної інформатики, що займається дослідженням складності алгоритмів для розв'язання задач на основі формально визначених моделей обчислювальних пристроїв.

Складність алгоритмів вимірюється за необхідними ресурсами, в основному, це тривалість обчислень або необхідний обсяг пам'яті. В окремих випадках досліджуються інші міри складності, такі як розмір мікросхем, або кількість процесорів необхідна для роботи паралельних алгоритмів.

Основи теорії систем (Basic Concept of a System). Система - це сукупність взаємодіючих компонентів, що працюють спільно для досягнення певних поставлених цілей. Система поділяються на прості, які складається з двох або кількох "апаратних" компонентів (наприклад: олівець), або складними, які складається з тисяч апаратних і програмних компонентів та операторів, що управляють інформаційними підсистемами. До складних систем відносять:

- многопозиційні системи автоматичного регулювання системи управління, в структуру яких входять цифрові і аналогові обчислювальні машини;

- економічні і соціальні системи – виробництво,попит, міграція населення;

- біологічні системи – багатоклітковіорганізми, популяції, біогеоценози, біосфера.

Визначальною ознакою системи є те, що властивості і поведінку системних компонентів, які впливають один на одного є надзвичайно складним і заплутаними. Тобто коректне функціонування кожного системного компонента залежить від функціонування багатьох інших компонентів. Так, операційне забезпечення може виконувати свої функції, якщо тільки працює процесор і відповідні апаратні засоби.

Процесор може виконувати обчислення, якщо тільки коректно інстальовані програмні системи, що задають ці обчислення.

Внутрішня організація комп'ютера(Computer Organization) базується на досягненнях в сфері розробок напівпровідникових електронних пристроїв, що виконують основні операції в комп'ютері. В основі таких операцій лежать логічні операції, а також операції здвигу та сумування. Керування роботою електронних пристроїв, що об'єднуючись створюють комп'ютер покладене на програмне забезпечення базового (системного) рівня і прикладне програмне забезпечення. Базове ПЗ організує процес обробки інформації в комп'ютері і забезпечує відповідне робоче середовище для прикладних програм. За допомогою прикладних програм на комп'ютері розв'язують конкретні задачі.

Комп'ютер зазвичай складається з наступних пристроїв: процесора, пам'яті, пристроїв вводу-виводу та тощо. Розглядаючи організацію комп'ютера можна виділити чотири абстрактних рівня. Макрорівень - формальний опис всіх функцій комп'ютера або набор інструкцій архітектури (ISA). Мікрорівень - реалізація ISA, тобто те яким чином відбуваються процеси в комп'ютері. Рівень електронних (логічних) схеми - рівень, на якому розглядається функціональний компонент мікроархітектури побудований для виконання простих задач на основі певних логічних правил. Фізичний рівень пристроїв - рівень, на якому розглядають фізичну побудову електронних пристроїв, що може бути на основі одно або багатоканальних метал-оксидних (CMOS) або галію-арсеніду (GaAs) напівпровідникових структур.

Кожен із рівнів абстракцію дозволяє здійснити перехід до верхнього рівня і залежить від рівня, що знаходиться нижче. Важливим в рівнях абстракції ISA є типи даних, інструкції, регістри, режими адресації, архітектура пам'яті, обробка

переривань, процеси вводу-виводу. В цілому, ISA визначає здатність комп'ютера і ті можливості, що доступні інженеру-програмісту при реалізації програмного забезпечення з урахуванням апаратних можливостей обраної комп'ютерної платформи.

Основи теорії компіляторів(Compiler Basics). Більша частина програм, що створюється пишеться на мовах високого рівня. Для виконання такої програми на комп'ютері потрібно перетворити початковий текст програми в еквівалентну об'єктну програму. Отримана таким чином еквівалентна об'єктна програма на машинній мові або вигляді автокода називається програмою, що виконується. В такому випадку програма, яка перетворює початкову програму в еквівалентну називається компілятором. Автокод дуже близький до машинної мови. Більшість команд на автокоді представляє точне символічне представлення машинних команд.

Процес компіляції починається з аналізі початкової програми, а закінчується синтезом еквівалентної об'єктної програми. Такий процес роботи компілятора можна розділити на декілька етапів серед яких можна виділити:

1) Лексичний аналіз. На цьому етапі відбувається аналіз, що має на меті прибрати все зайве (коментарі, роздільники), виділити лексеми, тобто лексичні одиниці, з яких будується програма на машинній мові та перетворити виділені структури на формат внутрішнього або проміжного подання. На цьому етапі йде активна робота з таблицями, в які заноситься інформація про розпізнаних лексем, їх типи, значеннях і т.д. Результатом є текст еквівалентний початкового тексту.

2) Синтаксичний аналіз. На цьому етапі при необхідності вводяться додаткові роздільники і замінюються існуючі для полегшення обробки.

3) Генерація проміжного коду (трансляція). На цьому етапі здійснюється контроль типу та виду всіх ідентифікаторів

та інших операндів. При цьому зазвичай перетворення початкової програми в проміжну (наприклад, польську) форму запису здійснюється

одночасно з синтаксичним аналізом. Синтаксичний аналізатор необхідний для того, щоб з'ясувати, чи задовольняють пропозиції, з яких складається вихідна програма, правилам граматики цієї мови. Поряд з перевіркою синтаксису паралельно відбувається генерація внутрішньої форми подання

програми.

4) Оптимізація коду.

5) Розподіл пам'яті для змінних в готовій програмі.

6) Генерація об'єктного коду та компоновка програмних сегментів. Генерування об'єктного коду та компоновка програмних сегментів - завершальна стадія компіляції. Змінним привласнюються конкретні адреси, при цьому визначається, які змінні будуть розміщені в регістрах. Потім проміжний код транслюється в цільовий тобто у програму, що виконується.

Операційні системи(Operating Systems Basics) - це базові комплекси програмного забезпечення, що виконують управління апаратними ресурсами комп'ютеризованої системи або віртуальної машини, а також забезпечують керування обчислювальним процесом і організовують взаємодію з кінцевим користувачем.

До складу операційної системи входять:

- ядро операційної системи, що забезпечує розподіл та управління ресурсами обчислювальної системи;
- базовий набір прикладного програмного забезпечення, системні бібліотеки та програми обслуговування.

За призначенням операційні системи розділяються на:

- універсальні (для загального використання);
- спеціальні (для розв'язання спеціальних задач);

- спеціалізовані (виконуються на спеціальному обладнанні);
- однозадачні (в окремий момент часу можуть виконувати лише одну задачу);
- багатозадачні (в окремий момент часу здатні виконувати більше однієї задачі);
- однокористувацькі (в системі відсутні механізми обмеження доступу до файлів та на використання ресурсів системи);
- багатокористувацькі (система впроваджує поняття «власник файлу» та забезпечує механізми обмеження на використання ресурсів системи (квоти)), всі багатокористувацькі операційні системи також є багатозадачними;
- реального часу (система підтримує механізми виконання задач реального часу, тобто такі, для яких будь які операції завжди виконуються за наперед передбачуваний і незмінний при наступних виконаннях час).

До основних функцій операційної системи відносять:

- виконання на вимогу програм користувача тих елементарних (низькорівневих) дій, які є спільними для більшості програмного забезпечення і часто зустрічаються майже у всіх програмах (введення та виведення даних, запуск і зупинка інших програм, виділення та вивільнення додаткової пам'яті тощо);
- стандартизований доступ до периферійних пристроїв (пристрої введення-виведення);
- завантаження програм у оперативну пам'ять і їх виконання.
- керування оперативною пам'яттю (розподіл між процесами, організація віртуальної пам'яті);
- керування доступом до даних енергонезалежних

носіїв (твердий диск, оптичні диски тощо), організованим у тій чи іншій файловій системі;

- забезпечення користувацького інтерфейсу;
- організацію мережевих операції, підтримка стеку мережевих протоколів.

Отже, для управління складними комп'ютеризованими системи та їх апаратним забезпеченням відбувається за допомогою операційних систем.

Теорія систем зв'язку (Network Communication Basics) – теорія, що призначена для опису структур та функціонування в них з'єднаного набору комп'ютерів в мережу, що дозволяє користувачам різних комп'ютерів обмінюватися інформаційними ресурсами з іншими користувачами. Мережа полегшує зв'язок між усіма підключеними комп'ютерами і в багатьох випадках може призвести до ілюзії одного, всеосяжного комп'ютера. Кожен комп'ютер або пристрій, підключений до мережі, називається мережевим вузлом.

Використанні комп'ютерів об'єднаних в одну мережу з'явилася технічна можливість з практичної реалізації розподілених обчислень, Інтернет-обчислень та хмарних обчислень.

Комп'ютерні мережі не всі однакові і можуть бути класифіковані за різними характеристиками, включаючи метод з'єднання в мережі, провідні технології, бездротові технології, масштаб, топологію мережі, функції та швидкість. Але класифікація, яка знайома більшості, базується на масштабах створення мереж та може бути розділена на наступні типи:

– *Персональна мережа/Домашня мережа (Personal Area Network/Home Network)* - це комп'ютерна мережа, що використовується для спілкування між комп'ютерами та різними інформаційно-технологічними пристроями,

близькими до однієї людини. Пристрої, підключені до такої мережі, можуть включати ПК, факси, КПК та телевізори. Це основа, на якій будується Інтернет речей.

– *Локальна мережа (LAN)* з'єднує комп'ютери та пристрої в обмеженій географічній зоні, наприклад, навчальному містечку, комп'ютерній лабораторії, офісній будівлі або групі будинків.

– *Мережа відокремленої території, що належить компанії та включає внутрішньофірмову інфраструктуру (Campus Network)* - це комп'ютерна мережа, що складається з взаємоз'єднаних локальних мереж в мережу на обмеженій географічній зоні.

– *Широкосмугова мережа (WAN)* - це комп'ютерна мережа, яка охоплює велику географічну зону, таку як місто чи країна, або навіть міжконтинентальні відстані.

– *Інтернет* - це глобальна мережа, яка з'єднує комп'ютери, розташовані у багатьох країнах.

Існують і інші класифікації, що розподіляють мережі за принципом організації керування мережею, типом зберігання інформації, віртуальні приватні мережі (VPN), бездротові мережі та мережу Інтернет речей.

Інтернет речей (англ. Internet of Things, IoT) - концепція обчислювальної мережі фізичних предметів («речей»), оснащених вбудованими технологіями для взаємодії один з одним або з зовнішнім середовищем, яка розглядає організацію таких мереж як явище, що здатне перебудувати економічні та суспільні процеси, що дозволяють виключати з процесу частини дій і операцій, що потребують участі людини.

Системи управління базами даних(Database Basics and Data Management). Бази даних використовуються при великому обсязі даних або коли логічні зв’язки між елементами даних є важливі. Фактори, що впливають на вибір

моделі організації бази даних при розробці програмного продукту включають в себе продуктивність, фактори доступу, цілісність структури та можливість відновлення після збоїв в роботі.

Системи управління базами даних (СУБД) (Database Management Systems (DBMS)) - це комплекс програмних та лінгвістичних засобів загального або спеціального призначення, що реалізують підтримку створення баз даних, централізованого управління й організації доступу до них різних користувачів в умовах прийнятої технології обробки даних.

Системи СУБД надають наступні можливості при їх використанні в складі програмного забезпечення:

- забезпечення цілісності, безпеки та відновлення інформації у разі її пошкодження;
- забезпечення контрольованого доступу користувачам до даних в базі для пошуку, зміни та формування;
- обслуговування бази даних для додавання, видалення, оновлення і коригування даних в базі даних у формі, що відповідає власним потребам.
- розробки додатків, що використовуються для розробки форм введення даних, запитів, звітів, таблиць.

Паралельні та розподілені обчислення(Parallel and Distributed Computing) використовуються в програмуванні для прискорення виконання задачі при наявності апаратних ресурсів та відповідної програмної можливості.

Паралельні обчислення (Паралелізм) – базується на сукупності математичних, алгоритмічних, програмних і апаратних засобів, що забезпечують можливість паралельного виконання задачі. Ґрунтуються на тому, що великі задачі можна розділити на кілька менших, кожну з яких можна

розв'язати незалежно від інших.

Розподілені обчислення – сукупність протоколів обміну та незалежних апаратних засобів (комп'ютерів, серверів), що представляються користувачу єдиним обчислювачем, придатним для вирішення складної задачі. Розподілені обчислення є окремим випадком паралельних обчислень, тобто одночасного розв'язання різних частин одного обчислювального завдання декількома процесорами одного або кількох комп'ютерів. Тому необхідно, щоб завдання, що розв'язується було сегментоване — розділене на підзадачі, що можуть обчислюватися паралельно. При цьому для розподілених обчислень необхідно враховувати можливу відмінність в обчислювальних ресурсах, які будуть доступні для розрахунку різних підзадач.

Основи аналізу психофізіологічних факторів(Basic User Human Factors) відносяться до аналізу людських факторів, що виникають при розробці програмного забезпечення. Програмне забезпечення розробляється людьми, використовується людьми і підтримується людьми. Якщо з програмним забезпеченням щось не так, люди несуть відповідальність за виправлення цих помилок. Таким чином, важливо створювати програмне забезпечення таким чином, щоб його було легко зрозуміти іншим розробникам. Засоби забезпечення того, щоб програмне забезпечення відповідало цієї мети, численні і варіюються від відповідної архітектури до конкретного типу кодування і використання змінних в модулях, що використовується.

Людські фактори (розробники ПЗ) (Basic Developer Human Factors) пов'язані з особливостями використання розробником одержаної інформації в процесі створення програмного продукту та пов'язаний на сам перед з його продуктивністю та комфортом роботи. Причому, з усіх факторів, що впливають на якість, ключовим є людський

фактор. Пояснюється це тим, що незацікавлений працівник не буде добре працювати навіть на хорошому обладнанні, а зацікавлений шукатиме, знаходити і використовувати будь-які можливості для підвищення своєї кваліфікації і досягнення високої якості готової програмної продукції. Така програмна продукція розробляється для комп'ютерів, але зчитується, аналізується та підтримується людьми. Тобто, важливо, створювати програму таким чином, щоб вона була зрозуміла не тільки для комп'ютера, але й для інших розробників програмного забезпечення(принаймні тих, що працюють над проектом). Засоби, що дозволяють створювати зрозумілий та "красивий" програмний код варіюються від правильної архітектури на макрорівні в певному стилі кодування до логічного використання змінних на макрорівні. Але два основних фактори, що впливають якість програмного коду - це коментарі (документація) і структура (програмні макети) програмного забезпечення.

Для більшості людей написання програми - це лише написання коду програми, але програмування також включає в себе написання коментарів і коментарі є невід'ємною частиною процесу програмування. Коментарі не використовуються комп'ютером і не є інструкціями для його дій, але вони покращують читабельність програм, пояснюючи зміст і логіку коду. Слід пам'ятати, що програми створенні для комп'ютера можуть бути використанні іншими розробниками при їх вивченні або зміні

Коментарі включають: опис коду, маркери коду, резюме коду, опис алгоритму код, і інформацію, яка не може бути виражена в самому коді.

Загальні рекомендації для написання "гарних" коментарів в програмі:

- кожна функція повинна бути пов'язана з коментарями, які пояснюють призначення функції

і її роль у загальній програмі;

- кожен логічний розділ пов'язується з коментарем для пояснення значення і цілі у розділу;
- коментарі повинні визначати можливості по внесенню змін до коду.

Структурування програми, як і коментарі призначені для покращення зрозуміння і модифікації самої програми. Існує багато способів структурування програм, які включають правильне використання білого простору, відступів і дужок для створення угруповань та порожніх рядків.

Безпека при розробці та обслуговуванні програмного забезпечення(Secure Software Development and Maintenance). Розвиток можливостей обчислювальної техніки, засобів, методів і форм автоматизації процесів обробки інформації є важливою проблемою та актуальною проблемою при розробці та обслуговуванні програмного забезпечення. До забезпечення виконання вже звичайних якостей програмного забезпечення (правильності і надійності функціонування) розробники програмного забезпечення повинні також звертати увагу на реалізацію заходів з забезпечення безпеки розроблюваного програмного забезпечення. Безпека програмного забезпечення та процесів його розробки підвищує безпеку програмного забезпечення. Забезпечення безпеки при обслуговуванні програмного забезпечення дозволяє гарантувати відсутність проблем з безпекою даних та елементів програмного забезпечення при його обслуговуванні. Створення безпечного програмного забезпечення обходиться дешевше і відбувається швидше, ніж ітераційне виправлення знайдених недоліки в процесі тестування та експлуатації програмного забезпечення. Забезпечення безпеки в програмному забезпеченні необхідно реалізовувати на всіх етапах життєвого циклу розробки програмного забезпечення. Зокрема, розробка безпечного

програмного забезпечення включає в себе безпеку програмного забезпечення, безпеку розробки програмного забезпечення та безпеку тестування програмного забезпечення. Крім того, безпека повинна також враховуватися при виконанні технічного обслуговування програмного забезпечення, збоїв безпеки та лазівки мають бути виправлені під час обслуговування програмного забезпечення.

2.15 МАТЕМАТИЧНІ ЗАСАДИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ (MATHEMATICAL FOUNDATIONS)

"Математичні засади програмної інженерії (Mathematical Foundations)" – це базові математичні концепції та поняття, що використовуються розробниками програмного забезпечення.

Область знань "Математичні засади програмної інженерії (Mathematical Foundations)" досить сильно відрізняється від типової арифметики, в якій оговорюються і розглядаються лише числа. Базовою основою математики, що використовується інженерами-програмістами є математичні обґрунтування й логіка. Математика допомагає в формальному описі систем з необхідною точністю, що дозволяє отримати програмісту однозначний алгоритм для створення програмного забезпечення. Такий опис, як будь-яка концепція, може бути представлена у вигляді множин чисел, символів, а також зображень, звуків, відео та іншого. Тобто, допомагає інженеру-програмісту відтворити у програмі з відповідною точністю необхідні властивості об’єкту або створити додаток, який відповідає заздалегідь визначеним

вимогам. А також і навпаки, інженер-програміст, використовуючи математичний апарат у своїй діяльності, має змогу розробляти точні абстракції програмного забезпечення, що створюється.

Розробка програмного забезпечення, яке призначене для моделювання роботи об'єктів або керування системами, неможливо створити використовуючи тільки стандартні засоби мов програмування. При створенні програмного забезпечення розробники використовують знання із суміжних наук, до складу яких входить математика. Застосування математичних методів, моделей, а також концепції допомагає інженерам в розумінні логіки роботи системи та створенні відповідного програмного забезпечення вибраною мовою програмування.

Область знань "Математичні методи програмного забезпечення (Mathematical Foundations)" містить у собі набір методів, які допомагають інженеру-програмісту зрозуміти суміжні області знань в контексті досліджуваної системи, представити необхідні процеси на стадії проектування програмного продукту.

Розділ програмної інженерії "Математичні засади програмної інженерії (Mathematical Foundations)" складається із 11 основних тем (рис 2.16), які розглядаються як сукупність логічно консолідованих тем про математичні методи програмної інженерії.

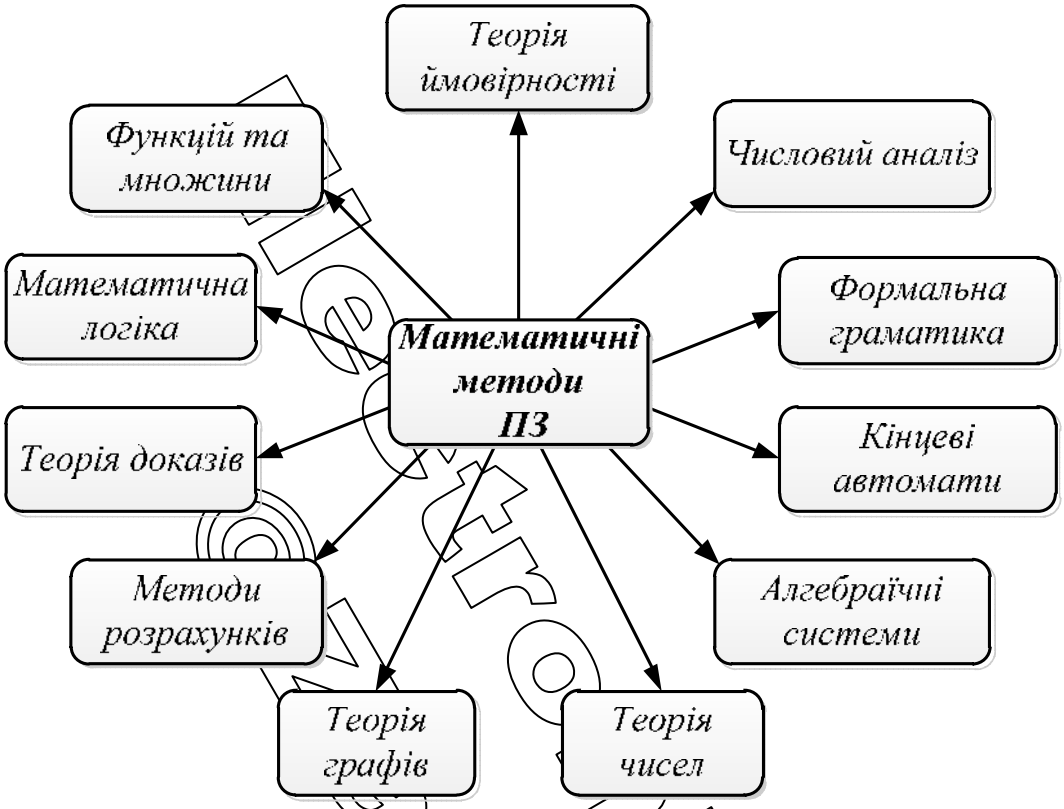


Рис. 2.16 – Склад розділу
"Математичні засади програмної інженерії
(Mathematical Foundations)"

Функції та множини(Set, Relations, Functions). Для зменшення невизначеності при описі систем застосовують математичні функції та елементи теорії множин.

Теорія множин – розділ математики, який вивчає загальні властивості множин і є основою практично всіх математичних теорій. Множиною називається сукупність елементів, що об'єднані за певною ознакою. Множина може бути представлена шляхом перерахування елементів між фігурними дужками, наприклад, S = {1, 2, 3}.

Функцією – залежність кожного елементу однієї множини (області завдання функції), від одного і тільки одного елементу іншої множини (області значень функції).

Математична логіка(Basic Logic) – це наука про математичне мислення і структуру математичних теорій за

допомогою якої довільну «логічну» операцію «можна виразити» через певні еквівалентні комбінації операцій диз'юнкції, кон'юнкції та заперечення.

Теорія доказів (Proof Techniques) – розділ математичної логіки, що вивчає властивості і перетворення формальних доказів, тобто формальних об'єктів, синтаксична правильність яких гарантує семантичну. Тобто в програмній інженерії структури доказів і структури, записані на алгоритмічній мові досить високого рівня, тісно пов'язані. Побудова структури програми та підпрограм в єдиній формі вкладається в доказ, а подібній формі для підпрограм в частину доказу (при умові відкиданням кроків і формул, потрібних лише для обґрунтування результату).

Методи розрахунків (Basics of Counting) – розділ математики, які включає коло питань, що пов’язані із пособами, прийомами, процедурами за допомогою яких здійснюється виконання наближених обчислень із використанням комп'ютерів. Класичні методи математичного аналізу використовуються самостійно (диференціювання і інтеграція) і в рамках інших методів (математичної статистики, математичного програмування). За ознакою отримання точного рішення всі математичні методи діляться на точні (єдине рішення отримується на основі критерію або без нього) і наближені (на основі стохастичної інформації). До оптимальних точних можна віднести методи теорії оптимальних процесів, деякі методи математичного програмування і методи дослідження операцій, до оптимізаційних наближених частину методів математичного програмування, дослідження операцій, економічної кібернетики, евристичні. До неоптимізаційних точних належать методи елементарної математики і класичні методи математичного аналізу, економічні методи, до неоптимізаційних наближених – метод статистичних

випробувань і інші методи математичної статистики.

Теорія графів (Graphs and Trees) - це одна з частин математичного апарату програмної інженерії, яка є зручною мовою для опису програмних (граф-схема алгоритму) та багатьох інших моделей. Стрункa система спеціальних термінів та визначень теорії графів дозволяють просто і доступно описувати складні та "тонкі речі". Зображення дозволяють дати пояснення в найбільш зрозумілій формі для людини, тобто на інтуїтивно зрозумілому для неї рівні. Причому, при використанні для побудови програмного продукту мов графічного програмування дають змогу візуально представити проходження потоків даних, що дозволяє оптимізувати програмний код й досягти усунення збитковостей коду.

Теорія ймовірності (Discrete Probability) – це розділ математики, що вивчає закономірності випадкових явищ, випадкові події, випадкові величини, їхні функції, властивості й операції над ними. Математичні моделі, що використовуються в програмній інженерії описують з деяким ступенем точності апріорної інформації про систему, тобто інформацією, яка може випадково змінюватися в залежності від умов, в яких існує система. Математичним апаратом теорії імовірності, який найчастіше використовується інженером-програмістом є комбінаторика, тобто математичні методи, які дозволяють вирішувати задачі за допомогою методів перестановки, розміщення, комбінація та розбиття, а також переліку або їх композиції.

Теорія кінцевих автоматів (Finite State Machines) базуючись на теорії математичних абстракцій дозволяє описувати шляхи зміни стану об’єкту в залежності від його поточного стану та вхідних даних, за умови, що кількість таких станів є кінцевою. Згідно теорії кінцевих автоматів регулярною подією називається подія, отримана за допомогою

застосування скінченої кількості елементарних операцій диз'юнкції, добутку й ітерації.

Формальна граматика (Grammars) використовується в математиці як спосіб опису формальної мови, тобто виділення деякої підмножини з множини всіх слів деякого скінченного алфавіту, що підкоряються певним закономірностям, послідовностям мовних одиниць та підпорядковуються визначеній синтаксичній структурі. На відміну від мов для спілкування, формальна мова базується на наборі правил. При цьому формальна мова є набір певної кількості слів або рядків, а граматика визначає правила формування послідовності цих слів або рядків.

Числовий аналіз (Numerical Precision, Accuracy, and Errors) базується на методах математичного аналізу основна мета, якого є розробка ефективних алгоритмів для обчислення точних числових значень функцій, рішення алгебраїчних і диференціальних рівнянь, задач оптимізації і т.д. Числовий аналіз базується на тому, що всі цифрові комп'ютери можуть зберігати тільки цифри кінцевої точності тобто комп'ютер не може зберігати нескінченно великі раціональні або будь-які реальні або комплексні числа. Таким чином, числовий аналіз дозволяє наближено проводити дії з числами кінцевої точності. Важливими складовими числового аналізу є точність і похибки, які виникають в результаті дій з числами з кінцевою точністю. В свою чергу, точність характеризує близькість, з якою обчислене значення узгоджується з істинним значенням. З іншого боку, точність - це близькість, з якою два або більше обчислених значення для тієї ж величини узгоджуються один з одним. Іншими словами, точність – це близькість обчисленого числа до його істинного або точного значення. Похибка в свою чергу є різницею між обчисленим значенням числа та його істинним або точним значення.

Алгебраїчні системи (Algebraic Structures) базуються на

теорії математичних абстракцій і дозволяють вивчати властивості аксіоматично заданих алгебраїчних систем, тобто множин(набір певної кількості зв'язаних однотипних елементів найчастище перерахункового типу) разом з визначеними на ній операціями та відношеннями, що задовольняють деякій прийнятій системі аксіом. Визначення множини, які використовуються в алгебраїчних системах дуже схожі на визначення масивів, але й мають свої відмінності:

- кожен елемент займає в масиві певну позицію, яка позначається його індексом, елементи масиву можна переставити, при цьому зміниться їх індекс. Положення елемента в множині залежить від реалізації та під час виконання програми змінюватися не може. При використані множин в програмі інформаційним є тільки наявність або відсутність у ньому елемента з потрібним значенням.

- у масиві може бути скільки завгодно елементів з однаковими значеннями. У множині кожен елемент входить тільки один раз.

Теорія чисел (Number Theory) – розділ математики, який пов'язаний з вивчення властивостей натуральних чисел, розглядом питань подільності і розв'язання алгебраїчних рівнянь з натуральними числами (як правило – це цілі, дробові або раціональні числа). В теорії чисел розглядаються різні числові типи, які включають: цілі числа (integer), реальні числа (real number), натуральні числа (natural number), комплексні числа (complex number), раціональні числа (rational number) і т.д.

Використання базисних аспектів із теорії чисел допомагає розробляти більш оптимізовані алгоритми в обчислювальних завданнях, а також успішно застосовувати чисельні методи при вирішенні різних завдань за допомогою обчислювальної техніки., розвиває вміння знаходити приховані взаємозв'язки між програмними структурами.

2.16 ОСНОВИ ІНЖЕНЕРНОЇ ДІЯЛЬНОСТІ В ПРОГРАМНІЙ ІНЖЕНЕРІЇ (ENGINEERING FOUNDATIONS)

"Основи інженерної діяльності в програмній інженерії (Engineering Foundations)" визначаються, як "застосування систематичного, дисциплінованого, кількісно підходу до структур, обчислювальних машин, програмних продуктів, інформаційних систем або процесів" та присвячена розгляду базових технічних навичок і методів, які використовуються інженерами-програмістами при створені та підтримці еволюції програмного продукту.

Розділ програмної інженерії "Основи інженерної діяльності в програмній інженерії (Engineering Foundations)" складається із 7 основних тем (рис 2.17), які розглядаються як сукупність логічно консолідованих тем про основи інженерної діяльності в програмній інженерії.

Емпіричні методи дослідження (Empirical Methods and Experimental Techniques) призначені для опису і зрозуміння невизначеності у вимогах до програмного продукту та його складових, виявлення джерел невизначеностей і прийняття рішенню по їх усуненню на відміну від класичних інженерних методів. В свою чергу, інженерні методи вирішення проблеми включають пропозиції по рішенню або моделі на основі методів, які використовуються для проведення експериментів або тестів по вивченню запропонованих рішень. Емпіричні методи дослідження широко використовуються в інженерній діяльності для проведення експериментів, спостережень та досліджень.

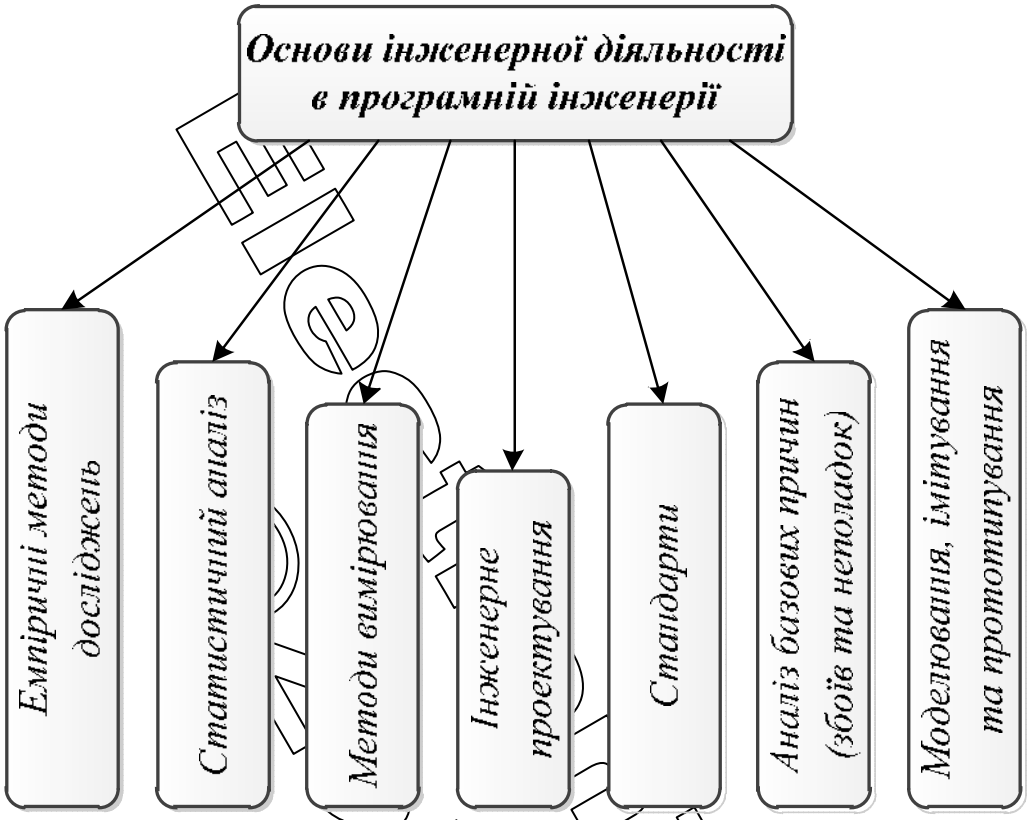


Рис. 2.17 – Склад розділу
"Основи інженерної діяльності в програмній інженерії
(Engineering Foundations)"

Статистичний аналіз (Statistical Analysis) - це процес вивчення, зіставлення та узагальнення інформації про отримані цифрові дані між собою та з іншими даними. При проведенні статистичного аналізу використовують методи: масового статистичного спостереження; групування; табличний; графічний; відносних величин; середніх величин; індексний; кореляційний; інші математичні методи, які використовуються для більш поглибленого вивчення взаємозв`язків між різноманітними явищами. Використання засобів статистичного аналізу інженерами-програмістами допомагає зрозуміти характеристики змінних процесів, а також досліджувати та встановлювати відношення між різними величинами або процесами. Важливо відзначити, що більшість досліджень проводяться на основі зразків або

імітаційних моделей. Тому, інженер-програміст в своїй діяльності повинен розвивати адекватне розуміння статистичних методів для збору достовірних даних, їх аналізу та узагальнення з ціллю прийняття рішень при створенні програмного продукту.

Методи вимірювання (Measurement) - це сукупність використовуваних способів та принципів вимірювань для створення вимірювальної інформації при дослідженні природи процесів, які описують об’єкти чи явища при розробці програмного проекту. Тобто вимірювання використовуються для ідентифікації сильних і слабких сторін процесу, а також для оцінки програмного забезпечення при реалізації або його еволюції. Процес вимірювання в практиці програмної інженерії починається з осмислення абстрактних понять, що пов’язана з процесом з наступним переходом до визначення методу вимірювання для отримання результату вимірювання. Кожен з етапів необхідний для отримання інформаційних даних про процес при розробці якісного програмного забезпечення. При цьому, окрім класичних вимірювань об’єктів чи явищ, використовуються вимірювання самого програмного забезпечення. Для такого класу вимірювань використовують метрики програмного забезпечення. Метрика програмного забезпечення (англ. software metric) — це міра, що дозволяє отримати числове значення деяких властивостей програмного забезпечення та його специфікацій. Кількісні методи оцінювання добре показали себе в інших сферах науки, а тому багато теоретиків та практиків в галузі інформаційних технологій, спробували перенести цей підхід в розробку програмного забезпечення. В загальному випадку застосування метрик дозволяє визначити складність розробленого проекту або проекту, що перебуває у розробці, оцінити об'єм робіт, стилістику розроблюваного проекту і зусилля, витрачені кожним розробником для реалізації того чи

іншого рішення.

Інженерне проектування (Engineering Design) в програмній інженерії можна визначити, як процес створення прототипу або моделі програмного проекту для знаходження оптимального способу реалізації програмного забезпечення. Останнє обумовлене тим, що вартість розробки програмного забезпечення значною мірою залежить від вибраної моделі конструювання програмного забезпечення. Конструкція програмного забезпечення в свою чергу залежить функціональних та нефункціональних вимог, що висуваються зацікавленими сторонами до проекту.

Моделювання, імітування та прототипування (Modeling, Simulation, and Prototyping) - є процеси, що знайшли широке використання при створенні складних програмних систем.

Моделювання є частиною процесу абстракції, використовуваний для представлення аспектів системи. При моделюванні створюють модель програмного забезпечення для проведення експериментів з цією моделлю для кращого розуміння поведінки програмного забезпечення в реальній роботі, відносини між підсистемами, а також можливості аналізу дизайну діалогового інтерфейсу програмного забезпечення. На відміну від моделювання де за основу береться математична модель процесу, в імітуванні за основу беруться реальні данні, що отримані про процес на основі емпіричних методів дослідження. Методи моделювання та імітування можуть бути використані для побудови теорій або гіпотез про поведінку системи, які використовуються для передбачення конфліктних ситуацій, збоїв або несправностей та їх запобіганню на стадії проектування та розробки.

Прототипування на відміну від процесів моделювання та імітування є процесом, який містить часткове уявлення про продукту або систему. Прототип в переважній більшості

випадків є первинним варіантом програмного забезпечення в якому нереалізовані всі функціональні можливості повноцінної фінальної версії.

Стандарти (Standards) - це документи, що встановлюють вимоги, специфікації, керівні принципи або характеристики, відповідно до яких можуть використовуватися матеріали, продукти, процеси та послуги, які підходять для цих цілей. Стандарти широко використовуються в програмній інженерії. Дотримання норм стандартів потрібно насамперед для забезпечення надійності, безпеки та боротьби з хаосом і плутаниною, що може виникнути в програмному забезпеченні. Тому створення складного програмного продукту неможливо без дотримання стандартів. Державні та галузеві стандарти відстежують тенденції розвитку програмування і дають обов'язкові рекомендації щодо їх дотримання.

Аналіз базових причин (Root Cause Analysis –RCA) – це метод, що використовується при методичній ідентифікації та виправленні корінних першопричин, що призводять до неполадок або збоїв в роботі програмного забезпечення при його розробці або еволюції. RCA зазвичай використовується в якості реактивного методу виявлення проблем та їх вирішення. Аналіз виконується після події. Також, RCA метод є корисним в якості способу прогнозування можливих несправностей на основі математичних або імітаційних моделей. Методологія RCA включає в себе безліч різних інструментів, процесів і філософії, які об’єднані спільними цілями для забезпечення безпечної та безперебійної роботи програмного забезпечення. Основною перевага RCA методу над іншими є пошук та корекція першопричин, що має на меті повністю запобігти повторенню проблеми на відміну від багатьох інших методів завдання яких є маскування проблеми.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть області знань SWEBOK інженерії розробки ПЗ.

2. Приведіть базові поняття SWEBOK.

3. Визначите цілі і завдання області знань "Вимоги до програмного забезпечення"

4. Визначите цілі і завдання області знань "Архітектура програмного забезпечення"

5. Визначите цілі і завдання області знань "Конструювання програмного забезпечення"

6. Визначите цілі і завдання області знань "Тестування програмного забезпечення"

7. Визначите цілі і завдання області знань "Супровід програмного забезпечення"

8. Визначите цілі і завдання області знань "Керування конфігурацією програмного забезпечення"

9. Визначите цілі і завдання області знань "Моделі та методи програмної інженерії"

10. Визначите цілі і завдання області знань "Якість програмного забезпечення"

11. Визначите цілі і завдання області знань "Професійна практика програмної інженерії"

12. Визначите цілі і завдання області знань "Економічні аспекти програмної інженерії"

13. Визначите цілі і завдання області знань "Основи обчислювальних технологій програмної інженерії"

14. З якими стандартами узгоджуються області знань SWEBOK?

15. Які галузі знань найбільш необхідні при розробці програмних систем і чому?.

16. Вкажіть, який зв'язок існує між ядром знань SWEBOK і стандартом ЖЦ.

РОЗДІЛ 3

ЖИТТЄВІ ЦИКЛИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

При вивченні цієї теми важливо пам’ятати, що життєвий цикл програмного забезпечення – це безперервний процес, який починається з моменту прийняття рішення про необхідність створення програмного забезпечення і закінчується в момент його повного вилучення з експлуатації. Існує багато моделей та методологій життєвих циклів, що можуть використовуватися при реалізації програмного забезпечення. Кожна модель і методологія визначає підходи до визначення фаз та робіт життєвого циклу програмного забезпечення. Залежно від обраної моделі життєвого циклу розробки програмного забезпечення, відрізняються підходи до визначення моменту переходу з однієї етапу життєвого циклу до іншого. Але всі життєві цикли містять загальні компоненти, що пов’язані з постановкою завдання, проектування, реалізацією та обслуговування програмного забезпечення.

Зміст розділу

3.1 ЖИТТЄВИЙ ЦИКЛ: ПОНЯТТЯ ТА СТАНДАРТИ

У світі спостерігається постійне зростання складності систем, створюваних людиною. Таке ускладнення призводить до ряду нових проблем, що виникають на всіх стадіях життєвого циклу системи і на різних рівнях її архітектурної деталізації. Джерелами проблем служать різнорідність складових елементів системи (обладнання, люди, ПО), комплексне використання комп'ютерних технологій, недостатня інтеграція застосовуваних дисциплін. Для подолання виникаючих проблем потрібен загальний підхід, що забезпечує ефективну взаємодію осіб, які створюють, використовують і керують сучасними системами. Основними групами задіяних осіб є менеджери, які керують створенням систем, і інженери, що створюють системи. Стандарт ISO/IEC/IEEE 15288:2015 Systems and software engineering – System life cycle processes (Системна і програмна інженерія – Процесси життєвого циклу системи) в якості підходу, що об'єднує ці групи, пропонує загальний набір практик, що охоплює весь життєвий цикл рукотворних систем, і наказує при роботі зі складною програмною системою мати опис її життєвого циклу.

Поняття життєвого циклу програмного забезпечення є одним з базових у програмній інженерії. Сенс життєвого циклу полягає у взаємозв'язку всіх дій, які треба виконати протягом усього «життя» програмного забезпечення. Сам, життєвий цикл програмного забезпечення визначається як послідовність етапів (фаз, стадій), що складаються з технологічних процесів, дій і операцій. Використання життєвого циклу дозволяє розглядати всі етапи у взаємозв'язку, що дозволяє скоротити терміни, вартісті та трудозатрати на програмне забезпечення.

Вперше про необхідність розглядати розробку програмного забезпечення з позицій його життєвого циклу було зазначено в 1968 р на першій конференції НАТО (First NATO Software Engineering Conference in Garmisch Partenkirchen, Germany) присвяченій питанням програмного забезпечення.

Історично основними стандартами на життєвих циклів для програмного забезпечення були:

1985 р. (уточнено в 1988 р) DOD-STD-2167 A – присвячений розробці програмних засобів для систем військового призначення. Перший формалізований і затверджений стандарт життєвого циклу для проектування програмних систем військового призначення на замовлення Міністерства оборони США. Цим документом регламентовано 8 фаз (етапів) створення складних та критичних програмних систем і близько 250 типових обов'язкових вимог до процесів і об'єктів проектування на цих етапах.

1994 р. MIL-STD-498. Присвячений розробці і документуванню програмного забезпечення. Прийнятий Міністерством оборони США як заміна DOD-STD-2167 A і ряду інших стандартів. Основне призначений для застосування всіма організаціями і підприємствами, які отримують замовлення Міністерства оборони США. У 1996 р затверджено дуже докладну версію (407 стор.) керівництва "Застосування і рекомендації до стандарту MIL-STD-498". Основну частину складають 75 підрозділів - рекомендацій щодо забезпечення і реалізації процесів ЖЦ складних та критичних програмних систем високої якості і надійності, що функціонують в реальному часі.

1995 р. IEEE 1074. Присвячений процесам життєвого циклу програмного забезпечення, що використовуються для його розвитку на всіх етапах життєвого циклу. Охоплює повний життєвий цикл програмних систем. Відповідно до

IEEE 1074 розділяють шість великих базових процесів. Ці процеси деталізуються 16 процесами, які деталізуються на 65 процесів.

Зміст кожного процесу починається з опису загальних функцій, завдань та переліку дій, а при подальшій деталізації описом робіт необхідних для успішного виконання процесу. Для кожного процесу в стандарті представлена вхідні і результуюча інформація про його виконання і короткий опис суті процесу. У стандарті увага зосереджена переважно на безпосередньому створенні ПС і на процесах попереднього проектування. В додатку подано чотири варіанти адаптації максимального складу компонентів ЖЦ ПС до конкретних особливостей типових проектів.

1995 р. ISO/IEC 12207:1995 (Information Technology - Software Life Cycle Processes) та ***2008 р. ISO/IEC 12207:2008*** (Systems and software engineering–Software life cycle processes). Ці два стандарти визначають організацію життєвих циклів програмного продукту як сукупність процесів, кожний з яких розбитий на дії, що складаються з окремих завдань, а також встановлює структуру (архітектуру) життєвих циклів програмного продукту у вигляді переліку процесів, дій і завдань.

Стандарт ISO / IEC 12207 розроблявся з урахуванням досвіду використання раніше розроблених стандартів. Основними результатами стандарту ISO 12207 є:

- введення єдиної термінології щодо розробки та застосування програмного забезпечення (призначений не тільки для розробників, але і для замовників, користувачів, всіх зацікавлених осіб);

- поділу понять життєвого циклу програмного забезпечення і моделі життєвого циклу програмного забезпечення. Життєвого циклу програмного забезпечення в стандарті вводиться як повна сукупність всіх процесів і дій по створенню і застосуванню програмного забезпечення, а модель життєвого циклу - конкретний варіант організації

життєвого циклу, обґрунтовано (розумно) обрана для кожного конкретного типу програмного забезпечення;

· опис організації життєвого циклу і його структури (процесів).

· виділення процесу адаптації стандарту для побудови конкретних моделей життєвого циклу.

Стандарт має бути введений в використання з 01.01.2016 як ДСТУ ISO/IEC 12207:2014 Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення (ISO/IEC 12207:2008, IDT)

1998 р. ISO /IEC TR 15504. Визначає множину процесів, названих процесами життєвого циклу, за допомогою яких може бути змодельований життєвий цикл програмної системи. В продовж 2002-2003 р.р. стандарт був імплементований для застосування на території України в якості низки стандартів:

- ДСТУ ISO/IEC TR 15504-1:2002 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 1. Концепції та вступна настанова. (ISO/IEC TR 15504-1:1998, IDT)

- ДСТУ ISO/IEC TR 15504-2:2002 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 2. Еталонна модель процесів та потужності процесу. (ISO/IEC TR 15504-2:1998, IDT)

- ДСТУ ISO/IEC TR 15504-3:2002 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 3. Виконання оцінювання. (ISO/IEC TR 15504-3:1998, IDT)

- ДСТУ ISO/IEC TR 15504-4:2002 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 4. Настановиз виконання оцінювання. (ISO/IEC TR 15504-4:1998, IDT)

- ДСТУ ISO/IEC TR 15504-6:2003 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів.

Частина 6. Настанови з визначення компетентності оцінювачів. (ISO/IEC TR 15504-6:1998, IDT)

- ДСТУ ISO/IEC TR 15504-9:2003 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 9. Словник термінів. (ISO/IEC TR 15504-9:1998, IDT)

ISO/IEC/IEEE 15288:2002 (уточнено в 2015р.) Системна інженерія. Процеси життєвого циклу систем (Preview Systems and software engineering -- System life cycle processes.) Розглядається узагальнення процессной моделі життєвого циклу (ISO / IEC 15288), що охоплює нові сфери життєвого циклу систем, в тому числі і програмного забезпечення. Детально вивчаються принципи побудови моделі стандарту, відмінності від принципів, закладених в основу ICO/IEC 12207. Коротко розглядається методичне керівництво SWEBOK як сукупне джерело інформації про технологічні стандарти в галузі інженерії програмного забезпечення.

Стандарт має бути імплементований в продовж 01.01.2016 – 01.01.2018 як ДСТУ ISO/IEC 15288:2014 Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем (ISO/IEC 15288:2008, IDT).

Розробка стандартів, які регламентують життєві цикли та їх практична імплементація до вирішення практичних задач стикалися з рядом проблем серед яких можливо виділити наступні:

· впровадження стандартів вимагало значні інвестиції коштів, які не завжди окупалися;

· неясність в необхідності процесів;

· різні типи програмного забезпечення (ІС, реального часу, бізнес системи) вимагали різних вимог;

· висока динаміка галузі і старіння стандартів;

· термінологічна неоднозначність різних державних і корпоративних стандартів;

· у багатьох випадках застосування стандартів було

викликано тільки вимогами замовників, хоча на практиці часто гальмувало виконання проектів.

Отже, ***життєвий цикл програмного забезпечення*** – період часу, який починається з моменту прийняття рішення про необхідність створення програмної системи і закінчується в момент її повного вилучення з експлуатації(ліквідації).

3.2 ПРОЦЕСИ ЖИТТЄВОГО ЦИКЛУ (ДСТУ ISO/IEC TR 15504)

Стандарт стандарт ISO / IEC TR 15504: Information Technology - Software Process Assessment (Оцінка процесів розробки ПЗ) визначає множину процесів, названих процесами життєвого циклу, за допомогою яких може бути змодельований життєвий цикл програмної системи. У цьому документі розглядаються питання атестації, визначення зрілості та удосконалення процесів життєвого циклу ПЗ. Один з розділів документа містить нову класифікацію процесів життєвого циклу, що є розвитком стандарту ISO / IEC 12207.

Зв'язок ISO / IEC TR 15504 зі стандартом ISO 12207 полягає в тому, що всі процеси стандарту ISO / IEC 15504 належать до однієї з наступних типів (рис.3.1):

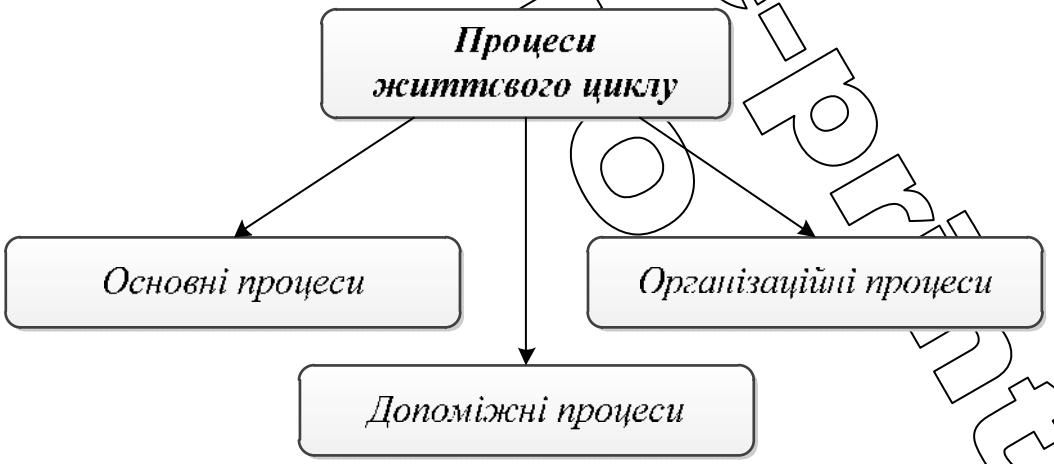


Рис. 3.1 – Процеси життєвого циклу розробки

а) основні процеси (придбання, доставка, розроблення, експлуатація, супровід);

б) організаційні процеси (управління, удосконалення, навчання);

в) допоміжні процеси (документування, забезпечення якості, верифікація, атестація, аудит, загальна оцінка тощо).

Основні процеси включають дві категорії процесів: "Споживач-постачальник" (CUS) і "Інженерна" (ENG). Категорія "Споживач-Постачальник" складається з процесів, безпосередньо впливають на споживача, що підтримують процес розробки програмного засобу і його передачі споживачеві і забезпечують можливість коректного використання програмного засобу або послуги. "Інженерна" категорія процесів складається з процесів, які безпосередньо визначають, реалізують або підтримують програмний продукт, його взаємодія з системою і документацію на нього. У тих випадках, коли система цілком складається з програмних засобів, інженерні процеси мають відношення тільки до створення і підтримання цих програмних засобів. Основні процеси включають наступні процеси(рис.3.2):

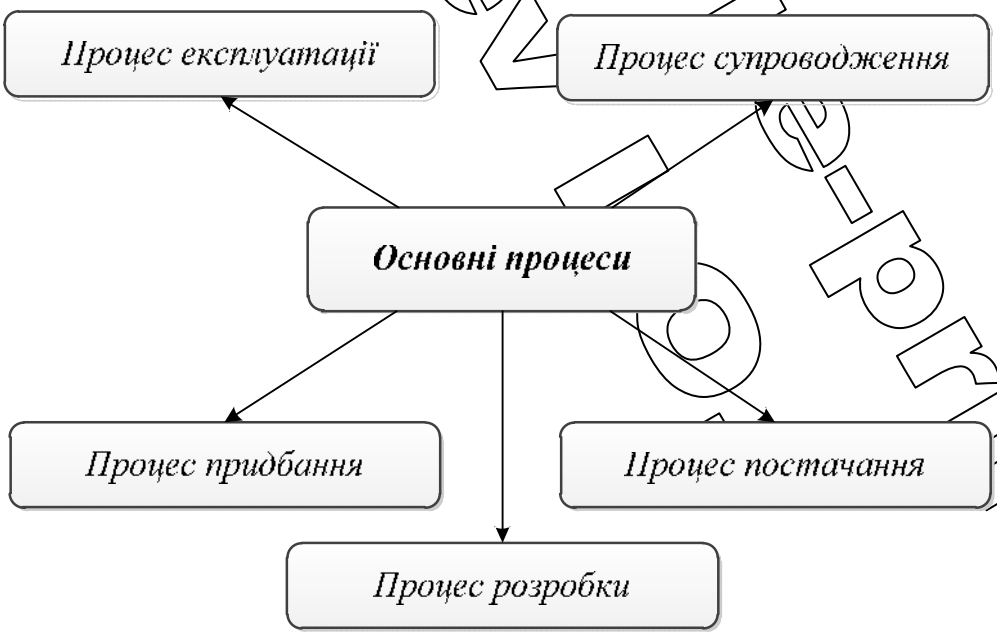


Рис. 3.2 – Основні процеси життєвого циклу розробки

програмного забезпечення

1. Процес придбання, що ініціює життєвий цикл ПЗ та визначає її покупця; передбачають виконання замовлення та постачання продукту замовнику.

2. Процес розроблення, що визначає дії організації – розробника інформаційного продукту; передбачає дії, що виконуються розробником, і охоплює роботи зі створення ПЗ та його компонентів відповідно до вимог, включаючи оформлення проектної й експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і відповідної якості програмних продуктів, матеріалів, потрібних для організації навчання персоналу.

3. Процес постачання, що визначає дії під час передачі розробленого продукту покупцеві;

4. Процес експлуатації, що означає дії з обслуговування системи під час її використання – консультації користувачів, вивчення їхніх побажань тощо;

5. Процес супроводження, що означає дії з керування модифікаціями, підтримки актуального стану та функціональної придатності, інсталяції та вилучення версій систем у користувача.

Процес розроблення ПЗ має забезпечити шлях від усвідомлення потреб замовника до передачі йому готового ПЗ (рис. 3.6) та складається наступних етапів (рис.3.3):

- визначення вимог – збір та аналіз вимог замовника виконавцем та подання їх у нотації, що зрозуміла як замовнику, так і виконавцю;

- проектування – перетворення вимог до розроблення у послідовність проектних рішень щодо способів реалізації вимог: формування загальної архітектури програмної системи та принципів її прив’язки до конкретного середовища

функціонування; визначення детального складу модулів кожної з архітектурних компонент;

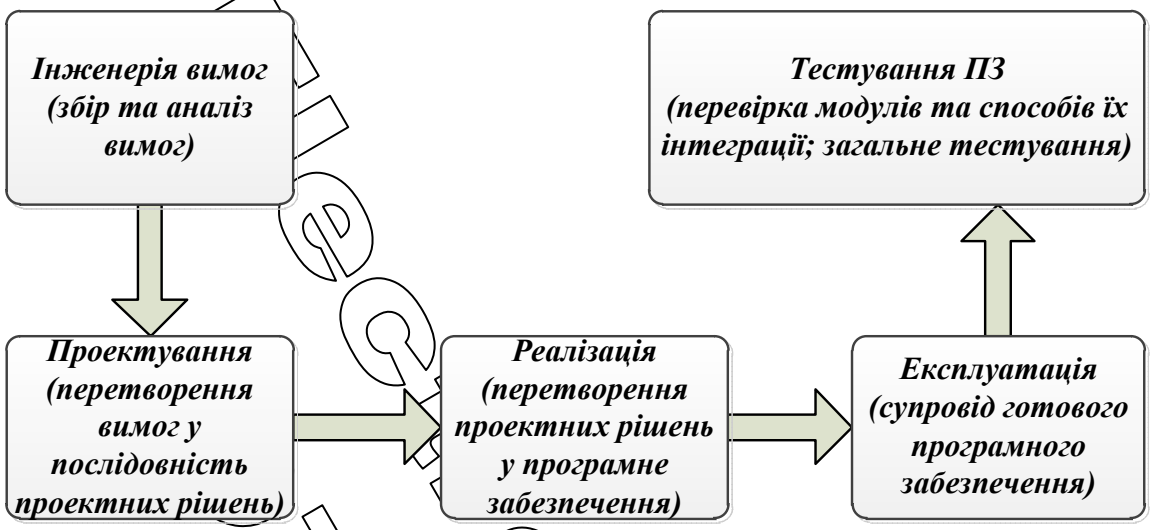


Рис. 3.3 – Основні процеси життєвого циклу розробки програмного забезпечення

- реалізації – перетворення проектних рішень у програмну систему, що реалізує означені рішення;

- тестування – перевірка кожного з модулів та способів їх інтеграції; тестування програмного продукту в цілому (так звана верифікація); тестування відповідності функцій працюючої програмної системи вимогам, що були до неї поставлені замовником (так звана валідація);

- експлуатації та супроводження готової системи.

Допоміжні процеси складаються з процесів, якими можуть користуватися будь-які інші процеси (включаючи інші допоміжні процеси) в різні моменти життєвого циклу програмних засобів. Допоміжні процеси включають (рис.3.4):

1. Процес документування, передбачає формалізований опис інформації, створеної протягом життєвого циклу програмної системи.

2. Процес управління конфігурацією передбачає, застосування адміністративних і технічних процедур протягом життєвого циклу програмної системи для визначення стану компонентів програмної системи, управління її модифікаціями.

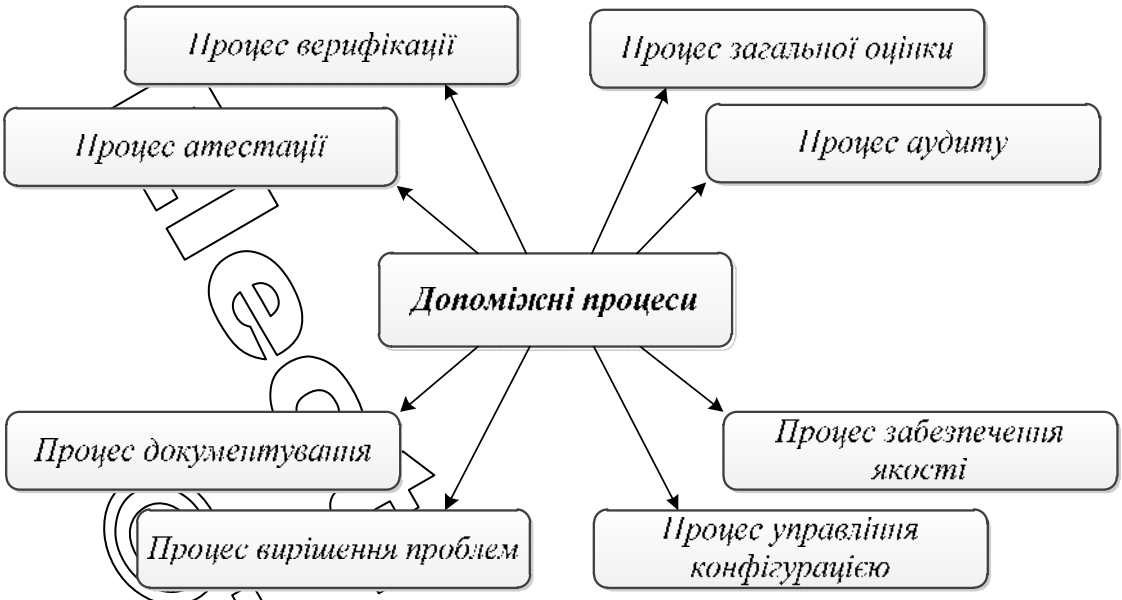


Рис. 3.4 – Допоміжні процеси життєвого циклу розробки програмного забезпечення

3. Процес забезпечення якості, забезпечення гарантій того, що програмна система і процеси її життєвого циклу відповідають заданим вимогам і затвердженим планам.

4. Процес верифікації, передбачає визначення того, що программи, які є результатами певної дії, повністю відповідають вимогам, які зумовлені попередніми діями.

5. Процес атестації, передбачає визначення повноти відповідності заданих вимог і створеної системи їх конкретному функціональному призначенню.

6. Процес загальної оцінки, передбачає оцінку стану робіт за проектом: контроль планування й управління ресурсами, персоналом, апаратурою, інструментальними засобами.

7. Процес аудиту, передбачає визначення відповідності вимогам, планам і умовам договору.

8. Процес вирішення проблем, передбачає аналіз і вирішення проблем, незалежно від їх походження або джерела, які виявлені під час розробки, експлуатації, супроводу або інших процесів.

Організаційні процеси складаються з процесів, що містять практики загального характеру, які можуть бути використані

кожним, хто керує будь-яким проектом чи процесом в ході життєвого циклу програмного продукту. Організаційні процеси включають (рис.3.5):

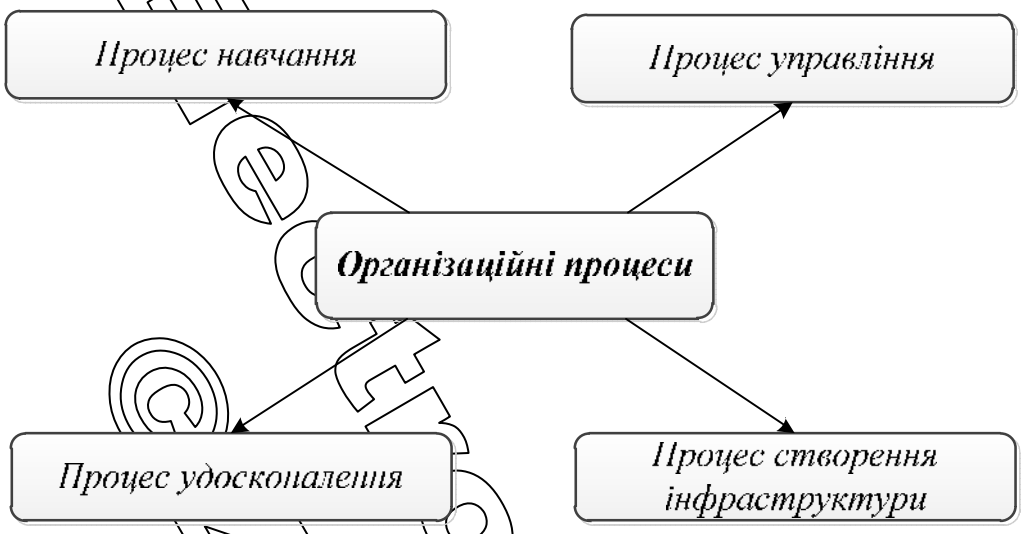


Рис. 3.5 – Організаційні процеси життєвого циклу розробки програмного забезпечення

1. Процес управління, передбачає дії і завдання, які можуть виконуватися будь-якою стороною, що управляє своїми процесами.

2. Процес створення інфраструктури, передбачає вибір і супровід технології, стандартів та інструментальних засобів, вибір та установка апаратних і програмних засобів, що використовуються для розробки, експлуатації або супроводу програмного забезпечення.

3. Процес удосконалення, передбачає оцінка, вимірювання, контроль і удосконалення процесів ЖЦ.

4. Процес навчання, передбачає початкове навчання і подальше постійне підвищення кваліфікації персоналу.

Кожен процес охоплює такі дії:

- ініціація;
- підготовка пропозицій;
- підготовка і коректування вимог;
- нагляд за діяльністю;
- приймання і завершення робіт.

Кожна дія включає низку завдань. Наприклад:

- формування вимог до системи;
- формування списку програмних продуктів;
- встановлення умов і угод;
- опис технічних обмежень;
- стадії життєвого циклу програмного забезпечення, взаємозв'язок між процесами і стадіями.

Процеси життєвого циклу програмних продуктів мають свої особливості та відмінності від процесів створення інших видів продукції, а саме:

– програмне забезпечення є не фізичним, а інтелектуальним продуктом людської діяльності. Тому при його створенні діють людські і логічні обмеження, а не фізичні закономірності, як при виробництві товарів;

– технічні вимоги до програмного забезпечення є стабільними, але при розробці програмне забезпечення найбільш характерні вимоги можуть видозмінюватися, втрачати актуальність і не реалізовуватися в програмне забезпечення, а також можуть з'являтися нові вимоги породженні іншими вимогами;

– продуктивність праці при створенні програмного забезпечення може варіюватися в широких межах, причому такі коливання більш характерні для окремих виконавців, ніж для творчих колективів;

– дефекти програмного забезпечення насамперед є наслідком людських помилок і низької якості або взагалі відсутності документації до створеного програмного забезпечення, а не низької якості матеріалів, які використовуються при створенні програмне забезпечення;

– цінність фізичних продуктів визначається їх технічними характеристиками, а програмне забезпечення оцінюють ще й з урахуванням їх інтерактивної функціональності;

– економічні аспекти якості програмного забезпечення

визначається переважно процесами узгодження вимог до програмного забезпечення та компонентів апаратної частини системи;

– ціна програмне забезпечення більшою мірою визначається процесами, що супроводжують його створення та підтримку, ніж їх відповідністю початковим вимогам;

– вартість безпосереднього виготовлення програмне забезпечення становить незначну частину повної вартості створення та підтримки програмне забезпечення на всіх життєвих стадіях й визначається в основному витратами на розробку, впровадження та випробування програмне забезпечення;

– статистичні методи оцінювання не можуть застосовуватися до процесу тиражування програмного забезпечення, оскільки всі копії, як правило, є однорідними за якістю;

– вартість підтримки програмного забезпечення, як складового елементу інформаційних систем підприємств, визначається методами відмінними від методів, що застосовуються для товарів та послуг, так як програмне забезпечення – це активи підприємств, що не піддаються капіталізації і амортизації.

Отже, *процеси життєвого циклу програмного забезпечення* – певна послідовність взаємопов'язаних дій, з моменту прийняття рішення про необхідність створення програмного забезпечення до повного вилучення програмного забезпечення з експлуатації. При цьому *процес* – сукупність взаємозв'язаних і взаємодіючих видів діяльності, які перетворюють входи у виходи. Процеси і дії життєвого циклу відбираються, відповідним чином налаштовуються і використовуються протягом стадії життєвого циклу для повного задоволення цілей і результатів на цій стадії.

3.3 ЕТАПИ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Етапи (Phase/Stage) життєвого циклу програмного забезпечення – це найбільші періоди життєвого циклу, асоційовані з програмною системою, і співвідносяться з станами опису процесів життєвого циклу програмної системи як набору достатніх продуктів або послуг для успішної реалізації даного етапу життєвого циклу програмної системи. Етапи описують основні контрольні точки просування програмної системи в її життєвому циклі. Такі сегменти дають можливість упорядкованого простеження і аналізу просування програмної системи через встановлені точки з можливістю переглядів виділених ресурсів на даному етапі з ціллю забезпечення зниження ризиків і забезпечення задовільного просування ПЗ по всьому життєвому циклу.

Етап життєвого циклу відносяться до періоду значного просунення системи і досягнення запланованих термінів протягом життєвого циклу. Вони можуть перекривати одна одну, можуть застосовуватися для побудови структур, за допомогою яких процеси життєвого циклу використовуються для моделювання безпосередньо життєвого циклу. Не зважаючи на різницю в життєвих циклах різних систем, існує базовий набір стадій життєвого циклу, які складають повний життєвий цикл будь-якої системи. Кожна стадія має певну ціль і вносить свій вклад в повний життєвий цикл і розглядається при плануванні і виконанні життєвого циклу програмної системи. Стадії життєвого циклу створюють структуру робіт для деталізованого моделювання життєвих циклів системи при використанні процесів життєвого циклу системи.

У таблиці 3.1 представлені приклади етапів життєвого циклу, які найбільш часто зустрічаються. У таблиці

відображені принципові цілі кожного з етапів і можливі варіанти рішень, що використовуються для управління досягненнями і ризиками, пов'язаними з розвитком програмної системи протягом життєвого циклу. Паралельне проходження етапів або їх проходження в різному порядку може призвести до форм життєвого циклу з абсолютно різними характеристиками. Часто в якості альтернативних варіантів використовуються послідовна, інкрементна або еволюційна форми життєвого циклу; в окремих випадках можуть бути розроблені комбінації цих форм життєвих циклів. Дії щодо розроблення ПЗ можуть включати: аналізування, розроблення проекту, кодування, тестування, а також інші дії та задачі, які виконують протягом різних фаз життєвого циклу ПЗ. Дії щодо керування можуть включати: планування проекту, відстеження, контролювання, аналізування ризиків тощо. Дії з підтримки можуть включати документацію з продукту, що постачаються, таку як посібник користувача, операційні довідники, мережні комунікації тощо.

Перед кожним керівником програмного проекту виникає задача з установлення відповідності між діями (таблиця 3.2) та моделями життєвого циклу ПЗ з наступним їхнім описом на достатньому рівні деталізації. Кожна з наведених у табл. 3.2 дій може бути деталізована з метою реалізації окремих деталей різними розробниками програмного проекту.

Вибір менеджером програмного проекту конкретної форми життєвого циклу залежить від ряду факторів, включаючи бізнес-контекст, природу і складність програмної системи, стабільність вимог, технологічні можливості, потреба в реалізації і підтримці різних системних можливостей та наявність бюджетних коштів і ресурсів для реалізації та підтримки програмної системи.

Таблиця 3.1 – Приклади етапів та їх цілей з схемами рішень

Етап життєвого циклу	Ціль	Схема рішень
Ідея створення	Визначити потреби правовласників Дослідити задуми Запропонувати життєздатні рішення	Варіанти рішення: - виконати наступну стадію; - продовжити дану стадію; - повернутись до попередньої стадії; - призупинити проект; - завершити роботи над проектом;
Розробка	Уточнити вимоги до системи Створити опис рішень Створити систему Провести верифікацію і валідацію системи	
Створення	Створити систему Проконтролювати і перевірити	
Впровадження	Забезпечити впровадження системи для задоволення потреб користувачів	
Підтримка	Забезпечити стійку реалізацію функціональних можливостей програмної системи	
Переведення в категорію непридатних для застосування	Зберігання, архівування, списання або повна ліквідація програмної системи	

Аналогічно тому, як всі системні елементи здійснюють внесок у систему як у єдине ціле, так і кожний етап життєвого циклу повинен бути врахований в інших стадіях життєвого циклу.

Програмне забезпечення в сучасній інформаційній системі підприємства використовується впродовж всього часу існування підприємства, незалежно від рівня автоматизації процесів на підприємстві.

Таблиця 3.2 – Перелік процесів та дій, які виконують на етапах життєвого циклу розробки ПЗ у відповідності з СОУ-Н ДКА 0061:2012

Етапи (фази) ЖЦ ПЗ	Процеси ЖЦ ПЗ	Дії процесів ЖЦ ПЗ
Планування моделі життєвого циклу розробки програмного забезпечення Керування проектом	1. Встановлення відповідності між типом моделі ЖЦ ПЗ та потребами проекту 2. Початок виконання програмного проекту	1. Ідентифікація типів моделей ЖЦ ПЗ 2. Вибір моделі проекту 3. Зіставлення дій та моделі ЖЦ ПЗ 4. Розподіл ресурсів проекту 5. Установлення середовища проекту 6. Керування планом проекту
Дії, що передують розробленню проекту	3. Відстеження та контроль проекту 4. Керування якістю ПЗ 5. Дослідження концепції 6. Системний розподіл	7. Аналізу ризиків 8. Планування непередбачених ситуацій 9. Керування проектом 10. Зберігання записів 11 Реалізація методу звітів з проблем 12. Планування керуванням якістю ПЗ 13. Визначення метричних показників 14. Керування якістю ПЗ 15. Ідентифікація потреби щодо поліпшування якості 16. Ідентифікація ідей або потреб 17. Формулювання потенційних підходів 18. Проведення досліджень щодо здійснення проекту 19. Планування системних переходів (за необхідності)

Етапи (фази) ЖЦ ПЗ	Процеси ЖЦ ПЗ	Дії процесів ЖЦ ПЗ
		20. Уточнення та завершення ідеї або потреби 21. Аналізування функцій 22. Розроблення системної архітектури 23. Декомпозиція системних вимог
Розробка ПЗ	7. Вимоги	24. Визначення та розробка вимог до ПЗ 25. Визначення вимог до інтерфейсу 26. Призначення пріоритетів та інтеграція вимог до ПЗ
	8. Розробка проекту	27. Розробка проекту архітектури 28. Проектування бази даних (за необхідності) 29. Проектування інтерфейсів
	9. Впровадження	30. Вибір або розробки алгоритмів (за необхідності) 31. Виконання деталізованої розробки проекту 32. Створення тестових даних 33. Розробка вихідного коду
Дії, які виконують після розробки ПЗ		34. Генерування об'єктного коду 35. Створення документації 36. План інтеграції
	10. Встановлення	37. Виконання інтеграції 38. План

Етапи (фази) ЖЦ ПЗ	Процеси ЖЦ ПЗ	Дії процесів ЖЦ ПЗ
		встановлення ПЗ 39. План інтеграції 40. Встановлення ПЗ
	11. Експлуатація та підтримка	41. Приймання ПЗ в операційному середовищі 42. Системні операції
	12. Супровід	43. Забезпечення технічної підтримки та консультацій
	13. Висновок з експлуатації	44. Журнал запитів про підтримку 45. Повторне використання ЖЦ при розробці ПЗ 46. Повідомлення користувачів 47. Здійснення одночасних операцій (за необхідності) 48. Висновок системи з експлуатації
Допоміжні процеси ЖЦ ПЗ	14. Атестація та верифікація	49. План атестації та верифікації 50. Виконання задач з атестації та верифікації 51. Збір і аналізування метричних даних 52. План проведення тестування 53. Розробка вимог до тестування
	15. Менеджмент конфігурації ПЗ	54. Виконання тестування 55. Планування менеджменту конфігурації 56. Ідентифікація конфігурації 57. Контроль

Етапи (фази) ЖЦ ПЗ	Процеси ЖЦ ПЗ	Дії процесів ЖЦ ПЗ
	16. Розробка документації	конфігурації 58. Облік стану 59. Планування документації 60. Застосування документації
	17. Навчання	61. Реалізація та структурування документації 62. Планування навчальної програми 63. Розроблення навчальних матеріалів 64. Перевірка навчальної програми 65. Реалізація навчальної програми

В залежності цілей застосування програмного забезпечення існує декілька підходів до визначення етапів життєвого циклу для програмного забезпечення інформаційної системи підприємства:

– вивчення особливостей господарської діяльності підприємства;

– проектування ПЗ інформаційної системи і узгодження проекту технічного завдання на виготовлення ПЗ з замовником;

– впровадження, супровід та удосконалення ПЗ в інформаційній системі підприємства.

При цьому відповідно до таблиць 3.1 та 3.2 можна виділити такі основні етапи життєвого циклу ПЗ інформаційної системи підприємства (рис. 3.6):

– початковий етап – вивчення особливостей господарської діяльності підприємства й аналіз вимог до ПЗ інформаційної системи;

– етап проектування ПЗ;

– етап створення прототипу і тестування прототипу ПЗ;

– етап експлуатації – впровадження, супровід та розвиток (еволюція) ПЗ;

– заключний етап – ліквідація або відмова від ПЗ інформаційної системи на підприємстві.

Існування певного програмного забезпечення в інформаційній системі підприємства обумовлене насамперед проведенням певного виду діяльності підприємством (1), що відразу створює (2) та впроваджує певні, не завжди досконалі вимоги до ПЗ інформаційну систему (2), що при наявності зацікавлених осіб веде до створення конкретного ПЗ (3) та його впровадження в інформаційну систему підприємства (4). При використанні ПЗ (5) в багатьох випадках виникає необхідність внесення змін в ПЗ інформаційної системи (6) чи її удосконалення при використанні на підприємстві, що обумовлено насамперед змінами у професійній діяльності підприємства. Для внесення змін в ПЗ, тобто його розвитку (еволюції) в деяких випадках проводяться додаткові дослідження діяльності підприємства (7), внесення змін в існуючий проект ПЗ (2), її реалізація (3) та впровадження (4), що породжує новий етап експлуатація ПЗ інформаційної системи підприємства. В процесі експлуатації ПЗ інформаційної системи в типових випадках обслуговується як працівниками підприємства, так і працівниками сторонніх організацій для забезпечення коректної та безперебійної роботи підприємства (5). Коли обслуговування інформаційної системи стає занадто складним або неекономічним, тоді виникає питання про її удосконалення (6) й розпочинається етап розробки та впровадження змін до інформаційної системи (7, 2-4). Значні витрати на обслуговування (5) чи удосконалення (6) ПЗ інформаційної системи можуть призвести до відмови від її використання (8, 9) та обумовити необхідність розробки нової

версії ПЗ інформаційної системи (10, 2-4) або розробки нового ПЗ (11, 2-4) для підприємства, оскільки без відповідного ПЗ інформаційна система підприємства не може існувати. Безумовно, що припинення існування самого підприємства припиняє існування і його ПЗ інформаційної системи.

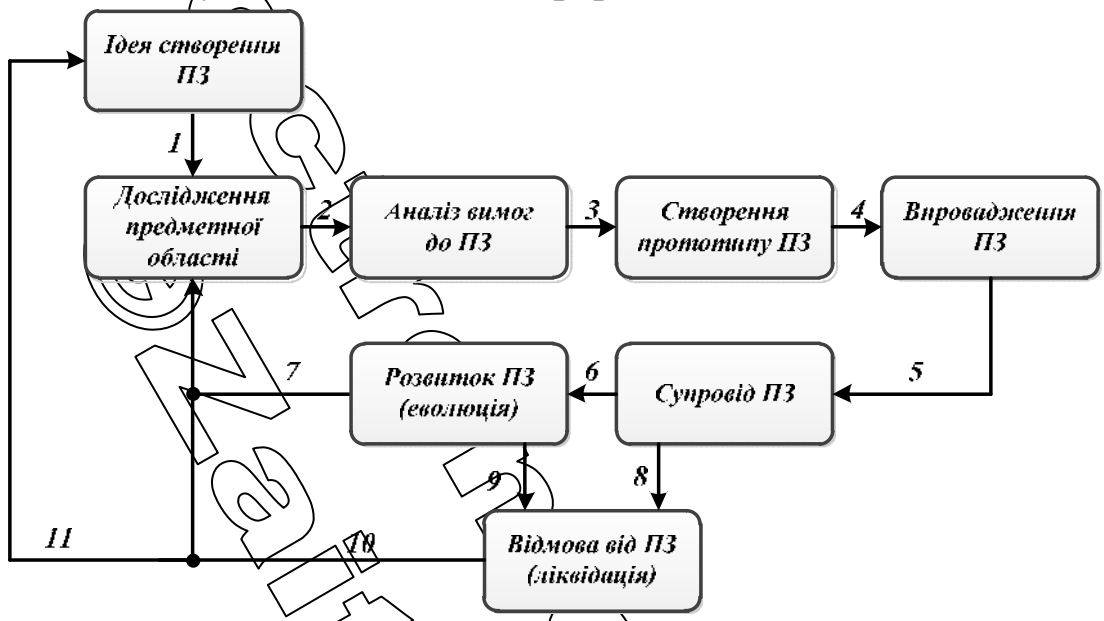


Рис. 3.6 – Основні етапи життєвого циклу програмного забезпечення інформаційної системи підприємства

Розглянемо найбільш типові етапи, які характерні для життєвого циклу будь-якого ПЗ інформаційної системи підприємства.

1. Початковий етап створення ПЗ інформаційної системи характеризується виконанням значного обсягу роботи по обґрунтуванню, розробці та узгодженню технічного завдання на ПЗ, що створюється між замовником (відомство, фірма, організація) і виконавцем - розробником і виробником ПЗ інформаційної системи або всієї системи (фірма, конструкторське бюро, підприємство-виробник). За основу на даному етапі береться технічне завдання погоджене з замовником виходячи в функціональних можливостей аналогічного за призначенням ПЗ інформаційних систем, які пройшли етап випробувань та успішної експлуатації. В

технічному завданні на ПЗ відображені всі етапи життєвого циклу, основні характеристики та параметри ПЗ необхідні для виконанні заданих функцій, часові інтервали проектування, випробувань, експлуатації та інші характеристики, включаючи характеристики надійності і інші. На початковому етапі визначається, по суті, часовий графік створення ПЗ з залученням числа розробників, рівня їх кваліфікації, обсягу спеціальних засобів і устаткування, виділення фінансових коштів і іншого.

З використанням методів теорії систем визначаються потенціальні досяжні характеристики створюваного ПЗ з урахуванням характерних особливостей його експлуатації при виконанні заданих функцій. Підготовча робота починається з вибору моделі життєвого циклу програмного забезпечення, що відповідає масштабові, значимості і складності проекту. Процес розроблення має відповідати обраній моделі. Розробник повинен вибрати, адаптувати до умов проекту і використовувати погоджені із замовником стандарти, методи й засоби розроблення, а також скласти план виконання робіт.

2. Етап проектування характеризується виконанням значного обсягом теоретичних, проектувальних і тестових досліджень для обґрунтування структури ПЗ інформаційної системи і розробки конструкторської, технологічної та технічної документації для реалізації ПЗ інформаційної системи в сучасну апаратно-програмну інформаційну систему підприємства.

На цьому етапі математичний апарат на підставі якого створюється програмного забезпечення інформаційної системи, використовується в повному обсязі. Так розробляються математичні та проектні моделі створюваного ПЗ, вирішуються завдання функціонування модулів ПЗ та ПЗ в складі системи для виконання заданих функцій. На основі результатів проектного моделювання проводиться

обґрунтування в деякому сенсі оптимального варіанта структури ПЗ інформаційної системи.

На даному етапі використовується в основному апріорні знання про майбутнє ПЗ, визначаються розрахункові характеристики, надійність ПЗ. Слід зазначити, що рівень інформаційного забезпечення проектування ПЗ забезпечує:

– підвищення якості розробки технічної та конструкторської документації за рахунок істотного зменшення помилок при проектуванні ПЗ;

– скорочення термінів розробки документації на базі використання сучасних автоматизованих інформаційних технологій проектування;

– розробку документації для створення модулів ПЗ інформаційної системи на сучасному.

Також на даному етапі проводиться розробка тестів для проведення тестування модулів, підсистем перед підготовкою до створення прототипу ПЗ.

3. Етап створення прототипу ПЗ і його тестування характеризується створенням прототипів ПЗ його модулів та підсистем.

При наявності документації на ПЗ інформаційної системи:

– підготовка до створення прототипів ПЗ його модулів та підсистем;

– при прототипів ПЗ виконуються операції контролю на сумісність та тестування окремих модулів та підсистем прототипу ПЗ;

– реалізується розроблена на етапі проектування методика і відповідні плани проведення тестувань для отримання характеристик і параметрів ПЗ інформаційної системи її модулів та підсистем;

– реалізується розроблена на етапі проектування методика статистичної обробки даних тестувань прототипів ПЗ його модулів та підсистем;

– за результатами порівняльного аналізу вхідних вимог до ПЗ відповідно до технічного завдання на розробку ПЗ і отриманих характеристик ПЗ при виконанні проектних та тесту вальних робіт вносяться необхідні корективи в проектну документацію до модулів та підсистем прототипу ПЗ.

На основі результатів виконаних робіт проводиться коригування ПЗ, в першу чергу проектної та конструкторської документації на розробку створення ПЗ інформаційної системи.

4. Етап виготовлення зразка ПЗ його сертифікація і передача в експлуатацію. На цьому етапі життєвого циклу ПЗ інформаційної системи виконуються наступні роботи:

– підготовка до створення фінальної версії ПЗ інформаційної системи з заданими характеристиками відповідно до коретованих вимог на етапах проектування, виготовлення та тестування;

– при виготовленні комерційних зразків ПЗ інформаційної системи виконується методика, яка розроблена на етапі проектування, налагодження різних модулів підсистем, перевірка їх функціонування для реалізації виконання заданих функцій;

– випробування виготовленого комерційного зразка ПЗ для діючої інформаційної системи щодо виконання заданих функцій відповідно до вимог (штатний режим функціонування).

5. Етап експлуатації. Даний етап життєвого циклу ПЗ інформаційної системи є найбільш тривалим за часом і відповідно використовується при виконанні наступних операцій:

– транспортування, налагодження та запуск в експлуатацію ПЗ на діючій інформаційній системі;

– поточний моніторинг функціонування ПЗ в діючій інформаційній системі відповідно до вимог;

– реалізації методик, які була розроблені на етапі проектування, побудови довгострокових підсистем моніторингу та визначення прогнозованих показників надійності функціонування ПЗ інформаційної системи;

– реалізація методики супроводу та розвитку системи;

– коригування математичних підсистем та окремих модулів ПЗ на діючій інформаційній системі, формування рекомендацій еволюції ПЗ інформаційної системи та самої системи з метою підвищення ефективності її використання;

– реалізація рекомендацій по розвитку функціональних можливостей ПЗ інформаційної системи та системи в цілому.

6. Заключний етап життєвого циклу системи. Даний етап є найбільш наукомісткий в життєвому циклі ПЗ інформаційної системи та складається з формування даних про можливості з глибинної модернізації ПЗ інформаційної системи, використання окремих підсистем ПЗ для створення нового ПЗ для діючої інформаційної системи або в випадку припинення існування самого підприємства припинення існування ПЗ інформаційної системи, як інтелектуального продукту. На даному етапі виконується також роботи по утилізації ПЗ інформаційної системи, яка вичерпала свій технічний ресурс або морально застаріла у зв'язку зі змінами у внутрішньому або зовнішньому середовищі підприємства.

В процесі створення та розвитку ПЗ інформаційної системи можливе використання різноманітних підходів, які базуються на врахуванні:

– змін, здійснених в організаційній структурі підприємства (виділяють підходи, що зберігають існуючу організаційну структуру підприємства, та підходи, що змінюють існуючу організаційну структуру, удосконалюючи її);

– характеру змін, внесених до ПЗ інформаційної системи підприємства (на базі галузевого удосконалення обліку на підприємстві, за яких удосконалюються окремі процеси

виконання облікових операцій та комплексні підходи, які удосконалюють інформаційну систему підприємства в цілому; на базі комплексного удосконалення ПЗ інформаційної системи підприємства при цьому передбачається послідовний розподіл системи, призначеної для розв'язання визначеного кола питань на окремі модулі, кожний з яких матиме певну функцію з наступним поєднанням отриманих модулів в єдине ціле, що і буде являти розвиток ПЗ інформаційної системи);

– способу реалізації удосконалення ПЗ (на базі підходів по удосконаленню ПЗ: удосконалення власними силами, удосконалення з використанням стандартизованих рішень та на базі замовлення індивідуального ПЗ).

При проектуванні ПЗ інформаційної системи слід враховувати певні принципи, дотримання яких дозволить значно підвищити ефективність роботи ПЗ інформаційної системи. Слід виділити такі принципи проектування ПЗ інформаційних систем, які поліпшують якість створення та розвитку ПЗ:

– дотримання принципів системності передбачає проведення аналізу предметної області функціонування ПЗ в цілому й діючої інформаційної системи зокрема, а також визначення загальних цілей і критеріїв потрібних для функціонування ПЗ;

– економічна доцільність передбачає приведення доказової бази з перевагами, що очікуються від використання ПЗ в інформаційній системи, які повинні перевищувати витрати на проектування, розробку впровадження та супровід ПЗ в діючій інформаційній системі;

– дотримання принципів гнучкості ПЗ інформаційної системи забезпечують достатній запас гнучкості, щоб ПЗ мало можливість реагування на зміни зовнішніх факторів;

– проведення контролю через тестування ПЗ, його окремих модулів та підпрограм;

– забезпечення системи захисту і безпеки даних дозволяє створювати та підтримувати розвиток ПЗ діючої інформаційної системи при захисті активів підприємства від нераціонального їх використання і забезпечувати надійність та безпеку інформації в інформаційній системі (якісне ПЗ інформаційної системи повинна виконувати такі функції щодо безпеки даних:

а) поділ доступу до функцій і даних в системі шляхом авторизації користувачів за паролем;

б) шифрування даних;

в) наявність контролю за входом до системи і ведення журналу робочого часу;

г) контроль за періодичністю створення резервних (архівних) копій інформації);

– забезпечення сумісності передбачає, що система повинна проектуватися з урахуванням людського фактора, організаційних особливостей підприємства та вже наявного апаратно-програмного забезпечення інформаційної системи підприємства;

– забезпечення універсальності дозволяє створювати ПЗ для вирішення не окремих задач, а ПЗ здатне виконувати стандартні процедури і обробляти конкретну задачу, як окремий випадок більш загальної;

– забезпечення безперервності розвитку ПЗ (еволюції ПЗ) передбачає постійне вдосконалення всіх видів функцій ПЗ (технічного, програмного, інформаційного та ін.) при розвитку підприємства або змін його виду діяльності виникають нові завдання управління, удосконалюються та змінюються вже існуючі задачі.

Отже, ***етап життєвого циклу*** програмного забезпечення - це відстань від однієї події в життєвому циклі програмного забезпечення до іншої.

3.4 МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Стандарти життєвого циклу встановлюють склад і організацію процесів, які можуть виконуються на різних етапах життєвого циклу ПЗ. У багатьох випадках ці процеси виявляються надлишковими і наведені стандарти (Додаток А) адаптуються до умов виконання конкретних проектів відповідно до вибираними моделями життєвого циклу, що враховують специфіку цих проектів.

Модель життєвого циклу розробки програмного забезпечення є єдиним видом процесу, в якому представлений порядок його здійснення. Відповідно до стандарту ISO / IEC 12207 модель життєвого циклу (life cycle model) визначається як структура, що складається з процесів, робіт і завдань, що включають в себе розробку, експлуатацію та супровід програмного продукту, що охоплює життя системи від встановлення вимог до неї до припинення її використання.

Модель життєвого циклу розробки програмного забезпечення (Software Life Cycle Model, SLCM) схематично пояснює, яким чином будуть виконуватися дії по розробці програмного продукту, за допомогою опису «послідовності» цих дій. Така послідовність може бути чи не бути лінійної, оскільки фази можуть слідувати одна за одною, повторюватися або реалізовуватися одночасно. На рис. 3.7 представлена проста узагальнена схема процесу розробки програмного забезпечення без врахування можливих ітерацій стадій на будь-якому етапі проектування.

При цьому, конкретні моделі визначаються особливістю завдань, обмеженнями на ресурси, досвідом розробників і т.д. Тобто вибір типу моделі життєвого циклу залежить від специфіки, масштабу і складності проекту і специфіки умов, в яких система створюється і функціонує (див. розділ 3.5).

Тобто вибір і адаптація моделі життєвого циклу розробки проекту впливає на методики розробки продукту, необхідні навички менеджеру проекту та задіяного персоналу. При виборі моделі життєвого циклу розробки програмного продукту менеджер проекту повинен насамперед мати уявлення про стандарти процесу, вміти оцінити їх застосовність по відношенню до даного проекту, оцінити альтернативні процеси і при необхідності адаптувати модель і процеси життєвого циклу до поточних потреб життєвого циклу програмного забезпечення.

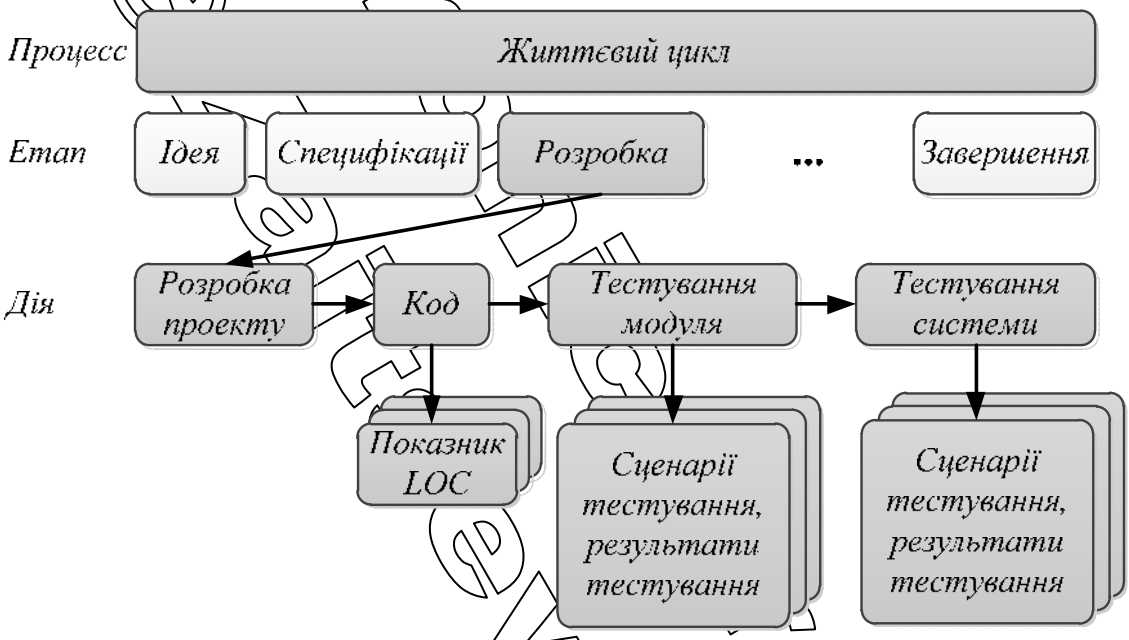


Рис. 3.7 – Узагальнена схема процесу розробки програмного забезпечення

Найбільш розповсюдженими моделями життєвого циклу програмного забезпечення, які знайшли застосування на практиці в певних умовах мають свої певні переваги та недоліки. Ці типові моделі встановлюють деякі принципи організації моделі життєвого циклу програмного забезпечення. До числа основних моделей та життєвого циклу програмного забезпечення та методологій, що їх використовують для реалізації життєвого циклу програмного забезпечення слід віднести: каскадна модель;V-подібна

модель;спіральна модель; інкрементна та ітеративні моделі; методологія швидкого еволюційного прототипування; методологія раціонального уніфікованого процесу (RUP); методологія швидкого розроблення та інші.

Кожна з наведених вище моделей життєвого циклу програмного забезпечення містить різний набір дій та задач, які реалізуються під час розробки основних компонентів програмного забезпечення. Дії підтримуючих процесів (менеджмент конфігурації, верифікація, розроблення документації) є ідентичними для всіх моделей. У таблиці 3.3 наведено список інтегральних процесів і відповідні їм задачі та дії, які є загальними для всіх базових моделей життєвих циклів.

Таблиця 3.3 – Інтегральні дії та задачі підтримуючих процесів ЖЦ ПЗ

Дії ЖЦ ПЗ	Задачі життєвого циклу програмного забезпечення
Відстеження та контролювання проекту	– аналізування ризиків – планування випадкових подій – керування проектом – зберігання записів – впровадження методу звіту з проблем
Керування якістю ПЗ	– планування керування якістю ПЗ – визначення метричних показників – планування якості ПЗ – ідентифікація потреб щодо поліпшування якості
Атестація та верифікація	– план атестації та верифікації – виконання задач атестації та верифікації – збір та аналізування метричних даних
Менеджмент конфігурації ПЗ	– план менеджменту конфігурації – ідентифікація конфігурації – контролювання конфігурації – облік стану
Розроблення документації	– документування плану – впровадження документації – створення та розподіл документації
Навчання	– планування навчальної програми – розроблення навчальних матеріалів – атестація навчальної програми – впровадження навчальної програми

Отже, ***модель життєвого циклу ПЗ*** – структура, що визначає послідовність виконання і взаємозв'язку процесів, дій і завдань на протягом життєвого циклу.

3.4.1 КАСКАДНА МОДЕЛЬ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Опис каскадної моделі (Waterfall model) вперше повно було викладено в 1970 році В.В. Ройсом (W.W. Royce) в його роботі "Managing the Development of Large Software Systems". На початковому етапі становлення програмної інженерії відігравала провідну роль як метод розробки складного програмного забезпечення. В продовж 70-80 р.р. XX століття модель була прийнята як стандарт міністерства оборони США.

Основною характеристикою каскадної моделі життєвого циклу програмного забезпечення є розбиття всієї розробки на етапи, причому перехід з одного етапу на наступний відбувається тільки після того, як буде повністю завершена робота на поточному етапі. Кожен етап завершується випуском повного комплекту документації, достатньої для того, щоб розробка могла бути продовжена командою фахівців на наступному етапі.

Каскадну модель наведено на рис. 3.8. ЖЦ ПЗ в каскадній моделі реалізують за допомогою впорядкованої послідовності кроків. У моделі передбачено, що кожна наступна фаза починається лише тоді, коли повністю завершено виконання попередньої фази. Кожна фаза має певні вхідні та вихідні дані. У результаті виконання всіх етапів ЖЦ генеруються внутрішні або зовнішні дані проекту, включаючи документацію та ПЗ.

Перехід від однієї фази до іншої здійснюється за допомогою підтримуючого огляду. Таким чином, замовник одержує загальне уявлення про процес розроблення, крім того,

завдяки огляду перевіряють якість програмного продукту. Як правило, етап огляду вказує на домовленість між командою розробників і замовником про те, що поточну фазу завершено і можна переходити до виконання наступної фази.

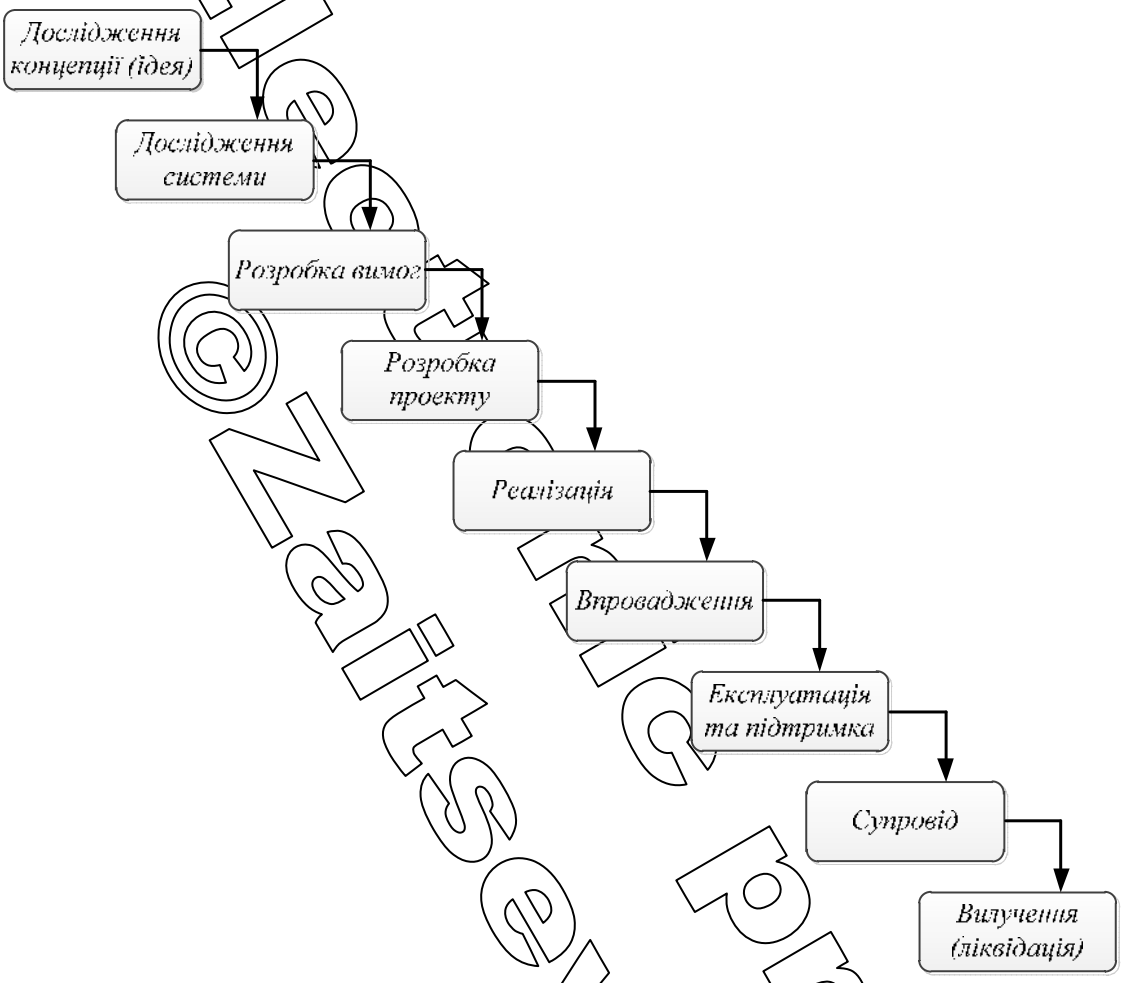


Рис. 3.8 – Структура каскадної моделі життєвого циклу програмного забезпечення

Характерні властивості каскадної моделі: модель являє собою формальний метод, різновид розроблення «зверху вниз», вона складається з незалежних фаз, які виконують послідовно, і підлягають частому огляду.

Наведено короткий опис кожної фази каскадної моделі:

– дослідження концепції – відбувається дослідження вимог на системному рівні з метою визначення можливості реалізації концепції;

– процес системного розподілу – може бути пропущений, якщо розробляють тільки ПЗ. У випадку необхідності розробки як апаратного, так і програмного забезпечення, необхідні функції застосовують до ПЗ та апаратного забезпечення відповідно до загальної архітектури системи;

– визначення вимог – визначають програмні вимоги, продуктивність і інтерфейси. За потреби до процесу також включають функціональний розподіл системних вимог до апаратного та програмного забезпечення;

– розроблення проекту – розробляють та формулюють логічно послідовну технічну характеристику програмної системи, включаючи структури даних, архітектуру ПЗ, інтерфейсні подання та алгоритмічну деталізацію;

– реалізація – в результаті виконання алгоритмічний опис ПЗ перетворюється у програмний продукт, при цьому створюється вихідний код, база даних та документація, які лежать в основі фізичного перетворення проекту. Якщо ПЗ є пакетом прикладних програм, основними діями щодо його реалізації повинні бути встановлення і тестування програм. Якщо ПЗ розробляють на замовлення, основними діями є програмування та тестування;

– впровадження – включає встановлення ПЗ, його перевіряння та офіційне приймання замовником для операційного середовища;

– експлуатація та підтримка – запуск користувачем системи і поточне забезпечення, зокрема надання технічної допомоги, обговорення виниклих питань із користувачем, реєстрування запитів користувача на модернізацію та внесення змін, а також корегування або усунення помилок;

– процес супроводу – аналізують та приймають рішення, пов'язані з усуненням програмних дефектів і аномалій, модернізацією та внесенням змін, що генеруються процесом

підтримки. Цей процес складається з ітерацій розроблення та допускає зворотний зв'язок з надання інформації про аномалії;

– процес вилучення з експлуатації – здійснюють вилучення існуючої системи з активного використовування або припиненням її роботи, або заміною новою системою чи модернізованою версією існуючої системи;

– інтегральні завдання – включають початок роботи над проектом, моніторинг проекту і його керування, керування якістю, верифікацію та атестацію, менеджмент конфігурації, розроблення документації і професійну підготовку протягом усього життєвого циклу.

Каскадна модель (рис. 3.8) забезпечує лінійний розподіл дій, наведених у табл. 3.4.

Таблиця 3. 4 – Перелік процесів, дій і задач, характерних для каскадної моделі життєвого циклу програмного забезпечення

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Дослідження концепції перевіряння вимог на системному рівні з метою визначення їх здійснення	– ідентифікація ідей або потреб – формулювання потенційних підходів – виконання досліджень здійсненності – планування системного переходу (за потреби) – уточнення та завершення ідеї або потреби
Процес системного розподілу — установлення відповідності між функціями та програмним або апаратним забезпеченням на основі загальної системної архітектури	– аналізування функцій – розроблення системної архітектури – декомпозиція системних вимог
Процес визначення вимог — визначення вимог до ПЗ для	– визначення та розроблення вимог до ПЗ

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
одержання інформації щодо області дії та функцій системи, поводження, продуктивності інтерфейсів, що застосовуються	– визначення вимог до розроблення інтерфейсу – розміщення пріоритетів та інтеграція вимог до ПЗ
Процес розроблення проекту — розроблення та подання логічно зв'язаної технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі	– розроблення проекту архітектури – проектування бази даних (за потреби) – проектування інтерфейсів – вибір або розроблення алгоритмів (за потреби) – виконання деталізованого розроблення проекту
Процес впровадження — втілення опису розроблення програмного проекту у програмний продукт, а також створення вихідного коду, баз даних і документації незалежно від того, чи були ці програмні продукти розроблені, придбані або мають змішане походження	– створення тестових даних – створення вихідного коду\ – генерування об'єктного коду – створення оперативної документації – план інтеграції
Процес інтеграції — встановлення та перевіряння ПЗ в операційному середовищі, а також одержання формального схвалення ПЗ замовниками	– план інтеграції – розподіл ПЗ – інтеграція ПЗ – приймання ПЗ в операційному середовищі
Процес експлуатації та підтримки — включає користувальницькі операції у системі та діючу підтримку, у тому числі технічну допомогу, надання консультацій користувачам, фіксування їхніх запитів з метою виконання розширень і змін, а також оброблення виправлень	– виконання операцій у системі – забезпечення технічної підтримки та проведення консультацій – підтримка журналу реєстрації запитів користувачів

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
або помилок	
Процес супроводження — аналізування запитів з метою виявлення програмних збоїв, помилок, розширень, а також змін, внесених під час здійснення процесу підтримки	– повторне застосування циклу розроблення ПЗ (початок процесу розроблення)
Процес виведення з експлуатації — припинення активного застосування системи, шляхом завершення операцій, що не використовуються, заміна новою системою або відновлення із застосуванням нової версії існуючої системи	– повідомлення користувачів – виконання паралельних операцій (за потреби) – виведення системи з експлуатації
Інтегральні дії — див. табл. 3.2	

Переваги каскадної моделі ЖЦ ПЗ:

– модель добре відома споживачам, які не мають відношення до розроблення та експлуатації програм, та кінцевим користувачам (її часто використовують інші підприємства для відстеження проектів, не пов'язаних з розробленням ПЗ);

– у рамках моделі упорядковано справляються зі складностями, що добре спрацьовує для тих проектів, які досить зрозумілі, але складні у виконанні;

– модель достатньо зрозуміла, тому що в ній чітко розподілено необхідні дії;

– модель є простою та зручною в застосуванні, тому що процес розроблення виконують поетапно;

– модель відрізняється стабільністю вимог, і нею може керуватися навіть не зовсім досвідчений персонал;

– модель є шаблоном, у який можна поміщати методи для аналізування, проектування, кодування, тестування та забезпечення;

– модель застосовна тоді, коли вимоги до якості домінують над вимогами до витрат і графіка виконання проекту;

– модель сприяє здійсненню чіткого контролювання та менеджменту проекту;

– за умови правильного використовування моделі дефекти можна виявити на ранніх етапах, коли їхнє усунення ще не вимагає великих витрат;

– модель полегшує роботу менеджерові проекту з розроблення плану та комплектування команди розробників;

– модель дозволяє учасникам проекту, що завершили дії на фазі, яку вони виконували, взяти участь у реалізації інших проектів;

– модель визначає процедури контролювання якості, кожні отримані дані підлягають огляду. Таку процедуру використовує команда розробників для визначення якості системи;

– етапи моделі добре визначені та зрозумілі;

– хід виконання проекту легко простежити за допомогою використовування часової шкали (або діаграми Ганта), оскільки момент завершення кожної фази використовується як точка звіту за результатами огляду.

Недоліки каскадної моделі ЖЦ ПЗ:

– в основі моделі лежить послідовна лінійна структура, в результаті чого кожна спроба повернутися на одну чи дві фази назад, щоб виправити яку-небудь проблему або недолік, призведе до значного збільшення витрат і збою у графіку виконання робіт;

– модель не може запобігати виникненню ітерацій між фазами, які часто зустрічаються під час розроблення ПЗ, оскільки сама модель створюється відповідно до стандартного циклу апаратного інжинірингу;

– модель може створити помилкове враження про роботу над проектом; вираз типу «50 % виконано» не несе ніякого змісту та не є показником для менеджера проекту;

– інтеграція всіх отриманих результатів відбувається у завершальній стадії роботи моделі; в результаті такого одиничного проходу через весь процес пов'язані з інтегруванням проблеми, як правило, дають про себе знати занадто пізно;

– замовник не має можливості ознайомитися із програмним забезпеченням заздалегідь, це відбувається лише в кінці життєвого циклу; замовник не має можливості скористатися доступними проміжними результатами і відкликати користувачів, а також не може повернути програмне забезпечення розробникам;

– оскільки готовий продукт є недоступним до закінчення процесу, користувач бере участь у процесі розроблення тільки з самого початку (під час формування вимог) і наприкінці (під час приймальних випробувань);

– користувачі не можуть бути впевненими в якості розробленого продукту до закінчення всього процесу розроблення; вони не мають можливості оцінити якість доти, доки не можна побачити готовий продукт розроблення;

– у користувача немає можливості поступово звикнути до програмного забезпечення;

– процес навчання відбувається наприкінці життєвого циклу, коли програмне забезпечення вже необхідно вводити в експлуатацію;

– проект можна виконати, застосовуючи впорядковану каскадну модель, і привести його у відповідність із заданими вимогами, але це не гарантує його введення в експлуатацію;

– кожна фаза є передумовою для виконання наступних дій, що перетворює такий метод у ризикований вибір для систем, які не мають аналогів, тому що він не підлягає гнучкому моделюванню;

– для кожної фази створюються результативні дані, які по завершенню вважають замороженими – це означає, що вони не повинні змінюватися на наступних етапах життєвого циклу ПЗ, якщо елемент даних якого-небудь етапу змінюється (що зустрічається досить часто), то на проект вплине зміна графіка, оскільки ні модель, ні план не були розраховані на дозвіл внесення зміни на пізніших етапах життєвого циклу програмного забезпечення;

– всі вимоги повинні бути відомі на початку життєвого циклу, але клієнти рідко можуть чітко сформулювати всі задані вимоги на момент розроблення, тобто модель не розрахована на динамічні зміни у вимогах протягом усього життєвого циклу, тому що одержані дані «заморожуються»;

– використовування моделі може викликати значні витрати, якщо вимоги в недостатній мірі відомі або підлягають динамічним змінам під час виконання життєвого циклу;

– виникає необхідність у твердому керуванні та контролюванні, оскільки в моделі не передбачено можливості модифікації вимог;

– модель засновано на документації, тому кількість документів може бути надлишковою;

– весь програмний продукт розробляють за один раз і виключається можливість розбити систему на частини, в результаті чого (через узяте розробниками зобов'язання – розробити систему в цілому за один раз) можуть виникнути

проблеми з фінансуванням проекту; відбувається перерозподіл більших коштів, а сама модель не дозволяє повторно розподілити кошти, не зруйнувавши при цьому проект у процесі його виконання.

Через недоліки каскадної моделі її застосовування необхідно обмежити ситуаціями, у яких вимоги та їхня реалізація максимально чітко визначені і зрозумілі.

Область застосування Каскадної моделі

Каскадна модель добре функціонує під час її застосовування у циклах розроблення програмного продукту, в яких використовується незмінне визначення продукту та цілком зрозумілі технічні методики.

Якщо є досвід побудови системи, в проекті, орієнтованому на побудову ще одного продукту такого ж типу, можливо, навіть заснованого на існуючих розробках, можна ефективно використати каскадну модель. Іншим прикладом застосування моделі може слугувати створення та випуск нової версії вже існуючого продукту, якщо внесені зміни цілком визначені і керовані. Перенесення уже існуючого продукту на нову платформу часто наводять як ідеальний приклад використання каскадної моделі у проекті.

До таких систем можна віднести складні розрахункові програмні системи з чітким алгоритмом функціонування, систем реального часу і інші подібні програмні системи. Однак, в процесі використання каскадної моделі для реалізації ПЗ ніколи повністю не вкладався в жорстку схему моделі, що перш за все викликано недоліками моделі. Тобто, у процесі створення ПЗ постійно виникає потреба в поверненні до попередніх етапів і уточнення або перегляд раніше прийнятих рішень або затверджених вимог. В результаті реальний процес створення ПЗ при використанні ітераційної моделі має вигляд, представлений на рис. 3.8.

Модифікована версія каскадної моделі є в значній мірі менш жорсткою, ніж класична модель (рис3.8). При цьому включаються ітерації між фазами, паралельні фази та менеджмент змін. Зворотні стрілки допускають можливість існування ітерацій між процесами. Незважаючи на те, що модифікована каскадна модель є значно гнучкішою, ніж класична модель, вона не є найкращим вибором для виконання проектів за умови прискореного розроблення.

Каскадні моделі протягом всього часу існування використовують під час виконання великих проектів, у яких задіяні кілька команд розробників.

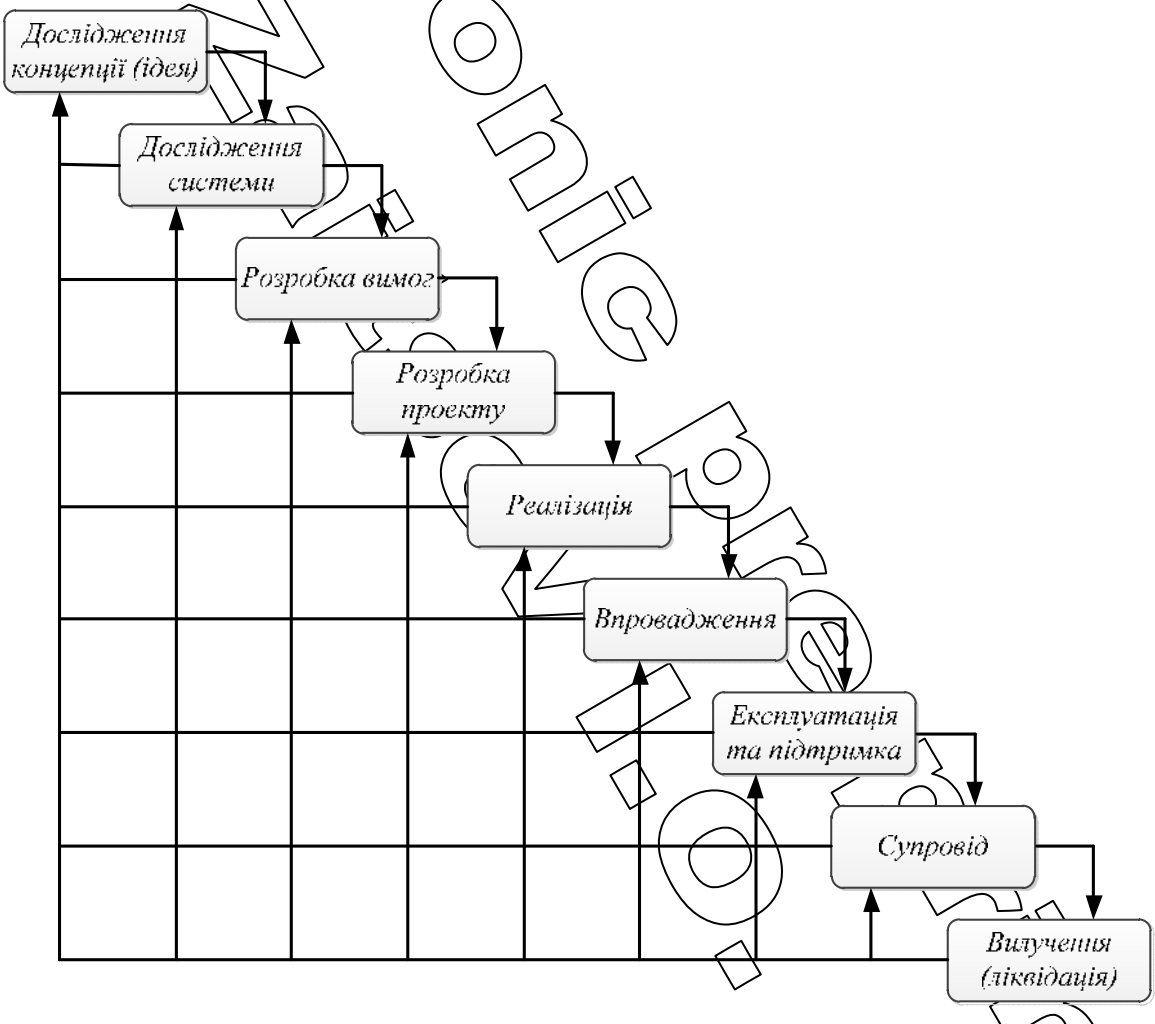


Рис. 3.9 – Структура модифікованої каскадної моделі життєвого циклу програмного забезпечення

Отже, ***каскадна модель*** – модель процесу розробки програмного забезпечення, життєвий цикл якої виглядає як потік, що послідовно проходить фази аналізу вимог, проектування. реалізації, тестування, інтеграції і підтримки. Процес розробки реалізується за допомогою впорядкованої послідовності незалежних кроків. Модель передбачає, що кожен наступний крок починається після повного завершення виконання попереднього кроку. На всіх етапах моделі виконуються допоміжні та організаційні процеси і роботи, включаючи управління проектом, оцінку і управління якістю, верифікацію і атестацію, управління конфігурацією, розробку документації. В результаті завершення кроків формуються проміжні продукти, які не можуть змінюватися на наступних кроках.

3.4.2 V-ПОДІБНА МОДЕЛЬ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Концепція V-подібної моделі була розроблена Німеччиною і США в кінці 1980-х років, щоб усунути недоліки каскадної моделі. Структуру типової V-подібної моделі наведено на рис. 3.10.

Основний принцип V-подібної моделі полягає в тому, що деталізація проекту зростає при русі зліва направо, одночасно з плином часу, і ні те, ні інше не може повернутися назад. Ітерації в проекті відбуваються по горизонталі, між лівою і правою сторонами літери. Модель базується на тому, що приймально-здавальні випробування грунтуються, перш за все, на вимогах, системне тестування – на вимогах та архітектурі, комплексне тестування – на вимогах, архітектурі та інтерфейсі, а компонентне тестування– на вимогах, архітектурі, інтерфейсі і алгоритмах. V-подібна модель надає

можливість значно підвищити якість ПЗ за рахунок своєї орієнтації на тестування, а також багато в чому вирішила проблему відповідності створеного продукту висунутим вимогам завдяки процедурам верифікації та атестації на ранніх стадіях розробки.

У цій моделі особливе значення надають діям, спрямованим на верифікацію та валідацію продукту. Вона демонструє, що тестування продукту обговорюють, проектують та планують на ранніх етапах життєвого циклу ПЗ. План випробування і приймання замовником розробляють на етапі планування, а план комплексного випробування системи – на етапах аналізування, розроблення проекту. Цей процес розроблення планів випробувань позначено на рис. 3.10 пунктирною лінією між прямокутниками V-подібної моделі.

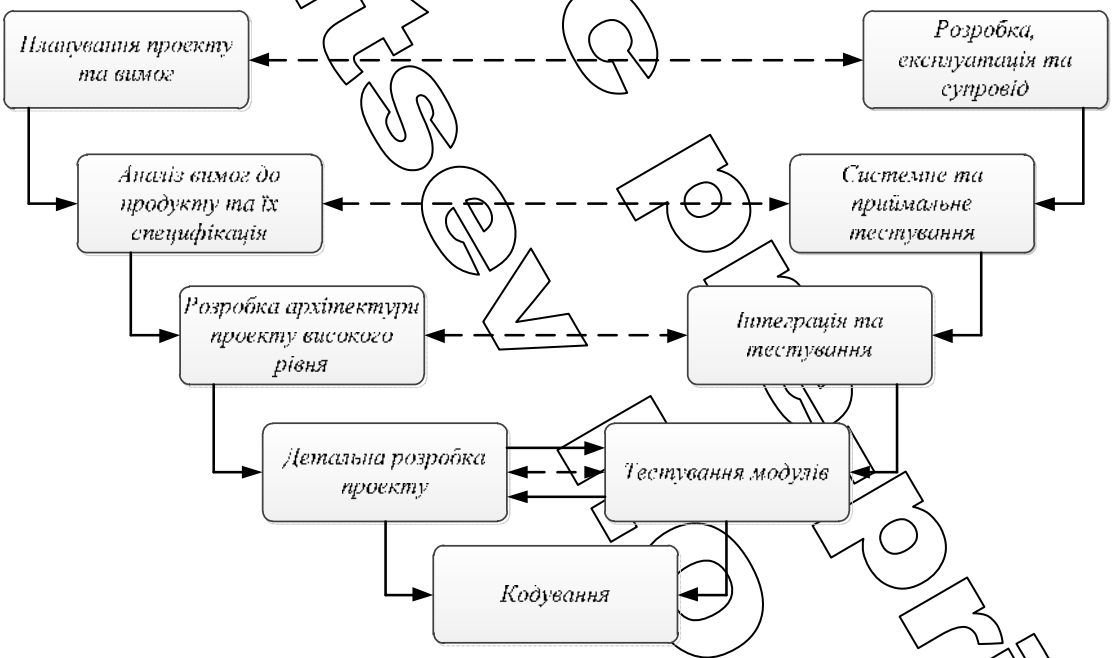


Рис. 3.10 – Структура V-подібної моделі життєвого циклу програмного забезпечення

Подібно каскадній моделі, V-подібна модель найкраще застосовна тоді, коли вся інформація про вимоги доступна заздалегідь. Модифікація V- подібної моделі, спрямована на

подолання її недоліків, містить у собі внесення ітераційних циклів для дозволу зміни у вимогах за рамками фази аналізування.

Нижче наведено короткий опис кожної фази V-подібної моделі:

– планування проекту та вимог – визначаються системні вимоги, а також розподіл ресурсів підприємства з метою їхньої відповідності поставленим вимогам; за потреби, на цій фазі виконують визначення функцій для апаратного та програмного забезпечення;

– аналізування вимог до продукту та специфікація – аналізують поточні на даний момент проблеми з ПЗ і завершують розроблення повної специфікації ПЗ, що створюється;

– розроблення архітектури проекту високого рівня – визначають, яким чином функції ПЗ повинні застосовуватися під час реалізації проекту;

– детальне розроблення проекту – визначають та документально обґрунтовують алгоритми для кожного компонента, який було визначено на фазі побудови архітектури; ці алгоритми згодом буде перетворено у код;

– кодування – виконують перетворення алгоритмів, визначених на етапі деталізованого проектування, у програмний код;

– тестування модулів – виконують перевіряння кожного закодованого модуля на наявність помилок;

– інтеграція та тестування – установлюють взаємозв'язок між групами раніше поелементно випробуваних модулів з метою підтвердження того, що ці групи працюють задовільно, як і модулі, випробувані незалежно один від одного на етапі елементного тестування;

– системне та приймальне тестування – виконують перевіряння функціонування програмної системи в цілому

(повністю інтегрована система), після введення в її апаратне середовище, відповідно до специфікації ПЗ;

– виробництво, експлуатація й супровід – ПЗ та систему запускають у виробництво. На цій фазі передбачено також модернізацію і внесення виправлень;

– приймальні випробування (не наведено на рис. А.2):

1) тестують функціональні можливості системи та ПЗ на відповідність вихідним вимогам;

2) після остаточного тестування ПЗ і його навколишнє апаратне забезпечення стають робочими.

V-подібна модель (рис. 3.10) визначає лінійний розподіл більшості дій, що наведені у табл. 3.5.

Таблиця 3.5 – Перелік процесів, дій і задач, характерних для V-подібна моделі життєвого циклу програмного забезпечення

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Планування проекту та вимог — визначення системних вимог і методів розподілу ресурсів організації з метою задоволення вимог	
Початок виконання проекту	– установлення відповідності між діями та планом ЖЦ ПЗ – розподіл ресурсів проекту – настроювання середовища проекту – планування керування проектом – дослідження концепції – ідентифікація ідей або потреб – формулювання потенційних підходів – дослідження здійсненності – планування системних переходів (за необхідності) – уточнення та завершення ідей або потреб – керування якістю ПЗ – планування керування якістю ПЗ

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
	– визначення метричних показників
Вимоги до продукту та аналізування специфікацій- аналізування і специфікація очікуваного зовнішнього поводження розроблення програмної системи	
Аналізування системного розподілу	– аналізування функцій – розроблення системної архітектури – декомпозиція системних вимог – ідентифікація вимог до ПЗ – визначення та розроблення вимог до ПЗ – визначення вимог до інтерфейсу – встановлення пріоритетів та інтеграція вимог до ПЗ
Архітектура та розроблення проекту на високому рівні — визначення порядку застосування програмних функцій для розроблення проекту	
Деталізоване розроблення проекту — визначення та документування алгоритмів для кожного компонента, який було визначено на фазі розроблення архітектури проекту	– вибір та розроблення алгоритмів (за необхідності) – виконання деталізованого розроблення проекту
Кодування – трансформування алгоритмів, визначених на фазі деталізованого розроблення проекту, в готове ПЗ	– створення вихідного коду – генерування об'єктного коду – створення оперативної документації
Тестування модуля — перевірення кожного створеного модуля на наявність помилок	– план тестування – розроблення тестових вимог – створення тестових даних – виконання тестів
Інтеграція та тестування —	– план інтеграції

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
перевіряння попередньо протестованих модулів, у результаті якого потрібно переконатися, що вони будуть поводитися так, як і у випадку незалежного тестування	– виконання інтеграції – план тестування розроблення тестових вимог – створення тестових даних – виконання тестів
Системне тестування — перевіряння програмної системи в цілому (повністю інтегрованої), яка вбудована в існуюче апаратне середовище, на відповідність специфікації програмних вимог	– план тестування – розроблення тестових вимог – створення тестових даних – виконання тестів
Тестування прийнятності — забезпечення можливості тестування функціональних властивостей системи користувачами відповідно до вихідних вимог. Після завершення тестування ПЗ та відповідне йому апаратне забезпечення стають працездатними. Наступним є етап супроводу системи	– план установлення ПЗ – розподіл ПЗ – установлення ПЗ – план тестування – розроблення тестових вимог – створення тестових даних – виконання тестів – приймання ПЗ в операційному середовищі
Виробництво, **експлуатація та супровід** — початок виробництва ПЗ, а також підтримка можливості виправлення та корегування	– виконання операцій у системі – технічна допомога та проведення консультацій – підтримка журналу запитів про технічну допомогу
Інтегральні дії — див. табл. 3.2	

Переваги V-подібної моделі ЖЦ ПЗ:

– у моделі особливе значення приділено плануванню, спрямованому на верифікацію та валідацію продукту на ранніх стадіях його розроблення:

1) фаза тестування модулів підтверджує правильність деталізованого проектування;

2) фази інтеграції та тестування реалізують архітектурне проектування або проектування високого рівня;

3) фаза тестування системи підтверджує правильність виконання етапу вимог до продукту і його специфікації;

– у моделі передбачено верифікацію та валідацію всіх зовнішніх і внутрішніх отриманих даних, а не тільки самого програмного забезпечення;

– у *V*-подібній моделі визначення вимог виконують перед розробленням проекту системи, а проектування ПЗ – перед розробленням компонентів;

– модель визначає продукти, які повинні бути отримані в результаті процесу розроблення, причому кожні отримані дані підлягають тестуванню;

– завдяки моделі менеджер проекту може відслідковувати хід процесу розроблення, тому що у цьому випадку цілком можливо скористатися часовою шкалою, а завершення кожної фази є контрольною точкою;

– модель є простою у використовуванні (щодо проекту, для якого вона є прийнятною).

Недоліки V-подібної моделі ЖЦ ПЗ:

– за допомогою моделі складно управляти паралельними подіями;

– у моделі не враховано ітерації між фазами;

– у моделі не передбачено внесення вимог динамічних змін на різних етапах життєвого циклу;

– тестування вимог у життєвому циклі відбувається занадто пізно, внаслідок чого неможливо внести зміни, не вплинувши при цьому на графік виконання проекту;

– у модель не входять дії, спрямовані на аналізування ризиків.

Область застосування Каскадної моделі

- в проектах, в яких існують часові та фінансові обмеження;

- для завдань, які передбачають більш широке, в порівнянні з каскадної моделлю, тестове покриття.

З метою подолання недоліків V-подібної моделі її можна модифікувати, включивши до неї ітераційні цикли, призначені для дозволу змін у вимогах за рамками фази аналізування. Структура модифікованої V-подібної моделі приведено на рис. 3.11.

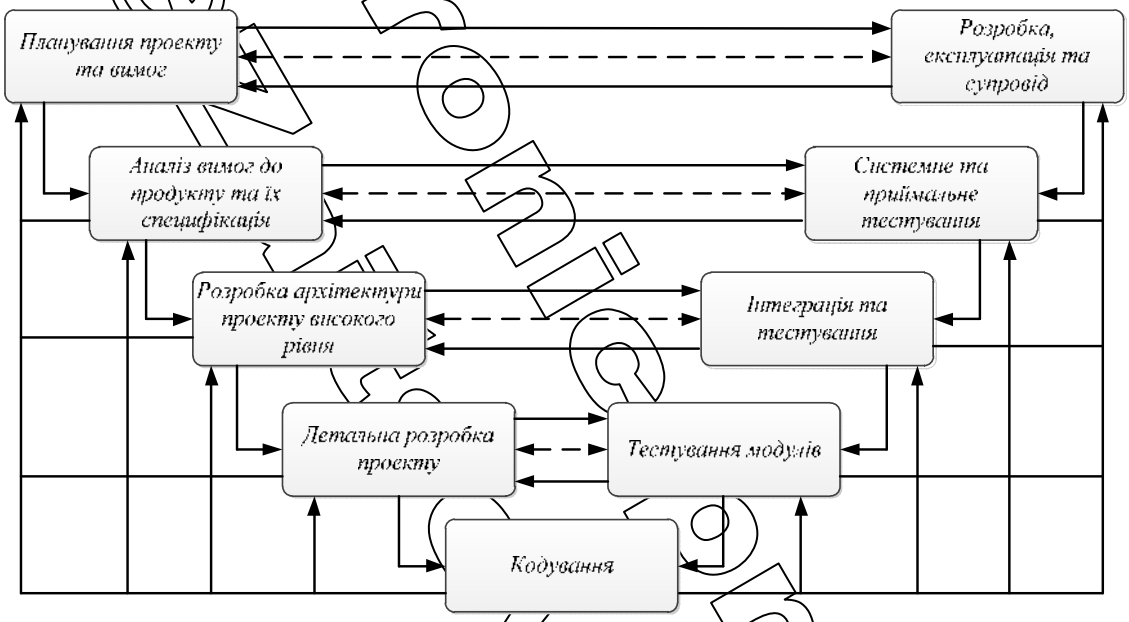


Рис. 3.11 – Структура модифікованої V-подібної моделі життєвого циклу програмного забезпечення

Отже, ***V-подібна модель*** є всього лише модифікацією каскадної моделі процесу розробки програмного забезпечення. Використання моделі ефективне у випадку, коли доступними є інформація про методи реалізації вимог та технології з реалізації, а персонал володіє необхідними знаннями і досвідом у роботі з цими технологіями.

3.4.3 СПІРАЛЬНА МОДЕЛЬ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Спіральна модель (англ. spiral model) була розроблена в середині 1980-х років Баррі Боем. Вона заснована на класичному циклі Демінга PDCA (plan-do-check-act — планування-дія-перевірка-коректування). При використанні цієї моделі ПЗ створюється в кілька ітерацій (витків спіралі) методом прототипування.

Спіральна модель, запропонована Баррі Боемом в 1986 р, стала істотним проривом у розумінні природи розробки ПЗ. На практиці, при вирішенні досить великої кількості завдань, розробка ПЗ має циклічний характер, коли після виконання деяких стадій доводиться повертатися на попередні. Можна вказати дві основні причини таких повернень:

– помилки розробників, допущені на ранніх стадіях і виявлення на пізніх стадіях - помилки аналізу, проектування, кодування, які виявляються, як правило, на стадії тестування.

– зміни вимог в процесі розробки («помилки» замовників). це або неготовність замовників сформулювати вимоги («Сказати, що повинна робити програма я зможу тільки після того, як побачу як вона працює »), або зміни вимог, викликані змінами ситуації в процесі розробки (зміни ринку, нові технології, тощо).

Спіральна модель Боема сфокусована на проектуванні. Власне розробка ПЗ відбувається лише на останньому витку спіралі за звичайною каскадною моделлю, однак цьому передує кілька ітерацій проектування на основі створення прототипів – при цьому кожна ітерація включає стадію виявлення і аналізу ризиків і найбільш складних завдань. Оскільки спіральна модель в основному охоплює саме проектування, то в первісному вигляді вона не отримала

широкого поширення в якості методу управління всім життєвим циклом створення ПЗ. Однак головна її ідея, яка полягає в тому, що процес роботи над проектом може складатися з циклів, які проходять одні й ті ж етапи, послужила вихідним пунктом для подальших досліджень і стала основою більшості сучасних моделей процесу розробки ПЗ.

У спіральній моделі враховані недоліки, викликані використовуванням каскадної моделі. Спіральна модель втілює в собі переваги каскадної моделі. При цьому, до неї також включено: аналізування ризиків, керування ними, а також процеси підтримки та менеджменту. В ній також передбачено розроблення програмного продукту під час використовування методу прототипування або швидкого розроблення додатків за допомогою застосування мов програмування.

На рис. 3.12 зображено спіральну модель життєвого циклу ПЗ. У кожний квадрант моделі входять основні та допоміжні дії, як наведено нижче:

– визначення проекту, альтернативних варіантів та обмежень:

1) визначають мету, робочі характеристики, функції, що виконуються, можливість внесення змін, які вирішують чинники досягнення успіху та апаратного/програмного інтерфейсу;

2) визначають альтернативні способи реалізації цієї частини продукту (конструювання, повторне використовування, закупівля, субдоговір тощо);

3) визначають обмеження застосування альтернативних варіантів (витрати, графік виконання, інтерфейс, обмеження, пред'явлені середовища тощо);

4) розробляють документацію, яка підтверджує ризики, пов'язані з недостатнім досвідом у даній сфері, застосуванням нової технології, твердими графіками, погано організованими процесами тощо;

– оцінювання альтернативних варіантів, ідентифікація та дозвіл ризиків:

1) оцінюють альтернативні варіанти, відносно цілей та обмежень;

2) виконують визначення і дозвіл ризиків, менеджмент ризиків, методика економічно вигідного вибору джерел дозволу, оцінювання інших пов'язаних з ризиком ситуацій, коли кошти може бути втрачено через продовження розроблення системи (рішення про припинення/продовження робіт над проектом тощо);

– розроблення продукту наступного рівня:

1) створення та критичне аналізування проекту,

2) розробка коду,

3) перевірка коду;

4) тестування і компонування продукту. Перша створена версія продукту грунтується на ознайомленні з нею замовника; потім починається фаза планування. Програму повертають у вихідний стан з метою врахування реакції клієнта. Кожна наступна версія більш точно втілює вимоги замовника, при цьому ступінь внесених змін від однієї версії програми до наступної зменшується з кожною новою версією, що призводить до одержання функціональної системи;

– планування наступної фази шляхом розроблення:

1) плану проекту (визначення проекту);

2) плану менеджменту конфігурацією;

3) плану тестування;

4) плану встановлення програмного продукту.

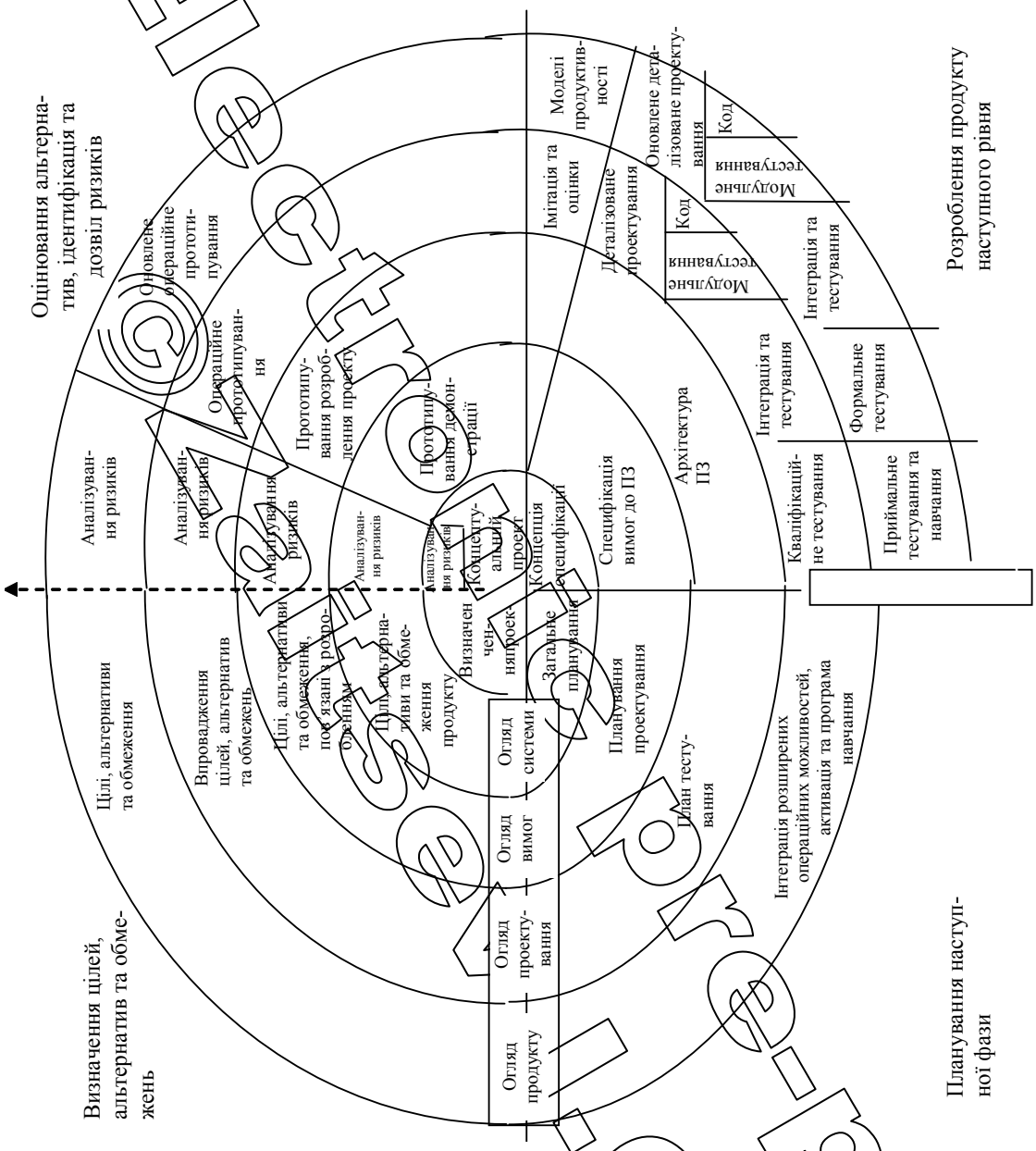


Рис. 3.12. – Структура спіральної моделі життєвого циклу програмного забезпечення

Спіральна модель (рис. 3.12) – це циклічна перестановка більшості дій з табл. 3.1, але дії виконують (проходять) за лінійним законом кілька разів, причому щоразу виробляють більш коректне програмне забезпечення, що є придатним для

задоволення зацікавлених сторін. У табл. 3.6 наведено процеси, дії та задачі спіральної моделі ЖЦ ПЗ.

Таблиця 3.6 – Перелік процесів, дій і задач, характерних для спіральної моделі життєвого циклу програмного забезпечення

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Перший прохід: цикл визначення проекту	
Визначення цілей, альтернатив і обмежень – визначення вимог і специфікацій для критичних частин системи щодо продуктивності, функціональних властивостей, здатності до сприйняття змін, програмно-апаратного інтерфейсу, критичних факторів успіху тощо **Визначення проекту** – визначення попередньої мети та розроблення плану проекту, за допомогою яких приблизно описують графіки і постачають продукти	
Початок виконання проекту	– установлення відповідності між діями та ЖЦ ПЗ – розподіл ресурсів проекту – установлення середовища проекту – керування планом проекту
Дослідження концепції	– ідентифікація ідей або потреб – формулювання потенційних підходів – проведення досліджень здійсненності – планування системних переходів – деталізація та заключне уточнення ідеї або потреби
Оцінювання альтернатив, а також ідентифікація та усунення ризиків –	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
оцінювання альтернатив, пов'язаних з цілями та обмеженнями; ідентифікація і усунення ризиків **Розроблення концептуального прототипу для обраної частини системи** – визначення вимог і специфікацій для потенційно найнебезпечніших частин системи з метою оцінювання та визначення ступеня ризику; поділ на окремі частини за ступенями ризику **Аналізування проекту** – розроблення проекту на основі раніше встановлених вимог	
Аналізування структури системи	– аналізування функцій – розроблення попередньої системної архітектури – декомпозиція попередніх системних вимог
Ідентифікація попередніх вимог до ПЗ	– проведення попередніх співбесід з користувачами щодо визначення та розроблення попередніх вимог до ПЗ – визначення попередніх вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог до ПЗ
Розроблення продукту наступного рівня – створення прототипу із застосовуванням інформації, отриманої на попередній фазі	– створення часткової специфікації вимог на системному рівні; включає концепцію операцій, які буде використано для побудови проекту демонстраційного прототипу
Планування наступної	– повторне планування керування

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
фази – застосовування інформації, яка відноситься до фази розроблення продукту на наступному рівні, до планування на наступні фази проекту	проектом – повторне формулювання потенційних підходів – повторне планування системного переходу – уточнення ідей або потреб – проведення огляду концепцій на системному рівні; приймання системної концепції демонстраційного прототипу
Другий прохід: цикл огляду вимог	
Визначення цілей, альтернатив і обмежень – визначення вимог і специфікацій для критичних частин системи щодо продуктивності, функціональних властивостей, здатності до сприйняття змін, програмно-апаратного інтерфейсу, критичних факторів успіху тощо	– аналізування функцій на рівні система/продукт – розроблення системної архітектури – декомпозиція системних вимог – уточнення та розроблення вимог до ПЗ – визначення вимог до інтерфейсу – розміщення пріоритетів та інтеграція програмних вимог на рівні система/ продукт
Оцінювання альтернатив, а також ідентифікація та усунення ризиків – оцінювання альтернатив, пов'язаних із цілями та обмеженнями; ідентифікація та дозвіл ризиків **Розроблення демонстраційного прототипу для обраної частини системи** – уточнення вимог та специфікацій для потенційно найнебезпечніших частин системи з метою оцінювання і визначення	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
ступеня ризику; поділ на частини за ступенем ризику **Вивчення здійсненності** — виконання імітацій і порівняльних тестів **Аналізування проекту** — проектування на основі попередньо сформульованих вимог	
Аналізування ризиків	– планування випадкових чинників – аналізування подальшої структури системи – подальший аналізування функцій – подальше розроблення попередньої архітектури системи – подальше декомпозиція попередніх системних вимог
Ідентифікація подальших вимог до програмного продукту	– подальше проведення співбесід з користувачами – подальше визначення та розроблення вимог до ПЗ – подальше визначення вимог до інтерфейсу – подальше розміщення пріоритетів і інтеграція системних вимог
Створення бази даних, ідентифікація попередніх елементів бази даних	– попереднє розроблення проекту архітектури – попереднє проектування бази даних
Проектування користувальницького інтерфейсу – визначення порядку взаємодії інтерфейсу з користувачем	– попереднє проектування інтерфейсів
Проектування алгоритмічних функцій	– вибір або розроблення алгоритмів (тільки «на папері»)
Розроблення продукту на наступному рівні – створення прототипу на базі інформації, отриманої	– створення завершеної специфікації вимог, включаючи деталі, які буде використано для створення прототипу проекту з метою його оцінювання

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
на попередній фазі	
Планування наступної фази — використовування інформації з наступного кроку фази продукту для виконання переходу, запланованого на кроці наступної фази проекту	– повторне планування керування проектом – повторне формулювання потенційних підходів – повторне планування системних переходів – уточнення ідей або потреб – виконання огляду вимог; – приймання проекту прототипу; – оцінювання проекту
Третій прохід: цикл огляду проекту	
Визначення цілей, альтернатив та обмежень – визначення вимог та специфікацій для критичних частин уявленої системи щодо продуктивності, функціональних властивостей, здатності до акомодації змін, програмно/апаратного інтерфейсу, критичних факторів успіху тощо	– аналізування функцій на рівні проекту – подальше розроблення системної архітектури – подальша декомпозиція системних вимог – подальше визначення та розроблення вимог до ПЗ – подальше визначення вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог на рівні проекту
Оцінювання альтернатив, а також ідентифікація та усунення ризиків – оцінювання альтернатив, пов'язаних з цілями і обмеженнями; ідентифікація та усунення ризиків **Розроблення**	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
демонстраційного прототипу для обраних частин системи – уточнення вимог та специфікацій для потенційно уразливих частин уявленої системи з метою оцінювання і визначення ступеня ризику; поділ на частини за ступенями ризику **Вивчення здійсненності –** виконання імітацій та порівняльних тестувань	
Аналізування проекту— розроблення проекту на основі:	– прийнятих вимог – аналізування ризиків – планування випадкових чинників
Подальше аналізування структури системи	– подальше аналізування функцій – подальше розроблення попередньої системної архітектури – подальша декомпозиція попередніх – системних вимог
Подальша ідентифікація вимог до програмного продукту	– подальше проведення співбесід з користувачем – подальше визначення та розроблення програмних вимог – подальше визначення вимог до інтерфейсу – подальше розміщення пріоритетів – уточнення розроблення проекту архітектури – уточнення проекту бази даних
Створення бази даних – ідентифікація елементів бази даних	– уточнення розроблення проекту інтерфейсу
Проектування користувальницького інтерфейсу – визначення методів взаємодії ПЗ з користувачем	– уточнення алгоритмів (тільки «на папері»)
Розроблення	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
алгоритмічних функцій **Розроблення продукту наступного рівня –** створення прототипу на основі інформації, отриманої на попередній фазі	
Похідне деталізоване розроблення проекту. Похідне деталізоване розроблення проекту ПЗ є похіднім від прийнятих вимог	– уточнення алгоритмів – виконання деталізованого розроблення проекту – генерування документа розроблення системного проекту
Планування наступної фази – використовування відомостей, отриманих на кроці фази продукту наступного рівня, для виконання переходу, запланованого для кроку наступної фази проекту	– повторне планування керування проектом – повторне формулювання потенційних підходів – повторне планування системного переходу – уточнення ідей або потреб – виконання огляду проекту; приймання оперативного проекту – прототипу проекту
Четвертий прохід: початковий цикл огляду можливостей продукту	
Визначення цілей, альтернатив і обмежень – визначення вимог і специфікацій для критичних частин уявленої системи щодо продуктивності, функціональних властивостей, здатності до акомодації змін, програмно/апаратного інтерфейсу, критичних факторів успіху тощо	– аналізування функцій на рівні продукту – подальше розроблення системної архітектури – подальша декомпозиція системних вимог – подальше уточнення та розроблення програмних вимог – подальше уточнення вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог на операційному рівні
Оцінювання альтернатив, а також	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
ідентифікація та усунення ризиків – оцінювання альтернатив, пов'язаних з цілями і обмеженнями; ідентифікація та оцінювання ризиків **Розроблення демонстраційного прототипу для обраної частини системи** – уточнення вимог та специфікацій для потенційно уразливих частин системи з метою виконання обчислень і оцінювання ризиків; поділ на окремі частини за ступенями ризику **Розроблення продукту наступного рівня** – створення прототипу на основі інформації, отриманої на попередній фазі **Створення операційного прототипу** – перетворення опису, який міститься в документі проекту ПЗ, у початкову операційну можливість програмного продукту, одержуючи при цьому вихідний код, бази даних і документацію незалежно від того, розроблені вони раніше, частково розроблені, придбані чи частково придбані **Вивчення здійсненності** — виконання імітації та	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
порівняльних тестів продуктивності	
Кодування — перетворення деталізованого проекту в операційну систему	– створення вихідного коду – генерування об'єктного коду – створення оперативної документації
Інтеграція — комбінування програмних компонентів	– план інтеграції – виконання інтеграції
Тестування – перевірення функціонуючого проекту	– план тестування, розроблення тестових вимог – створення тестових даних – виконання тестів
Установлення виробничої системи- установлення та перевіряння ПЗ в операційному середовищі, виконання необхідного настроювання з метою здійснення формального прийняття функціонуючого ПЗ замовником	– план установлення – поширення ПЗ – установлення ПЗ
Планування наступної фази – використовування інформації, отриманої на етапі наступного рівня продукту, з метою виконання переходу до планування (етап наступної фази проекту)	
Оцінювання системи — огляд операційного прототипу	– повторне планування керування проектом – повторне формулювання потенційних підходів – повторне планування системного

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
	переходу – уточнення ідеї або потреби – виконання огляду продукту; – приймання операційного прототипу
П'ятий прохід: заключний цикл огляду операційних можливостей продукту	
Визначення цілей, альтернатив і обмежень – визначення вимог і специфікацій для критичних частин системи щодо продуктивності, функціональних властивостей, здатності до сприйняття змін, програмно-апаратного інтерфейсу, критичних факторів успіху тощо	– аналізування функцій – подальше розроблення системної архітектури – подальша декомпозиція системних вимог – подальше визначення та розроблення вимог до ПЗ – подальше визначення вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог на заключному – операційному рівні
Оцінювання альтернатив, а також ідентифікація та усунення ризиків – оцінювання альтернатив, пов'язаних з цілями і обмеженнями; ідентифікація та оцінювання ризиків **Розроблення заключного проекту системи з операційними можливостями** – уточнення операційного прототипу для зазначеної системи	
Аналізування проекту — розроблення проекту на основі прийнятого проекту продукту	– аналізування ризиків – планування випадкових чинників – подальше аналізування структури системи – подальше аналізування функцій – подальше розроблення системної архітектури продукту

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Створення бази даних — ідентифікація елементів бази даних	– уточнення розроблення проекту архітектури – уточнення проекту бази даних
Проектування користувальницького інтерфейсу – визначення порядку взаємодії ПЗ з користувачем	– уточнення проекту інтерфейсу
Проектування алгоритмічних функцій	– уточнення алгоритмів
Розроблення продукту на наступному рівні – створення прототипу на базі інформації, отриманої на попередній фазі **Створення завершальних операційних можливостей системи** – перетворення опису документа оновленого програмного проекту в завершальні операційні можливості програмного продукту, при цьому створюють вихідний код, базу даних, документацію, а також проводять навчання, незалежно від того, розроблені вони раніше, частково розроблені, придбані чи частково придбані **Вивчення здійсненності** — виконання імітацій та порівняльного тестування	
Кодування — перетворення деталізованого розроблення проекту в операційну систему	– створення вихідного коду – генерування об'єктного коду – створення оперативної документації

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Інтеграція — комбінування програмних компонентів	– план інтеграції – виконання інтеграції
Тестування — перевіряння завершальної операційної можливості для даного екземпляра проекту	– план тестування – розроблення тестових вимог – створення тестових даних – тестування
Встановлення виробничої системи – встановлення та перевіряння ПЗ в операційному середовищі, настроювання (за необхідності) для здійснення формального прийняття замовником виробничого ПЗ	– план установлення
Планування наступної фази — використовування інформації етапу наступного рівня продукту для планування переходу на етап наступної фази проекту	
Оцінювання системи – виконання огляду операційного прототипу	– повторне планування – повторне формулювання потенційних підходів – повторне планування системних переходів – уточнення ідеї або потреби – огляд продукту, прийняття операційної системи
Експлуатація та супровід – переміщення ПЗ у виробничий стан	– поширення ПЗ – установлення ПЗ – приймання ПЗ в операційному середовищі

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
	– виконання операцій у системі – надання технічної допомоги та консультацій – ведення журналу запитів про допомогу
Процес виведення з експлуатації – припинення активного використовування існуючої системи шляхом відмови від виконання відповідних операцій, шляхом заміни новою системою або за допомогою застосування обновленої версії існуючої системи	– повідомлення користувачів – виконання паралельних операцій – виведення системи з експлуатації
Інтегральні дії – див. табл. 3.2	

Спіральна модель не використовує традиційних структурних методів, які з'являються наприкінці (тобто в зовнішньому циклі) спіралі. Версія, одержана внаслідок проведення користувачем приймальних випробувань, готова перед настанням етапу кінцевої придатності для експлуатації, як це також відбувається у каскадній моделі. Як показано на рис. 3.12, після аналізування та оцінювання ризиків у великому об'ємі у «хвості» спіральної моделі зображено етапи процесу, що нагадують каскадну модель.

Модель відображає базову концепцію, яка полягає в тому, що кожний цикл являє собою набір операцій, якому відповідає така ж кількість етапів, як і в процесах каскадної моделі. Але й має свої особливості. Особливістю спіральної моделі є увага до ризиків, що впливають на організацію життєвого циклу програмного забезпечення, серед них:

– дефіцит фахівців;

– нереалістичні терміни і бюджет;

– реалізація невідповідної функціональності;

– розробка неправильного користувацького інтерфейсу;

– безперервний потік змін;

– брак інформації про зовнішні компоненти, що визначають оточення системи або залучених в інтеграцію;

– недоліки в роботах, виконуваних зовнішніми (по відношенню до проекту) ресурсами;

– недостатня продуктивність одержуваної системи;

– розрив між кваліфікацією фахівців і вимогами проекту.

Велика частина цих ризиків пов'язана з організаційними та процесними аспектами взаємодії фахівців у проектній команді.

Кожен виток спіралі відповідає створенню фрагмента або версії програмного забезпечення, на ньому уточнюються цілі і характеристики проекту, визначається його якість і плануються роботи наступного витка спіралі. Таким чином поглиблюються і послідовно конкретизуються деталі проекту і в результаті вибирається обгрунтований варіант, який доводиться до реалізації.

На кожному витку спіралі можуть застосовуватися різні моделі процесу розробки ПЗ. В кінцевому підсумку на виході отримуємо готовий продукт. Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне завдання – якомога швидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог. Основна проблема спірального циклу- визначення моменту переходу на наступний етап. Для її вирішення необхідно ввести тимчасові

обмеження на кожен з етапів життєвого циклу. Перехід здійснюється відповідно до планів, навіть якщо не вся запланована робота закінчена. План складається на основі статистичних даних, отриманих в попередніх проектах, і особистого досвіду розробників.

Переваги спіральної моделі ЖЦ ПЗ:

– модель дозволяє користувачам «побачити» систему на ранніх етапах, що забезпечується за допомогою використовування прискореного прототипування в життєвому циклі розроблення ПЗ;

– забезпечується визначення максимальних ризиків без особливих додаткових витрат;

– модель дозволяє користувачам активно брати участь у плануванні, аналізуванні ризиків, розробленні, а також виконанні оцінювальних дій;

– модель забезпечує розбиття великого потенційного об'єму роботи з розроблення продукту на невеликі частини, у яких спочатку реалізуються вирішальні функції з високим ступенем ризику, що дозволяє усунути необхідність продовження роботи над проектом (таким чином, за потреби, стає можливим припинити роботу над проектом і зменшити витрати);

– у моделі передбачено можливість гнучкого проектування, оскільки в ній втілено переваги каскадної моделі і в той же час, дозволено ітерації на всіх фазах цієї моделі;

– реалізовано переваги інкрементної моделі, а саме випуск інкрементів, скорочення графіка за допомогою перекриття інкрементів, розсортованих за версіями, і незмінюваність ресурсів під час поступового зростання системи;

– зворотний зв'язок здійснюється з високою частотою і на ранніх етапах моделі, що забезпечує створення потрібного продукту високої якості;

– відбувається вдосконалення адміністративного керування процесом забезпечення якості, правильністю виконання процесу розроблення, витратами, дотриманням графіка, а також кадрових забезпечень, що досягається шляхом виконання огляду наприкінці кожної ітерації;

– підвищується продуктивність завдяки використовуванню придатних для повторного використовування компонентів;

– підвищується ймовірність передбачуваного поводження системи за допомогою уточнення поставлених цілей;

– за умови використовування спіральної моделі не потрібно розподіляти заздалегідь всі необхідні для виконання проекту фінансові ресурси;

– можна виконувати часте оцінювання сукупних витрат, а зменшення ризиків пов'язане зі зменшенням витрат.

Недоліки спіральної моделі ЖЦ ПЗ:

– якщо проект має низький ступінь ризику або невеликі розміри, модель може виявитися дорогою; оцінювання ризиків після проходження кожної спіралі пов'язане з більшими витратами;

– модель має ускладнену структуру, тому може бути ускладнене її застосування розробниками, менеджерами й замовниками;

– модель вимагає високопрофесійних знань для оцінювання ризиків;

– спіраль може тривати нескінченно, оскільки кожна відповідна реакція замовника на створену версію може породжувати новий цикл, що віддаляє закінчення роботи над

проектом, для якого необхідне ухвалення загального рішення про припинення процесу розроблення;

– велика кількість проміжних етапів може призвести до необхідності оброблення додаткової внутрішньої та зовнішньої документації;

– використовування моделі може виявитися дорогим і навіть недопустимим щодо засобів, тому що час, витрачений на планування, повторне визначення цілей, аналізування ризиків і прототипування може бути надмірним;

– під час виконання дій на етапі поза процесом розроблення виникає необхідність у перепризначенні розробників;

– можуть виникнути труднощі під час визначення цілей та етапів, що вказують на готовність продовжувати процес розроблення на наступній ітерації;

– відсутність прийнятного засобу або методу прототипування може зробити використовування моделі незручним;

– у виробництві використовування спіральної моделі ще не одержало широкого масштабу, як застосування інших моделей.

Спіральну модель доцільно застосовувати як модель ЖЦ ПЗ у випадках:

– коли створення прототипу являє собою прийнятний тип розроблення продукту;

– коли важливо повідомити, яким чином буде відбуватися збільшення витрат, і підрахувати витрати, пов'язані з виконанням дій із квадранта ризику;

– коли організація має навички, необхідні для адаптації моделі;

– для проектів, виконання яких пов’язане із середнім і високим ступенем ризику;

– коли недоцільно братися за виконання довгострокового проекту через потенційні зміни, які можуть відбутися в економічних пріоритетах, і коли така невизначеність може викликати обмеження в часі;

– коли мова йде про застосування нової технології та коли необхідно протестувати базові концепції;

– коли користувачі не впевнені у своїх потребах;

– коли вимоги занадто складні;

– за умови розроблення нової функції або нової серії продукту;

– коли очікуються істотні зміни, наприклад, під час вивчення специфікацій проекту, або в дослідницькій роботі;

– коли важливо сконцентрувати увагу на незмінних або відомих частинах, коли збір інформації про частини, що змінюються, ще не закінчено;

– у випадку великих проектів;

– для організацій, які не можуть собі дозволити заздалегідь виділити всі необхідні для виконання проекту кошти, і коли у процесі розроблення відсутня фінансова підтримка;

– коли переваги розроблення неможливо точно визначити, а досягнення успіху не гарантовано;

– з метою демонстрації якості та досягнення цілей за період часу;

– коли у процес залучено нові технології;

– за умови розроблення систем, які вимагають великого об'єму обчислень, наприклад, систем прийняття рішень;

– під час виконання бізнес-проектів, а також проектів у галузі аерокосмічної промисловості, оборони та інжинірингу, де використовування спіральної моделі вже одержало визнання.

Еволюціонування спіральної моделі пов'язано з питаннями деталізації робіт. Особливо варто виділити акцент

на більш важливих питаннях - вимог, дизайну та коду, тобто надання більшої важливості питанням ітерацій, в тому числі, збільшення їх кількості при скороченні тривалості кожної ітерації. У результаті, можна визначити загальний набір контрольних точок в сьогоднішній спіральної моделі:

– Concept of Operations (COO) - концепція (використання) системи;

– Life Cycle Objectives (LCO) - цілі та зміст життєвого циклу;

– Life Cycle Architecture (LCA) - архітектура життєвого циклу; тут же можливо говорити про готовність концептуальної архітектури цільової програмної системи;

– Initial Operational Capability (IOC) - перша версія створюваного продукту, придатна для дослідної експлуатації;

– Final Operational Capability (FOC) - готовий продукт, розгорнутий (встановлений і налаштований) для реальної експлуатації.

Отже, ***спіральна модель*** є всього лише модифікацією каскадної моделі процесу розробки програмного забезпечення, в якій на кожному витку спіралі виконується створення чергової версії продукту, уточнюються вимоги проекту, визначається його якість і плануються роботи наступного витка. Особлива увага приділяється початковим етапам розробки - аналізу та проектуванню, де реалізація тих чи інших технічних рішень перевіряється і обгрунтовується за допомогою створення прототипів (макетування).

3.4.4 ІНКРЕМЕНТНА ТА ІТЕРАЦІЙНІ МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Альтернативними моделями каскадній моделі є так звані модель ітеративної і інкрементної розробки (англ. iterative and incremental development, IID)RUP, що мають за визначенням Т. Гілба назву еволюційної моделі. При розробці ПЗ та програмних систем ці дві моделі часто використовують паралельно. Відмінність моделей полягає в тому, щоб розробити систему шляхом циклів, що повторюються (ітеративний) та в менші проміжки часу (інкрементний), даючи змогу розробнику скористатися знаннями, що були отримані підчас розробки попередніх порцій або версій системи. Навчання приходить як з процесу розробки так і з процесу використання системи, де можливі ключові кроки процесу починаються з простої реалізації підмножини вимог до програмного продукту та ітеративного вдосконалення версій, які еволюціонують доки повна система не буде реалізована. На кожному кроці (ітерації) або за певні проміжки часу(інкременти) виконуються зміни дизайну та додаються нові функціональні можливості до програмного забезпечення.

Тобто, інкрементна розробка поділяє систему на інкременти(порції). У кожній порції доставляється певна частина функціональності. Уніфікований процес поділяє порції/ітерації на такі фази: початок, створення плану, побудова та перехід.

Кожна з фаз може бути розділена на 1 або більше ітерацій, які, зазвичай, обмежені у часі, а не у функціональності. Архітектори та аналітики працюють на одну ітерацію попереду програмістів та тестувальників, для того, щоб тримати їхній перелік завдань повним.

Певною модифікацією ітеративного процесу розробки програмного забезпечення, що розроблене Rational Software є Раціональний уніфікований процес (RUP).

3.4.4.1 Інкрементна модель

Інкрементна модель (Incremental model) є класичним прикладом інкрементної стратегії конструювання. Вона об'єднує елементи послідовної каскадної моделі з ітераційною філософією макетування (запропонована Б. Боемом як удосконалення каскадної моделі). Кожна лінійна послідовність тут виробляє певний інкремент ПЗ.

Ключові етапи цього процесу – проста реалізація підмножини вимог до програми і вдосконалення моделі в серії послідовних релізів до тих пір, поки не буде реалізовано ПЗ в усій повноті. В ході кожної ітерації організація моделі змінюється, і до неї додаються нові функціональні можливості.

Розроблення інкрементної моделі є процесом часткової реалізації всієї системи та повільного нарощування функціональних можливостей або ефективності. Цей підхід дозволяє зменшити витрати, понесені до моменту досягнення рівня вихідної продуктивності. За допомогою цієї моделі прискорюється процес створення функціонуючої системи. Цьому сприяє застосований принцип компонування зі стандартних блоків, завдяки якому забезпечується контролювання процесу розроблення вимог, що змінюються.

Інкрементна модель діє за принципом каскадної моделі з перекриттями, завдяки чому функціональні можливості продукту, придатні до експлуатації, формуються раніше. Для цього може знадобитися повний заздалегідь сформований набір вимог, які виконують у вигляді послідовних, невеликих

за розміром проектів, або виконання проекту може початися з формулювання загальних цілей, які потім уточнюють та реалізують групи розробників.

Інкрементний метод описується як процес об'єднання елементів каскадної моделі та прототипування і реалізується за принципом нарощування функціональних можливостей продукту. Подібне вдосконалювання каскадної моделі ефективне під час використовування як для надзвичайно великих, так і невеликих проектів.

Наприклад, перший інкремент призводить до отримання базового продукту, реалізує базові вимоги (правда, багато допоміжні вимоги залишаються нереалізованими). План наступного інкремента передбачає модифікацію базового продукту, що забезпечує додаткові характеристики і функціональність. За своєю природою інкрементний процес ітеративний, але, на відміну від макетування, інкрементна модель забезпечує на кожному Інкременті працюючий продукт. Нові дані виходять як в ході розробки ПЗ, так і в ході його використання, де це можливо.

Лінійну послідовність може бути розподілено відповідно до календарного графіка, причому в результаті виконання кожної послідовності може створюватися результативний інкремент програмного продукту, як зображено на рис. 3.13.

Інкрементна модель описує процес, під час виконання якого першорядну увагу приділяють системним вимогам, а потім їхній реалізації групами розробників. Як правило, згодом інкременти зменшуються та реалізують щоразу меншу кількість вимог. Кожна наступна версія системи додає до попередньої певні функціональні можливості доки не буде реалізовано всі заплановані можливості. У цьому випадку можна зменшити витрати, контролювати вплив вимог, що змінюються, і прискорити створення функціональної системи

завдяки використовуванню методу компонування зі стандартних блоків.

Передбачається, що на ранніх етапах життєвого циклу (планування, аналізування і розроблення проекту) виконують конструювання системи в цілому. На цих етапах визначають стосовні до них інкременти та функції. Кожний інкремент потім проходить через інші фази життєвого циклу: кодування, тестування і постачання.

Спочатку виконують: конструювання, тестування та реалізацію набору функцій, що формують основу продукту, або вимоги першорядної важливості, що відіграють основну роль для успішного виконання проекту чи знижують ступінь ризику. Наступні ітерації поширюються на ядро системи, поступово полінцтючи її функціональні можливості або робочу характеристику. Додавання функцій здійснюють за допомогою виконання істотних інкрементів з метою комплексного задоволення потреб користувача. Кожну додаткову функцію атестують відповідно до набору вимог.

Інктементну модель розробки можна комбінувати з іншими моделями. Найчастіше його поєднують зі спіральною, V-подібною, а також каскадною моделями, що дозволяє знизити витрати та ризики під час розроблення системи.

Інкрементна модель (рис 3.13) – лінійна модель, у якій використовують також дії, наведені в табл. 3.1. В інкрементних моделях завершену систему розробляють на ранніх стадіях проекту, а потім реалізують у вигляді версій дискретних компонентів, набір функціональних властивостей яких постійно розширюється. Найчастіше відбувається перекриття дій з розроблення кожного дискретного компонента. Кількість компонентів, що створюються, визначають на ранніх стадіях проекту. Задачі та дії у прикладі інкрементної моделі, в якій визначено створення трьох компонентів для завершеної системи, наведено в табл. 3.7

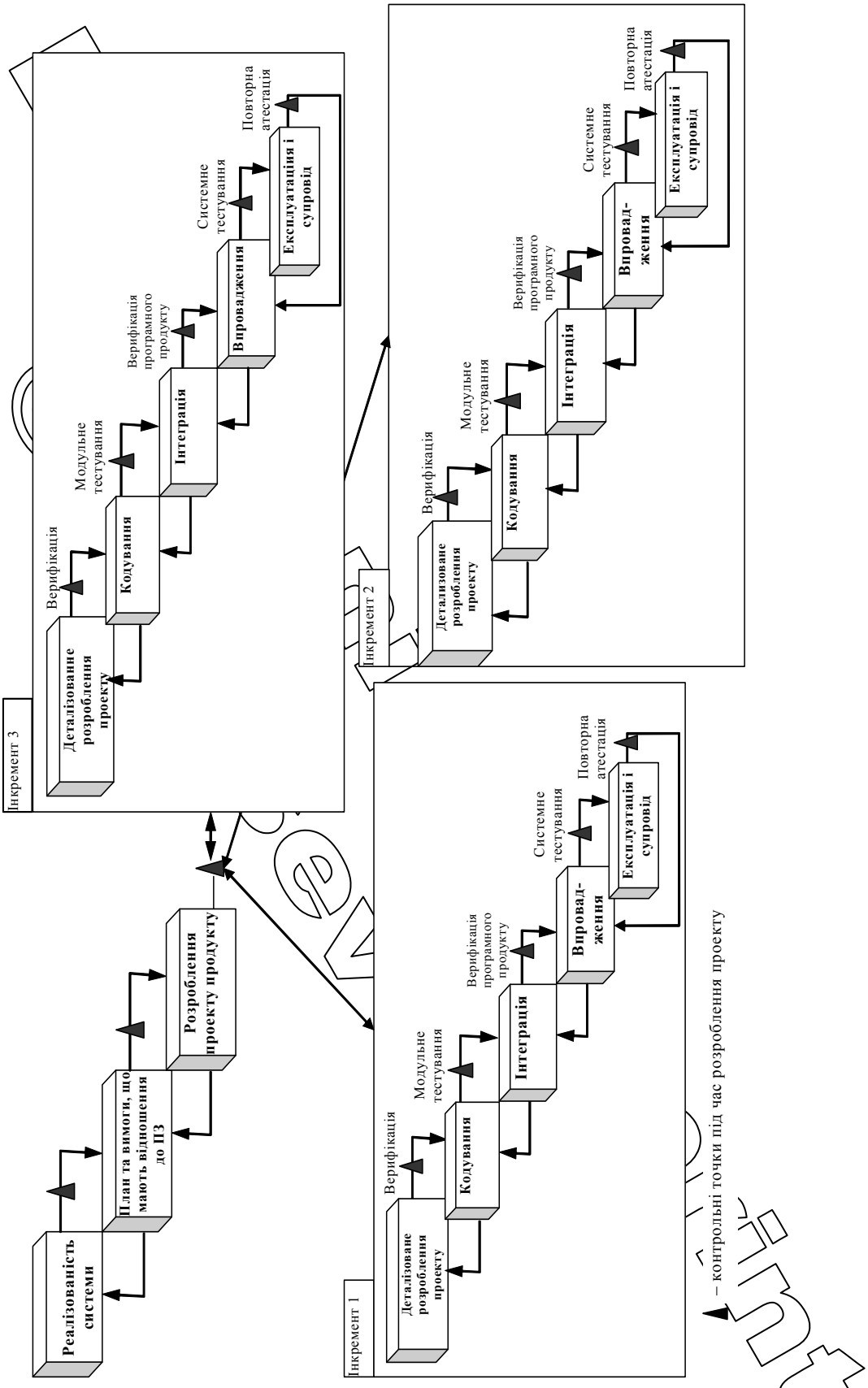


Рис. 3.13 – Структура інкрементної моделі
життєвого циклу програмного забезпечення

Таблиця 3.7 – Процеси, дії та задачі інкрементної моделі

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Планування та активізація проекту	
Початок виконання проекту	– встановлення відповідності між діями та ЖЦ ПЗ – розподіл ресурсів проекту – встановлення середовища проекту – планування керування проектом
Здійсненність системи – дослідження вимог на системному рівні з метою визначення їх здійсненності	– ідентифікація ідей або потреб – формулювання потенційних підходів – вивчення здійсненності – планування системного переходу – уточнення й завершення ідеї та потреби
Процес визначення структури системи – установлення відповідності між функціями та програмно-апаратним забезпеченням на основі загальної системної архітектури	– аналізування функцій – розроблення архітектури системи – декомпозиція системних вимог
Процес визначення вимог – визначення вимог до ПЗ для інформаційної області, функцій, поводження, продуктивності, а також інтерфейсів системи	– визначення та розроблення вимог до ПЗ – визначення вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог до ПЗ
Процес розроблення проекту – розроблення та подання логічно несуперечливої технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також деякі процедурні (алгоритмічні) деталі	– проектування бази даних – проектування інтерфейсів – вибір або розроблення алгоритмів

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Розроблення проекту продукту: перший етап	
Процес розроблення проекту – розроблення та подання логічно несуперечливої технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі	
Потенційні дії	– вибір або розроблення алгоритмів – виконання деталізованого розроблення проекту
Процес кодування – перетворення опису розроблення програмного проекту в програмний продукт.	– генерування тестових даних – створення вихідного коду – генерування об'єктного коду
Процес інтеграції – комбінування елементів з наступним створенням робочого компонента системи	– план інтеграції – виконання інтеграції
Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Процес експлуатації та підтримки – включає користувальницькі операції в системі та поточній підтримці, у тому числі технічну допомогу, консультації користувачів, реєстрування користувальницьких запитів про розширення і зміни, а також оброблення виправлень або помилок	

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Операційний процес – використовування системи у виробництві	– експлуатація системи – надання технічної допомоги та проведення консультацій – підтримка журналу запитів про технічну допомогу
Процес супроводу – аналізування запитів з метою виявлення програмних помилок, збоїв, недоліків, розширень і змін, згенерованих у процесі супроводу	– повторне застосування циклу розроблення ПЗ (початок –виконання проекту)
Розроблення проекту продукту: другий етап	
Процес розроблення проекту — розроблення та подання логічно несуперечливої технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі	– вибір або розроблення алгоритмів – виконання деталізованого розроблення проекту
Процес кодування – перетворення опису процесу розроблення проекту ПЗ в програмний продукт	– генерування тестових даних – створення вихідного коду – генерування об'єктного коду
Процес інтеграції – комбінування елементів у складі робочого компонента системи	– план інтеграції – виконання інтеграції
Процес упровадження – перетворення програмних компонентів у початкові версії встановлених програмних продуктів з	– створення документації – планування встановлення ПЗ – поширення ПЗ – встановлення ПЗ – приймання ПЗ в операційному

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
документацією. Виконують установлення та перевіряння ПЗ в операційному середовищі, а також формальне приймання ПЗ на другому етапі	середовищі
Процес експлуатації та супроводу – містить користувальницькі операції в системі та поточній підтримці, включаючи забезпечення технічної допомоги, консультації користувачів, фіксування користувальницьких запитів щодо розширень і змін, а також оброблення виправлень або помилок	
Процес експлуатації – використовування системи у виробничих цілях	– виконання операцій у системі – надання технічної допомоги та консультацій –ведення журналу запитів про технічну підтримку
Процес супроводу – аналізування запитів з метою виявлення програмних помилок, збоїв, дефектів, розширень і змін, згенерованих у процесі підтримки	– повторне застосовування циклу розроблення (ініціалізація циклу розроблення)
Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Розроблення проекту продукту: третій етап	
Процес розроблення проекту – розроблення та подання логічно несуперечливої технічної специфікації програмної	– вибір або розроблення алгоритмів – виконання деталізованого розроблення проекту

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі	
Процес кодування – перетворення опису програмного дизайну в програмний продукт	– створення тестових даних – створення вихідного коду – генерування вихідного коду
Процес експлуатації та підтримки – залучення користувальницьких операцій у системі та поточній підтримці, включаючи надання технічної допомоги, проведення консультацій користувачів, фіксування запитів користувачів щодо розширень і змін, а також оброблення виправлень або помилок	
Процес експлуатації – використовування системи у виробництві	– виконання операцій у системі – надання технічної допомоги та консультацій – ведення журналу запитів про підтримку
Процес супроводу - аналізування запитів з метою виявлення помилок, збоїв, дефектів, розширень, а також змін, згенерованих у ході процесу підтримки	– повторне застосовування циклу розроблення ПЗ (початок розроблення проекту)
Процес виведення з експлуатації – припинення активного використовування існуючої системи або шляхом відмови від	– повідомлення користувача (користувачів) – проведення паралельних операцій – виведення системи з експлуатації

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
виконання відповідних операцій, заміни новою системою, або за допомогою застосування оновленої версії існуючої системи	
Інтегральні дії — див. табл. 3.2	

Переваги інкрементної модель ЖЦ ПЗ:

– не потрібно заздалегідь витрачати засоби, необхідні для розроблення всього проекту, оскільки спочатку розробляють та реалізують основну функцію або функції із групи високого ризику;

– у результаті виконання кожного інкремента виходить функціональний продукт;

– уроки, отримані в результаті виконання кожного інкрементного постачання, можуть сприяти вдосконалюванню наступних постачань. Замовник має у своєму розпорядженні можливість висловитися із приводу кожної розробленої версії системи;

– використовування послідовних інкрементів дозволяє об'єднати отриманий користувачами досвід у вигляді вдосконаленого продукту, затративши при ньому набагато менше засобів, ніж потрібно для виконання повторного розроблення;

– можливість розбиття проблеми, що виникла, на керовані частини, дозволяє уникати формування громіздких переліків вимог, висунутих команді розробників;

– у процесі розроблення допускається обмеження кількісті персоналу таким чином, щоб над постачанням кожного інкремента послідовно працювала одна і та сама команда, і всі задіяні у процесі розроблення команди не

припиняли роботи над проектом (графік розподілу робочої сили може вирівнюватися за допомогою розподілу за часом обсягу роботи над проектом);

– можливість почати побудову наступної версії проекту на перехідному етапі попередньої версії згладжує зміни, викликані зміною персоналу;

– наприкінці кожного інкрементного постачання необхідно переглянути ризики, пов'язані з витратами та дотриманням установленого графіка;

– знижуються витрати на попереднє постачання програмного продукту;

– прискорюється початковий графік постачання, що дозволяє відповідати зростаючим вимогам ринку;

– знижується ризик невдачі та зміни вимог;

– потреби клієнта краще підлягають керуванню, оскільки час розроблення кожного інкремента є незначним;

– замовник може звикати до нової технології поступово;

– замовники можуть розпізнавати найважливіші та корисні функціональні можливості продукту на більш ранніх етапах розроблення;

– ризик розподіляється на менші за розміром інкременти і не зосереджений в одному великому проекті;

– поліпшується розуміння вимог для пізніших інкрементів, що забезпечується можливістю одержання користувачем подання про раніше отримані інкременти на практичному рівні;

– інкременти функціональних можливостей приносять більше користі та простіші під час тестування, ніж продукти проміжного рівня під час розроблення за принципом «донизу».

Недоліки інкрементної моделі ЖЦ ПЗ:

– у моделі не передбачено ітерації у рамках кожного інкремента;

– визначення повної функціональної системи повинне здійснюватися на початку життєвого циклу, щоб забезпечити визначення інкрементів;

– оскільки створення деяких модулів буде завершено значно раніше інших, виникає необхідність у чітко визначених інтерфейсах;

– формальне критичне аналізування і перевіряння набагато складніше виконати для інкрементів, ніж для системи в цілому;

– може виникнути тенденція до відтягування рішень важких проблем на майбутнє з метою демонстрування керівництву успіху, досягнутого на ранніх етапах розроблення;

– замовник повинен усвідомлювати, що загальні витрати на виконання проекту не буде знижено;

– використовування на етапі аналізування загальних цілей замість повністю сформульованих вимог, що може виявитися незручним для керівництва.

Інкрементну модель доцільно застосовувати як модель ЖЦ ПЗ у випадках:

– якщо більшість вимог можна сформулювати заздалегідь, але їхня поява очікується через певний період часу;

– якщо існує потреба швидкого постачання на ринок продукту, що має функціональні базові властивості;

– для проектів, на виконання яких передбачено великий період часу розроблення (біля одного року);

– за умови рівномірного розподілу властивостей різного ступеня важливості;

– за умови розроблення програм, пов'язаних з низьким або середнім ступенем ризику;

– за умови виконання проекту із застосовуванням нової технології, що дозволяє користувачеві адаптуватися до системи шляхом виконання більш дрібних інкрементних кроків, без різкого переходу до застосування основного нового продукту;

– коли під час розгляду ризику, фінансування, графіка виконання проекту, розміру програми, її складності або необхідності реалізації на ранніх фазах виявляється, що найоптимальнішим варіантом є застосування принципу пофазового розроблення;

– коли однопрохідне розроблення системи пов'язане з великим ступенем ризику;

– коли результативні дані виходять через регулярні інтервали часу.

Отже, застосування ***інкрементної моделі*** при розробці поділяє систему на інкременти(порції) в кожному із яких реалізується певна частина функціональності програмного забезпечення.

3.4.4.2 Ітераційна модель

Ітеративний підхід (англ. Iteration, «повторення») в розробці програмного забезпечення – це виконання робіт паралельно з безперервним аналізом отриманих результатів і коригуванням попередніх етапів роботи. Проект при цьому підході в кожній фазі розвитку проходить повторюваний цикл PDCA (англ. Plan-do- check-act cycle): Планування – Реалізація – Перевірка – Оцінка.

Вперше запропонована Філіпом Крачтеном в 1995 р, дана модель об'єднує головні переваги спіральної, інкрементної, каскадної моделей, а також методів розробки на основі створення прототипів і об'єктно-орієнтованого підходу. Вона завоювала більшу популярність і в тому чи іншому вигляді використовується в багатьох сучасних проектах.

Відповідно до ітеративної моделі (рис.3.14) є чотири основні фази життєвого циклу розробки ПЗ (початок, дослідження, побудова та впровадження). На кожній фазі проект проходить безліч ітерацій, що призводять до створення працездатних версій: на початкових створюються прототипи, уточнюються вимоги, опрацьовуються найбільш складні проблеми; кінцеві приводять до створення продукту, його вдосконалення і розширенню функціональності.

Ітеративна модель, крім основних фаз, виділяє ще дві групи процесів: робочі (управління вимогами, аналіз і проектування, реалізація, тестування, розгортання) і допоміжні (управління конфігурацією і змінами, проектом і процесом). Кількість і суть процесів варіюються в залежності від потреб розробника, вони також можуть мати свої цикли, які не обов'язково навіть відповідають основним фазам. Однак результатом робочих процесів завжди є створення версій продукту.

Рис. 3.14 – Структура ітераційної моделі
життєвого циклу програмного забезпечення

Ітеративний процес припускає, що різні види діяльності не прив'язані намертво до певних етапів розробки, а виконуються в міру необхідності, іноді повторюються, до тих пір, поки не буде отриманий потрібний результат. Тобто, каскадна модель з можливістю повернення на попередній крок при необхідності переглянути його результати, стає ітеративною(рис.3.15). Разом з гнучкістю і можливістю швидко реагувати на зміни, ітеративні моделі привносять додаткові складності в управління проектом та відстеження його ходу. При використанні ітеративного підходу значно складніше стає адекватно оцінити поточний стан проекту та спланувати довгостроковий розвиток подій, а також передбачити терміни і ресурси, необхідні для забезпечення певної якості результату. При ітераціях доводиться відкидати частину зробленої раніше роботи.

Ітеративна модель подібно до спіральної дає можливість успішно справлятися з ризиками. Якщо під час роботи над черговою версією буде встановлено, що трудовитрати на реалізацію необхідної функціональності занадто великі, то перевищення бюджету і порушення термінів можна буде уникнути шляхом співвідношення пріоритетів розробки та трудовитрат на початку кожної ітерації. Таким чином, дана модель добре

підходить для більшості типів програмних проектів, але особливо її переваги помітні при роботі над продуктами, призначеними для виходу на вільний ринок, в силу початкової орієнтації на випуск послідовних версій.

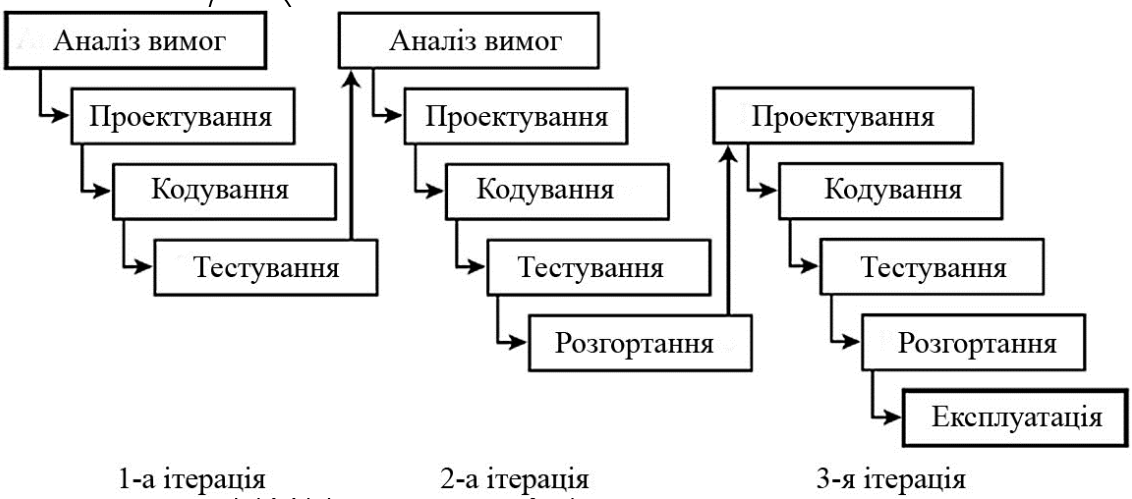


Рис.3.15 – Структура ітераційної моделі
життєвого циклу програмного забезпечення на базі каскадної моделі

Ітеративна модель є основою RUP (Rational Unified Process) – одного з найбільш поширених методів комплексного управління процесом розробки ПЗ. На її ж основі розроблений головний конкурент RUP з боку Microsoft – MSF (Microsoft Solutions Framework), а також аналогічний підхід компанії Borland – ALM (Application Lifecycle Management).

Переваги ітеративного підходу:

– зниження впливу серйозних ризиків на ранніх стадіях проекту, що веде до мінімізації витрат на їх усунення;

– організація ефективного зворотного зв'язку проектної команди з споживачем (а також замовниками, стейкхолдерами) і створення продукту, який реально відповідає його потребам

– акцент зусиль на найбільш важливі і критичні напрямки проекту;

– безперервне ітеративне тестування, що дозволяє оцінити успішність всього проекту в цілому;

– раннє виявлення конфліктів між вимогами, моделями і реалізацією проекту;

– більш рівномірне завантаження учасників проекту;

– ефективне використання накопиченого досвіду;

– реальна оцінка поточного стану проекту і, як наслідок, велика впевненість замовників і безпосередніх учасників в його успішному завершенні;

– витрати розподіляються по всьому проекту, а не групуються наприкінці.

Переваги ітеративної моделі:

– раннє створення працюючого ПЗ;

– гнучкість – готовність до зміни вимог на будь-якому етапі розробки;

– кожна ітерація – маленький етап, для якого тестування і аналіз ризиків забезпечити простіше, ніж для всього життєвого циклу продукту.

Недоліки ітеративної моделі:

– кожна фаза – самостійна, окремі ітерації НЕ накладаються;

– можуть виникнути проблеми з реалізацією загальної архітектури системи, оскільки не всі вимоги відомі до початку проектування.

Ітеративну модель доцільно застосовувати як модель ЖЦ ПЗ у випадках:

– коли відомі, принаймні, ключові вимоги;

– коли вимоги до проекту можуть змінюватися в процесі розробки.

У математичних термінах, ітеративна модель є реалізацією методики послідовної апроксимації – тобто, поступове наближення до образу готового продукту:

– ітеративна модель не передбачає повного обсягу вимог для початку робіт над продуктом;

– розробка може початися з вимог до функціонала, який може змінюватися;

– процес повторюється;

– за результатами кожної ітерації приймається рішення – чи будуть результати використовуватися як точка входу для наступної ітерації.

Ключ до успішного використання цієї моделі – сувора валідація вимог і ретельна верифікація функціональності, що розробляється в кожній з ітерацій. У кожній ітерації створюється програмне забезпечення, яке вимагає тестування на всіх рівнях.

Отже, ***ітеративна модель*** є ключовим елементом так званих «гнучких» (Agile) підходів до розробки програмного забезпечення.

3.4.4.3 Раціональний уніфікований процес (RUP)

Раціональний уніфікований процес (Rational Unified Process, RUP) – одна з кращих методологій розробки програмного забезпечення. Грунтуючись на досвіді багатьох успішних програмних проектів, RUP дозволяє створювати складні програмні системи, грунтуючись на індустріальних методах розробки. Передумови для розробки RUP зародилися на початку 1980-х рр. в Rational Software corporation. На початку 2003 р Rational придбала IBM. Одним з основних

стовпів, на які спирається RUP, є процес створення моделей за допомогою уніфікованої мови моделювання (UML).

RUP – одна з спіральних методологій розробки ПЗ. Методологія підтримується і розвивається компанією Rational Software. Як мова моделювання в загальній базі знань використовується мова Unified Modelling Language (UML). Ітераційна і інкрементна розробка програмного забезпечення в RUP передбачає поділ проекту на кілька проектів, які виконуються послідовно, і кожна ітерація розробки чітко визначена набором цілей, які повинні бути досягнуті в кінці ітерації. Кінцева ітерація передбачає, що набір цілей ітерації повинен в точності збігатися з набором цілей, зазначених замовником продукту, тобто всі вимоги повинні бути виконані.

Процес передбачає еволюціонування моделей; ітерація циклу розробки однозначно відповідає певній версії моделі програмного забезпечення. Кожна з ітерацій містить елементи управління життєвим циклом програмного забезпечення: аналіз і дизайн (моделювання), реалізація, інтегрування, тестування, впровадження. У цьому сенсі RUP є реалізацією спіральної моделі, хоча досить часто зображується у вигляді графіка-таблиці.

На рис. 3.16 представлені два виміри: горизонтальна вісь представляє час і показує тимчасові аспекти життєвого циклу процесу; вертикальна вісь представляє дисципліни, які визначають фізичну структуру процесу. Видно, як з плином часу змінюються акценти в проекті. Наприклад, в ранніх ітераціях більше часу відводиться вимогам; в пізніх ітераціях більше часу відводиться реалізації. Горизонтальна вісь сформована з часових відрізків – ітерацій, кожна з яких є самостійним циклом розробки; мета циклу – принести в кінцевий продукт деяке, заздалегідь визначене,відчутне доопрацювання, корисне з точки зору зацікавлених осіб.

По осі часу життєвий цикл ділиться на чотири основні фази.

Фаза 1. Початок (Inception) – формування концепції проекту, розуміння того, що ми створюємо, уявлення про продукт (Vision), розробка бізнес-плану (business case), підготовка прототипу програми або часткового вирішення. Це фаза збору інформації та аналізу вимог, визначення способу проекту в цілому. Мета – отримати підтримку і фінансування. У кінцевій ітерації результат цього етапу – технічне завдання.

Фаза 2. Проектування, розробка (Elaboration) – уточнення плану, розуміння того, як ми це створюємо, проектування, планування необхідних дій і ресурсів, деталізація особливостей. Завершується етап визначенням архітектури, коли всі архітектурні рішення прийняті і ризики враховані. виконувана архітектура є працюючим програмним забезпеченням, яке демонструє реалізацію основних архітектурних рішень. У кінцевій ітерації це – технічний проект.

Фаза 3. Реалізація, створення системи (Construction) – етап розширення функціональності системи, закладеної в архітектурі. У кінцевій ітерації це – робочий проект.

Фаза 4. Впровадження, розгортання (Transition), створення кінцевої версії продукту. Фаза впровадження продукту, постачання товару конкретному користувачу (тиражування, доставка і навчання).

Вертикальна вісь складається з дисциплін, кожна з них може бути більш детально розписана з точки зору виконуваних завдань, відповідальних за них ролей, продуктів, які подаються завданням на вхід і випускаються в ході їх виконання і т.д.

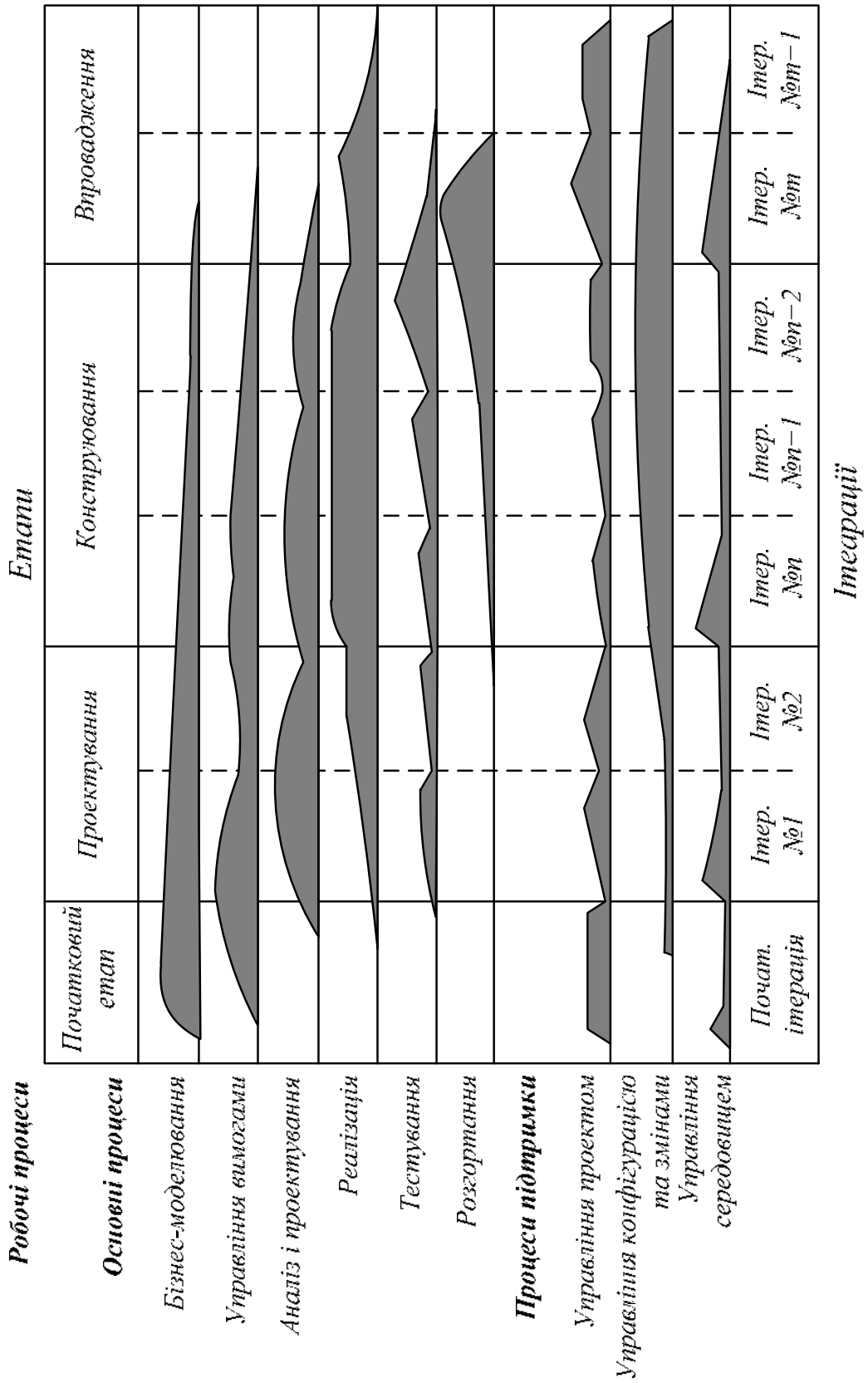


Рис. 3.16 – Структура раціонального уніфікованого процесу життєвого циклу програмного забезпечення

1. Бізнес-аналіз та моделювання (Business modeling) забезпечує реалізацію принципів моделювання з метою вивчення бізнесу організації та накопичення знань про нього, оптимізації бізнес- процесів і прийняття рішення про їх часткову або повну автоматизацію.

2. Управління вимогами (Requirements) присвячено отриманню інформації від зацікавлених осіб та її перетворенню в набір вимог, що визначають зміст запропонованої системи і детально описують очікування від того, що система повинна робити.

3. Аналіз і проектування (Analysis and design) охоплює процедури перетворення вимог в проміжні описи (Моделі), що пояснюють, як ці вимоги повинні бути реалізовані.

4. Реалізація (Implementation) охоплює розробку коду, тестування на рівні розробників і інтеграцію компонентів, підсистем і всієї системи відповідно до встановлених специфікацій.

5. Тестування (Test) присвячено оцінці якості створюваного продукту.

6. Розгортання (Deployment) охоплює операції, що мають місце при передачі продуктів замовникам і забезпеченні доступності продукту кінцевим користувачам.

7. Конфігураційне управління та управління змінами (Configuration management) присвячено синхронізації проміжних і кінцевих продуктів і управління з їх розвитком в ході проекту і пошуком прихованих проблем.

8. Управління проектом (Management) присвячено плануванню проекту, управлінню ризиками, контролю ходу його виконання та безперервній оцінці ключових показників.

9. Управління середовищем (Environment) включає елементи формування середовища розробки інформаційної системи і підтримки проектної діяльності.

Залежно від специфіки проекту можуть бути використані будь-які засоби IBM Rational, а також третіх фірм.

У RUP рекомендовано дотримуватися 6 практик, що дозволяє успішно розробляти проект: ітеративна розробка; управління вимогами; використання модульних архітектур; візуальне моделювання; перевірка якості; відстеження змін.

Невід'ємну частину RUP складають артефакти (artefact), прецеденти (precedent) і ролі (role).

Артефакти (artefact) – це деякі продукти проекту, породжувані або використовуються в ньому при роботі над остаточним продуктом.

Прецеденти (precedent) – це послідовності дій, виконуваних системою для отримання спостережуваного результату. Фактично будь-який результат роботи окремих людей чи групи є артефактом, будь то документ аналізу, елемент моделі, файл коду, тестовий скрипт, опис помилки і т. д.

Для підтримки процесів RUP в IBM розроблений широкий набір інструментальних засобів:

– Rational Rose – CASE – засіб візуального моделювання інформаційних систем, що має можливості генерування елементів коду. Спеціальна редакція продукту – Rational Rose.
– RealTime – дозволяє на виході отримати виконуваний модуль.
– Rational Requisite Pro – засіб управління вимогами, який дозволяє створювати, структурувати, встановлювати пріоритети, відстежувати, контролювати зміни вимог,що виникають на будь-якому етапі розробки компонентів додатку.
– Rational ClearQuest – продукт для управління змінами та відстеження дефектів в проекті (bug tracking), тісно інтегрується із засобами тестування і управління

вимогами і представляє собою єдине середовище для зв'язування всіх помилок і документів між собою.

– Rational SoDA – продукт для автоматичного генерування проектної документації, що дозволяє встановити корпоративний стандарт на внутріньофірмові документи, можливо також приведення документації до вже існуючих стандартів (ISO, CMM).
– Rational Purify, Rational Quantify Rational PureCoverage, – засоби тестування і налагодження.
– Rational Visual Quantify – засіб вимірювання характеристик для розробників додатків і компонентів, що програмують на C / C ++, Visual Basic і Java; допомагає визначати і усувати вузькі місця в продуктивності ПЗ.
– Rational Visual PureCoverage – автоматично визначає області коду, які не схильні до тестування.
– Rational ClearCase – продукт для управління конфігурацією програм (Rational's Software Configuration Management, SCM), що дозволяє проводити версійність ,контроль всіх документів проекту. З його допомогою можна підтримувати кілька версій проектів одночасно, швидко переключаючись між ними. Rational Requisite Pro підтримує поновлення і відстежує зміни у вимогах для групи розробників.
– SQA TeamTest – засіб автоматизації тестування;
– Rational TestManager – система управління тестуванням,яка об'єднує всі пов'язані з тестуванням інструментальні засоби, артефакти, сценарії і дані;
– Rational Robot – інструмент для створення, модифікації та автоматичного запуску тестів;

– SiteLoad, SiteCheck -для тестування Web-сайтів на продуктивність і наявність непрацюючих посилань;
– Rational PerformanceStudio – вимір і передбачення характеристик продуктивності систем.

Цей набір продуктів постійно вдосконалюється і поповнюється. Так, наприклад, недавній продукт IBM Rational Software Architect (RSA) є частиною IBM Software Development Platform – набору інструментів, що підтримують життєвий цикл розробки програмних систем. Продукт IBM Rational Software Architect призначений для побудови моделей програмних систем, які розробляються, з використанням уніфікованої мови моделювання UML 2.0, перш за все моделей архітектури розроблюваних додатків. Проте, RSA об'єднує в собі функції таких програмних продуктів, як Rational Application Developer, Rational Web Developer і Rational Software Modeler, тим самим надаючи можливість архітекторам і аналітикам створювати різні уявлення розробляється інформаційної системи з використанням мови UML 2.0, а розробникам – виконувати розробку J2EE, XML, веб-сервісів і т.д.

Крім того, Rational Software Architect підтримує технологію розробки, керованої моделями (model-driven development, MDD), яка дозволяє моделювати програмне забезпечення на різних рівнях абстракції з можливістю трасуванню.

3.4.5 МЕТОДОЛОГІЯ ШВИДКОГО ЕВОЛЮЦІЙНОГО ПРОТОТИПУВАННЯ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методологія швидкого еволюційного прототипування (evolutionary rapid development) життєвого циклу програмного забезпечення запропонована в SPC-97057-CMC, що був виданий 1997 р. під авторством R.M. De Santis, J. Blyskal, A. Moini та M. Tappan.

Прототипування – це процес побудови робочої експериментальної моделі системи (прототипу).

Реалізація еволюції ПЗ спрямована на досягнення високої продуктивності. Цей метод також допускає, що протягом усього процесу розроблення елементів ПЗ у ньому бере участь користувач.

«Швидка» часткова реалізація системи створюється перед етапом визначення вимог або впродовж цього етапу. Кінцеві користувачі системи використовують прискорений прототип, а потім шляхом зворотного зв'язку повідомляють про своє рішення команді, яка працює над проектом, для подальшого уточнення вимог до системи. Процес уточнення триває доти, поки користувач не одержить того, що йому потрібно. Після завершення процесу визначення вимог шляхом розроблення прискорених прототипів, одержують детальний проект системи, а прискорений прототип доопрацьовують, у результаті чого одержують кінцевий робочий продукт.

Можна сформувати модель прототипування високої якості, не заощаджуючи на документації, аналізуванні, проектуванні, тестуванні тощо. Така модель ЖЦ ПЗ одержала назву моделі швидкого еволюційного прототипування (рис. 3.17).

Початок життєвого циклу розроблення розташовано в центрі еліпса. Користувач і програміст розробляють попередній план проекту, керуючись при цьому попередніми вимогами.

Використовуючи методи швидкого аналізування, користувач і програміст спільно працюють над визначенням вимог і специфікацій для найважливіших частин майбутньої системи.

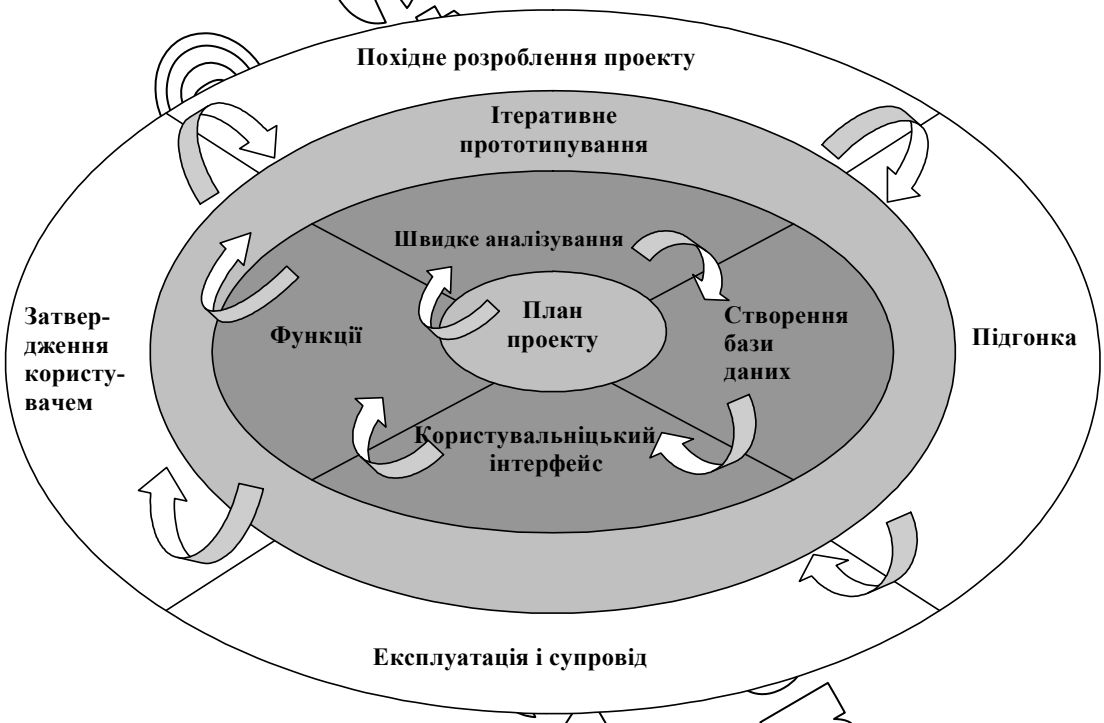


Рис. 3.17 – Методологія швидкого еволюційного прототипування

Планування проекту – це перша дія на етапі швидкого аналізування, за допомогою якого одержують документ, що описує загалом зразкові графіки та результативні дані.

Таким чином, створюють план проекту, а потім виконують швидке аналізування, після чого проектують базу даних, користувальницький інтерфейс і розроблення функцій.

Друга дія – це швидке аналізування, протягом якого попередні опитування користувачів використовують для розроблення навмисно неповної високорівневої моделі

системи на рівні документації. У результаті виконання цього завдання одержують документ, який містить часткову специфікацію вимог, яку використовують для побудови вихідного прототипу, що створюється на наступних трьох етапах. Дизайнер конструює модель ПЗ з використанням інструментальних засобів, тобто часткове подання системи, що містить у собі тільки ті базові властивості, які необхідні для задоволення вимог замовника. Потім починається ітераційний цикл швидкого прототипування. Розробник проекту демонструє прототип (модель), а користувач оцінює його функціонування. Цей процес триває доти, доки користувач не буде задоволений тим, яким чином система відображає поставлені до неї вимоги. Команда розробників проекту продовжує виконувати цей процес, доки користувач не погодиться, що швидкий прототип у точності відображає системні вимоги. Створення бази є першою із цих двох фаз.

Після створення вихідної бази даних починають розроблення меню, після чого розробляють функції для створення робочої моделі. Модель демонструють користувачеві з метою одержання пропозицій щодо її вдосконалення, які поєднуються в послідовні ітерації, доки робоча модель не стане задовільною. Потім одержують офіційне схвалення користувачем функціональних можливостей прототипу для створення документа попереднього проекту системи.

Основною є фаза ітерації прототипу, протягом якої під час використовування сценаріїв, наданих робочою моделлю, користувач може вимагати послідового уточнення моделі, поки не буде виконано всі функціональні вимоги. Одержавши схвалення користувача, швидкий прототип перетворюють у детальний проект і систему настроюють на виробниче використовування. Саме на цьому етапі настроювання прискорений прототип стає повністю діючою системою, що

заміняє собою часткову систему, отриману в ітераційному циклі прототипування.

Деталізований проект можна також одержати на основі прототипів. У цьому випадку настроювання прототипу виконують під час використовування коду або зовнішніх утиліт. Розробник використовує затверджені вимоги як основу для проектування виробничої версії ПЗ.

Під час розроблення виробничої версії програми може виникнути необхідність у додатковій роботі. Може знадобитися більш високий рівень функціональних можливостей, різні системні ресурси, необхідні для забезпечення повного робочого навантаження або обмеження в часі. Після цього починають тестування у граничних режимах, визначення вимірювальних критеріїв і настроювання, а потім починають функціональний супровід.

Заключною фазою є функціонування та супровід, які відображають дії, спрямовані на переміщення системи у стадію виробничого процесу.

Не існує «правильного» способу використовування методу прототипування. Отриманий результат може не знадобитися, його може бути використано як підставу для наступної модернізації або перетворено у продукт застосовних процесів та бажаної якості залежно від вихідних цілей. За умови використання еволюційного прототипування знижуються витрати та оптимізується дотримання графіків, оскільки кожний з його компонентів знаходить своє застосування.

Структурована модель еволюційного швидкого прототипування (див. рис 3.17) є результатом циклічної перестановки більшості дій, описаних у табл. 3.1. При цьому треба враховувати, що цю модель застосовують кілька разів, і кожний раз створюється більш коректний продукт, що

постачається. У табл. 3.8 наведено процеси, задачі та дії, які виконують у моделі швидкого прототипування.

Таблиця 3.8 – Процеси, дії та задачі, які виконуються у моделі швидкого прототипування

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Спільне планування проекту – користувач та розробник проекту спільно розробляють попередній план проекту, у якому надають схематичні графіки постачання продукту	
Початок виконання проекту	– встановлення відповідності між діями та планом ЖЦ ПЗ – розподіл ресурсів проекту – установлення середовища проекту – планування керування проектом – дослідження концепції – ідентифікація ідей або потреб – формулювання потенційних підходів – дослідження здійсненності – планування системних переходів – уточнення та завершення ідей або потреб
Розробка «паперового» прототипу — спільна робота користувача та розробника проекту з метою розроблення вимог та специфікацій для критичних частин ПЗ **Швидке аналізування** — користувач та розробник спільно розробляють проект на основі попередньо сформульованих вимог	
Аналізування структури системи	– аналізування функцій – розроблення попередньої системної архітектури – декомпозиція попередніх

	системних вимог – ідентифікація попередніх програмних вимог – проведення попередньої співбесіді з користувачами – визначення та розроблення попередніх вимог до ПЗ – розроблення попередніх вимог до інтерфейсу, що створюється – розміщення пріоритетів та інтеграція вимог до ПЗ
Створення бази даних- спільна ідентифікація попередніх елементів бази даних	– попереднє розроблення проекту архітектури – проектування попередньої бази даних
Проектування користувальницького інтерфейсу – спільне визначення порядку взаємодії ПЗ з користувачем	– проектування попередніх інтерфейсів
Розробка алгоритмічних функцій	– вибір або розроблення алгоритмів (тільки на папері)
Часткова **специфікація вимог –** використовують під час створення початкового прототипу **Створення програмного прототипу системи –** застосування специфікації швидкого аналізування вимог **Впровадження система-** розроблення ПЗ на основі специфікацій та результатів вимірювань	– створення тестових даних – створення вихідного коду – генерування об'єктного коду – план установлення – установлення ПЗ
Оцінювання системи — спільний огляд програмного прототипу Повторне **виробництво програмного прототипу** – включення змін, вивчених на етапі оцінювання (за необхідності). Етап може повторюватися кілька разів.	

Використовують авторитетні думки для планування циклів навчання	
Відновлення **специфікації вимог** – застосовують для створення виправленого прототипу Повторення **швидкого аналізування** – користувач та розробник виконують спільно повторне розроблення проекту на основі виправлених вимог	– аналізування структури системи – проведення попередньої співбесіди з користувачами – уточнення та подальше розроблення вимог до ПЗ – ідентифікація виправлених вимог до ПЗ – визначення виправлених вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог до ПЗ
Коригування бази даних – спільна ідентифікація скорегованих елементів бази даних	– розроблення скорегованого проекту архітектури – корегування розроблення проекту бази даних
Корегування користувальницького інтерфейсу – спільне повторне визначення порядку взаємодії інтерфейсу з користувачем	– розроблення попередніх інтерфейсів
Корегування алгоритмічних функцій	– вибір або розроблення алгоритмів
Приймання функціональних властивостей прототипу ПЗ – формальне схвалення користувачем функціональних властивостей прототипу **Успадкування деталізованого розроблення проекту на етапі**	– уточнення алгоритмів – деталізоване розроблення проекту – створення документа, що визначає розроблення проекту виробничої системи

виробництва ПЗ — успадкування деталізованого розроблення проекту на основі прийнятого швидкого прототипу	
Впровадження– трансформування опису розроблення проекту системи у програмний продукт. При цьому створюють вихідний код, базу даних та документацію незалежно від того, придбано, розроблено чи частково розроблено продукт	
Кодування — перетворення деталізованого розроблення проекту у виробничу систему	– створення вихідного коду – генерування об'єктного коду – створення оперативної документації
Інтеграція — об'єднання програмних компонентів	– план інтеграції – виконання інтеграції
Тестування — перевіряння впровадження розробленого проекту	– план тестування – розроблення тестових вимог – створення тестових даних – виконання тестів
Установлення системи – установлення та перевіряння ПЗ в операційному середовищі, виконання необхідного настроювання з метою формального приймання ПЗ замовником	– план установлення – розподіл ПЗ – установлення ПЗ – приймання ПЗ у виробничому операційному середовищі
Процес експлуатації та підтримки – виконують операції в системі та поточній підтримці,	– виконання операцій у системі – забезпечення технічної допомоги та виконання консультацій – підтримка журналу запитів про

включаючи забезпечення технічної підтримки, консультації користувачів, фіксування користувальницьких запитів на розширення і зміну, а також оброблення виправлень або помилок	допомогу
Процес супроводу — аналізування запитів з метою знаходження програмних помилок, збоїв, недоліків, розширень, а також змін, здійснених у процесі супроводу	– повторне застосування циклу розроблення ПЗ (початок нового циклу розроблення ПЗ)
Процес виведення з експлуатації – припинення активного використовування існуючої системи або шляхом завершення системних операцій за допомогою впровадження нової системи або шляхом використовування оновленої версії існуючої системи	– повідомлення користувача (користувачів) – виведення системи з експлуатації
Експлуатація та супровід — переведення ПЗ у виробничий стан	– поширення ПЗ – установлення ПЗ – приймання ПЗ в операційному середовищі – операції в системі – забезпечення технічної допомоги та консультації – підтримка журналу запитів про допомогу
Інтегральні дії –див. табл. 3.2	

Переваги швидкого еволюційного прототипування:

– кінцевий користувач може побачити системні вимоги у процесі їхньої реалізації командою розробників; таким чином,

взаємодія замовника із системою починається на ранньому етапі розроблення;

– виходячи з реакції замовників на демонстрацію продукту, що розробляється, розробники одержують відомості про один або кілька аспектів поводження системи, завдяки чому мінімізується кількість неточностей у вимогах;

– знижується можливість перекручування інформації або непорозумінь під час визначення системних вимог, що призводить до створення більш якісного кінцевого продукту;

– у процес розроблення можна внести нові вимоги користувача, що часом необхідно, тому що реальність може відрізнятися від концептуальної моделі реальності;

– модель є формальною специфікацією, втіленою в робочу модель;

– модель дозволяє виконувати гнучке проектування та розроблення, включаючи кілька ітерацій на всіх фазах життєвого циклу;

– під час використовування моделі створюються постійні, видимі ознаки прогресу у виконанні проекту, завдяки чому замовники почувають себе впевнено;

– можливість виникнення розбіжностей під час спілкування замовників з розробниками мінімізована;

– очікувану якість продукту визначають за активної участі користувача на ранніх фазах розроблення;

– можливість спостерігати ту або іншу функцію в дії створює умови для розроблення функціональних додаткових можливостей;

– завдяки меншому обсягу доопрацювань зменшуються витрати на розроблення;

– скорочуються загальні витрати завдяки тому, що проблеми виявляють до залучення додаткових ресурсів;

– забезпечується керування ризиками;

– документацію сконцентровано на кінцевому продукті, а не на його розробленні;

– беручи участь у процесі розроблення протягом усього життєвого циклу, користувачі більшою мірою будуть удоволені отриманими результатами.

Недоліки швидкого еволюційного прототипування:

– модель може бути відхилена через уявлення про проект ПЗ, отриманого з її допомогою, як про проект, розроблений «швидкоруч»;

– розроблені «швидкоруч» прототипи, на відміну від еволюційних прискорених прототипів, можуть мати неадекватну або неповну документацію;

– якості або експлуатаційній надійності ПЗ може бути приділено недостатньо уваги;

– іноді в результаті використовування моделі одержують систему з низькою робочою характеристикою, особливо якщо у процесі її виконання пропущено етап підгонки;

– на ітераційному етапі прототипування швидкий прототип являє собою частину системи; якщо виконання проекту завершують достроково, у кінцевого користувача залишиться лише часткова система;

– розбіжність уявлень замовника та розробників про використовування прототипу може привести до створення іншого користувальницького інтерфейсу;

– замовник може виявити бажання одержати прототип, замість того, щоб чекати появи повної, добре продуманої версії;

– прототипування викликає залежність і може тривати занадто довго; недостатньо досвідчені розробники можуть потрапити у так званий цикл «кодування-усунення помилок» (code-and-fix cycle), що призводить до дорогих незапланованих ітерацій прототипування;

– розробники та користувачі не завжди розуміють, що, коли прототип перетворюється у кінцевий продукт, існує необхідність у традиційній документації; якщо вона відсутня, модифікування моделі на більш пізніх етапах може виявитися більш дорогим, ніж відмова від створеного прототипу;

– коли замовники, вдоволені прототипом, вимагають його негайного постачання, у менеджера програмного проекту виникає бажання піти їм назустріч;

– на замовників може вплинути той факт, що вони не мають інформації про точну кількість ітерацій, які будуть необхідні;

– на розроблення системи може бути витрачено занадто багато часу, тому що ітераційний процес демонстрації прототипу і його перегляд можуть тривати нескінченно без належного керування процесом; у користувачів може виникнути прагнення поповнювати список елементів, призначених для прототипування доти, поки проект не досягне обсягу, що значно перевищує рамки, встановлені аналізуванням здійснення проектного рішення;

– під час вибору інструментальних засобів прототипування (операційні системи, мови тощо) розробники можуть зупинити свій вибір на менш прийнятному рішенні, щоб продемонструвати їхню здатність;

– структурні методи не використовуються, щоб не перешкодити аналізуванню; під час прототипування необхідно проаналізувати реальні вимоги, здійснити проектування та звернути увагу на якість із метою створення програми, яка допускає супровід, так само, як і в будь-якій іншій моделі життєвого циклу (хоча на ці дії може знадобитися менше часу та ресурсів).

Методологію швидкого еволюційного прототипування доцільно застосовувати як модель ЖЦ ПЗ, якщо:

– вимоги не відомі заздалегідь або підлягають уточненню;

– вимоги непостійні або можуть бути невірно тлумачені чи невдало сформульовані;

– існує необхідність розроблення користувальницьких інтерфейсів;

– потрібне перевіряння концепції;

– здійснюються тимчасові демонстрації;

– побудоване за принципом структурної моделі швидке еволюційне прототипування можна успішно використати у великих системах, у яких деякі частини підлягають прототипуванню, а деякі розробляються більш традиційним способом;

– виконується нове, що не має аналогів, розроблення (на відміну від експлуатації продукту на вже існуючій системі);

– потрібно зменшити неточності у визначенні вимог, тобто зменшується ризик створення системи, що не має цінності для замовника;

– вимоги підлягають швидким змінам, коли замовник неохоче погоджується на фіксований набір вимог або якщо про прикладну програму відсутнє чітке уявлення;

– розробники не впевнені у тому, яку оптимальну архітектуру або алгоритми треба застосовувати;

– алгоритми або системні інтерфейси ускладнені;

– потрібно демонструвати технічну здійсненність, коли технічний ризик великий;

– розробляється ПЗ, особливо у випадку програм, коли проявляється середній і високий ступінь ризику;

– застосовують цю модель в комбінації з каскадною моделлю. На початковому етапі проекту використовується прототипування, а на останньому – фази каскадної моделі з метою забезпечення функціональної ефективності системи та якості;

– прототипування завжди треба використовувати разом з елементами аналізування та проектування, які застосовують під час об’єктно-орієнтованого розроблення;

– швидке прототипування доцільно використовувати для розроблення інтенсивно застосованих систем користувальницького інтерфейсу таких як індикаторні панелі для контрольних приладів, інтерактивні системи, нові продукти, а також системи забезпечення прийняття рішень і системи керування.

Отже, ***методологія швидкого еволюційного прототипування*** - це концепція інтеграції програмних систем на основі повторного використання компонентів та використання програмних та архітектурних шаблонів.

3.4.6 МЕТОДОЛОГІЯ ШВИДКОГО РОЗРОБЛЕННЯ ДОДАТКІВ (RAPID APPLICATION DEVELOPMENT)

Концепція RAD стала відповіддю на незграбні методи розробки програм 1970-х і початку 1980-х років, такі як «каскадна модель» та її модифікації. Ці методи передбачали настільки повільний процес створення програми, що часто навіть вимоги до програми встигали змінитися до закінчення розробки. Засновником RAD вважається співробітник IBM Джеймс Мартін, який в 1980-х роках сформулював основні принципи RAD, грунтуючись на ідеях Баррі Бойма і Скотта Шульца. А в 1991 році Мартін опублікував відому книгу, в якій детально виклав концепцію RAD і можливості її застосування. В даний час RAD стає загальноприйнятою схемою для створення засобів розробки програмних продуктів.

Під час застосування методу швидкого розроблення додатків (*Rapid Applic*ation Development – RAD) користувач є учасником розроблення на всіх фазах життєвого циклу проекту, а не тільки під час визначення вимог. Участь користувача у процесах ЖЦ стає активною завдяки використовуванню засобу розроблення середовища, який дозволяє оцінювати продукт на всіх стадіях його розроблення. Це забезпечується наявністю засобів розроблення графічного користувальницького інтерфейсу та генераторів коду.

Характерною рисою RAD є короткий час переходу від визначення вимог до створення повної системи. Метод ґрунтується на послідовності ітерацій еволюційної системи або прототипів, критичне аналізування яких обговорюється із замовником. У процесі такого аналізування формуються вимоги до продукту. Розроблення кожного інтегрованого продукту чітко обмежується певним періодом часу, що, як правило, становить 60 днів і називається часовим блоком.

Чинники, що дозволяють створити систему за короткий час без зниження якості, містять у собі використовування інструментальних засобів розроблення та високий рівень чинника повторного використовування.

Принципи RAD технології спрямовані на забезпечення трьох основних її переваг – високої швидкості розробки, низької вартості і високої якості. Досягти високої якості програмного продукту досить непросто ,бо одна з головних причин виникаючих труднощів полягає в тому, що розробник і замовник бачать предмет розробки (ПЗ) по-різному.

– інструментарій повинен бути націлений на мінімізацію часу розробки;

– створення прототипу для уточнення вимог замовника;

– циклічність розробки: кожна нова версія продукту ґрунтується на оцінці результату роботи попередньої версії замовником;

– мінімізація часу розробки версії, за рахунок перенесення вже готових модулів і додавання функціональності в нову версію;

– команда розробників повинна тісно співпрацювати, кожен учасник повинен бути готовий виконувати кілька обов'язків;

– управління проектом має мінімізувати тривалість циклу розробки.

Принципи RAD застосовуються не тільки при реалізації, а й поширюються на всі етапи життєвого циклу, зокрема на етап обстеження організації, побудови вимог, аналіз і дизайн.

Модель RAD, наведена на рис. 3.8, демонструє етапи розроблення ПЗ та відображає участь користувача на всіх етапах цього процесу (крива пунктирна лінія).

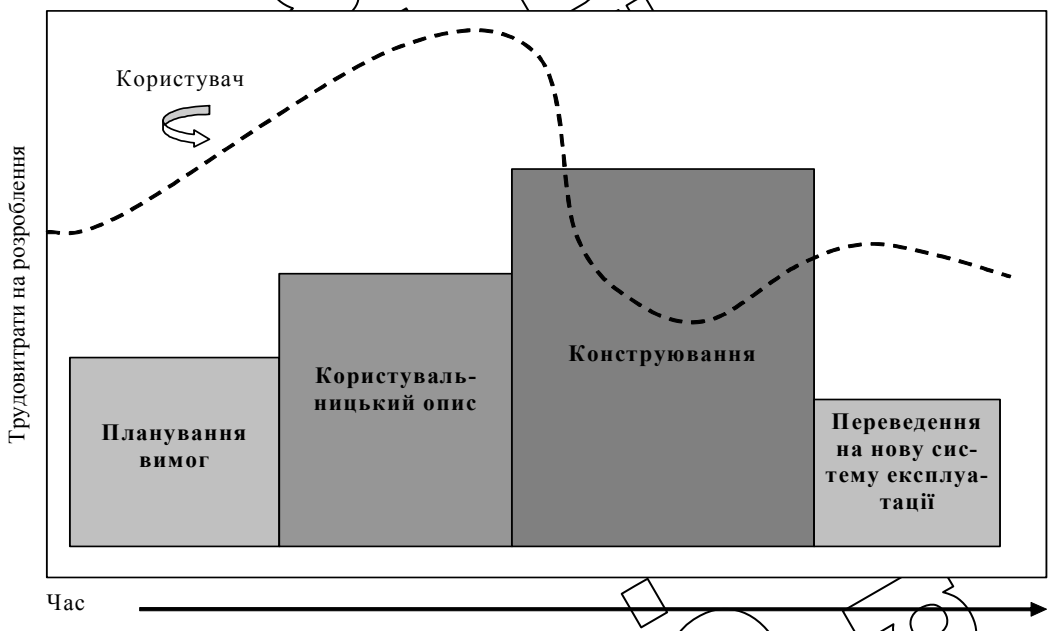


Рис. 3.18 – Структура моделі життєвого циклу програмного забезпечення за методологією швидкого розроблення додатків (RAD)

Базовими етапами в методологія RAD є:

– етап планування вимог – збір вимог виконують під час використовування робочого методу, названого спільним

плануванням вимог, що являє собою структурне аналізування і обговорення наявних комерційних завдань;

– користувальницький опис – спільне проектування додатка використовують з метою залучення користувачів; на цій фазі проектування системи, що не є промисловою, команда, що працює над проектом, найчастіше використовує автоматичні інструментальні засоби, які забезпечують збір користувальницької інформації;

– конструювання («до повного завершення») – поєднує у собі деталізоване проектування, реалізацію (кодування та тестування), а також постачання програмного продукту замовникові за певний час; строки виконання цієї фази значною мірою залежать від використовування генераторів коду, екранних генераторів та інших типів виробничих інструментальних засобів;

– переклад на нову систему експлуатації – включає проведення користувачами приймальних випробувань, установлення системи та навчання користувачів.

Модель RAD (рис. 3.18) – є спеціальним випадком каскадної моделі. Головною рисою цієї моделі є те, що для неї властивий надзвичайно короткий цикл розроблення ПЗ. Основна область застосовування цієї моделі – розроблення додатків для інформаційних систем з архітектурою клієнт/сервер. Процеси та дії, які наведено в табл. 3.1, застосовують також і для моделі RAD. Процеси, задачі та дії моделі RAD зазначено в табл. 3.9.

Таблиця 3.9 – Процеси, дії та задачі, які виконуються у моделі RAD

Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Планування та активізація	

проекту	
Початок виконання проекту	– установлення відповідності між діями та планом ЖЦ ПЗ – розподіл ресурсів проекту – установлення середовища проекту – планування керування проектом
Дії життєвого циклу програмного забезпечення	Задачі життєвого циклу програмного забезпечення
Дослідження концепції — дослідження вимог на системному рівні з метою визначення здійсненності із застосуванням підходу спільного планування вимог, ідентифікація ідей або потреб	– формулювання потенційних підходів – вивчення здійсненності – планування системного переходу – уточнення/завершення ідеї та потреби
Процес визначення структури системи — установлення відповідності між функціями та програмно-апаратним забезпеченням на основі загальної системної архітектури та із застосуванням спільного планування вимог	– аналізування функцій – розроблення архітектури системи – декомпозиція системних вимог
Процес визначення вимог — визначення вимог до ПЗ для інформаційної області та функцій системи	– визначення та розроблення вимог до ПЗ – визначення вимог до інтерфейсу – розміщення пріоритетів та інтеграція вимог до ПЗ
Процес розроблення проекту – розроблення та подання логічно несуперечливої технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі; у цьому випадку використовують підхід, застосований під час	– проектування інтерфейсів – вибір або розроблення алгоритмів – фаза конструювання

спільного розроблення додатків, розроблення проекту архітектури, проектування бази даних	
Розроблення проекту – розроблення та подання логічно несуперечливої технічної специфікації програмної системи, включаючи структури даних, програмну архітектуру, подання інтерфейсу, а також процедурні (алгоритмічні) деталі, створені за обмежений період часу	– розроблення проекту архітектури – проектування бази даних – проектування інтерфейсів – вибір або розроблення алгоритмів – виконання деталізованого дизайну
Процес упровадження – перетворення опису розроблення проекту ПЗ у програмний продукт, створення вихідного коду, баз даних та документації незалежно від того, розроблені вони раніше, частково розроблені, придбані чи частково придбані	– генерування тестових даних – створення вихідного коду – генерування об'єктного коду – створення документації – планування інтеграції – виконання інтеграції – планування тестування – розроблення тестових вимог – виконання тестів
Процес установлення – установлення та перевіряння ПЗ в операційному середовищі, а також виконання формального приймання замовником	– план установлення – поширення ПЗ – інсталяція ПЗ – установлення ПЗ в операційному середовищі
Інтегральні дії - див. табл. 3.2	

Переваги методології RAD:

– час циклу розроблення для всього проекту можна скоротити завдяки використовуванню потужних інструментальних засобів;

– потрібна менша кількість фахівців, оскільки розроблення системи виконують зусиллями команди, обізнаної у предметній галузі;

– існує можливість зробити швидкий початковий огляд продукту;

– зменшуються витрати на розроблення завдяки скороченому часу ЖЦ і вдосконаленій технології, а також меншій кількості задіяних у процесі розробників;

– зменшуються витрати та ризики, пов'язані з дотриманням графіка;

– модель забезпечує ефективне використовування засобів та структур, які є в наявності;

– залучення замовника на постійній основі мінімізує ризик його невдоволеності розробленим продуктом. Крім того, це гарантує, що система буде відповідати комерційним потребам, а сам програмний продукт буде надійним в експлуатації;

– до складу кожного часового блоку входить аналізування, проектування та впровадження;

– основну увагу перенесено з документації на код, при цьому діє принцип «одержуєте те, що бачите» (what you see is what you get, WYSIWYG);

– у моделі використовують такі принципи та інструментальні засоби моделювання:

1) ділове моделювання (методи передавання інформації, місце генерування інформаційних потоків, ким і куди направляється, яким чином обробляється);

2) моделювання даних (відбувається ідентифікація об'єктів даних і атрибутів, а також взаємозв'язків);

3) моделювання процесу (виконується перетворення об'єктів даних);

4) генерування додатка (методи останніх поколінь компіляторів);

– у моделі повторно використовують компоненти вже існуючих програм.

Недоліки методології RAD:

– якщо користувачі не можуть постійно брати участь у процесі розроблення протягом усього життєвого циклу, це може негативно позначитися на кінцевому продукті;

– під час використовування цієї моделі необхідна достатня кількість висококваліфікованих розробників, що вміють користуватися обраними інструментальними засобами розроблення для прискорення часу розроблення;

– виникає потреба в системі, яка може бути змодельована коректним (правильним) чином;

– використовування моделі може виявитися невдалим у випадку, якщо відсутні придатні для повторного використовування компоненти;

– для реалізації моделі потрібні розробники та замовники, готові до швидкого виконання дій з твердими часовими обмеженнями;

– команди, які розробляють комерційні проекти за допомогою моделі RAD, можуть «затягувати» розроблення програмного продукту настільки, що його своєчасне постачання кінцевому користувачеві буде під загрозою;

– існує ризик, що роботу над проектом ніколи не буде завершено; у зв'язку із цим менеджер проекту повинен співробітничати як з командою розробників, так і з замовником, що дозволить уникнути появи замкнутого циклу.

Модель RAD доцільно застосовувати як модель ЖЦ ПЗ у випадках:

– у системах, які підлягають моделюванню, а також у масштабованих системах;

– у системах, вимоги до яких у достатній мірі добре відомі;

– у випадках, коли кінцевий користувач може брати участь у процесі розроблення протягом усього життєвого циклу;

– коли користувачі хочуть брати активну участь у використовування автоматичних інструментальних засобів;

– під час виконання проектів, розроблення яких необхідно виконати у скорочений строк (як правило, не більш ніж за 60 днів);

– коли придатні до повторного використовування компоненти можна одержати з автоматичних сховищ програмних продуктів;

– у системах, які є некритичними або мають невеликий розмір;

– коли витрати та дотримання графіка не є найважливішим питанням (наприклад під час розроблення власних інструментальних засобів);

– за умови невисокого ступеня технічних ризиків;

– в інформаційних системах;

– при наявності висококваліфікованих і вузькоспеціалізованих архітекторів. Бюджет проекту великий, щоб оплатити цих фахівців разом з вартістю готових інструментів автоматизованого складання. RAD-модель може бути обрана при впевненому знанні бізнес-цілей і необхідності термінового виробництва системи протягом 2-3 місяців..

Отже, ***методологія швидкої розробки додатків (RAD)*** – концепція створення засобів розробки додатків, програмних продуктів, що приділяє особливу увагу швидкості й зручності програмування, створенню технологічного процесу, що дозволяє програмістові максимально швидко створювати комп'ютерні програми.

3.5 ЗАСОБИ ВИБОРУ МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ

Вибір моделі ЖЦ ПО відповідно до варіанту виконується відповідно до організаційного процесу ЖЦ ПЗ інформаційної системи, який регламентується СОУ-Н ДКА 0061:2012 та здійснюється шляхом виконання наступних дій:

– аналізування проекту відповідно до таблиць 1.1 – 1.4:

1) вимоги до ПЗ – таблиця 3.10;
2) команда розробників – таблиця 3.11;
3) колектив користувачів – таблиця 3.12;
4) тип проекту та ризики – таблиця 3.13;

– підготовка відповідей на питання, наведені для кожної категорії;

– ранжирування за ступенем важливості категорій, що відносяться до кожної категорії, щодо проекту, для якого вибирають модель ЖЦ;

– усунення протиріч, які виникають під час порівняння моделей, якщо загально отримані показники подібні або однакові.

Таблиця 3.10 містить питання щодо вимог, які пред'являє користувач до проекту. У термінології їх іноді називають властивостями системи, які будуть підтримуватися цим проектом.

По можливості, до складу команди розробників найкраще відібрати персонал ще до того, як буде обрано модель життєвого циклу. Характеристики такої команди (таблиця 3.11) відіграють важливу роль у процесі вибору, оскільки команда відповідає за вдале виконання ЖЦ і може надати допомогу в процесі вибору.

Таблиця 3.10 – Результати опитів щодо вимог до програмного забезпечення по моделях життєвого циклу програмного забезпечення

Питання користувача щодо вимог до ПЗ	Відповіді щодо вимог до ПЗ по моделях ЖЦ					
	кас-кадна	V-подібна	прототи-пування	RAD	інкре-ментна	спі-ральна
Чи легко визначити вимоги і/або вони добре відомі?	Так	Так	Ні	Так	Ні	Ні
Чи можна вимоги заздалегідь визначити?	Так	Так	Ні	Так	Так	Ні
Чи часто будуть змінюватися вимоги в життєвому циклі ПЗ?	Ні	Ні	Так	Ні	Ні	Так
Чи потрібно демонструвати вимоги з метою визначення?	Ні	Ні	Так	Так	Ні	Так
Чи потрібне для демонстрації можливостей перевіряння концепції?	Ні	Ні	Так	Так	Ні	Так
Чи будуть вимоги відображати складність системи?	Ні	Ні	Так	Ні	Так	Так
Чи має вимога функціональні властивості на ранньому етапі?	Ні	Ні	Так	Так	Так	Так

На початкових фазах проекту можна одержати чітке подання про колектив користувачів (таблиця 3.13) і його майбутній взаємозв'язок з командою розробників протягом усього проекту. Таке подання допоможе під час вибору моделі ЖЦ ПЗ, оскільки деякі моделі вимагають посиленої участі користувачів у процесі розроблення та вивчення проекту.

Таблиця 3.11 – Результати опитів щодо вимог до програмного забезпечення по моделях життєвого циклу програмного забезпечення

Питання до команд розробників проекту	Відповіді на питання розробників щодо моделі ЖЦ					
	каскадна	V-подібна	прототи-пування	RAD	інкре-ментна	спіральна
Чи є проблеми предметної галузі проекту новими для більшості розробників?	Ні	Ні	Так	Ні	Ні	Так
Чи є технологія предметної галузі проекту новою для більшості розробників?	Так	Так	Ні	Ні	Так	Так
Чи є інструменти, які використовують у проекті, новими для більшості розробників?	Так	Так	Ні	Ні	Ні	Так
Чи змінюються ролі учасників проекту під час життєвого циклу?	Ні	Ні	Так	Ні	Так	Так
Чи можуть розробники проекту пройти навчання?	Ні	Так	Ні	Так	Так	Ні
Чи є структура більш значуща для розробників, ніж гнучкість?	Так	Так	Ні	Ні	Так	Ні
Чи буде менеджер проекту чітко відслідковувати прогрес команди?	Так	Так	Ні	Ні	Так	Так

Питання до команд розробників проекту	Відповіді на питання розробників щодо моделі ЖЦ					
	каскадна	V-подібна	прототипування	RAD	інкрементна	спіральна
Чи важлива легкість розподілу ресурсів?	Так	Так	Ні	Так	Так	Ні
Чи прийме команда огляди та інспекції, менеджмент/огляди замовників?	Так	Так	Так	Ні	Так	Так

Таблиця 3.12 – Результати опитів щодо вимог до програмного забезпечення по моделях життєвого циклу програмного забезпечення

Питання до користувачів	Відповіді на питання користувачів					
	каскадна	V-подібна	прототипування	RAD	інкрементна	спіральна
Чи буде присутність користувачів обмежена в ЖЦ?	Так	Так	Ні	Ні	Так	Так
Чи будуть користувачі ознайомлені визначеннями ПЗ?	Ні	Ні	Так	Ні	Так	Так
Чи будуть користувачі ознайомлені із проблемами предметної галузі?	Ні	Ні	Так	Так	Так	Ні
Чи буде залучено користувачів у всі фази життєвого циклу?	Ні	Ні	Так	Так	Ні	Ні
Чи буде замовник відслідковувати хід виконання проекту?	Ні	Ні	Так	Ні	Ні	Так

У таблиці 3.13 уточнено тип проекту та ризики, що було розглянуто як елементи, визначення яких здійснюють на фазі планування. У деяких моделях ЖЦ ПЗ передбачено менеджмент ризиків високого ступеня, у той час як в інших його не передбачено взагалі. Вибір моделі, яка робить можливим менеджмент ризиків, не означає, що не потрібно встановлювати план дій, спрямований на мінімізування виявлених ризиків. Така модель забезпечує схему, у рамках якої обговорюється та виконується план певних дій.

По закінченню попереднього вибору моделі ЖЦ ПЗ на підставі таблиць 3.10 - 3.13 адаптують моделі відповідно до потреб проекту. Це означає, що обрані реальні фази та дії повинні допомогти керівникові проекту співвіднести проект із обраною моделлю. Після завершення адаптації модель здобуває більший ступінь значущості для команди розробників та колективу користувачів. Її можна використовувати як опорну точку під час оглядів стану процесу розроблення, оцінювання ризиків і поставки кінцевого продукту.

Стадії процесу вибору життєвого циклу розроблення ПЗ та його наступної адаптації визначають в такій послідовності:

– ознайомлення з різними моделями ЖЦ ПЗ;

– аналізування можливих видів робіт: розроблення, модернізація, супровід тощо;

– вибір найбільш підходящої моделі ЖЦ із використовуванням матриць критеріїв: високий ступінь ризику, користувальницький інтерфейс, висока надійність, час поставки на ринок/випуску продукту, пріоритети користувача, уточнення вимог, очікуваний строк експлуатації системи, технологія, розмір і складність, можливий паралелізм, а також інтерфейси для існуючих і нових систем;

– вибір фаз та дій для вибраної моделі ЖЦ ПЗ;

– визначення дій щодо оглядів, інспектування, верифікації та валідації, а також визначення контрольних точок проекту;

– оцінювання ефективності моделі ЖЦ ПЗ та її модифікації.

Таблиця 3.13 – Результати опитів щодо вимог до програмного забезпечення по моделях життєвого циклу програмного забезпечення

Тип проекту й ризики	Модель життєвого циклу					
	каскадна	V-подібна	прототипу-вання	RAD	інкремен-тна	спірал ь-на
Чи буде проект ідентифікувати новий напрямок продукту для організації?	Ні	Ні	Так	Ні	Так	Так
Чи буде проект мати тип системної інтеграції?	Ні	Так	Так	Так	Так	Так
Чи буде проект розширенням існуючої системи?	Ні	Так	Ні	Так	Так	Ні
Чи буде фінансування проекту стабільним протягом всього життєвого циклу?	Так	Так	Так	Так	Ні	Ні
Чи очікується тривала експлуатація продукту в організації?	Так	Так	Ні	Ні	Так	Так
Чи потрібен високий ступінь надійності?	Ні	Так	Ні	Ні	Так	Так

Тип проекту й ризики	Модель життєвого циклу					
	каскадна	V-подібна	прототипу-вання	RAD	інкреме-н-тна	спірал ь-на
Чи буде система змінюватися, можливо, із застосовуванням непередбачених методів, на етапі супроводу?	Ні	Ні	Так	Ні	Так	Так
Чи є графік обмеженим?	Ні	Ні	Так	Так	Так	Так
Чи є «прозорими» інтерфейсні модулі?	Так	Так	Ні	Ні	Так	Ні
Чи є доступними компоненти, які використовують повторно?	Ні	Ні	Так	Так	Ні	Так
Чи є достатніми ресурси (час, кошти, інструменти, персонал)?	Ні	Ні	Так	Ні	Ні	Так

Отже, ***вибір моделі ЖЦ*** залежить від специфіки, масштабу, складності програмного проекту й набору вимог до програмного забезпечення.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть три основні групи процесів життєвого циклу.
2. Назвіть організаційні і допоміжні процеси ЖЦ.
3. Охарактеризуйте процес вибору типу життєвого циклу.
4. Який стандарт визначає перелік і зміст процесів життєвого циклу програмного забезпечення?
5. Чи всі процеси стандарту повинні бути застосовані в розробці програмного забезпечення?
6. Охарактеризуйте суть моделі життєвого циклу і основні види моделей.
7. Опишіть каскадну і спіральну моделі життєвого циклу програмного забезпечення.
8. Охарактеризуйте еволюційну модель моделі життєвого циклу програмного забезпечення.
9. Назвіть методології життєвого циклу програмного забезпечення.
10. Охарактеризуйте методології розробки моделі життєвого циклу програмного забезпечення?
11. Охарактеризуйте засоби адаптації моделей моделі життєвого циклу програмного забезпечення до потреб проекту.
12. Назвіть основні типи моделей життєвого циклу програмного забезпечення та їх відмінності.
13. Які існують моделі ЖЦ ПЗ?
14. Які основні принципи побудови ПЗ при використанні ітераційної моделі ЖЦ ПЗ?
15. Які основні принципи побудови ПЗ при використанні каскадної та V-подібної моделі ЖЦ ПЗ?
16. Які основні особливості розробки ПЗ при використанні циклічної моделі ЖЦ ПЗ?
17. Використання "прототипу" при проектуванні ПЗ?

РОЗДІЛ 4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

При вивченні цієї теми важливо пам’ятати, що у загальному випадку під вимогами до програмного забезпечення розуміють властивості, які повинно мати програмне забезпечення для виконання покладених на нього функцій всіма зацікавленими особами. Вимоги до програмного забезпечення можуть поділятися на високорівневі твердження про функціональні можливості та вимоги, щодо обмеження залежності від системи або мати деталізований формальний математичний опис всіх функцій. В залежності від життєвого циклу розробки програмного забезпечення вимоги поділяються на групи (типи, категорії) вимог (системні, програмні, функціональні, не функціональні).

Зміст розділу

4.1 ПОНЯТТЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Успішне функціонування програмного забезпечення багато в чому залежить від правильної організації процесу виконання робіт з визначення та аналізу вимог до нього. Як було зазначено в розділі 2.1, вимоги виражають потреби людей (замовників, користувачів, розробників), зацікавлених у його створенні. Відповідно до стандарту 610.12-1990 (IEEE Standard Glossary of Software Engineering Terminology), ***вимоги*** це:

а) умова або здатність, необхідна користувачеві, щоб вирішити проблему або досягти поставленої мети;

б) умова або здатність, якими повинно володіти ПЗ або його компонент для задоволення умов контракту, стандарту, специфікації або іншого формально встановленого документа;

в) задокументоване уявлення умови або здатності, що зазначена у визначенні (а) або (б).

У відповідності до Rational Unified Process (RUP) вимоги до програмного забезпечення визначаються як умови або можливості, яким повинна задовольняти програмна система.

Основні характеристики, яким повинні відповідати вимоги до програмного забезпечення:

– процес розробки вимог є ітеративним;

– виявлені вимоги ведуть до вибору конкретних варіантів моделей життєвого циклу програмного забезпечення, які в свою чергу призводять до виникнення нових вимог при реалізації чи експлуатації програмного забезпечення;

– повний набір вимог можна отримати визначивши функції і атрибути ПЗ, а також атрибути зовнішніх впливів;

– у вимоги не слід включати загальну інформацію (графіки, плани розробки, виділені кошти, тести), а також інформацію, що стосується проектування;

– обмеження проектування стосується тільки варіантів

проектування системи або виконання процесів, які призначені для розробки програмної системи.

Отже, ***вимога до програмної системи*** - це результат аналізу побажань всіх зацікавлених осіб в програмному забезпеченні.

4.2 КЛАСИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до програмного забезпечення включають умови користувачів щодо функціональності та зовнішнього інтерфейсу програмного забезпечення і погляди розробників на деякі параметри програмного забезпечення. При цьому термін користувач стосується осіб, зацікавлених у створенні системи.

Для класифікації вимог на різні рівні, використовується їх поділення на вимоги, що призначені для користувача (user requirements) та вимоги, що призначені для детального опису виконуваних системою функцій – високорівневі узагальнюючі вимог і системні вимоги (system requirements). Крім вимог цих двох рівнів, застосовується ще більш деталізований опис вимог до програмного забезпечення – проектна системна специфікація (software design specification), яка слугує мостом між етапом розробки вимог і етапом проектування програмного забезпечення.

Тому вимоги до програмного забезпечення класично поділяються на користувацькі, системні та проектні (рис.4.1).

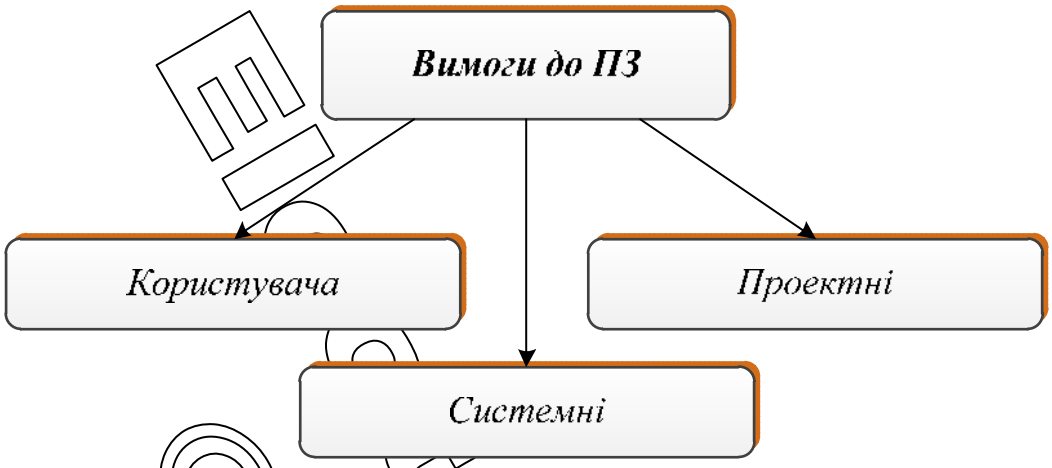


Рис. 4.1 – Класифікація вимог до програмного забезпечення

Наведені на рис. 4.1. вимоги можна визначити наступним чином:

1. Вимоги користувача – вимоги, які мають опис функцій що виконуються системою і обмежень, що накладаються на неї природною мовою (можливо при використанні діаграм, графіків, математичних формул). Тобто, вимоги користувачів (user requirements) задаються умовами досягнення цілей і задач, віддзеркалюють вимоги споживачів до спектра розв'язуваних майбутньою системою задач. Під вимогою замовника розуміються потреби або очікування, які:

– встановлено, як необхідні для функціонування програмного забезпечення;

– передбачаються (маються на увазі, задаються неявно), як необхідні для функціонування програмного забезпечення;

– є обов'язковим для реалізації.

Наприклад: У бесіді замовник висловив сподівання, що звучить так: «Зробіть мені редактор листів, щоб я міг виділяти різні слова різним кольором». В даному випадку:

– встановлено: програма повинна вміти читати формат листів, редагувати листи і зберігати їх;

– малося на увазі: «природно, я хочу, щоб це був

вбудований в почтовий клієнт» редактор (в ідеалі повинно бути встановлено, а отже, сформульовано аналітиком явно);

– обов'язково: виділяти різні слова різним кольором.

Наведений приклад показує, що недостатньо просто зафіксувати побажання замовника так, як він його висловив.

2. Системні вимоги – деталізований опис системних функцій і обмежень, які ще називають функціональними. Вони використовуються для укладення контракту між замовником програмного забезпечення і його розробниками.

3. Проектна системна специфікація – узагальнений опис структури програмного забезпечення, яка є основою для більш детального проектування програмного забезпечення і його реалізації. Ця специфікація доповнює і деталізує специфікацію системних вимог.

Різниця між призначеними для користувача і системними вимогами показані в прикладі, представленому в табл. 4.1. В таблиці 4.1 показано, як призначені для користувача вимоги можуть бути перетворені в системні.

Вимоги до програмного забезпечення, також, часто, класифікуються за рівнями на функціональні, нефункціональні та вимоги до предметної області (рис. 4.2).

Наведені на рис. 4.2. вимоги можна визначити наступним чином:

1. Функціональні вимоги. Це перелік сервісів, які повинна виконувати система, причому повинне бути зазначене, як система реагує на ті або інші вхідні дані, як вона поводиться в певних ситуаціях і т.д. У деяких випадках вказується, що система не повинна робити.

2. Нефункціональні вимоги. Описують характеристики системи і її оточення, а не поводження системи. Тут також може бути наведений перелік обмежень, що накладаються на

дії й функції, виконувані системою. Вони включають тимчасові обмеження, обмеження на процес розробки системи, стандарти й т.д.

Таблиця 4.1. Вимоги користувача і системні вимоги

Вимоги користувача	***Системні вимоги***
ПЗ повинно надавати засіб доступу до зовнішніх файлів, створені в інших програмах.	Користувач повинен мати можливість визначати тип зовнішніх файлів.
	Для кожного типу зовнішнього файлу повинно бути відповідний засіб, що може бути використаний для цього типу файлів.
	Зовнішній файл кожного типу повинен бути представлений відповідною піктограмою на дисплеї користувача.
	Користувачеві має бути надана можливість самому визначати піктограму для кожного типу зовнішніх файлів.
	При виборі користувачем піктограми, що представляє зовнішній файл, до цього файлу має бути застосовано засіб, асоційований з зовнішніми файлами даного типу.

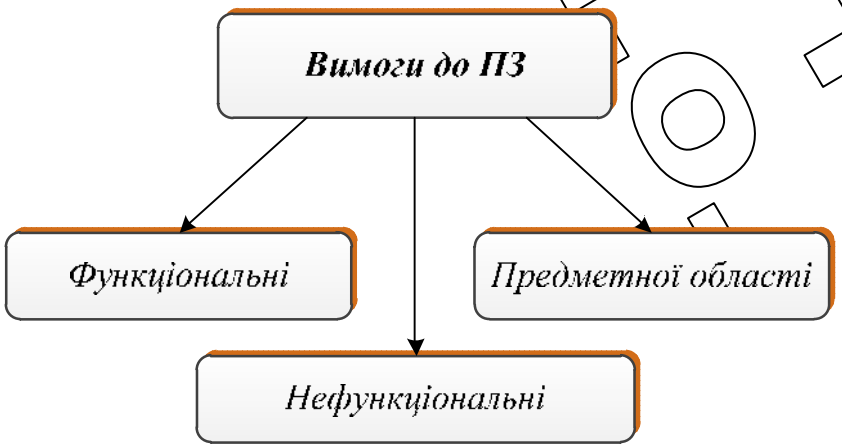


Рис. 4.2 – Класифікація вимог до програмного забезпечення

3. Вимоги предметної області. Характеризують ту предметну область, де буде експлуатуватися система. Ці вимоги можуть бути функціональними й нефункціональними.

Існують і інші класифікації вимог до програмного забезпечення. Так, наприклад, на рис 4.3. зображено процес визначення та класифікація вимог у відповідності до методології RUP.

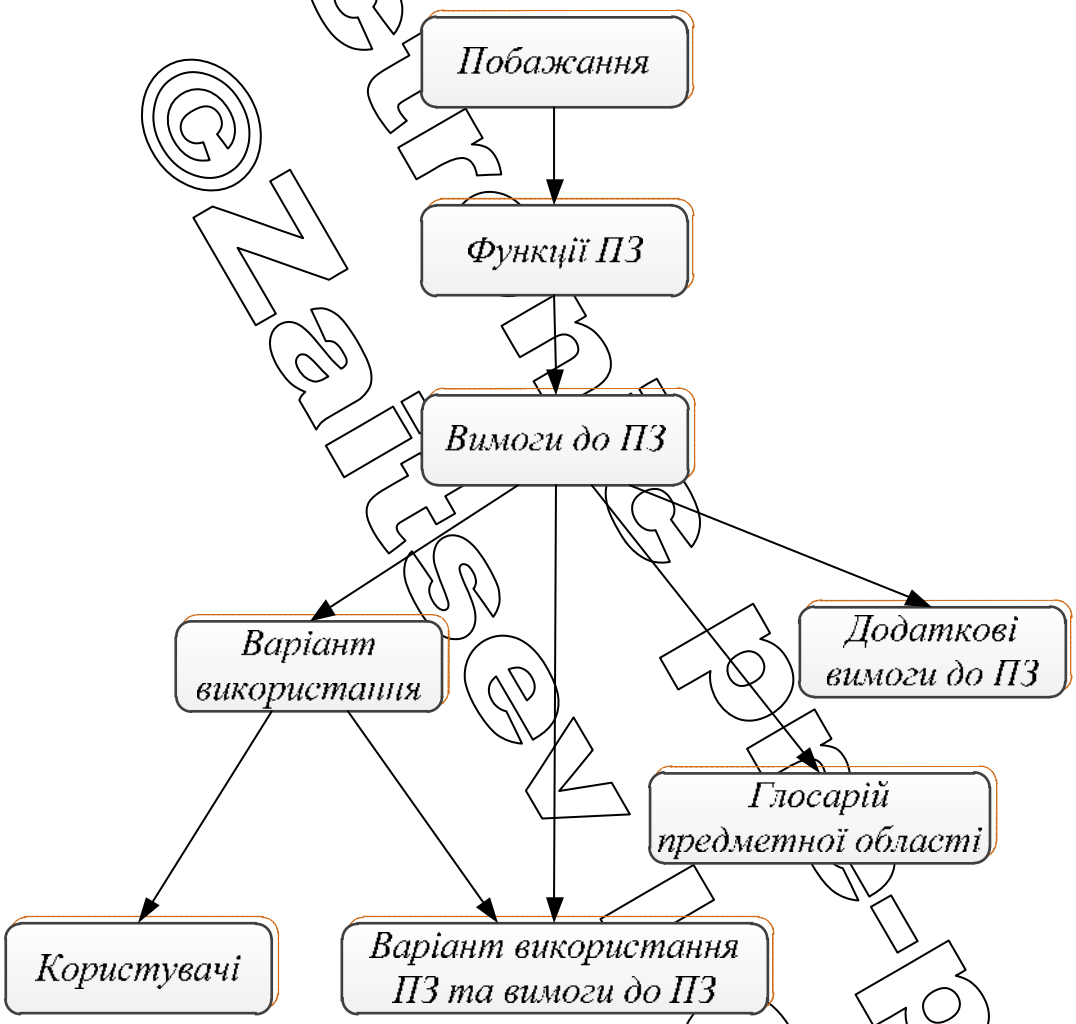


Рис. 4.3 – Класифікація вимог до програмного забезпечення за методологією RUP

У відповідності до К. Вігерса, класифікація рівнів вимог виглядає іншим чином (рис. 4.4).

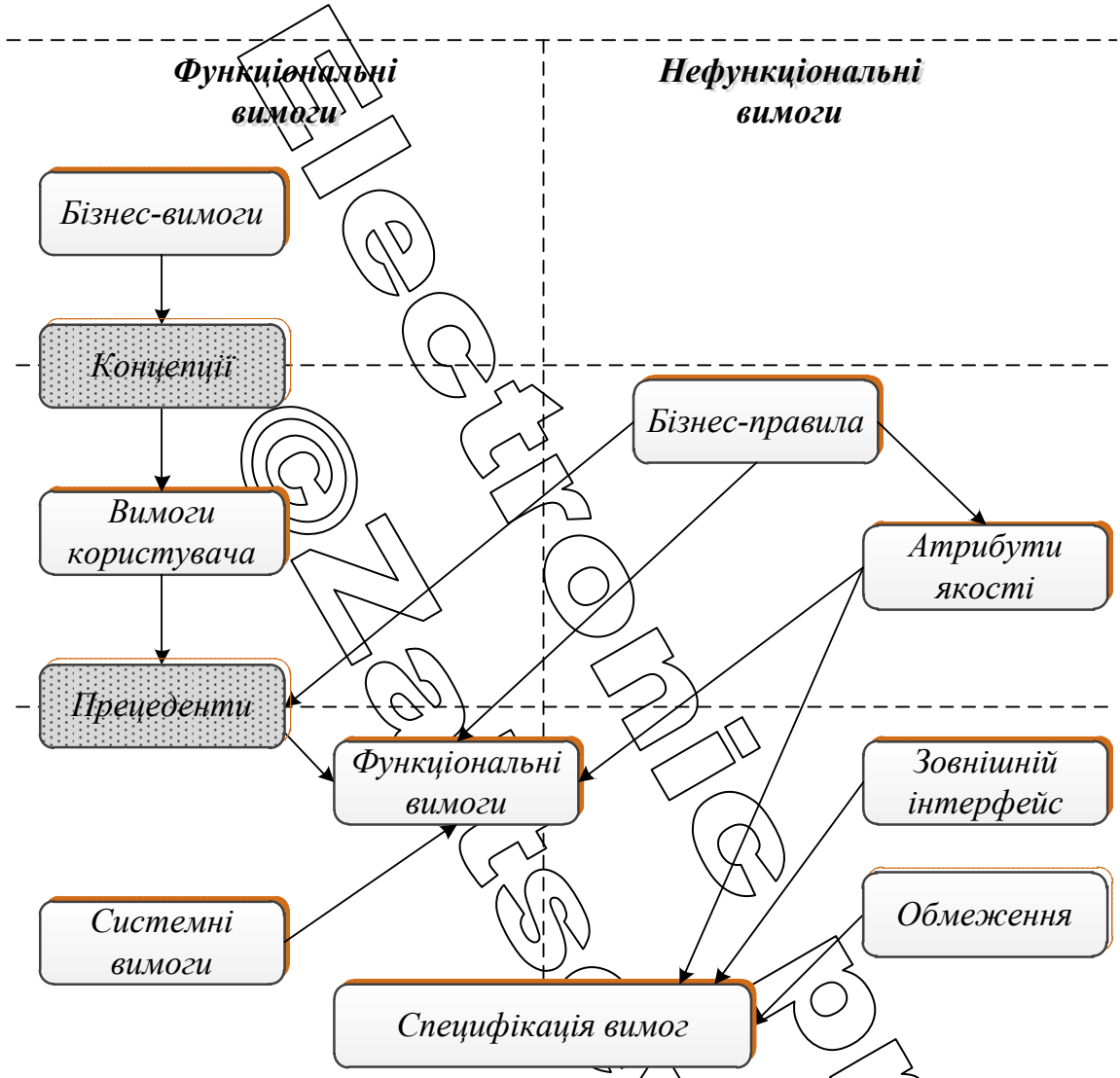


Рис. 4.4 – Класифікація вимог до програмного забезпечення за К. Вігерсом

Відповідно до класифікації рівнів вимог К. Вігерса вимоги до програмного забезпечення складаються з 3-х рівнів(рис.4.4):

– бізнес-вимоги;

– вимоги користувачів;

– функціональні вимоги.

Модель представлення вимог К. Вігерса, приведена на рис. 4.4, схематично показує організацію вимог, в якій світлий прямокутник – визначає тип інформації для вимог, темний прямокутник спосіб зберігання інформації (документи, діаграми, бази даних).

Відповідно до моделі К. Вігерса вимоги за типом інформації поділяються на:

1. *Бізнес-вимоги* (business requirements) містять високорівневі цілі організації або замовників ПЗ.Як правило, їх висловлюють, ті хто фінансує проект, покупці системи, менеджери користувачів. В цьому документі пояснено для чого необхідна така система, тобто описано цілі які організація має намір досягнути. Часто бізнес вимоги записують в формі документа про образ і межі проекту, який називають уставом проекту (project charter) або документом ринкових вимог (market requirement document). Визначення меж проекту є першим етапом управління загальними проблемами «розповзання» меж.
2. *Вимоги користувачів* (user requirements) описують цілі і задачі, які дозволить користувачу вирішувати дана система. До способів представлення таких вимог відносяться варіанти використання, сценарії і таблиці «подія-відгук». В цьому документі вказано, що клієнти можуть робити з допомогою системи.
3. *Функціональні вимоги* (functional requirements) визначають функціональність ПЗ, яку розробники повинні забезпечити, щоб користувачі змогли виконати свої завдання в межах бізнес-вимог.
4. *Бізнес-правила (business rules)* включають корпоративну політику, постанови управління, промислові стандарти і чисельні алгоритми. Бізнес-правила не є вимогами до ПЗ, тому що вони знаходяться зовні меж будь-якої системи ПЗ. Однак вони часто накладають обмеження, визначаючи хто може виконувати конкретні варіанти використання, або якими функціями повинна володіти система, яка підлягає відповідним правилам. Інколи бізнес-правила є

джерелом атрибутів якості, які реалізуються в функціональності.

5. *Атрибути якості* (quality attributes) є додатковим описом функцій продукту, опис характеристик важливих для користувачів або розробників. До таких характеристик відносяться легкість і простота використання, простота переміщення, цілісність, ефективність і стійкість до збоїв. Інші не функціональні вимоги описують зовнішні взаємодії системи і середовища, а також обмеження дизайну і реалізації.
6. *Обмеження(constraints)* стосуються вибору можливості розробки зовнішнього вигляду і структури продукту.
7. *Характеристика (feature)* – це набір логічно зв'язаних функціональних вимог, які забезпечують можливості для користувача і відповідають бізнес-цілям. В області комерційного ПЗ характеристика є групою вимог, яку вирізняють всі зацікавленні особи, які важливі при прийняті рішення про покупку. Характеристики продукту, які перераховує клієнт, не еквівалентні тим які необхідні для вирішення задач користувача.

Хоча в моделі класифікації вимог, що наведено на рис. 4.4 потік процесів визначення та класифікації вимог показано в напрямку зверху вниз, проте, на практиці, слід очікувати і циклів, і ітерацій між бізнес-вимогами, вимогами користувачів та функціональними вимогами.

Іншою класифікацією вимог є класифікація D. Firesmith із Інституту програмної інженерії Карнегі-Меллон, Піттсбург, штат Пенсільванія, США, яка наведена на рис. 4.5.

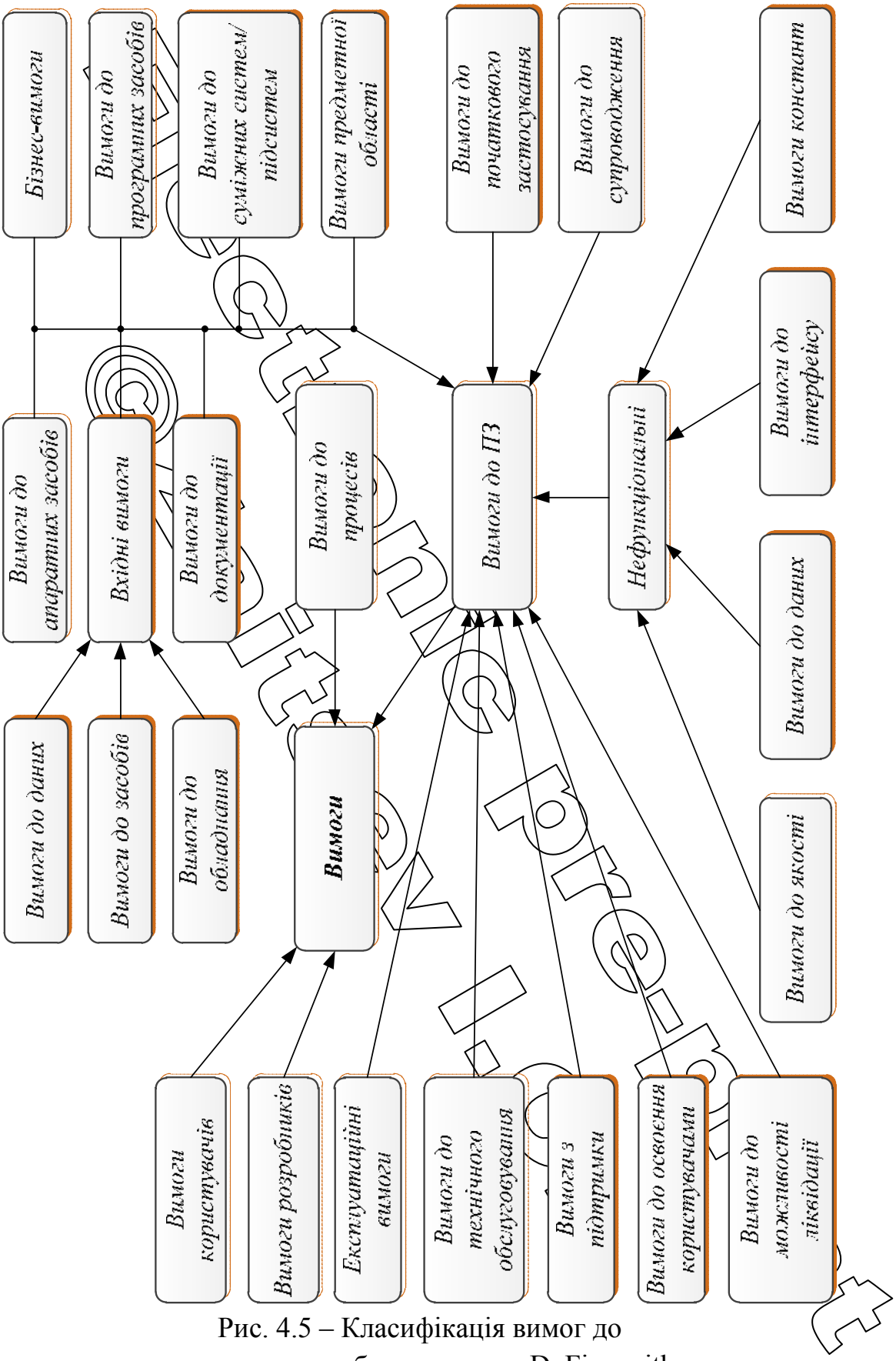


Рис. 4.5 – Класифікація вимог до програмного забезпечення за D. Firesmith

У дійсності чіткої границі між типами вимог та їх класифікації не існує. Наприклад, користувальницькі вимоги, що стосуються безпеки системи, можна віднести до нефункціональних. Однак при більш детальному розгляді така вимога можна віднести до функціональних, оскільки вони породжують необхідність включення в систему засобу авторизації захисту користувача. Тому, розглядаючи далі ці види вимог, необхідно завжди пам'ятати, що дані класифікації в значній мірі є штучними.

Отже, ***класифікація вимог*** за рівнями та типами відображає принципи взаємодії проектованої системи з іншими середовищами, платформами і загальносистемним забезпеченням.

4.3 ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги (Functional Requirements) – вимоги, які описують поводження системи й сервіси (функції), внутрішню роботу, поведінку, роботу з даними та інші специфічні функції, які мають виконуватися в програмному забезпеченні. Функціональні вимоги з'ясовуються на першій стадії процесу розроблення ПЗ, тобто на етапі аналізу вимог.

Функціональні вимоги складаються (рис 4.6):

– бізнес-вимог (Business Requirements) - визначають високорівневі цілі організації або клієнта (споживача) - замовника програмного забезпечення, що розробляється.

– вимог користувача (User Requirements) - описують цілі / завдання користувачів програмного забезпечення, які повинні досягатися / виконуватися користувачами за допомогою програмного забезпечення, що створюється.

– функціональні вимоги (Functional Requirements) –

визначають функціональність (поведінку) програмної системи, яка повинна бути створена розробниками для надання можливості виконання користувачами своїх обов'язків в рамках бізнес-вимог і в контексті призначених для користувача вимог.

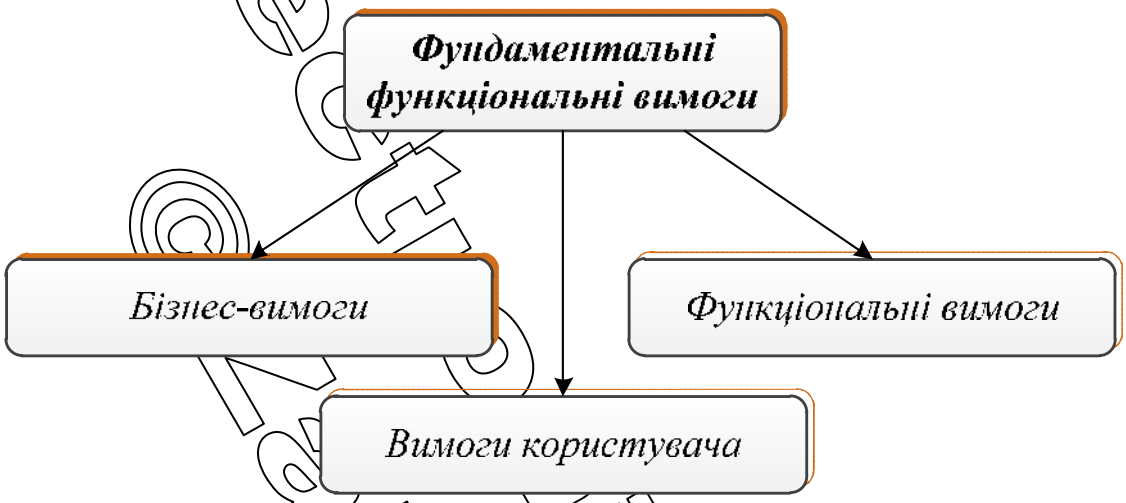


Рис. 4.6 – Класифікація функціональних вимог до програмного забезпечення

Функціональні вимоги описують сервіси, які надаються програмним середовищем, його поведінку в певних ситуаціях, реакцію на ті чи інші вхідні дані і дії які система дозволить виконувати користувачам.

Кожний програмний продукт призначений для виконання визначених функцій. Для того щоб визначити, підходить та чи інша програма для розв’язання задач, необхідно мати чіткий набір критеріїв, на основі яких можна зробити правильний вибір.

При написанні функціональних вимог необхідно враховувати те, що чим вони детальніші, тим більш точна оцінка робіт по строкам і вартості буде проведена перед розробкою технічного завдання на створення програмного забезпечення. Якщо на наступних етапах розробки ПЗ не виникає доповнень до початково сформульованих функціональних вимог, то оцінка буде достатньо точною.

Одночасно при описанні вимог не потрібно заглиблюватись в будь-які дрібні деталі. Необхідно описувати саме функції програми, а не те, яку кнопку треба натиснути, щоб отримати результат. Такі деталі повинні бути детально пророблені вже в процесі розробки технічного завдання.

Функціональні вимоги до програмного забезпечення можуть бути описані різними способами. Розглянемо для приклада функціональні вимоги до бібліотечної системи університету, призначеної для замовлення книг і документів з інших бібліотек.

1. Користувач повинен мати можливість проводити пошук необхідних йому книг і документів або по всій множині доступних каталожних баз даних або по певній їхній підмножині.

2. Система повинна надавати користувачеві підходящий засіб перегляду бібліотечних документів.

3. Кожне замовлення повинен бути постачений унікальним ідентифікатором (ORDER_ID), що копіюється у формуляр користувача для постійного зберігання.

Функціональні користувальницькі вимоги визначають властивості, якими повинна володіти система. Вони взяті з документа, що містить користувальницькі вимоги, і показують, що функціональні вимоги можуть бути описані з різним рівнем деталізації (порівняєте першу й третю вимоги). Так, наприклад, якщо функціональні вимоги оформлені як користувальницькі, вони, як правило, описують системи в узагальненому виді. На противагу цьому функціональні вимоги, оформлені як системні, описують систему максимально докладно, включаючи її вхідного й вихідного дані, виключення й т.д.

Специфікація функціональних вимог повинна бути комплексна й несуперечлива. Під комплексністю мається на увазі опис (визначення) всіх системних функцій. На практиці

для великих і складних систем украй важко розробити комплексну й несуперечливу специфікацію функціональних вимог. Причина криється частково в складності самої розроблювальної системи, а частково у неузгоджених опорних точках зору на те, що повинна робити система. Ця непогодженість може не виявитися на етапі первісного формулювання вимог - для її виявлення необхідний більше глибокий аналіз специфікації.

Функціональні вимоги документуються в специфікації вимог до програмного забезпечення, де описуються як можна більш повно очікувану поведінку системи.

Необхідно, щоб функціональна специфікація програмного засобу була математично точною. Бажано навіть, щоб при її розробці використовувались математичні методи і формалізовані мови. Вона повинна базуватись на чітких поняттях і твердженнях, що однозначно розуміються розробниками і замовниками програмного продукту.

Функціональна специфікація складається з трьох частин:

1. Описання зовнішнього інформаційного середовища, з якою буде взаємодіяти програмне забезпечення. Повинні бути визначені всі канали вводу і виводу і всі інформаційні об’єкти де застосовується дане програмне забезпечення, а також суттєві зв’язки між цими інформаційними об’єктами.
2. Визначення функцій програмного забезпечення, визначених на множині станів цього інформаційного середовища. Вводяться позначення всіх функцій, специфікуються їх вхідні дані і результати виконання, з вказуванням типів даних і завдань всіх обмежень, які повинні задовольняти ці дані і результати. Визначається вміст кожної з цих функцій.

3. Описання виключних ситуацій, якщо такі можуть виникнути при виконанні програм, і реакцій на ці ситуації, які повинні забезпечити відповідні програми. Повинні бути перераховані всі існуючі випадки, коли програмне забезпечення не зможе нормально виконати ту чи іншу свою функцію. Для кожного такого випадку повинна бути визначена реакція програми.

Отже, ***функціональні вимоги*** – це вимог, що встановлюють, що програмне забезпечення має робити.

4.4 НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Нефункціональні вимоги (англ. Non-Functional Requirements) – це:

– вимоги до програмного забезпечення, які задають критерії оцінювання якості його роботи;

– опис обмежень на функціонування системи, що відповідає на питання «в яких зовнішніх умовах повинно працювати програмне забезпечення?»;

– опис обмежень на функціонування програмного забезпечення, що відповідає на питання «в яких внутрішніх умовах повинно працювати програмне забезпечення?»;

– опис очікуваних якісних параметрів роботи програмного забезпечення, що відповідає на питання «які кількісні характеристики повинно мати програмне забезпечення?»;

– опис обмежень на реалізацію функцій програмного забезпечення, що відповідає на питання «як (з використанням чого) програмне забезпечення

повинно щось робити?»;

– вимоги до даних, з якими повинно працювати програмне забезпечення;
– вимоги до протоколів, з використанням яких повинно працювати програмне забезпечення;
– вимоги до інтерфейсів, за якими повинно працювати програмне забезпечення для взаємодіяти з іншими системами та користувачем;
– додаткові вимоги до функціональних вимог.

Нефункціональні вимоги не позв'язані безпосередньо з функціями, виконуваними системою. Вони пов'язані з такими інтеграційними властивостями системи, як надійність, час відповіді або розмір системи. Крім того, нефункціональні вимоги можуть визначати обмеження на систему, наприклад на пропускну здатність пристроїв введення-виведення, або формати даних, використовуваних у системному інтерфейсі.

Нефункціональні вимоги, як і функціональні з'ясовуються на етапі аналізу вимог до програмного забезпечення. За призначенням нефункціональні вимоги до програмного забезпечення спрямовані на покращення безпеки програмного забезпечення, його надійності, швидкодії, зручності у використанні тощо, а також на вдосконалення (масштабування, відновлюваність та інші) властивостей програмного забезпечення.

Нефункціональні вимоги відносяться до системи в цілому, а не до окремих її засобів. Це означає, що вони більш значимі й критичні, ніж окремі функціональні вимоги. Помилка, допущена у функціональній вимозі, може знизити якість системи, помилка в нефункціональних вимогах може зробити систему непрацездатною.

Нефункціональні вимоги вкрай критичні при реалізації системи. Їх виконання - запорука досягнення бажаної якості, продуктивності та задовільних швидкісних характеристик

програмного забезпечення.

Для початку виявлення нефункціональних вимог бажано:

1. Зробити огляд програмного забезпечення з точки зору майбутнього використання. Наприклад: на яких версіях операційних систем ПЗ може працювати? Які додаткові компоненти необхідні для стабільної роботи ПЗ?

2. Описати очікувані якісні параметри роботи програмного забезпечення, що відповідають на запитання «які кількісні характеристики повинна мати система?». Наприклад: Яка пропускна здатність цих каналів необхідна? З якою швидкістю повинна передаватися інформація? Чи є вимоги до захисту інформації? Який протокол передачі даних використовується?

Нефункціональні вимоги відображають користувальницькі потреби при цьому вони ґрунтуються на бюджетних обмеженнях, ураховують організаційні можливості компанії-розроблювача й можливість взаємодії розроблювальної системи з іншими програмними й обчислювальними системами, а також такі зовнішні фактори, як правила техніки безпеки, законодавство про захист інтелектуальної власності й т.п.

Базові типи нефункціональні вимог поділяються на (рис.4.7):

– *Вимоги до оточення* – вимоги, що визначають фізичну середу (природну або створену), в якій працюватиме програмне забезпечення. У деяких випадках, це також може відображати політичну чи економічну обстановку, в якій виконується робота або система буде функціонувати.

– *Фізичні вимоги* – вимоги, що визначають форму програмного забезпечення. Наприклад, зазначення розміру, форми, забарвлення, ваги або інших аналогічних властивостей програмного забезпечення або систем.

– *Вимоги до інтерфейсів* – вимоги, що визначають дані, структуру і фізичну форму інтерфейсів між компонентами (апаратними засобами, програмним забезпеченням і людьми). Тут також можуть зазначатися вимоги по взаємодії з існуючими системами або використанню деяких стандартних інтерфейсів. Деякі аналітики виділяють інтерфейсні вимоги в окрему групу, не включаючи їх у нефункціональні вимоги.

– *Вимоги по обмеженню* – вимоги, які веліли умови або обмеження на те, як програмне забезпечення може бути побудовано або як і в якому контексті повинні застосовуватися інші вимоги. Нетехнічні аспекти, такі як терміни або бюджет можуть також обмежувати проекти з розробки програмного забезпечення.

– *Вимоги до факторів якості* – вимоги, які стосуються інших якісних чинників програмного продукту чи процесу.

На рис. 4.8 показана класифікація нефункціональних вимог за Соммервіллем.

Нефункціональні вимоги, показані на рис. 4.8, розділяються на три великі групи.

1. Вимоги до продукту. Описують експлуатаційні властивості програмного продукту. Сюди ставляться вимоги до продуктивності системи, обсягу необхідної пам'яті, надійності (визначає частоту можливих збоїв у системі), переносимості системи на різні комп'ютерні платформи й зручності експлуатації.

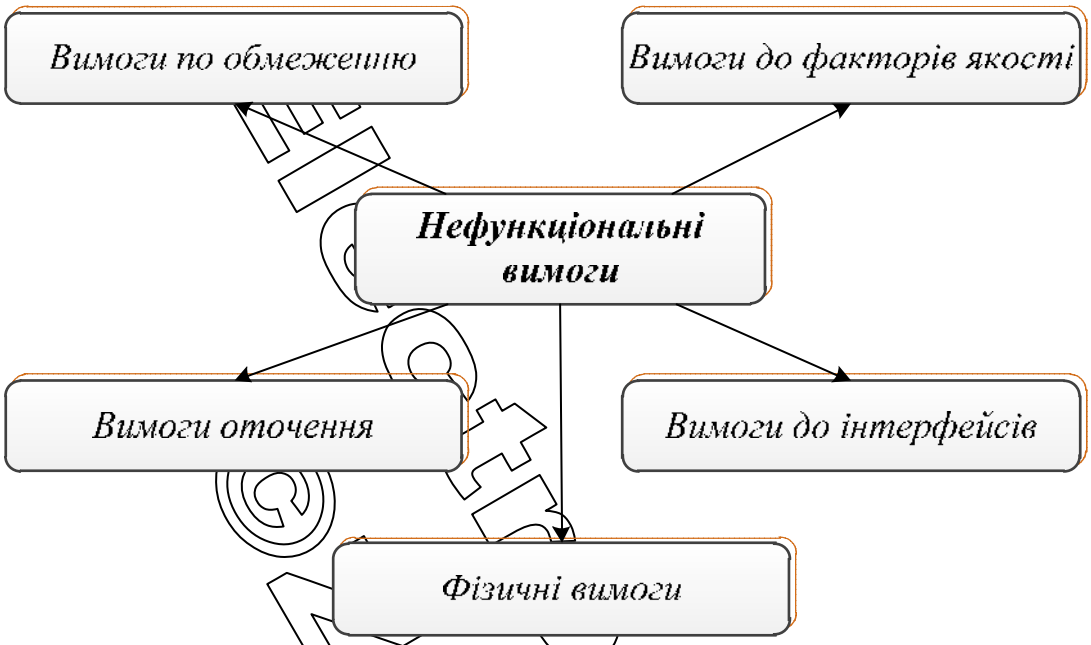


Рис. 4.7. Базові типи нефункціональних вимог

2. Організаційні вимоги. Відображають політику й організаційні процедури замовника й розроблювача ПЗ. Вони включають стандарти розробки програмного продукту, вимоги до реалізації ПЗ (тобто до мови програмування й методам проектування), вихідні вимоги, які визначають строки виготовлення програмного продукту, і супровідну документацію.

3. Зовнішні вимоги. Враховують фактори, зовнішні стосовно розроблювальної системи й процесу її розробки. Вони включають вимоги, що визначають взаємодію даної системи з іншими системами, юридичні вимоги, проходження яким гарантує, що система буде розроблятися й функціонувати в рамках існуючого законодавства, а також етичні вимоги. Останні повинні гарантувати, що система буде прийнятна для користувачів або замовника.

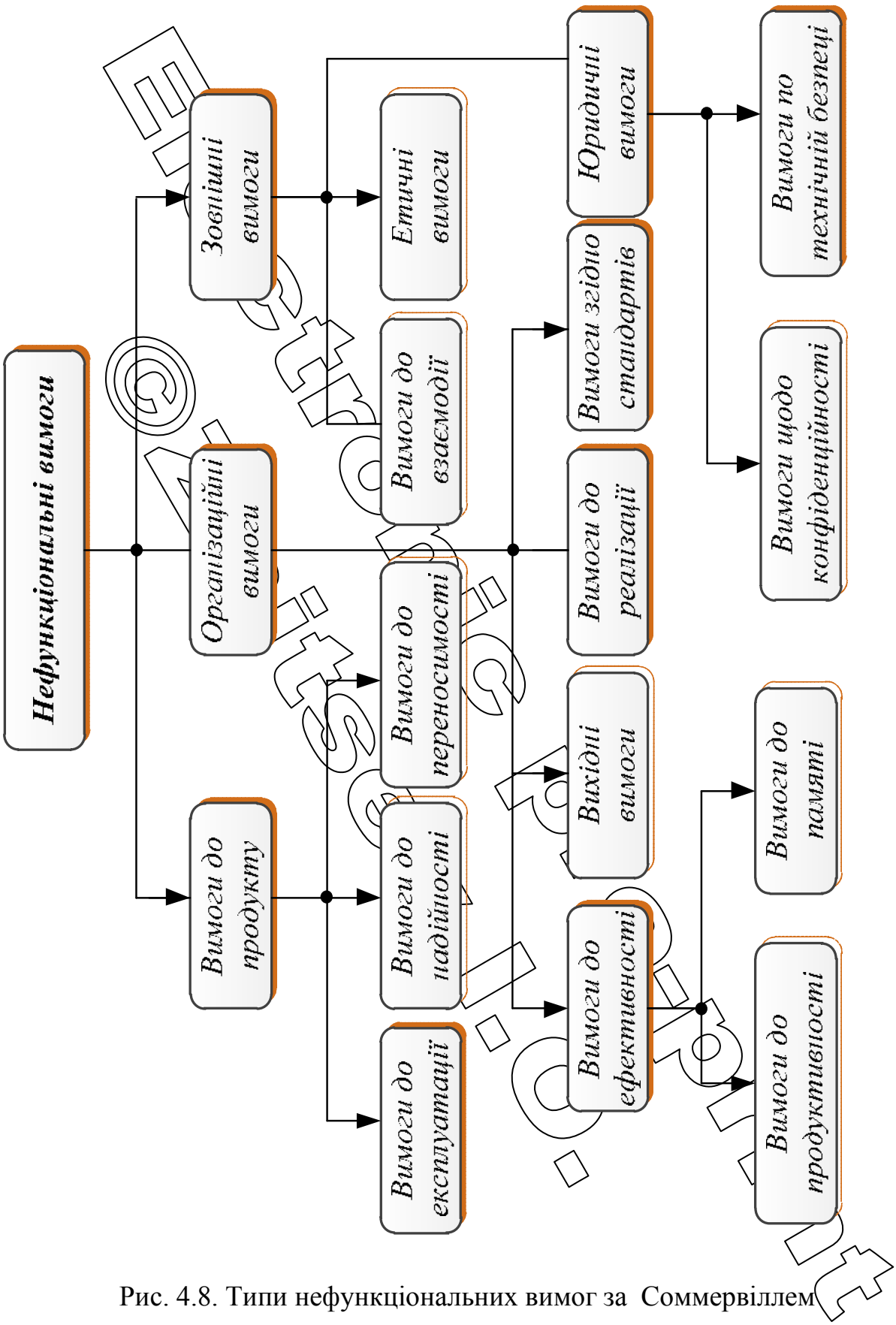


Рис. 4.8. Типи нефункціональних вимог за Соммервіллем

У відповідності до IEEE SWEBOK нефункціональні вимоги поділяються на наступні типи (рис. 4.9):

1. Бізнес-правила (Business Rules) – включають вимоги пов'язані з корпоративними регламентами, політиками, стандартами, законодавчими актами, внутрішньокорпоративними ініціативами, обліковими практиками, алгоритмами обчислень і т.д. Бізнес-правила часто визначають розподіл відповідальності в системі, відповідаючи на питання "хто буде здійснювати конкретний варіант, сценарій використання" або диктують появу деяких функціональних вимог.

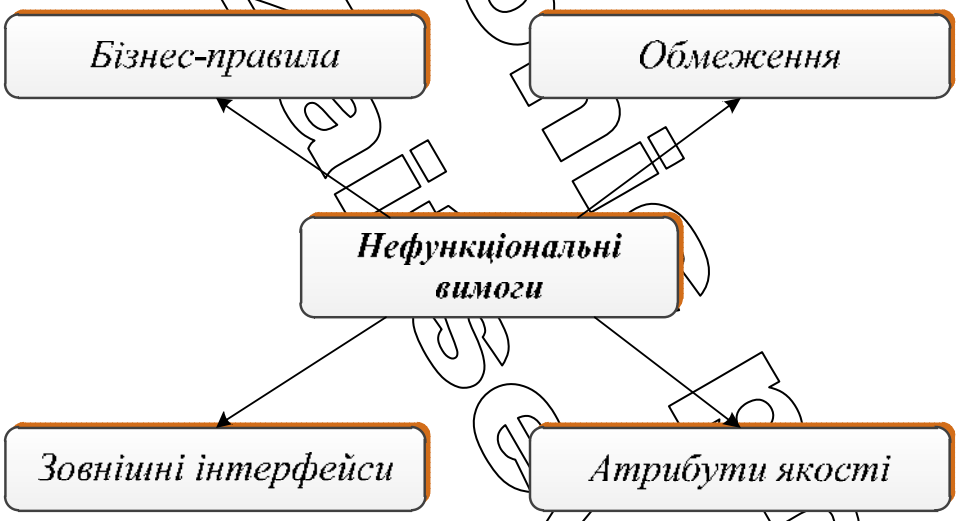


Рис. 4.9. Типи нефункціональних вимог за SWEBOK

2. Зовнішні інтерфейси (External Interfaces) – вимоги, які найчастіше розглядаються, як вимоги, що стосуються зовнішнього вигляду програми, концепції і дизайну графічного інтерфейсу системи. Основне джерело виявлення таких вимог - інтерв'ювання майбутніх користувачів системи. Важливо розуміти, що не всі зацікавлені особи будуть виступати як майбутні користувачі, і слід обмежувати їх в прагненні вплинути на дизайн користувальницького інтерфейсу. Слід чітко розуміти, що аналітик не здатний і не повинен розробляти кінцевий зовнішній вигляд інтерфейсів, його завдання - створити і узгодити з користувачами системи

прототип графічного інтерфейсу(GUI). Остаточний дизайн екранних форм виконує дизайнер графічних інтерфейсів.

3. Атрибути якості (Quality Attributes) - описують додаткові характеристики продукту в різних "вимірах", важливих для користувачів і/або розробників. Атрибути стосуються питань прозорості взаємодії з іншими системами, цілісності, стійкості та ін.

4. Обмеження (Constraints) - формулювання умов, що модифікують вимоги або набори вимог, звужуючи вибір можливих рішень по їх реалізації. Зокрема, до них можуть ставитися параметри продуктивності, що впливають на вибір платформи реалізації та / або розгортання (протоколи, сервери додатків, баз даних, ...), які, в свою чергу, можуть ставитися, наприклад, до зовнішніх інтерфейсів

Види нефункціональних вимог до програмного забезпечення:

1) Вимоги до інтерфейсу (Interface Requirements) поділяють на:

– апаратні інтерфейси (Hardware Interfaces) – необхідні для підтримки роботи програмно-апаратної системи, зокрема її логічної структури, фізичних адрес і очікуваної поведінки;
– інтерфейси ПЗ (Software Interfaces) – інтерфейси, з якими наявна аплікація має взаємодіяти з користувачем під час її використання;
– комунікаційні інтерфейси (Communications Interfaces) – інтерфейси для комунікацій (взаємодії) програмного забезпечення з іншими програмно-апаратними системами та/або пристроями.

2) Апаратні та програмні вимоги (Hardware/Software Requirements) – опис апаратної та програмної платформ, необхідних для надійної роботи (і підтримки) програмно-апаратної системи.

3) Операційні вимоги (Operational Requirements) відповідають за:

– безпеку та конфіденційність (Security and Privacy) даних у програмному забезпеченні;
– надійність (Reliability) роботи програмного забезпечення у штатних і позаштатних ситуаціях;
– відновлюваність (Recoverability) програмного забезпечення після аварійного завершення роботи;
– програмну продуктивність (Performance) програмного забезпечення у різних наборах вхідних даних;
– потенціал (місткість) (Capacity) програмного забезпечення стосовно обсягу оброблюваних даних;
– збереження даних (Data Retention) в оперативній пам'яті програмного забезпечення;
– управління помилками (Error Handling) в програмному забезпеченні, які виникають внаслідок його роботи, але не були виявлені під час тестування;
– правила перевірки (Validation Rules) функціональних можливостей програмне забезпечення;
– узгоджені стандарти (Convention Standards), згідно з якими було розроблено програмне забезпечення.

Основна проблема не функціональних вимог полягає в тому, що їх виконання важко перевірити. Часто вони пишуться для того, щоб відобразити загальні цілі замовника системи, такі, як простота експлуатації, можливість відновлення після збоїв або швидку відповідь на запити користувача. Реалізація подібних вимог може виявитися складним для системних розробників, оскільки вони нечітко сформульовані і відкривають простір для різних тлумачень.

При описі нефункціональних вимог природною мовою виникають різні проблеми, серед яких:

1. Відсутність чіткості викладу. Іноді нелегко викласти яку-небудь думку природною мовою чітко й недвозначно, не

зробивши при цьому текст багатослівним і важким для читання.

2. Змішання вимог. У користувальницьких вимогах відсутнє чіткий поділ на функціональні та нефункціональні вимоги, на системні цілі й проектну інформацію.

3. Об'єднання вимог. Кілька різних вимог до системи можуть описуватися як єдина користувальницька вимога.

В ідеалі нефункціональні вимоги повинні виражатися через кількісні показники, які можна об'єктивно виміряти. На практиці висловити нефункціональні вимоги за допомогою кількісних показників досить важко. Часто замовник програмного забезпечення не може оформити своє бачення майбутньої системи за допомогою вимог, виражених кількісними показниками. Або деякі системні вимоги, наприклад зручність супроводу, взагалі не можна виразити через кількісні показники. Крім того, витрати на об'єктивне вимір кількісних не фуикціональних вимог можуть виявитися вкрай високими. Тому часто документ, специфікує вимоги до системи, містить опис системних цілей спільно з чітко сформульованими вимогами. Ці цілі корисні, оскільки відображають уявлення (і пріоритети) замовника про майбутню систему.

Різниця між функціональними і нефуикціональними вимогами формується на базі розумінням функціональності програмного забезпечення розробником і аналітиком. Для розробника функціональність - це всі функції/методи будь-яких класів програмної системи, навіть якщо ці функції не приносять ніякого корисного результату. Для аналітика функції програмного забезпечення – варіанти використання.

Отже, ***нефункціональні вимоги*** встановлюють, що саме програмне забезпечення повинно робити.

4.5 ЗБИРАННЯ ТА АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Процес збирання вимог – це один з найбільш важливих етапів процесу створення будь-якого програмного забезпечення. Перед початком збирання вимог, необхідно виявити всіх зацікавлених осіб, які зацікавленні в створенні та використанні програмного забезпечення. Чим точніше буде цей список зацікавлених осіб, тим повніше будуть вимоги і тим якісніше буде реалізовано програмне забезпечення. Тому що вимоги відбивають потреби зацікавлених осіб (замовників, користувачів, розробників).

В процесі виявлення вимог замовник і розробник спільно обговорюють проблеми проекту із реалізації програмного забезпечення, проводять збирання вимоги, їх аналіз, при необхідності здійснюють перегляд вимог, а також визначають необхідні обмеження в залежності від обмежень предметної області та апаратних можливостей системи в якій буде використовуватися програмне забезпечення.

В обговоренні вимог до програмного забезпечення *беруть участь*:

– представники замовника з декількох професійних груп;

– оператори, що обслуговують систему;

– аналітики і розробники майбутньої системи.

Обговорення проекту програмного забезпечення відбувається з метою вивчення думки і вироблення перших висновків щодо доцільності реалізації програмного забезпечення, прогнозування реальності його виконання в заданий термін і за кошти, що надає замовник.

Обговорення проекту з розробки програмного забезпечення *проводиться для визначення*:

– спектра проблем, що буде вирішуватися в програмному забезпеченні;

– функціонального змісту програмного забезпечення;

– регламенту операційного обслуговування програмного забезпечення тощо.

Природно, особа, яка замовила проект системи, бажає отримати від розробника набір необхідних послуг, за якими будуть звертатися різні категорії користувачів: оператори, менеджери та інші. Розробники системи можуть оцінити можливість реалізації послуг у проекті із реалізації програмного забезпечення, що замовляється, у заданий термін і бюджет.

Погоджена сфера дій у програмному проекті дає можливість оцінити необхідні інвестиції в проект, заздалегідь визначити можливі ризики і здатності розробників щодо виконання проекту. Підсумком обговорення можливих реалізацій програмного проекту може бути рішення про розгортання робіт з реалізації програмного забезпечення або відмови реалізації.

У вимогах до програмного забезпечення, крім системних, формулюються реальні потреби замовника щодо функціональних, операційних і сервісних можливостей майбутнього програмного забезпечення та системи на якій програмне забезпечення може використовуватися. Результати дії із дослідження й аналізу предметної області фіксуються в документі з опису вимог(див. розділ 5) і в договорі між замовником і виконавцем проекту.

Первісними джерелами для виявлення та збирання вимог є:

– мета і задачі проекту, що формулює замовник майбутньої системи, і які повинні осмислюватися розробниками;

– колектив, який виконує реалізацію функцій системи.

Вивчення і фіксація реалізованих функціональних можливостей у діючому програмному забезпеченні є підґрунтям для накопичення досвіду для формулювання нових

вимог до неї або аналогічного за призначення програмного забезпечення. При цьому необхідно відокремлювати нові вимоги до програмного забезпечення від старих вимог, щоб не повторити невдалі реалізації щодо старої системи в новому її виконанні.

Вимоги до програмного забезпечення формулюються замовником у термінах понять його предметної області з урахуванням відомих словників, стандартів, існуючих умов середовища функціонування майбутньої системи, а також трудових і фінансових ресурсів, виділених на розробку системи, що треба враховувати розробнику на початку роботи над визначенням вимог до програмного забезпечення, їх аналізу та специфікації.

Методи збирання вимог до програмного забезпечення поділяють на такі:

– інтерв'ю із зацікавленими сторонами щодо виявлення їхніх вимог до майбутнього програмного забезпечення (проведення інтерв'ю із представниками від зацікавлених сторін є традиційним підходом, що використовується під час аналізу вимог до програмного забезпечення. Таке уточнення вимог слугує одним із шляхів отримання цілісного та достовірного знання, яке часто неможливо виявити в спільних сесіях із замовниками і розробниками програмного забезпечення, де їхня увага підпорядкована забезпеченню більш крос-функціонального контексту. Інтерв'ю із конкретними респондентами надає аналітику можливість зрозуміти більш детально предметну область використання майбутнього програмного забезпечення);

– вивчення та аналіз вдалих та невдалих варіантів виконання функцій, розділення ролей відповідальних осіб, які пропонують ці варіанти, або тих, що взаємодіють із апаратною частиною системи при її функціонуванні для раніше розроблюваного програмного забезпечення;

– спостереження за роботою діючого програмного забезпечення для виявлення його особливих властивостей.

Процес *аналізу вимог* починається після обговорення проблематики програмного проекту. Серед обговорюваних при аналізі вимог можуть виявитися:

– неочевидні або не однаково важливі вимоги, які були взяті з застарілих джерел і документів замовника;

– різні типи вимог, що відповідають різним рівням деталізації програмного проекту і вимагають застосування методів керування ними;

– постійно змінювані або уточнюванні вимоги, залежно від унікальних властивостей або значень функціональних можливостей, що повинні бути реалізовані в програмному забезпеченні;

– складні за формою і змістом вимоги, що вимагають складних процедур узгодження між замовником та розробником (наприклад із-за різного розуміння предметної області).

Для здійснення успішного аналізу вимог до програмного забезпечення розробники повинні мати відповідні знання в предметній області програмного забезпечення і вміти здійснювати:

– аналіз проблем, задач предметної області, а також потреб замовника і користувачів системи,

– виявлення функцій системи, що мають бути реалізовані в проекті,

– внесення змін в окремі елементи вимог у процесі їх виконання.

Отже, ***визначення та аналіз вимог*** – це процес за допомогою яких визначаються вимоги, щодо нового, чи зміненого продукту, враховуючи всі можливі конфліктні всіх зацікавлених осіб в розробці та використання програмного забезпечення.

4.6 ПОКАЗНИКИ ЯКОСТІ ВИМОГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В процесі роботи з функціональними і нефункціональними вимогами для забезпечення якості вимог до них застосовуються показники якості програмного забезпечення – тобто опис тих якостей, яким повинні задовольняти якісні вимоги.

Показники якості вимог програмного забезпечення – сукупність властивостей програмного забезпечення, що встановлюють його спроможність задовольнити вимоги (запити) замовника, які він висловив у вигляді користувацьких вимог на початкових етапах розроблення ПЗ.

Відповідно із міжнародними та вітчизняними стандартами(див. Додаток А) оцінювання рівня якості вимог до програмного забезпечення, розрізняють два процеси забезпечення якості програмного забезпечення впродовж його життєвого циклу:

1) гарантія якості програмного забезпечення, що є результатом певних дій на кожній стадії життєвого циклу реалізації програмного забезпечення з перевірки й підтвердження відповідності його стандартам і процедурам, орієнтованим на досягнення певних показників якості програмного забезпечення, й програмного продукту в цілому;
2) інженерія якості програмного забезпечення як процес надання програмному забезпеченню, яке потрібно розробити або яке знаходиться у процесі розроблення якісних характеристик із надійності, супроводу та тощо.

Ці процеси забезпечення якості вимог до програмного забезпечення потребують:

– оцінювання стандартів і процедур, що

використовуються під час розроблення програмного забезпечення;

– ревізії процесів управління, розроблення та забезпечення якості програмного забезпечення, а також усієї проектної документації (звітів, графіків розроблення, повідомлень);
– контролю за процесом проведення формальних інспекцій та оглядів певних дій на кожному етапі реалізації програмного проекту;
– аналізу й контролю за процесом проведення тестування (випробування) ПЗ.

Основні показники якості програмного забезпечення відповідно до процесів (рис.4.10) – це повнота, однозначність, коректність, узгодженість, можливість верифікації, необхідність, корисність, здійсненність, можливість модифікації, можливість перевірки, впорядкованість за важливістю і стабільності, наявність кількісної метрики.

Характеристики якості вимог до програмного забезпечення приводяться в стандартах та директивах IEEE Standard Glossary of Software Engineering Terminology IEEE Std 610.12-1990 та IEEE Standard 830-1998 "IEEE Recommended Practice for Software Requirements Specifications" (рис. 4.10).

Розглянемо характеристики докладніше.

1. Повнота (окремої вимоги і системи вимог до програмного забезпечення) означає, що вимога повинна містити всю необхідну інформацію для її реалізації. Відповідно до цієї характеристики у вимоги включається вся інформація про описуваний параметр, відома на момент виявлення та збирання вимог на будь-якому із етапів життєвого циклі програмного забезпечення. Також, система вимог також не повинна містити невиявлених і не визначених вимог. Тому, причини неповноти опису слід явно оголошувати.

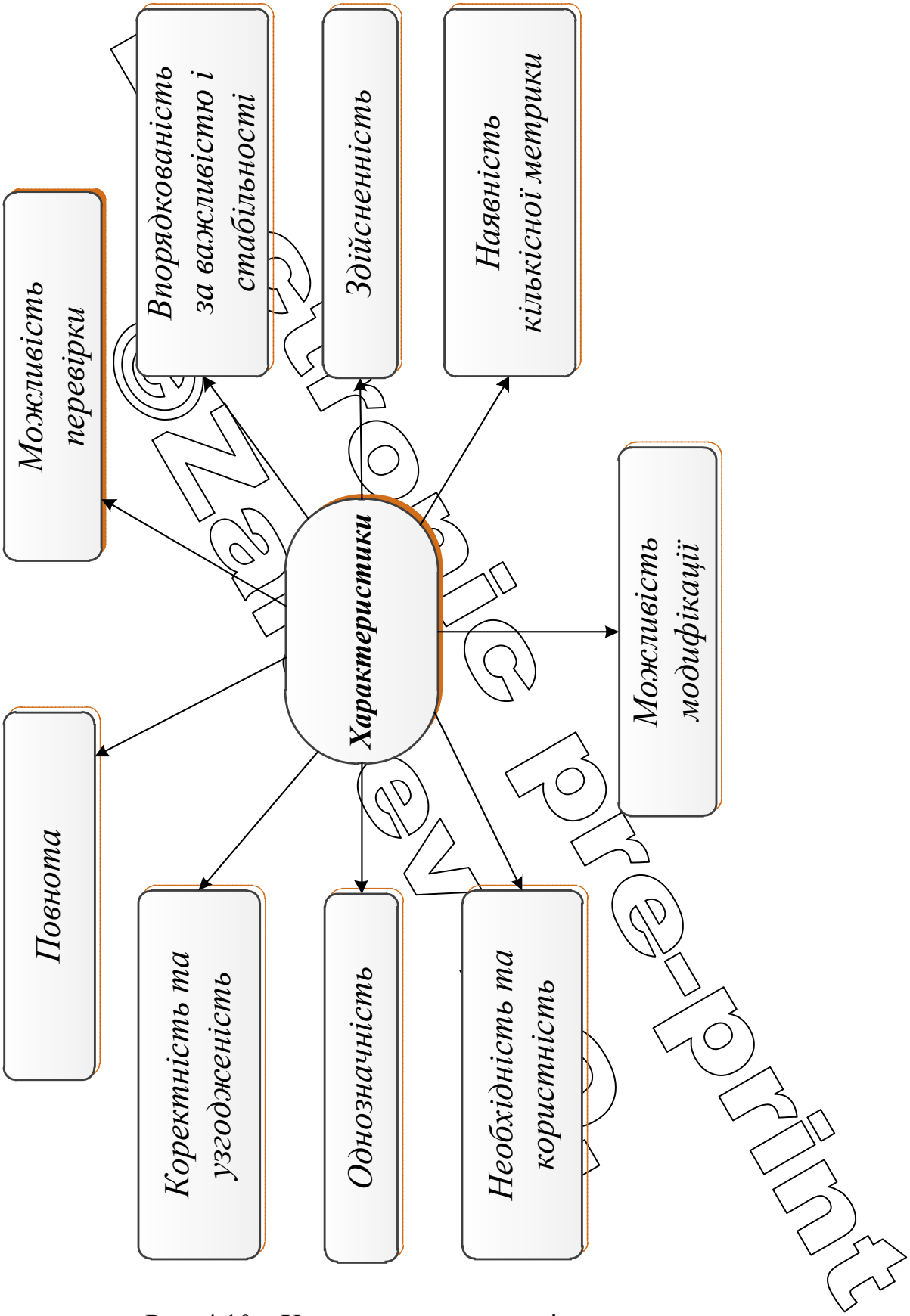


Рис. 4.10 – Характеристики якості вимог

Як відомо з теорії штучного інтелекту, неповнота - одна з фундаментальних властивостей людського знання. При створенні програмних систем доводиться мати справу з характеристиками ще неіснуючої системи, тобто з ще неіснуючою інформацією, що призводить в майбутньому до появи ризиків. Тому, ідея про те, що необхідно сформулювати всі вимоги повністю, тобто вичерпним чином, до початку проектування, а тим більше - реалізації програмного забезпечення є утопічною, і пов'язана з використанням класичної каскадної моделі життєвого циклу, який підтримував послідовну модель реалізації програмного забезпечення. Спіральна модель життєвого циклу, на якому базується більшість сучасних методологій, передбачає поетапне виділення і деталізацію вимог на всьому протязі циклу розробки програмного забезпечення.

Проте, вимога повноти пред'являється до вимог, що формулюються до програмного забезпечення.

Повнота окремого вимоги - властивість, що означає, що текст вимоги не вимагає додаткової деталізації, тобто в ньому передбачені всі необхідні нюанси, особливості та деталі даної вимоги.

Повнота системи вимог - властивість, що означає, що сукупність артефактів, що описують вимоги, вичерпним чином описує все те, що повинно бути реалізовано в програмному забезпеченні.

2. Однозначність (недвозначність, визначеність, ясність) означає, що вимога повинна бути внутрішньо несуперечливо і всі, хто працює з нею повинні розуміти її однаково. Вимоги слід висловлювати просто, стисло і точно, використовуючи відомі терміни. Зазвичай базові знання читачів специфікації вимог до програмного забезпечення розрізняються. Тому в її склад включають розділ з визначенням понять прикладної області, що використовуються при визначенні вимог. На

практиці ясність вимог досягається в тому числі і в процесі консультацій, в ході яких відбувається змісту понятті вимог усіма учасниками розроблення вимог до програмного забезпечення. Гарною підмогою в цьому служить узгоджений сторонами глосарій ключових понять предметної області. Так, наприклад, К.Вігерс дає наступну пораду щодо підвищення ясності документів: "Пишіть документацію просто, коротко і точно, застосовуючи лексику, зрозумілу користувачам". Ще однією стороною поняття "ясність вимоги" є простежуваність. Вимога, яке сформульована ясно, може бути простежено, починаючи від того документа, де вона сформульована вперше, аж до робочих специфікацій програмного забезпечення.

3. Коректність і узгодженість (несуперечність) означає, що вимоги не повинні містити в собі невірної, неточної інформації, а окремі вимоги в системі вимог не повинні суперечити одна одній. Коректність - одне з найважливіших властивостей вимог. Так, К. Вігерс в вводить поняття коректності вимоги через точність опису функціональності. У цьому сенсі коректність певною мірою конкурує з повнотою. Але є і відмінність: якщо властивість повноти носить скоріше якісний характер: абсолютна повнота представляє недосяжний ідеал, до якого можна наближатися, то властивість коректності носить оціночний характер і задає дихотомію: кожне з вимог або коректно, або ні. Крім того, можна міркувати про взаємну коректності вимог або узгодженості (несуперечності): якщо два вимоги вступають в конфлікт, значить - як мінімум одне з них некоректно. В ієрархії вимог можна виділити вертикальну і горизонтальну узгодженість. Іншими словами, вимоги не повинні суперечити, відповідно, вимогам свого рівня ієрархії і вимогам "батьківського" рівня. Так, вимоги користувачів не повинні суперечити бізнес-вимогам, а функціональні вимоги - вимогам користувача.

4. Можливість перевірки (простежуваності, можливості трасування, верифікації) - означає, що існує кінцевий і розумний за вартістю процес ручної або машинної перевірки того, що програмне забезпечення задовольняє вимогам. Кожна вимога (особливо нефункціональне) має містити достатньо інформації для однозначної перевірки його реалізації. При цьому в ході перевірки у сторін (приймаючої і здає роботу) не повинно виникнути нерозв'язних суперечностей в оцінках. Так як добре сформульовані вимоги складають основу успішного створення програмного забезпечення. Вимоги до програмного забезпечення представляють основу контракту між Замовником та Виконавцем та якщо дані вимоги не можна перевірити - значить і контракт не має ніякого сенсу, отже, успіх або невдача проекту будуть залежати тільки від емоційних оцінок сторін і їх здатності домовитися, а це - надто хитка основа для здійснення робіт.

Інакше, факт реалізації буде грунтуватися на думці, а не на аналізі, що призведе до проблем при здачі готового програмного забезпечення. Для характеристики верифікації можна вважати наявність чисельних значень характеристик якості програмного забезпечення. Так, характеристика верифікації істотно пов'язана з властивостями ясності і повноти: якщо вимога викладена мовою, зрозумілою і однаково сприймається учасниками процесу створення програмного забезпечення, причому вимоги є повними, тобто жодна з важливих для реалізації деталей не упущена - значить, цю вимогу можна перевірити.

Також, процес верифікації дозволяє проаналізувати корисність виявлених вимог: або ці вимоги несуть недостатню корисне навантаження і можуть бути проігноровані, або мають місце помилки проектування: пропущені відповідні вимоги або вони недостатньо були описані(деталізовані для реалізації). Інша мета верифікації – підвищити керованість

проектом: при зміні окремо взятої вимоги є зрозумілим, які з проектних, робочих або інших вимог або ресурсів підлягають зміні.

5. Необхідність і корисність при експлуатації – це вимоги, які відображають можливість або характеристику програмного забезпечення, які дійсно необхідні користувачам, або які витікають із інших вимог. Необхідність і корисність при експлуатації – це одні з найбільш суб'єктивних властивостей вимог, які важко перевіряються.

Так, для бізнес-вимог вимоги формулюють перші особи, що представляють Замовника, і навряд чи хто-небудь краще них знає, яким умовам повинна відповідати створюване програмне забезпечення, щоб відповідати бізнес-цілям підприємства. Проте, якщо у представника Розробника виникають сумніви в необхідності тої чи іншої бізнес-вимоги, викликані інтуїтивними міркуваннями, або досвідом впровадження програмного забезпечення аналогічного призначення на підприємствах, він повинен проявити ініціативу і зібрати спільну нараду сторін. Аргументи на користь відсутності необхідності вимоги безсумнівно будуть сприйняті, особливо якщо вони будуть мотивовані в бізнес-термінології Замовника і підтверджені викладками, які прогнозують співвідношення витрат на виконання вимоги і очікуваної ефекту від реалізації.

Необхідність реалізації вимог користувача може випливати з відповідних бізнес-вимог. Крім того, вимоги користувача можуть мотивуватися ергономічністю продукту і особливостями використання програмного забезпечення, а також у зв’язку із недостатньою повнотою розкриття на попередньому рівні ієрархії визначення вимог до програмного забезпечення.

Слабшою, в якісному рівні за "необхідність" є властивість "корисність при експлуатації". Розмежування між даними

властивостями можна провести наступним чином. Необхідними слід вважати властивості, без виконання яких неможливо, або утруднене виконання вимог користувачів, щодо функціональних властивостей програмного забезпечення, а корисними при експлуатації слід вважати будь-які властивості, що підвищують ергономічні якості програмного забезпечення

6. Здійсненність – означає, що вимога повинна бути здійсненна при заданих обмеженнях, які можуть бути накладеним внутрішнім та зовнішнім середовищем на програмне забезпечення (наприклад: операційне середовище). Здійсненність вимог перевіряється в процесі аналізу здійсненності реалізації вимог розробником. Зокрема, для функціональних вимог перевіряється можливість досягнення зазначених чисельних значень при існуючих обмеженнях в системі.

Можна сформулювати вимогу, здійсненність якої обмежена науковими уявленнями або здійсненність якого обмежується сьогоднішнім рівнем розвитку техніки і технології, хоча багато з того, що було неможливо десять років тому, цілком реально сьогодні, хоча і фундаментальні уявлення іноді змінюються, нехай і не так швидко, як розвиток IT-технологій.

Здійснимість вимоги на практиці визначається розумним балансом між цінністю (ступенем необхідності та корисності) і потреби ресурсів. Так, якщо вартість контракту на розробку програмного забезпечення становить $ 10000, а витрати на виконання нової вимоги, що виникло в момент, коли проект виконаний наполовину, оцінюється в $ 4000, чи є воно нездійсненним? Швидше за все, так, якщо Виконавець доведе Замовнику новизну вимоги (вимога не входила в узгоджені специфікації) і складність його виконання. Але, якщо вимога є критично важливою, необхідною, але не була виявлена перед

підписанням контракту на розробку програмного забезпечення і Замовник готовий виділити додаткове фінансування, а Виконавець - трудові ресурси - значить, вимога здійсненна. Таким чином, вимога є здійсненною в ряді випадків які є суб'єктивним, а критерії оцінки залежать від області домовленостей між Замовником та Виконавцем. Чудовою ілюстрацією балансування між цінністю і здійсненністю є так званий трикутник компромісів (рис. 4.11). Залежність між ресурсами проекту (людськими і фінансовими), календарним графіком (часом) його виконання і реалізованими можливостями (рамками) утворюють трикутник, показаний на рис. 4.11. Досягнення рівноваги в цьому трикутнику при зміні будь-якої з його сторін вимагає для підтримки балансу модифікацій інших (двох) сторін. При цьому необхідно забезпечувати можливість модифікації вимог, якщо знадобиться, і підтримки історії змін. Для цього всі вимоги повинні мати унікальні позначення для однозначної їх ідентифікації. Кожна вимога має бути записано в специфікації тільки один раз. Інакше є можливість отримати неузгодженість, змінивши тільки одну із сторін трикутника. Збереження вимог в базі даних робить придатними їх для повторного використання та управління.

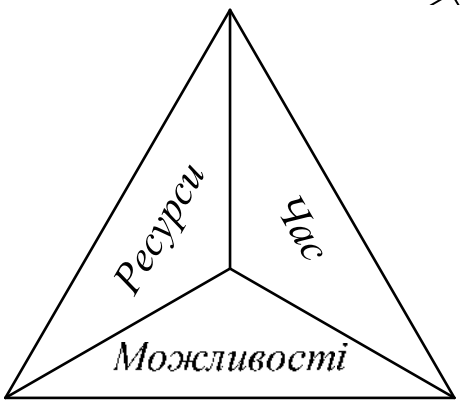


Рис. 4.11 – Трикутник компромісів

7. Можливість модифікації – означає, що вимога може змінюватися тоді і тільки тоді, коли її структура і стиль такі,

що будь-яка зміна вимоги може відбуватися, не порушуючи існуючої структури і стилю всієї множини вимог до програмного забезпечення.

8. Упорядкованість за важливістю і стабільністю – означає, що вимоги мають оцінку ступеня значимості (важливості) та незмінності в часі. Пріоритети вимог зазвичай призначає представник Замовника. Розробник, відштовхуючись від пріоритетності вимог, управляє процесом реалізації програмного забезпечення. Стабільність пріоритету вимоги характеризує прогнозну оцінку незмінності вимоги в часі та пріоритету її реалізації при створенні програмного забезпечення.

Пріоритети вимог зазвичай розрізняються на:

- Hot (терміново) - вимога необхідно реалізувати терміново, навіть раніше, ніж вимоги з високим пріоритетом. Пріоритет використовується в основному для прийняття управлінських рішень. Введено для можливості управління черговістю опрацювання, реалізації і тестування вимог, щоб вимоги з пріоритетами «середній» і «низький» можна було підняти в черзі над вимогами з пріоритетом «високий».
- High (високий) - вимога з максимальним пріоритетом. Зазвичай пріоритет визначенні вимозі розробники надають їм пріоритет разом із замовником.
- Medium (середній) - вимога середнього пріоритету. Зазвичай пріоритет визначенні вимозі розробники надають їм пріоритет разом із замовником. Значення за замовчуванням.
- Low (низький) - вимога низького пріоритету. Зазвичай пріоритет визначенні вимозі розробники надають їм пріоритет разом із замовником.

9. Наявність кількісної метрики мають важливу роль у

верифікації та атестації програмного забезпечення. В першу чергу це відноситься до нефункціональних вимога, які, як правило, повинні мати під собою кількісну основу (запит повинен відпрацьовуватися не більше, ніж ___ секунд; середнє напрацювання на відмову повинна становити не менше, ніж ___ годин). Функціональні вимоги також можуть характеризуватися кількісними метриками.

Багато з характеристик якостей вимог програмного забезпечення є взаємозалежні.

Отже, ***якість вимог*** до програмного забезпечення безпосередньо визначає якість програмного забезпечення, що розробляється.

4.7 АТРИБУТИ ЯКОСТІ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для кожного із розглянутих раніше типів вимог до програмного забезпечення можна застосувати однаковий набір атрибутів, які відповідно розділяють на дві великі групи:

– перша група (Run Time) – це атрибути, які відносяться до роботи програмного забезпечення;

– друга група (Design Time) – це атрибути, які визначають ключові аспекти проектування програмного забезпечення.

До першої групи належать такі атрибути якості:

– *доступність* – атрибут якості, що визначає час безперервної роботи програмного забезпечення (визначається, як максимально допустимий час простою системи);

– *надійність* – вимога, що описує поведінку програмного забезпечення в позаштатних ситуаціях (приклади: автоматичний перезапуск, відновлення роботи, збереження даних, дублювання важливих даних, резервування логіки);

– вимоги *до часу зберігання* даних (наприклад, використання БД в якості постійного сховища даних, тривалість зберігання даних);

– *масштабованість* – вимога до горизонтального і / або вертикального масштабування програмного забезпечення (Вертикальна масштабованість – вимога до вертикальної архітектури, можливості перенесення додатків на більш потужні SMP-системи, підтримка великого обсягу пам'яті і файлів; горизонтальної масштабованість – вимоги до горизонтальної архітектури, можливості використання технологій кластеризації, дозволяє підвищити відмовостійкість системи);

– вимоги *до зручності використання* програмного забезпечення (з точки зору користувача) і вимоги до зручності і простоти підтримки (Usability);

– вимоги *до безпеки*, як правило, включають в себе три великі категорії: вимоги, пов'язані з розмежуванням доступу, вимоги, пов'язані з роботою з приватними даними, і вимоги, спрямовані на зниження ризиків від зовнішніх атак;

– вимоги *до конфігурації* додатків, взаємодії та розташування компонентів умовно розділяють на чотири рівні:

1. на основі визначеного набору параметрів (predefined configurability), коли необхідний рівень модифікації досягається шляхом зміни значень параметрів із визначеного набору;
2. на основі визначеного набору базових об'єктів (framework constrained configurability), коли необхідний рівень модифікації досягається шляхом перекомпонування визначеного набору процесів, сутностей і службових процедур;
3. шляхом реалізації нових базових об'єктів (basis reimplementation), коли забезпечується розширення

набору процесів і сутностей;

4. шляхом нової реалізації системи (system reimplementation), коли система повинна встановлюватися і налаштовуватися з нуля;

– вимоги *до продуктивності* рішення, які визначаються в термінах кількості одночасно працюючих користувачів, що обслуговуються транзакцій, часу реакції, тривалості обчислень, а також швидкості і пропускної здатності каналів зв'язку

– *обмеження на обсяг* доступної пам'яті, процесорного часу, дискового простору, пропускну здатність мережі, при яких програма має ефективно виконувати покладені на нього завдання

– *складність реалізації* вимоги (Complexity) (в складності реалізації вимоги не враховується складність тестування, наприклад підготовка тестових стендів і/ або тестових даних).

До другої групи належать такі атрибути якості:

– вимоги *до повторного* використання реалізації або компонентів програми або системи (Reusability);

– вимоги *до розширюваності* (Extensibility) програмного забезпечення в зв'язку з появою нових функціональних вимог, тісно пов'язано переносимість коду;

– вимоги *до переносимості* (Portability) додатки або системи на інші платформи;

– вимоги *до взаємодії* між компонентами програмного забезпечення, між зовнішніми компонентами, використання стандартних протоколів і технологій взаємодії (Interoperability). (наприклад, до таких вимог можна віднести можливість використання декількох стандартних протоколів для обміну даними між однією з підсистем розроблюваної системи і зовнішньою системою-постачальником даних);

– вимоги *до підтримки* системи або додатки (Supportability) – це параметри такі як дешевизна і швидкість

розробки, прозорість поведінки додатку, простота аналізу помилок і проблем в роботі;

– вимоги *до модульності* програми або системи (Modularity) вказують, яким чином програмне забезпечення повинно бути розділено на модулі, або перераховують список обов'язкових модулів, які повинні входити до складу системи;

– вимоги *до можливості тестування* (Testability) програмного забезпечення визначають обсяг вимог до автоматичного і ручного тестування, наявність необхідного інструментарію;

– вимоги *до можливості і простоті локалізації* (Localizability) програмного забезпечення визначають можливості і специфічні архітектурні вимоги, що накладаються процесом локалізації та містять перелік мов, на яких передбачається виконувати локалізацію програмного забезпечення.

Склад атрибутів вимог може бути доповнюватися або скорочуватися, що залежить від масштабу проекту по створенню програмного забезпечення. Значення та склад атрибутів визначаються спільно всіма членами проектної команди та використовуються менеджером проекту для визначення ризиків при реалізації програмного забезпечення.

Отже, ***атрибути вимог*** до програмного забезпечення дозволяють приймати рішення реалізації програмного забезпечення на всій етапах його життєвого циклу.

4.8 СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Специфікація вимог до програмного забезпечення (Software Requirements Specification (SRS)) – специфікація всіх виявлених вимог до програмного забезпечення, що були виявлені та проаналізовані на початковому етапі розробки програмного забезпечення. Специфікація вимог містить:

– повний і чіткий опис програмного забезпечення;

– повний опис поведінки програмного забезпечення.

Основне призначення специфікації – це отримання зворотного зв'язку із зацікавленими особами та спрощують процес узгодження вимог із зацікавленими особами і командою з розробки програмного забезпечення. Тому вимоги в специфікації мають бути записані зрозумілою мовою для всіх зацікавлених осіб, що приймають участь у їх розробці.

Специфікація вимог до програмного забезпечення потрібна для:

– отримання точної оцінки вартості, ризиків і витрат часу;

– можливості клієнтом більш чітко сформувати власне бачення програмного забезпечення;

– реалізації однакового уявлення замовником і виконавецем про майбутнє програмне забезпечення;

–виявлення оптимального набору функцій програмного забезпечення;

– формування основи для реалізації іншої технічної документації;

– оптимізації процесу розробки програмного забезпечення;

– мінімізовані витрати часу на розробку програмного забезпечення;

– запобігання дублювання вимог при розробці програмного забезпечення;
– запобігання виникнення протиріч вимог при розробці програмного забезпечення;
– структурувати проблеми, що виникають при розробці програмного забезпечення та здійснити оптимальний пошук шляхів для вирішення цих проблем;
– допомагає зрозуміти, які результати будуть вважатися оптимальними при тестуванні програмного забезпечення.

Відповідно до стандарту IEEE 830 (див. Додаток А) рекомендована наступна структура специфікації вимог до програмного забезпечення (Software requirements specification-SRS):

Вступ:

– цілі
– угоди про терміни
– аудиторія та послідовність сприйняття
– масштаб проекту
– посилання на джерела

Загальний опис:

– бачення продукту
– функціональність продукту
– класи і характеристики користувачів
– середовище функціонування продукту (операційне середовище)
– рамки, обмеження, правила і стандарти
– документація для користувачів
– допущення і залежності
– функціональність системи:
– функціональний блок X (таких блоків може бути кілька)

опис і пріоритет
– причинно-наслідкові зв’язки, алгоритми (рух процесів, workflows)
– функціональні вимоги
– вимоги до зовнішніх інтерфейсів
– інтерфейси користувача (UX)
– програмні інтерфейси
– інтерфейси обладнання
– інтерфейси зв’язку і комунікації
– нефункціональні вимоги
– вимоги до продуктивності
– вимоги до збереження (даних)
– критерії якості програмного забезпечення
– вимоги до безпеки системи
– інші вимоги

Додаток А: Словник

Додаток Б: Моделі процесів і предметної області та інші діаграми

Додаток В: Список ключових завдань

Інщою специфікацією вимог до програмного забезпечення є класифікацічя наведена в єдиній системі програмної документації (ЕСПД), яка має наступну структуру:

Вступ

– Призначення, мета. Визначити продукт, вимоги до якого описані в цьому документі. Описати межі продукту, зокрема якщо цей документ описує лише частину системи чи окрему підсистему.
– Посилання. Список документів чи Web адрес, на які посилається цей документ. Може включати зразки користувацьких інтерфейсів, контракти, стандарти, варіанти використання.

Загальний опис.

– Перспективи продукту. Описати контекст і витоки продукту, який специфікується цим документом. Наприклад, стан продукту, як члена сімейства продуктів, заміна існуючих систем, чи новий самодостатній продукт. Якщо SRS визначає компонент великої системи, пов'язати вимоги великої системи із функціональністю цього продукту і визначити інтерфейси між ними. Ефективними є прості діаграми, що показують основні компоненти загальної системи, взаємозв'язок підсистем чи зовнішні.
– Характеристики продукту. Резюмувати основні характеристики продукту чи істотні функції, що він здійснює чи дозволяє здійснювати користувачу. Деталі представляються в Розділі 3, тому тут потрібне узагальнення вищого рівня. Організувати функції, щоб вони були зрозумілими будь-якому читачу документа. Ефективним є представлення основних груп пов'язаних вимог і їхніх зв'язків діаграмами потоків даних чи діаграмами класів.
– Класи користувачів та їх характеристики. Визначити різні класи користувачів, які як ви очікуєте, будуть використовувати продукт. Класи користувачів можуть диференціюватись, базуючись на частоті використання, підмножині функцій продукту, яка використовується, технічній експертизі, рівнях безпеки чи привілеїв, рівню освіти чи досвіду. Описати доцільні характеристики кожного класу користувачів. Відділити пріоритетні класи користувачів від тих, що є менш важливими.
– Середовище функціонування. Описати середовище, в якому буде функціонувати продукт, включаючи апаратну платформу, операційну систему і версії, і

будь-які інші програмні компоненти чи аплікації, з якими воно має коректно співіснувати.

– Обмеження проектування і реалізації. Описати питання чи пункти, що будуть обмежувати можливості, доступні розробникам. Може містити: корпоративні чи регуляторні правила; апаратні обмеження (часові вимоги, вимоги щодо пам'яті); взаємодії з іншими аплікаціями; специфічні технології, інструменти, бази даних; паралельні операції; мовні вимоги; протоколи комунікацій; обговорення безпеки; проектні програмні стандарти (на приклад, якщо організація замовника буде відповідальною за супровід продукту).
– Документація користувача. Список компонент документації користувача (такі як керівництво по використанню, on-lineдопомога, чи інструкції), що буде надаватися разом з програмним продуктом. Визначити формати та стандарти документації^
– Припущення та залежності. Список припущень (які суперечать відомим фактам), що можуть мати вплив на вимоги встановлені в SRS. Може включати комерційні компоненти, які планується використовувати, факти щодо розробки середовища функціонування. Визначити залежність проекту від зовнішніх факторів, таких як програмні компоненти з інших проектів, які планується повторно використовувати.
– Характеристики програмного забезпеченя (системи). Ця частина ілюструє організацію функціональних вимог до продукту через характеристики системи, основні сервіси, які надає продукт. Це розділ можна організувати через варіанти використання, режими операцій, користувацькі класи, об'єкти класів,

функціональну ієрархію, чи їх комбінації, залежно від того, що найбільш логічно придатно для продукту.

Характеристика системи 1 (2,3).

3.1.1 Опис і пріоритет

Надайте короткий опис характеристики і відзначте, якого вона пріоритету Високого, Середнього, чи Низького. Можна також включати специфічні оцінки компоненти, такі як користь, штраф, ціна чи ризик (кожна оцінена у відповідній шкалі від 1 до 9).

3.1.2 Послідовності дія/відгук

Список послідовностей дій користувача і відгуків системи, що спричиняють режим визначений для цієї характеристики. Це відповідає елементам діалогу асоційованим з варіантами використання.

2.1.3 Функціональні вимоги

Перелічити детальні функціональні вимоги асоційовані із цією характеристикою. Це можливості продукту, які мають бути реалізовані, щоб користувач скористався сервісами чи виконав варіант використання. Включаючи, як продукт повинен реагувати на помилкові умови чи неправильні введення. Вимоги мають бути короткими, повними, недвозначними, верифіковуваними і необхідними.

Кожна вимога має бути унікально ідентифікована номером чи значущою міткою певного виду, наприклад

0-1:

0-2: .

Вимоги зовнішніх інтерфейсів

– Користувацькі інтерфейси. Описати логічні характеристики кожного інтерфейсу між ПЗ та користувачами. Може включати зразки зображень

екрану, GUIстандарти чи керівництва стилів сімейства продуктів, яких треба дотримуватись, розмітка екрану, стандарти кнопок і функцій (наприклад, допомога), що з'являються на кожному вікні, комбінації клавіш, стандарти відображення повідомлень про помилки і т.п. Визначити програмні компоненти, для яких потрібні користувацькі інтерфейси. Деталі розробки користувацького інтерфейсу мають бути документовані в окремій специфікації користувацького інтерфейсу.

– Апаратні інтерфейси. Описати логічні та фізичні характеристики кожного інтерфейсу між ПЗ та апаратними компонентами системи. Може включати типи підтримуваних пристроїв, природу даних і керуючих взаємодій між ПЗ та апаратними засобами і комунікаційні протоколи, які будуть використані.
– Програмні інтерфейси. Описати зв'язок між продуктом і іншими специфічними програмними компонентами (назва і версія), включаючи бази даних, операційні системи, інструменти, бібліотеки і інтегровані комерційні компоненти. Визначити дані і повідомлення, які поступають в систему і виходять з неї, описати мету кожної. Описати необхідні сервіси і природу комунікацій. Навести документи, що описують протоколи програмних інтерфейсів додатку. Визначити дані, які будуть спільно використовуватись програмними компонентами.
– Комунікаційні інтерфейси. Описати вимоги, що пов'язані з комунікаційними функціями: електронна пошта, веб-броузер, мережеві протоколи,

електронні форми і т.п. Визначити прийнятні формати повідомлень. Визначити комунікаційні протоколи, які будуть використовуватись (FTP, HTTP). Визначити безпеку комунікацій чи питання шифрування, швидкість передачі даних, і механізми синхронізації.

– Інші нефункційні вимоги.
– Вимоги продуктивності. Якщо є вимоги продуктивності до продукту в різних середовищах, описати їх та пояснити. Визначити часові залежності для систем реального часу. Описати такі вимоги настільки точно, як це можливо.
– Вимоги надійності. Визначити вимоги, що пов’язані з можливими втратами, пошкодженнями, що можуть виникати при використанні продукту. Визначити заходи безпеки чи дії, які треба прийняти для запобігання цьому.
– Вимоги безпеки. Визначити вимоги, що стосуються безпеки чи питань секретності щодо використання продукту чи захисту даних, які використовуються чи створюються. Визначити вимоги аутентифікації користувачів.
– Атрибути якості програмного продукту. Визначити додаткові якісні характеристики до продукту, важливі для замовників чи розробників. Наприклад, адаптовуваність, придатність, коректність, гнучкість, функціональна сумісність, супроводжуваність, портативність, надійність, стійкість, тестопридатність, зручність використання.
– Інші вимоги. Визначити інші вимоги, що не розкриті в SRS. Це може включати вимоги бази даних, вимоги інтернаціоналізації, юридичні вимоги і т.п.

Додаток А: Словник. Визначити всі терміни необхідні для правильної інтерпретації SRS, включаючи абревіатури та скорочення.

Додаток В: Моделі аналізу. Опційно, включити будь-які необхідні таблиці й структури опису моделейжиттєвого циклу,аналізу процесів та ризиків необхідних для аналізу й оптимізації розробки програмного додатку для обрано предметної області,.

Додаток С: Список пропозицій. Динамічний список відкритих пунктів вимог, які потрібно вирішити, незакінчені рішення, інформація, яка є потрібною, конфлікти, які очікують розв'язання.

Отже, *специфікація вимог до програмного забезпечення* – це закінчений опис функціонального призначення та поведінки програмного забезпечення, яке потрібно розробити.

4.9 КЕРУВАННЯ ВИМОГАМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Процес керування вимогами мало чим відрізняється від керування будь-якими іншими процесами. Перш, ніж почати будь-який програмний проект, необхідно зрозуміти, які саме функції мають бути реалізовані в програмному забезпеченні і, що має бути отримано в результаті реалізації програмного проекту. Необхідно уявити характер робіт, які потрібно буде виконувати на всіх етапах життєвого циклу програмного забезпечення, необхідно зрозуміти, чи будуть існувати будь-які взаємозв'язки між окремими функціями, а також необхідно знати якими професійними навичками повинні володіти розробники програмного забезпечення.

Важливо пам'ятати, що реалізація будь-якого програмного забезпечення залежить від трьох факторів (рис 4.11):

– можливості реалізації програмного забезпечення;
– вартість робіт з реалізації програмного забезпечення;
– термінів завершення робіт з реалізації програмного забезпечення.

Ці три фактори дуже тісно пов'язані між собою. Будь-яка зміна вимог до програмного забезпечення призводить до зміни хоча б одного з цих факторів призводить до зміни як мінімум ще одного фактора. Кожне прийняте рішення, що до зміни вимоги або її модифікації призводить до необхідності заново позиціонувати програмний проект щодо цих трьох фундаментальних факторів, що призводить до появи проблем із управління процесами керування вимогами при реалізації проектів зі створення програмного забезпечення. Серед останніх можна виділити три основні:

– Недостатній досвід керування вимогами до програмного забезпечення – зумовлений тим, що небагато організацій мають добре поставлений внутрішній процес керування вимогами при реалізації програмного забезпечення. В результаті люди, що працюють з вимогами, виявляються не готовими до цього. У такій ситуації дати оцінку планованого часу і трудовитрат важко, оскільки одним з основних елементів гарної оцінки є відповідний досвід з керування вимогами до програмного забезпечення та процесами з їх реалізації командою розробників. Як наслідок люди, що не мають досвід в керуванні вимогами, можуть навіть і не здогадуватися про те, яку роботу необхідно запланувати для реалізації вимог.
– Не розрізнення вимог користувача та системних вимог до програмного забезпечення, системних вимог і

системних специфікацій вимог до програмного забезпечення. Тобто, розробники починають відразу визначати детальне технічне рішення, замість того, щоб спочатку розробити вимоги, які не залежать від реалізації.

– Керування вимогами в значній мірі залежить від типу організації конкретної команди розробників програмного забезпечення. Тобто процес формування і управління вимогами та їх взаємозв'язками буде відрізнятися в залежності від типу організації:
  – Організація-покупець створює систему для задоволення своїх власних потреб. Головним завданням для організацій такого типу є розробка і керування призначеними для користувача вимогами, які згодом використовуються для приймання програмного забезпечення.
  – Організація-постачальник задовольняє запити організації-покупця або організації-постачальника вищого рівня. Організації такого типу зазвичай отримують вимоги і на їх основі розробляють системні вимоги на основі, яких розробляються специфікації до програмного забезпечення.
  – Організація-розробник – організація, яка розробляє програмне забезпечення та продає його. Організація такого типу формує призначені для користувача вимоги під впливом ринку, а не вивчаючи потреби конкретної групи користувачів. Виявленням вимог в такій організації зазвичай займається відділ маркетингу. Організація-виробник розробляє програмне забезпечення відповідно до вимог користувача (ринковими) і потім продає його. По суті, організація такого типу одночасно виступають в ролі покупця і постачальника, проте зв’язки між

різними департаментами всередині даної організації, відрізняються від стандартних взаємин між звичайною компанією-постачальником і компанією-покупцем.

Для керування вимогами, часто використовують план керування вимогами. План керування вимогами призначений для документування порядку аналізу, документування та керування вимогами на всьому життєвому циклі розробки програмного забезпечення. Типовий план керування вимогами до програмного забезпечення може включати:

– порядок планування, відстеження та складання звітів про дії з вимогами;
– дії з управління конфігурацією, такі як порядок ініціювання змін вимог до програмного забезпечення, послуги або результату, порядок аналізу ризиків та їх впливів на програмне забезпечення, їх виявлення, відстеження і складання звітів, а також рівні повноважень, необхідні для схвалення цих змін;
– процес розстановки пріоритетів вимог;
– структуру відстеження, тобто які параметри вимог будуть відображені в матриці відстеження.

Структура типового плану керування вимогами може мати наступну структуру

1. Вступ
2. Загальний опис методології аналізу, розробки та керування вимогами
3. Аналіз і моделювання
3.1. Огляд основних моделей
3.2. Управління артефактами (моделями) при аналізі
4. Розробка вимог
4.1. Типи вимог
4.2. Типізація не функціональних вимог
4.3. Атрибути вимоги

5. Управління вимогами

5.1. Методологія керування

5.1.1. Атрибути вимог

5.1.2. Розподіл повноважень при роботі з вимогами

5.1.3. Обговорення вимог

5.1.4. Узгодження вимог

5.1.5. Затвердження вимог

5.2. Життєвий цикл вимог

5.2.1. Вимоги зацікавлених осіб

5.2.2. Всі інші типи вимог

5.3. Трасування вимог

6. Керування змінами у вимогах

6.1. Обробка запитів на зміну вимог

6.2. Базові версії вимог (BaseLines)

7. Специфікації вимог

8. Керування процесом аналізу та розробкою вимог

8.1. Список основних артефактів і діяльностей з керування вимогами

8.2. Передача інформації бізнес-аналітику

8.3. Навчання і передача знань системному аналітику

8.4. Постійні поліпшення процесу аналізу і розробки вимог

Отже, *керування вимогами* є одним із найважливіший процесів в управлінський діяльності при реалізації програмного забезпечення. Постійне порівняння досягнутих результатів з запланованими паралельно з контролем витраченого часу і ресурсів дає можливість оцінювати поточне виконання плану. Ігнорування ж результатів, а використання лише контролю часу і ресурсів, формує дуже спотворене уявлення про хід робіт з реалізації програмного забезпечення і призводить до зриву програмного проекту або створення не якісного програмного забезпечення.

4.10 АНАЛІЗ ТА КЕРУВАННЯ РИЗИКАМИ

Важливою частиною при систематизації вимог програмного забезпечення є робота з оцінки ризиків, які можуть вплинути на графік робіт або на якість створюваного програмного забезпечення, а також розробка заходів щодо запобігання виникненню ризиків.

Ризики впливають на розподіл використання ресурсів, на графік робіт, які необхідні для виконання проекту. Конкретні типи ризиків, які можуть вплинути на даний проект, залежать від виду створюваного програмного забезпечення й від організаційного оточення, де реалізується програмне забезпечення. Разом з тим існує багато типів ризиків, які наведені в табл. 4.2, що здатні вплинути на якість та строки виконання будь-якого програмного забезпечення.

Таблиця 4.2. Можливі ризики

Ризик	Тип ризику	Опис ризику
Плинність розробників	Ризик для проекту із створення програмного забезпечення	Досвідчені розробники залишають програмний проект до його завершення
Зміна в керуванні організацією	Ризик для проекту із створення програмного забезпечення	Організація змінює свої пріоритети в керуванні створення програмного забезпечення
Неготовність апаратних засобів	Ризик для проекту із створення програмного забезпечення	Апаратні засоби, які необхідні для програмного проекту, не надійшли вчасно або не готові до експлуатації
Зміна вимог	Ризик для проекту із створення ПЗ й для ПЗ, що розробляється	Поява великої кількості непередбачених змін у вимогах, пропонованих до розроблювального програмного забезпечення

Ризик	Тип ризику	Опис ризику
Затримка в розробці специфікації	Ризик для проекту із створення ПЗ й для ПЗ, що розробляється	Специфікації основних інтерфейсів програмної підсистем не *надійшли* до розроблювачів відповідно до графіка робіт
Недооцінка масштабу розроблювального програмного забезпечення	Ризик для проекту із створення ПЗ й для ПЗ, що розробляється	Розмір системи значно перевищив первісну оцінку
Недостатня ефективність CASE-засобів	Ризик для проекту із створення ПЗ й для ПЗ, що розробляється	CASE-засоби, призначені для підтримки проекту, виявилися менш ефективними, ніж очікувалося
Зміни в технології розробки ПЗ	Ризик для проекту із створення програмного забезпечення	Основні технології побудови програмної системи заміняються новими
Поява конкуруючого програмного продукту	Бізнес-Ризик	На ринку програмних продуктів до закінчення проекту з'явилася конкуруюча програмна система

Схема процесу керування ризиками складається із чотирьох стадій.

Визначення ризиків. Визначаються можливі ризики для проекту, для розроблювального продукту й бізнес ризики.

Аналіз ризиків. Оцінюється ймовірність і послідовність появи ризикових ситуацій.

Планування ризиків. Плануються заходи щодо запобігання ризиків або мінімізації їхнього впливу на проект.

Моніторинг ризиків. Постійне оцінювання ймовірностей ризиків і виконання заходів щодо зм'якшення наслідків прояву ризикової ситуацій.

Процес керування ризиками, як і інші процеси планування, є ітераційним, виконуваним протягом усього строку реалізації проекту. Спочатку розробляються плани керування ризиками, потім постійно відслідковується ситуація навколо реалізації проекту. При надходженні нової інформації про можливі ризики заново проводиться аналіз ризиків і першорядна увага приділяється новим ризикам. У міру надходження нової інформації також змінюються плани заходів щодо запобігання й пом'якшення ризиків.

Результати процесу керування ризиками документуються у вигляді планів керування ризиками. Вони повинні включати опис можливих проектних ризиків, їхній аналіз і перелік заходів, необхідних для керування ризиками при створенні програмного забезпечення.

Визначення ризиків може виконуватися в режимі командної роботи з використанням підходу "мозковий штурм" або ґрунтуватися на досвіді менеджера. При визначенні ризиків можливе використання категорій ризиків наведених в табл. 4.3.

1. *Технологічні ризики.* Виникають із програмних і апаратних технологій, на основі яких розробляється система.

2. *Ризики, пов'язані з персоналом.* Пов'язані зі членами команди розроблювачів.

3. *Організаційні ризики.* Виникають із організаційного оточення, у якому виконується проект.

4. *Інструментальні ризики.* Пов'язані з використовуваними CASE-засобами й іншими засобами підтримки процесу створення ПЗ.

Таблиця 4.3. Категорії ризиків

Категорія ризиків	Приклади ризиків
Технологічні ризики	База даних, що використовується в програмній системі, не забезпечує обробку очікуваного обсягу транзакцій. Програмні компоненти, які використовуються повторно, мають дефекти, що обмежують їхні функціональні можливості
Ризики, пов'язані з персоналом	Неможливо підібрати працівників з необхідним професійним рівнем. Провідний розроблювач занедужав у самий критичний час. Неможливо організувати необхідне навчання персоналу
Організаційні ризики	В організації, що виконує розробку ПЗ, відбулася реорганізація, у результаті чого змінилися пріоритети в керуванні проектом. Фінансові труднощі в організації привели до зменшення бюджету проекту
Інструментальні ризики	Програмний код, що генерується CASE-засобами, не ефективний. CASE-Засобу неможливо інтегрувати з іншими засобами підтримки проекту
Ризики, пов'язані із системними вимогами	Зміни вимог приводять до значних повторних робіт із проектування системи. Первісне нечітке формулювання користувальницьких вимог привело до значних змін системних вимог, що виявилися на пізніх стадіях розробки проекту

5. *Ризики, пов'язані із системними вимогами.* Проявляються при зміні вимог, пред’явлених до розроблювальної системи.

6. *Ризики оцінювання.* Зв'язані оцінюванням характеристик програмної системи й ресурсів, необхідних для реалізації проекту.

При аналізі для кожного певного ризику підраховується ймовірність його прояву й збиток, що він може нанести.

Результати аналізу ризиків представляються у вигляді таблиці ризиків, упорядкованих по ступені можливого збитку. У табл. 4.4 наведений упорядкований список ризиків, описаних у табл. 4.4, в якій зазначені ймовірності цих ризиків. В табл. 4.4 ймовірності ризиків і ступінь збитку від них зазначені довільно. На практиці для їхнього визначення необхідна докладна інформація про програмний проект, технологію створення програмного забезпечення, команду розробників і про саму організацію.

Таблиця 4.4. Список ризиків після проведення їхнього аналізу

Ризик	Імовірність	Ступінь збитку
Фінансові труднощі в організації привели до зменшення бюджету проекту	Низька	Катастрофічна
Неможливо підібрати працівників із професійним рівнем, що вимагається	Висока	Катастрофічна
Провідний розроблювач занедужав у самий критичний час	Середня	Серйозна
Програмні компоненти, використовувані повторно, мають дефекти, що обмежують їхні функціональні можливості	Середня	Серйозна
Зміни вимог приводять до значних повторних робіт із проектування системи	Середня	Серйозна
В організації, що виконує розробку ПЗ, відбулася реорганізація, у результаті чого змінилися пріоритети в керування	Висока	Серйозна
База даних, що використовується в програмній системі, не забезпечує обробку очікуваного обсягу транзакцій	Середня	Серйозна
Недооцінки часу виконання проекту	Висока	Серйозна

CASE-засоби неможливо інтегрувати з іншими засобами підтримки проекту	Висока	Терпима
Первісне нечітке формулювання користувальницьких вимог привело до значних змін системних вимог, що виявилися на пізніх стадіях розробки проекту	Середня	Терпима
Неможливо організувати необхідне навчання персоналу	Середня	Терпима
Швидкість виявлення дефектів у системі нижче раніше запланованої	Середня	Терпима
Розмір системи значно перевищує спочатку розрахований	Висока	Терпима
Програмний код, що генерований CASE-засобами, неефективний	Середня	Незначна

Отже, *аналіз та керування* ризиками полягає у забезпечення успішності з реалізації розробки програмного забезпечення в умовах виникнення можливих ризиків які можуть вплинути на графік робіт або на якість створюваного програмного забезпечення.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яка різниця між функціональними та нефункціональними вимогами?

2. Які чотири стадії процесу керування ризиками?

3. Які є основні групи нефункціональних вимог?

4. Що характеризують вимоги предметної області?

5. Розкрийте зміст поняття "програмні вимоги (Software Requirements)".

6. Розкрити зміст поняття "група нефункціональних вимог (NonFunctional Requirements)".

7. Розкрийте зміст поняття "бізнес-правила (Business Rules)".

8. Розкрийте зміст поняття "обмеження (Constraints)".

9. Розкрийте зміст поняття "системні вимоги (System Requirements)".

10. Розкрийте зміст поняття "незалежні властивості (Emergent Properties)".

11. Розкрийте зміст поняття "вимоги з кількісною оцінкою (Quantifiable Requirements)".

12. Розкрийте зміст поняття "системні вимоги та програмні вимоги (System Requirements and Software Requirements)".

13. Поясніть процес роботи з вимогами (Requirements Process).

14. Розкрийте зміст поняття "модель процесу визначення вимог".

15. Поясніть техніку витягу вимог (Elicitation Techniques).

16. Розкрийте зміст поняття "аналіз вимог (Requirements Analysis)".

17. Розкрийте зміст поняття "класифікація вимог (Requirements Classification)".

18. Розкрийте зміст поняття "архітектурне проектування та розподіл вимог (Architectural Design and Requirements Allocation)".

19. Розкрийте зміст поняття "специфікація вимог (Requirements Specification)".

20. Як називається етап ЖЦ розроблення ПЗ, на якому фіксується контракт між замовником і виконавцем розробки?

21. Назвіть дійових осіб процесу формування вимог.

22. Назвіть джерела відомостей про вимоги.

23. Яка послідовність кроків по використанню діючої системи в новій розробці?

24. Дайте пояснення для нотації діаграми сценаріїв і базових відносин у них.

25. Розкажіть про принципи взаємин між замовником і розробником вимог до програмного забезпечення.

26. Розкрийте зміст поняття "вимоги до продукту та процесу (Product and Process Requirements)".

27. Розкрийте зміст понять "функціональні та нефункціональні вимоги (Functional and Non-functional Requirements)".

28. Розкрийте зміст поняття "бізнес-вимоги (Business Requirements)".

29. Розкрийте зміст поняття "користувальницькі вимоги (User Requirements)".

РОЗДІЛ 5
ПРОГРАМНА ДОКУМЕНТАЦІЯ

При вивченні цієї теми важливо пам’ятати, що *документація програмного забезпечення (англ. software documentation)* - супроводжуючі документи до програмного забезпечення, які містять в собі інформацію, яка описує загальні положення необхідні для розуміння функцій та призначення програмного забезпечення і дозволяє забезпечити потреби розробників й користувачів, залучених до створення та використання програмного забезпечення необхідною та самодостатньою інформацією потрібною для розробки, виготовлення, експлуатації та супроводу програмного забезпечення. Така документація є важливою і описує не тільки яким чином правильно використовувати програмне забезпечення, а й пояснює основні алгоритми використання програмного забезпечення і його призначення. В залежності від складності кожного окремого програмного забезпечення, специфіки, а також ліцензії, що використовуються при його створенні – програмна документація може варіюватися за обсягом і змістом.

Зміст розділу

5.1 Документообіг у життєвому циклі програмного забезпечення.

5.2 Цілі та задачі документування.

5.3 Вимоги до документації програмного забезпечення

5.4 Типи та види програмної документації

5.5 Оновлення документації

5.1 ДОКУМЕНТООБІГ У ЖИТТЄВОМУ ЦИКЛІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Забезпечення програмних продуктів відповідною технічною та користувацькою документацією є важливою та невід'ємною стадією, яка супроводжує етапи створення, тестування, впровадження та успішне використання розробленого ПЗ в інформаційних системах збору, обробки та аналізу даних для систем технічної діагностики або простих програмних додатків призначених для вирішення безлічі організаційних проблем, які гальмують ділові процеси і, як наслідок, знижують ефективність діяльності підприємства в цілому.

Протягом життєвого циклу програмного забезпечення може створюватися до півсотні різних типів документів, щоб полегшити планування, відстеження та складання звітів відповідно до етапів проекту. Документи починаються від аналізу можливості реалізації, планів по залученню та використанню ресурсів, фінансових планів і проектних планів контрактів з постачальниками, аналізів можливого функціонування впровадженої системи, форм запитів про внесення змін і звітів про статус проекту.

Як правило, документація до програмного забезпечення створюється під час виконання певного етапу проектування. Відповідно до класичного представлення життєвого циклу чотирма етапами документацію програмного проекту можна розділити на фази:

1. Фаза опису або створення концепції проекту – фаза, що присвячена створенню основи програмного проекту, визначаються вимоги до програмного забезпечення, цілі проекту, що є ключовим для забезпечення успішної реалізації програмного забезпечення на всіх фазах.

2. Фаза проектування проекту – фаза, що присвячена аналізуванню можливих несподіванок (аналізу ризиків). Документи цієї фази демонструють план проекту, розподіл ресурсів, містять угоди з клієнтами, сценарії управління ризиками та містять в собі стратегічні деталі проекту.

3. Фаза виконання(тестування та впровадження) проекту – фаза, що присвячена створенню, тестуванню та впровадженню програмного забезпечення. Документи цієї фази надають реальні дані, дозволяють відстежувати витрати ресурсів і їх розподілення, хід робіт, а також відстежувати виникаючі проблеми та методи їх вирішення при реалізації та впровадженні програмного забезпечення.

4. Фаза підтримки проекту – фаза, що присвячена огляду результатів впровадження та використання програмного забезпечення, вибору методів та сценаріїв, які будуть застосовувати надалі до програмного забезпечення. Серед сценаріїв найтиповішими є закриття та створення нового або еволюція вже існуючого програмного забезпечення. Документи цієї фази містять невирішені питання і / або результати впровадження та використання,

Фази проекту є покажчиками того, які саме програмні документи відіграють важливу роль на певному етапі життєвого циклу програмного проекту. Відсутність програмної документації може привести до появи ряду перешкод, серед яких можна виділити:

• Нестача ясності – виникає коли менеджери проекту та замовники мають нечітке уявлення про вимоги, стан робіт над проектом та ризиків, що можуть виникнути при реалізації проекту. Документи програмного проекту розглядаються як окремі документи, між якими не має взаємозв’язку та обміну інформацією. Ці документи ведуть до надлишку інформації, що в свою чергу призводить до неоднозначності в програмній

документації та браку при створенні програмного забезпечення при реалізації програмного проекту.

• Слабка безпека – означає, що відсутні бізнес-правила і технологічні процеси для роботи з важливими елементами програмного проекту, а відповідно і не створюються важливі програмні документи, що містять вимоги до безпеки та методи реалізації захисту в програмному продукті. Це може привести до втрати або витоку важливої інформації про проект.

• Втрата даних – виникає при неможливості зберегти всю документацію проекту в одному сховищі. Інформація в програмних документах в принципі може загубитися, стати важкодоступною, що призводить до відсутності цілісності даних та є потенційною загрозою до появи невірних звітів та прийняття невірних рішень в проекті.

• Обмежене співробітництво – виникає коли програмні документи або їх частини зберігаються як неструктуровані дані у вигляді електронних та паперових листів, звітів, специфікацій, вимогах тощо.

Один із варіантів створення документації певного типу та змісту у відповідності з фазами життєвого циклу процесу розробки програмного забезпечення наведено на рис. 5.1.

Значна частина документації створюється і застосовується для внутрішнього використання розробниками на різних етапах реалізації програмного проекту (рис. 5.1).

Проектна документація є засобом демонстрації просування робіт над проектом і підвищення особистої відповідальності його учасників за результат виконання проекту на багатьох стадіях, як підтвердження успішного виконання проекту. Крім цього, в реальних випадках факт власноручного викладу результатів робіт розробником в письмовій формі є декларацією його здібностей.

Документація, яка створюється під час реалізації програмного проекту використовується для зберігання і

передачі знань і фактів різноманітного характеру - від «глобальних» – на зразок вимог до системи, до знань обмеженого використання (наприклад, список дефектів, знайдених в ході певного раунду тестування). Документація використовується для формалізації домовленостей і зобов'язань, під якими можна розуміти широкий спектр обіцянок, намірів або завдань, що курсують між однією чи кількома організаціями (наприклад: технічне завдання на розробку програмного забезпечення).

Документація, створена на етапі тестування призначена для перевірки роботи програмного продукту відповідно до вимогами та специфікацій, затвердженими замовником та розробником. На етапі тестування розробляються сценарії реалістичного тестування, створюється документація відповідно до результатам тестування й розробляється експлуатаційна документація на межі переходу до етапу впровадження.

Документація створена на етапі підтримки, використовується для ведення історії змін, містить методи контролю за внесенням змін та план контролю якості програмного забезпечення. План контролю якості за внесеними змінами визначає, яким чином програмне забезпечення має досягти відповідності встановленому рівню якості.

Отже, ***документація програмного забезпечення (англ. software documentation)*** – сукупність документів, які містять в собі, необхідну та самодостатню інформацію для розробки, виготовлення, експлуатації та супроводу програмного забезпечення.

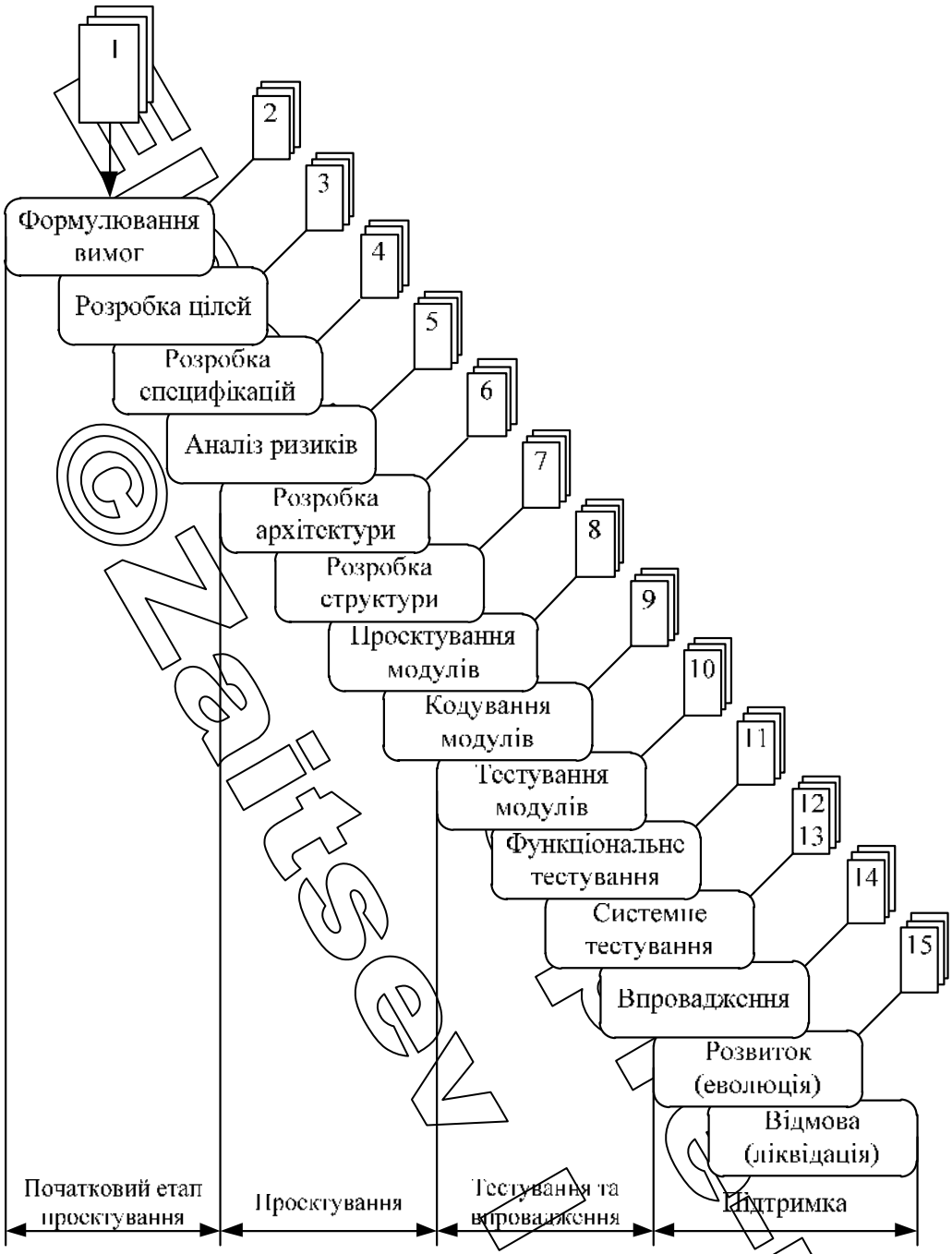


Рис. 5.1 – Схема відповідності етапів створення документації ПЗ до етапів життєвого циклу ПЗ:

1 – завдання на розробку ПЗ; 2 – перелік вимог; 3 – перелік цілей; 4 – перелік специфікацій; 5 – план керування ризиками; 6 – опис архітектури; 7 – опис структури: 8 – специфікації інтерфейсу і бази даних; 9 – специфікації мови кодування; 10 – сценарії тестування модулів та результати тестування; 11 – сценарії функціонального тестування та результати тестування; 12 – сценарії системного тестування та результати тестування; 13 – керівництво користувача (оператора) та керівництво з технічного обслуговування; 14 – план контролю якості; 15 – перелік історії змін.

5.2 ЦІЛІ ТА ЗАДАЧІ ДОКУМЕНТУВАННЯ

Цілі створення документації ПЗ досить різнопланові та залежать від широкого кола факторів, серед яких найвагомішими є вимоги та потреби персоналу, залученого до розробки програмного продукту, а також специфіка етапів з розробки або підтримки ПЗ, на якому є необхідність створення та ведення певного типу програмної документації.

Цінність правильності і повноти документації зростає з масштабами програмного проекту у зв’язку з тим, що в великих проектах з розробки програмних систем залучається велика кількість спеціалістів різних спеціальностей, які залежать один від одного. Отже, забезпечення надійного зв'язку є важливим для розробки надійного ПЗ, і, крім того, документація є частиною кінцевого етапу розробки програмного продукту.

Основна ціль будь-якої документації з програмного забезпечення – інформувати зацікавленого читача про функції ПЗ на необхідному рівні, що забезпечує розуміння описуваних функцій відповідно до завдань, вирішуються в ПЗ.

У загальному випадку програмна документація відповідає ряду істотних вимог, серед яких основними є наступні:

1) відповідність запитам кількох категорій технічного персоналу, пов'язаного з проектуванням і експлуатацією розробленого ПЗ: керівник проекту, системний програміст, аналітик, програміст, користувач та інші;

2) можливість використання на різних етапах проектування і експлуатації програмного продукту;

3) відповідність якостям, що сприяють розумінню структури і уможливлюють використання її на будь-якому етапі проектування і експлуатації програмної системи;

4) можливість до оновлення при використанні;

5) автоматізація пошуку і здійснення корекції у разі необхідності.

Кожна з категорій персоналу, що використовує програмну документацію, має різні цілі і задачі з використання та створення документації:

Користувач (оператор) – використовує програмну документацію для розуміння того, як працювати з ПЗ та в деяких випадках для розуміння способів реалізації вирішення задач в ПЗ. До такої документації можна віднести:

– керівництво користувача до ПЗ;

– матеріал спеціального призначення, що містить суб'єктивні знання експертів з предметної області та описує способи реалізації вимог до ПЗ;

– технічний матеріал, в якому використана мова високого рівня, з коментарями, орієнтованими на користувача.

У випадку, коли користувач використовує ПЗ лише для роботи за призначенням йому потрібна інформація лише для того, щоб використовувати програму в процесі обробки даних, знати можливості програми, межі її функціонування і накладені обмеження на дані. У зв'язку з тим, що початкові вимоги користувача підлягають різнобічній перевірці і в них можуть бути виявлені помилки, неузгодженості або протиріччя, у випадках коли користувач безпосередньо бере участь в розробці ПЗ, йому необхідно ознайомитися з процедурою діагностики помилок на різних етапах створення програмного проекту.

Менеджер проекту з розробки програмного продукту та системний аналітик – це може бути одна особа, в залежності від масштабів проекту, яка використовує та створює документацію до програмного забезпечення, яку застосовує для розуміння етапів розробки, структури та архітектури ПЗ від вищого до детального рівнів. Така документація має бути зрозумілою та детальною до рівня,

який дозволяє менеджеру здійснювати планування і контроль за ходом виконання процесу розробки ПЗ для програмної системи (кодуванням, контролем, тестуванням окремих компонентів ПЗ), впровадженням та використанням ПЗ.

Менеджеру проекту також – необхідно мати навички та знання про кодування критичних модулів програми або алгоритми їх функціонування в системі на рівні обґрунтованого прийняття рішення щодо проблем, пов'язаних з використанням ПЗ. Він повинен володіти чітким уявленням про всі нюанси розробки ПЗ для програмної системи, починаючи від етапу початкового формулювання вимог, тестування та супроводу закінчуючи підтримкою програмного забезпечення при його експлуатації та ліквідації

Як менеджер, так і системний аналітик, використовуючи та створюючи документацію, повинен знати мову програмування, принципи функціонування ПЗ на рівні елементарних складових, методи конструювання та архітектури ПЗ в достатньому обсязі. Це гарантує в разі необхідності прийняття відповідальних рішень про зміни і корекції в програмному проекті на основі раніше розробленої повної та якісної документації до ПЗ та втручання менеджера в хід розробки, проектування та підтримки ПЗ на професійному рівні. Крім перерахованих вище вимог, менеджер проекту, звісно, має й інші обов'язки, пов'язані з контролем графіку виконання і вартості розробки ПЗ.

Виконання перерахованих вище обов'язків неможливе за відсутності добре організованого документування процесів розробки та використання програмної системи.

Системний програміст (програміст) – використовує та створює на різних етапах розробки програмного забезпечення великий обсяг різноманітної документації.

На вхідному рівні системний програміст повинен швидко отримати достатню інформацію для створення ПЗ. У зв'язку з

тим, що може мати місце плинність кадрів розробників, наявність повної і якісної документації є одним з основних факторів вирішення проблеми передачі розробки від одного програміста іншому.

Системний програміст повинен мати детальні знання про структуру ПЗ або про окремі її компоненти, в розробці яких бере участь, про внутрішню взаємодію окремих компонентів ПЗ між собою та з операційною системою, що використовується на в апаратно-програмному забезпеченні інформаційної системи збору, обробки та аналізу даних. Системному програмісту для успішної роботи необхідні знання та навички з таких областей знань як:

– основи інформатики;
– основи вищої математики;
– основи теорії алгоритмів;
– основи побудови структур даних;
– основи побудови інтерфейсу користувача;
– моделі даних і їх організацію;
– мов системного програмування;
– принципів побудови мов запитів і маніпулювання даними;
– синтаксису, семантики і формальних способів опису мов програмування;
– конструкції розподіленого і паралельного програмування;
– архітектурних особливостей і основ побудови сучасних електронно-обчислювальних машин;
– методів і етапів трансляції;
– принципів побудови експертних систем;
– способів і механізмів управління даними;
– принципів організації, складу і схем роботи операційних систем;
– принципів управління ресурсами, методи організації

файлових систем;

– основних методів розробки програмного забезпечення;

– інформаційного законодавства;

– законодавства про авторські та суміжні права;

– правил і норм з охорони праці, техніки безпеки.

Вирішення питань адаптації документації до потреб користувача та розробників лежить на шляху створення ієрархічної структури, де кожен рівень документації (рис. 5.2) має свій рівень деталізації.

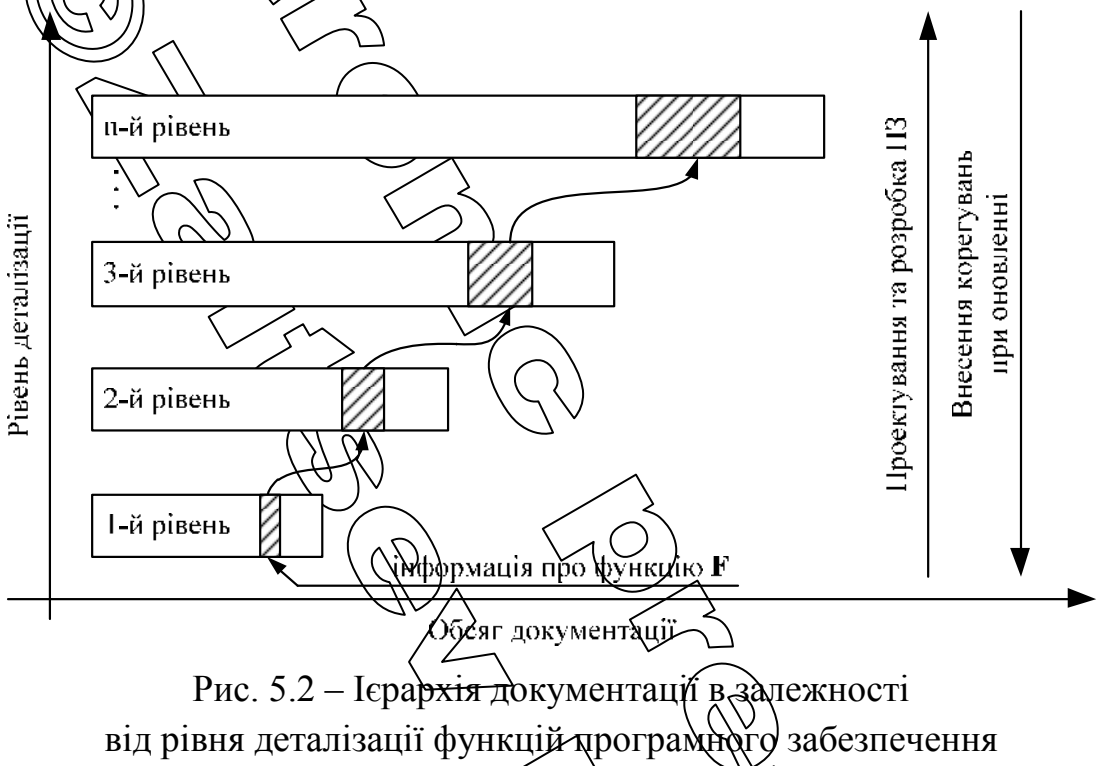


Рис. 5.2 – Ієрархія документації в залежності від рівня деталізації функцій програмного забезпечення

На рис. 2.1 схематично зображено формування детальної інформації про функцію **F**, яка для вирішує завдання в деякій предметній області. На кожному рівні деталізації має бути отримана єдина структура документації з метою забезпечення простих посилань від документації вищих рівнів до відповідної документації, в якій розглядаються структури деталізованих нижчих рівнів. Створення такого механізму є важливим також і тому, що необхідно забезпечити працездатний механізм оновлення (корегування) документації.

Таким чином, ***основною ціллю створення документації програмного забезпечення*** є забезпечення потреб розробників і користувачів, залучених до створення та використання програмного забезпечення необхідною та самодостатньою інформацією для розробки, виготовлення, експлуатації та супроводу програмного забезпечення.

5.3 ВИМОГИ ДО ДОКУМЕНТАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до документації, її переліку, змісту окремих документів та їх структури залежить насамперед від масштабу проекту з розробки програмного продукту.

Проведення аналізу масштабу проекту із створення програмного забезпечення необхідно для встановлення в договорі між замовниками і розробниками масштабу, допустимого для розробки переліку програмної документації. Деякі вимоги можуть вимагати зміни (зазвичай збільшення) масштабу, і відповідно зміни кількості видів документації, що використовується на етапах ЖЦ програмного продукту для забезпечення необхідної функціональної придатності програмного забезпечення.

Таким чином, вимоги до функціональної придатності, до конструктивних характеристикам ПЗ, до форматів, структури і номенклатури документації мають бути узгоджені з масштабом проекту і доступними ресурсами для його реалізації.

Формуванню вимог до програмного продукту супроводжується створення вимог, що відображають документообіг при реалізації програмного проекту.

Від масштабу ПЗ безпосередньо залежать витрати

ресурсів для їх документування і не завжди доцільно створювати і використовувати в реальних проектах весь комплекс документів.

Масштаб проекту і специфікація вимог до ПЗ, безпосередньо позначаються на складі, змісті та обсязі документації, яка є необхідною різним учасникам проекту. Кожен з розробників в тій чи іншій мірі має залучатися до управління вимогами, як до проекту, так і до документації проекту.

Розробникам необхідно виробити професійні прийоми для розуміння і викладу в документах потреб користувачів, управління масштабом проекту, побудовою системи і документації, які відповідають досить повно ці потреби, які включають:

- команду розробників ПЗ, отримує уявлення про складність і розміру створюваного продукту, склад його документації;

- Менеджери проекту, - базу для розрахунку вмісту специфікацій, графіків, витрат і ресурсів;

- Група тестування, - плани тестування, варіанти випробувань і процедури перевірок;

- Фахівці з супроводу та підтримки, отримують уявлення про функціональність кожної складової частини продукту;

- Клієнти відділу маркетингу і фахівці з продажу, будуть мати подання про кінцевий програмний продукт;

- Укладачі документації, що створюють шаблони документів, керівництва для користувачів і довідки на підставі специфікації вимог до ПС отримають проект призначеного для користувача інтерфейсу;

- Фахівці, відповідальні за навчання персоналу, отримають специфікації вимог до ПЗ і документацію для користувачів, а також для розробки навчальних матеріалів;

- Персонал, який займається юридичною стороною проекту, перевірить, чи відповідають вимоги до продукту існуючим законам і постановам.

Розмір або масштаб комплексів програм в даний час приводиться в публікаціях в різних одиницях, що може змінювати їх чисельні значення для одних і тих же програм в кілька разів. В якості таких одиниць найчастіше використовуються чисельні значення: рядків тексту програми на мові програмування, пропозицій на асемблері в тексті програми, байт або команд в об'єктному коді після трансляції. Кожна з цих одиниць виміру може мати деякі переваги при певних цілях дослідження або проектування. Вплив одиниць виміру розміру програм на оцінку раціонального обсягу документації можна значно змінюватися, якщо врахувати їх принципові особливості і, перш за все, виділити дві групи одиниць вимірювання масштабу проектів ПЗ:

- Групу, що характеризує розмір початкового програмного коду, які розробляються й аналізуються фахівцями та відображає складність, трудомісткість і тривалість створення ПЗ, його компонентів та основних документів до ПЗ;

- Групу, яка відображатиме розмір програм і даних, що розміщуються та реалізуються, характеризує обсяг пам'яті і продуктивність апаратно-програмних засобів, необхідних для робочого функціонування і виконання всі функцій у відповідності із призначенням програмного забезпечення.

Ці дві групи одиниць відображають розмір програм і документів, з різних позицій, і повинні використовуватися в залежності від цілей аналізу і застосування значень масштабу проекту. Хоча між ними є взаємозв'язок, але в загальному випадку розміри програми, виміряні різними групами одиниць, важко спів ставляти через наявність між ними проміжної ланки - перетворювача (транслятора). Це обумовлено особливостями роботи трансляторів, які

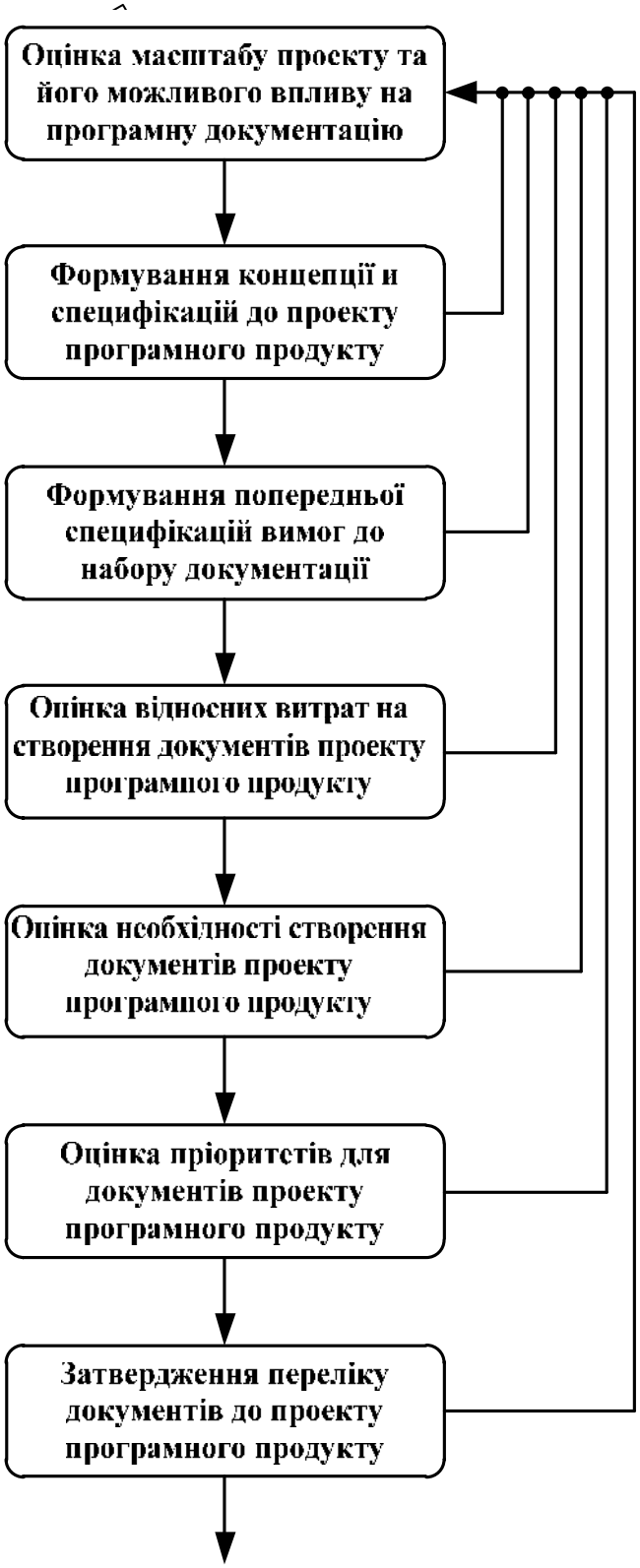


Рис. 5.3 – Алгоритм оцінки номенклатури документів до програмного продукту

перетворюють тексти програм, розроблені людиною - програмістом, в команди, що реалізує ЕОМ, а також особливостями мов програмування.

Основна мета оцінки масштабу ПЗ – підготувати можливість прийняти обґрунтованого рішення про допустимість подальшого просування проекту в область аналізу, збору та специфікації вимог, попереднього проектування і створення документації. Якщо виявляється, що розрахований масштаб і необхідні ресурси не можуть бути забезпечені для продовження проекту, то можливі кардинальні рішення: або зміна деяких виділених ресурсів, або припи-нення проектування даного

ПЗ (рис. 5.3).

Один із шляхів оцінки розміру ПЗ і складу комплексу документів, полягає в порівнянні його функціональних завдань і властивостей з уже існуючими аналогами. Звичайно, даний метод не є точним, оскільки при написанні компонентів прототипів і документів могли використовуватися різні мови програмування, що відносяться до різних областей додатків. Також могли застосовуватися різноманітні алгоритми, що мають різні рівень складності і функціональні властивості. Їх вибір залежить від конкретних умов розробки проекту, а також від можливостей використання інформації про раніше створених прототипах з близькими функціональними характеристиками ПЗ. Коли вперше розглядається масштаб нового проекту ПЗ, інтуїтивні і експертні оцінки його масштабу і складу необхідної документації замовниками і розробниками можуть відрізнятися від кінцевого значення в кілька разів. Така вірогідність оцінок обумовлена рівнем невизначеності на початковому етапі знань про функції, зміст і можливої складності програмного продукту.

Для зменшення цих невизначеностей і можливих методичних помилок необхідно визначити основні поняття і одиниці вимірювання масштабу комплексів програм. З самого початку роботи над проектом ПС важливо вести постійний облік даних про його дійсної трудомісткості, вартості, зміст і комплексі документів і порівнювати ці дані з реальними оцінками цих характеристик аналогічних проектів. Слід узгоджувати цілі оцінювання масштабу документування з потребами в інформації, зафіксованої в документах, що сприяє прийняттю рішень для планування розвитку проекту, витрат праці та інших ресурсів при подальшому застосуванні.

Для продовження розробки номенклатури документів після оцінки масштабу комплексу програм, необхідно

виконати цикл поетапного визначення та формування сукупності специфікацій вимог до компонентів і документації проекту ПЗ. Першим етапом є створення Концепції проекту ПЗ і комплексу первинних вимог специфікацій до ієрархічним набору функцій, на які можуть бути розбиті передбачувані фактичні компоненти ПЗ. Надалі розбиття може структуруватися і деталізуватися, формуючи спрощений або більш точний рівень абстракції і взаємодії компонентів. Мета документа - концепція, яка полягає в зборі, аналізі та визначенні високорівневих потреб користувачів, функцій і документів програмного продукту. Основна увага повинна приділятися можливостям і функцій, в яких потребують майбутні розробники і користувачі, і причин існування цих потреб.

Після первинної оцінки масштабу проекту ПЗ ітераційно виконується поетапна розробка специфікацій вимог до проекту і документів ПЗ. Обмеження при прогнозуванні вимог до документів, визначаються, перш за все, наявними даними, які можуть бути використані в якості вихідних аналогів або узагальнених характеристик. Будучи кінцевим сховищем комплексу вимог до продукту і документів, специфікація вимог до ПЗ повинна бути ясною і зрозумілою, щоб у розробників і замовників не залишалося жодних можливостей для різночитання. Не обов'язково складати специфікацію для всього продукту до початку розробки, але необхідно зафіксувати вимоги для кожної складової перед її створенням. При роботі над будь-яким проектом необхідно досягти узгодженості рішень по кожному набору вимог і документів до початку їх реалізації розробниками. Узгодженість рішень являє собою процес вивчення і схвалення документів до ПЗ. У загальному випадку для оцінки, прогнозування та обґрунтування специфікацій вимог нового комплексу документів необхідні вихідні дані:

- Узагальнені характеристики використаних ресурсів для документування та техніко-економічні показники завершених розробок - прототипів ПЗ, а також оцінки впливу на них функцій, різних факторів об'єктів і середовища розробки;

- Реалізовані плани документування та узагальнені переліки виконаних робіт, реальні графіки проведених раніше оцінок і розробок, різних ПЗ і документів;

- Цілі і зміст приватних робіт в процесі створення попередніх, складних комплексів програм і різних документів для забезпечення необхідної якості ПЗ в цілому;

- Структура і зміст повного комплекту документів, що був результатом виконання окремих робіт конкретного проекту.

Складати специфікацію вимог до документації ПЗ слід таким чином, щоб всі зацікавлені в проекті особи змогли в ній розібратися:

- Розділи, підрозділи і окремі вимоги повинні бути названі узгоджено;

- Корисно використовувати засоби візуального виділення (такі, як напівжирний, підкреслення, курсив і різні шрифти) послідовно, ієрархічно і в розумних межах;

- Створюйте зміст, а також алфавітний покажчик, щоб полегшити користувачам пошук необхідної інформації;

- Нумерувати всі малюнки і таблиці, посилання на них, використовуючи присвоєні номера.

Щоб легко відстежувати і модифікувати документи, кожне функціональне вимога повинна бути подана унікально і незмінно. Це дозволить посилатися на певні вимоги документів в запиті на зміни, в хронології змін, в перехресних посиланнях або матриці для відстеження реалізації вимог. При цьому також спрощується багаторазове використання документованих вимог в декількох проектах. Для усунення або зниження помилок складу і ризиків документації до допустимих меж може бути необхідна зміна вимог до

функціональної придатності та/або до конструктивним характеристикам і доступним ресурсам проекту ПЗ. Тому на етапах проектування послідовно повинні визначатися:

- при виборі концепції і первинній оцінці масштабу проекту - попередні вимоги до призначення, функціональної придатності, до складу і номенклатурі необхідних конструктивних характеристик якості і до первинного переліку набору документів ПЗ;

- при попередньому проектуванні - уточнена оцінка масштабу проекту, вимоги до функціональних і конструктивним характеристикам якості, до орієнтовною структурою і змістом набору шаблонів документів в життєвому циклі ПЗ з урахуванням загальних обмежень ресурсів;

- при детальному проектуванні - докладні специфікації вимог до функціональних і конструктивним характеристикам якості з детальним обліком і розподілом реальних обмежених ресурсів, а також повний склад і зміст шаблонів документів в життєвому циклі програмного забезпечення.

Розробку та затвердження специфікацій вимог до функціональних характеристик, до якості, складу і номенклатурі документів ПЗ, доцільно проводити ітераційно на етапах попереднього і детального проектування.

Повна формалізація вимог до характеристик і комплекту документів на початку життєвого циклу складного ПЗ зазвичай неможлива, перш за все, через різних уявлень замовника і розробників про деталі його призначення, функцій і можливостей реалізації при доступних ресурсах.

Чим більше і складніше проект ПЗ і відповідно вище його вартість, тим ретельніше слід розробляти вимоги до його характеристикам, до складу та змісту документів, а також розподіляти ресурси на їх реалізацію.

Для середньої і відносно невисокої складності ПЗ у багатьох випадках можна задовольнятися подібний аналіз вимог до комплексу програм, відповідного попереднього проектування ПЗ.

Для великих і особливо складних проектів потрібен більш детальний аналіз факторів, що впливають на розробку набору документів і вимог їх оптимізації за критерієм якість/витрати в детальному програмному проекті.

При первинному визначенні вимог до функціональної придатності і до конструктивним характеристикам, задані замовником обмеження ресурсів не завжди можуть враховувати ряд особливостей проекту, що може зумовити неприпустиме зниження (або завищення) вимог до деякими характеристиками і документам ПС. Крім того, можливо, що деякі характеристики суперечливі або принципово не реалізовуються в даному проекті. В результаті, не збалансовані вимоги і доступні ресурси виявляться у вигляді втрат в якості або в потреби додаткових ресурсів.

Залежно від складності проекту остаточним результатом робіт при попередньому або детальному документуванні проекту повинні бути деталізовані і затверджені документи специфікацій до номенклатури, властивостей, значень якості і документації проекту ПЗ, які є достатніми для його повноцінного робочого проектування і подальшої, ефективної експлуатації. Ці вимоги до документів закріплюються в контракті і технічному завданні, за якими розробник згодом повинен звітувати перед замовником при завершенні проекту. Однак на наступних етапах життєвого циклу і при конфігураційному управлінні, документи можуть змінюватися за погодженням між замовником і розробником, які найчастіше приурочуються до підготовки нової версії ПЗ. Для цього необхідний моніторинг масштабу проекту, комплекту документів, специфікацій вимог і реалізацій характеристик

протягом всього життєвого циклу програмного забезпечення.

Для розробників особливо важливо формалізувати вимоги до якості і узгодити їх із замовником при затвердженні контракту і технічного завдання на проект ПЗ. Вимоги до функціональних характеристик, якості, складу та змісту документів, затверджені після попереднього проектування, можуть бути закріплені в технічному завданні як обов'язкові для детального і робочого проектування. Ці дані можуть використовуватися при подальшому оцінюванні якості документації при їх зіставленні з вимогами в процесі кваліфікаційних випробувань або сертифікації ПЗ.

Для великомасштабних і дорогих проектів може знадобитися уточнення вимог до якості документації при детальному проектуванні з позиції поліпшення співвідношення значень якості і витрат ресурсів, які необхідні або допустимі для їх реалізації ПЗ.

Для замовника і користувачів може мати значення не тільки визначення функціональної придатності, а й оцінка потенційного попиту на ринку конкретного програмного продукту, а також його конкурентоспроможності з іншими аналогічними за функціями ПЗ з урахуванням його якості та вартості. Ця обставина може визначати необхідність уточнення охоронних вимог до окремих характеристиках і документами не тільки для їх реалізації розробниками в ЖЦ ПЗ, але також для оцінювання інтегрального якості готового програмного продукту, що поставляється на ринок. Зазвичай замовники і розробники спочатку встановлюють вимоги до кожної характеристиці і до номенклатури документів ПЗ без урахування відносних витрат на їх досягнення, а також без детального аналізу їх спільного впливу на повну функціональну придатність у споживете лей. Це може призводити до значних перекосів і до незбалансованим значенням вимог до окремих, взаємопов'язаним

характеристиками і документам, на які не раціонально використовуються обмежені ресурси ЖЦ ПЗ, або до неадекватно низьким їх значенням.

Для узагальненого оцінювання якості ПЗ необхідний врахування відносного впливу кожної конструктивної характеристики і документа, на функціональну придатність. При цьому не завжди враховуються ресурси для їх реалізації в конкретно ПЗ. Це часто призводить до не раціонально вимогам і документам, які значно відрізняються: або за ступенем впливу на функціональну придатність, або за величиною ресурсів, необхідних для їх реалізації як повноцінних документів.

Отже, ***вимоги до програмної документації*** потрібні для розуміння можливої складності та необхідної номенклатури програмної документації, що забезпечує потреби розробників і користувачів, залучених до створення та використання програмного забезпечення необхідною та самодостатньою інформацією.

5.4 ТИПИ ТА ВИДИ ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Типи та види програмної документації в будь-якому програмному проекті або продукті на сам перед призначені для відповіді на наступні питання:

- що повинно бути зроблено, крім власне програми?
- що і як має бути оформлено у вигляді документації?
- що передавати користувачам, а що службі супроводу?
- як управляти всім цим процесом?
- що має входити в саме завдання на програмування?

Процеси розробки програмного забезпечення та його

підтримки безпосередню пов'язані з використанням програмної документації та впливають на поділ останню за видами.

Кожний вид програмної документації в залежної від специфіки її використання можна умовно поділити на наступні дві великі групи (рис. 5.4) документацію, що використовується на стадії реалізації програмного проекту та документацію програмного продукту.

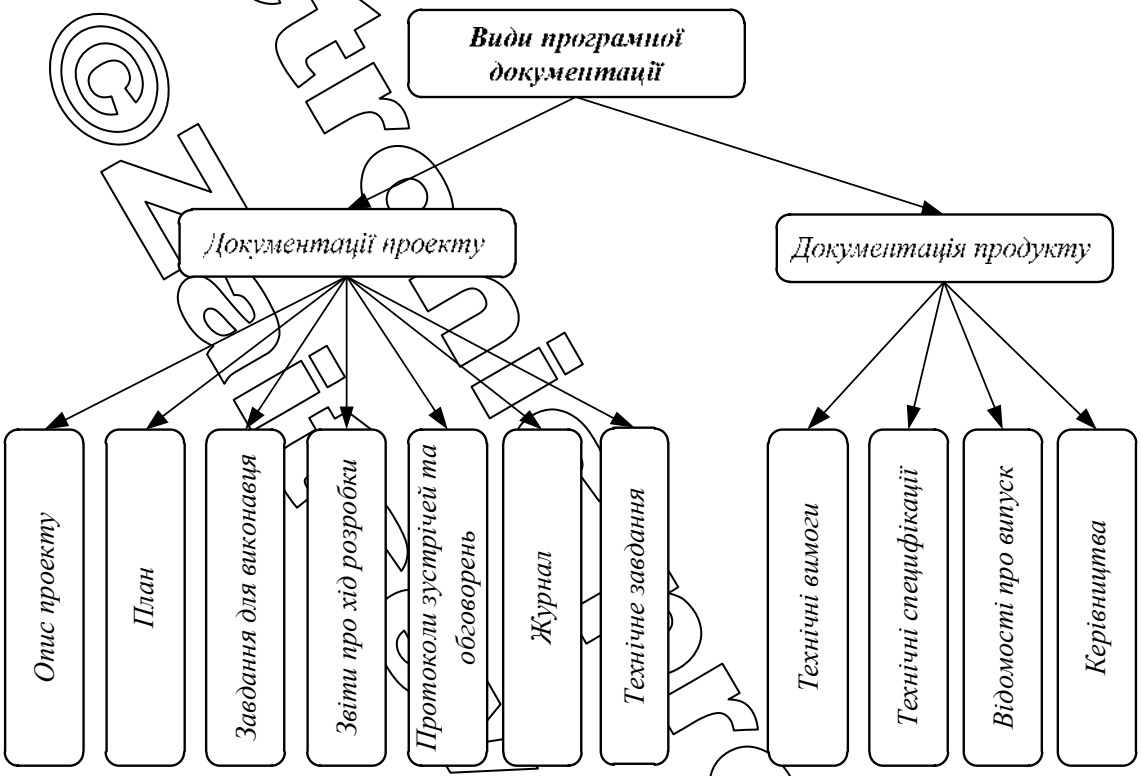


Рис. 5.4 – Види програмної документації

Документація, що використовується на стадії реалізації програмного проєкту в свою чергу ділиться на:

1. *Опис проекту (project statement),* містить основну інформацію про суть і призначення проекту, включаючи постановку задачі, оцінки, пріоритети і обмеження технічного, бюджетного та тимчасового характеру і критерії, при виконанні яких проект буде вважатися успішним. При цьому приділяється увага перерахуванню меж - активностей, які слугують вхідними вимогами для початку реалізації проекту, а

також тих, які не будуть використовуватися в рамках даного проекту але є допоміжними для розуміння реалізуємості функціоналу в програмному проекті.

В різних проектних групах в залежності від організації чи корпорації документи такого роду називаються по-різному: технічне завдання, на розробку, обґрунтування та ін. Однак, незважаючи на різницю в назвах, їх мета ідентична - описати ключові положення контрактного характеру. Ця характеристика звичайно слугує додатком до контракту та основою для виділення ресурсів (коштів) під реалізацію проекту, а тому повинна бути розроблена з максимальною достовірністю. Оскільки успішне виконання проекту є основною задачею виконавця (будь то спеціалізована організація або внутрішнє підрозділ замовника), різниці в оцінці успішності реалізації проекту можуть призвести до важких репутаційних та фінансових наслідків. Тому уточнення вимог до проекту та якісне керування процесами використання вимог при створенні документів і внесення необхідних доповнень при необхідності, як і узгодження вимог і доданих вимог при роботі над реалізацією проекту як із сторони замовника/користувача, так із сторони розробника (виконавця) відноситься до найважливіших інтересів як виконавця, та і замовника і виконується зазвичай менеджером проекту.

2. План – це документ, що фіксує систему цілей, задач і засобів, які необхідні для успішного завершення роботи над проектом. Необхідними атрибутами планів є:

– опис необхідних дій;

– інформація про події, що ініціалізують початок дії;

– опис взаємної залежності дії між собою подій і дій;

– інформація про розробників;

– інформацію про ймовірні дати початку та завершення робіт над проектом.

Найбільш поширеним виглядом планів є план-графік проекту в нотації діаграм Ганта. При роботі над дослідницькими проекти, а також проектами, що мають пряме керування часом можуть бути реалізовані без планів-графіків. У таких проектах планування може мати поверхневій характер, а плани майбутніх періодів можуть фіксуватися при складанні регулярних звітів.

Теорія розробки програмного забезпечення при використанні модель CMMI (Capability Maturity Model Integration – Інтеграція моделі можливостей зрілості) та ISO/IEC15504 (SPICE (Software Process Improvement and Capability Estimation – Покращення процесу розробки програмного забезпечення та оцінки можливостей) також згадують безліч інших планів, якими має супроводжуватися розробка програмного забезпечення: план управління вимогами, план управління змінами, план управління ризиками, план управління налаштуваннями, тестування та ін. На практиці в "одиночному" проекті невеликої складності, більшість із цих планів можуть носити загальний (ознайомчий) характер і мати більшу орієнтацію на опис правил взаємодії з замовником, ніж на опис внутрішніх функцій та вимог до програмного забезпечення, що проектується. Риски, пов'язані з відсутністю більшості планів, крім плану-графіки та плану тестування, мають значний вплив тільки при реалізації великих або складних проектів.

3. Завдання для виконавців/розробників (task) – документ, що встановлює основне призначення, показники якості, техніко-економічні та спеціальні вимоги до програмного забезпечення та його основних функцій, стадії розробки та складу програмної документації. Завдання, які видаються виконавцям, підлягають документуванню функціональних та не функціональних вимог до програмного забезпечення з метою виключення їх «забування» і двоякого тлумачення.

Документальні форми, які використовуються для накопичення і запису завдань, на практиці використовуються рідко і зазвичай в якості джерела інформації про видані завданнях застосовуються: плани-графіки; спеціально виділені розділи в регулярних звітах; завдання, прислані через Microsoft Outlook або через системи управління (такі, як Microsoft Project, OpenProj і ін.).

4. Звіти про хід робіт (status report) – це документи, що застосовуються для інформування керівництва і замовника про статус ходу реалізації проекту. Звіти можуть створюватися як кінцевими розробниками, так і менеджером проекту.

5. Протоколи зустрічей та обговорень – це документи, що містять результати формальних та неформальних, запланованих і спонтанних обговорень, присвячених найрізноманітніших питанням пов'язаним із створенням та реалізацією проекту по створенню програмного забезпечення. Ці обговорення можуть відбуватися у формі очних зустрічей, по телефону, а також листуванням. Підсумки усних обговорень доцільно фіксувати в формі протоколу, якщо: прийняті рішення, змінюють або доповнюють опис проекту; прийнято багато різноманітних рішень; висловлені нові ідеї будь-якого характеру, які можуть впливати на реалізацію проекту в подальшому. Найчастіше зафіксовані рішення в протоколах, відбивається в звітах, що подаються на розгляд особі, що приймає рішення по доцільності реалізації прийнятих рішень в проекті. Основними заходами прийняття рішення про доцільність є: аналіз здійсненності, аналіз альтернатив реалізації, огляд (review) проектної документації, тестування і приймання. Протоколи цих активностей, як правило, містять відомості про час, об'єкті, рамках і характер> активності, середовищі, в якій проводилася активність, досягнутих результатах і рекомендованих рішеннях. Звіт

служить відправною точкою і засобом мінімізації тривалості майбутніх обговорень, так що в деяких випадках рішення може бути прийнято по листуванню. Практика показує, що при відсутності звіту обговорення носить хаотичний і безпредметний характер, непродуктивно відволікаючи час задіяних осіб в реалізації проекту.

6. Журнали – це документи, що містять накопичувальне перерахування тих чи інших однотипних подій або фактів, що виникають в ході проекту. Типовими об'єктами, що накопичуються в журналах, є ризики, проблеми, запити на зміни, дефекти (як документації, так і власне продукту). План-графік може замінюватися або доповнюватися «журналом завдань», в разі, якщо ці завдання носять локальний характер. Журнали рідко оформляються у вигляді окремого документа. Журнали ризиків і проблем зазвичай є розділом в звіті про стан проекту; ведення журналів змін і дефектів організовується за допомогою систем управління змінами. Необхідно відзначити, що ведення журналів дефектів має сенс тільки в разі, коли інспекції або тестування здійснюються особою, яка не є автором коду програмного забезпечення. У більшості випадків розробник, який знайшов помилку в своєму творінні, вважатиме за краще тут же виправити її, не переймаючись якимось документуванням.

7. Технічне завдання (ТЗ) – це затверджений між замовником та розробником документ, на основі якого виконується розробка проекту та оцінка його вартості. У ньому максимально точно і детально описані вимоги до компонентів та характеристик майбутнього програмного продукту. Типово ТЗ містить наступні основні розділи:

– *призначення* – в даному розділі дається детальний опис для чого і як використовуватиметься майбутнє програмне забезпечення.

– *цілі і завдання* – в даному розділі описується навіщо вам потрібен цей проект? У чому його важливість для вашої компанії? Чому він вартий реалізації? Які проблеми він може вирішити? Яку користь принесе використання програмного забезпечення? Один проект може мати кілька цілей, кожна з яких супроводжується багатьма завданнями. Завданням – це так звана «дорожня карта», яка допоможе дістатися до «пункту призначення», тобто до реалізації цілі.

– *цільова аудиторія* – даний розділ дає детальний опис про те, хто сидітиме перед екранами, використовуючи ваше програмне забезпечення? Хто є вашими потенційними споживачами? До якої вікової категорії і статі вони належать? Де проживають? Який рівень доходів? Чого вони потребують найбільше і чого очікують від вас?

– *конкретні вимоги* – в даному розділі детально описуються вимоги до програмного забезпечення, до безпеки, до пошукової оптимізації та інші всі можливі аспекти. Для наочності, щоб уточнити кольорову гамму, розташування блоків та інші нюанси інтерфейсу, можливості розробки макетів, прототипів, використання шаблонів та їх елементів. Чим більше конкретики ви внесете, чим більше деталей продумаєте і точніше їх сформулюєте, тим бажаніший і очікуваніший результат отримаєте. Дуже часто виникає потреба врахувати нову інформацію, яка виявилася уже в ході роботи над проектом. У таких випадках технічне завдання може доопрацьовуватися, доповнюватися правками, додатками.

Документація програмного проекту ділиться на:

1. Технічні вимоги, або вимоги до системи (system requirements specification) – це документи в які містять опис функцій, які має містити програмний продукт, а також очікування замовника щодо продуктивності, відмовостійкості, надійності системи, середовища, в якій воно повинно

працювати і т.д. Часто вимоги є частиною контрактної документації чи технічного завдання.

Виконавець зацікавлений в деталізації і фіксації вимог якомога раніше, щоб уникнути перепланування, пов'язаного зі зміною вимог. На практиці, найчастіше, цього не відбувається - кожен проект в ході розробки в тій чи іншій мірі піддається зміні відповідно до змін у вимогах до програмного продукту.

2. Технічні специфікації (technical specification) – це документи, що містять опис архітектури програмного продукту і застосованих в ньому технічних рішень. Детальність і зміст технічних специфікацій в основному залежить від складності предметної області, нестандартності застосованих рішень, кваліфікації розробників і розподілу завдань між ними.

Мінімальний набір технічних специфікацій має містити загальний опис архітектури та інтерфейсів між компонентами, що створюються та мають підтримуватися при участі різних розробників. Тому розумним є такий рівень детальності технічних специфікацій, який здається злегка надмірним для розробників, які працюють над розробкою.

3. Відомості про випуск (release note) – це документи, що містять інформацію про випуски програмного продукту та описують фактично реалізовану функціональність і помилки, виправлені в даному випуску програмного продукту, а також різні особливості випуску: середовище, в якій продукт тестувався, відхилення від вимог, невиправлених помилок тощо. У певному сенсі відомості є звітним документом. При визначенні детальності опису функціональності та помилок в цьому документі в першу чергу необхідно брати до уваги вимоги замовника, а також мету створення відомостей для: первинної оцінки якості розробки, аналізу, підтримки тощо.

4. Керівництва – це документи, що призначені для надання користувачам допомоги в використанні програмного

забезпечення. Керівництва можуть бути для оператора-користувача, навчальна та адміністратора. Необхідність і формат подання керівництва оператора-користувача майже завжди диктуються замовником (більш детально керівництво оператора-користувача розглянуто далі). Керівництва або інструкції адміністратора можуть не створюватися в разі, якщо адміністративну підтримку системи передбачається доручити розробникам. При експлуатації інформаційної системи керівництва адміністратора забезпечує підтримку первинної інсталяції, штатного функціонування та відновлення програм і даних після збоїв і відмов. Керуюча діяльність адміністратора полягає в маніпулюванні керованими об'єктами і повинна описуватися, аналізуватися і регламентуватися сукупністю вимог і документів. Для цього необхідна повна документація про компонентах інформаційної системи (комп'ютерах, мережевих пристроях), які мають свої особливості в управлінні за допомогою спеціальних програмних компонентів, що підтримують адміністрування та управління системою. До основних функцій системи адміністрування, документи для яких підлягають розробці та оцінюванню, належать:

- планування використання пам'яті і продуктивності обчислювальної системи в робочому режимі застосування ПЗ, оперативне управління та розподіл ресурсів інформаційної системи;
- інсталяція та генерація робочих версій ПЗ для оперативних користувачів;
- управління та облік зовнішнього середовища при виконанні адаптації та реконфігурації конкретного ПЗ;
- виявлення, реєстрація та накопичення даних про збої та дефектах функціонування програм і даних;
- управління засобами захисту інформації та санкціонованого доступу користувачів, аналіз спроб злому

системи захисту, відновлення програм та інформації баз даних при спотвореннях;

- збір та узагальнення статистики про якість функціонування ПЗ в складі системи обробки інформації.

При оцінці необхідної деталізації інструкцій беруть до уваги, що підтримка, як правило, здійснюється персоналом, який часто змінюється і не має постійного контакту з розробниками. Відсутність інструкцій ускладнює передачу знань про програмне забезпечення та його призначення, що може призвести до спотворення розуміння функцій програмного забезпечення та їх використання, а наявність в системі не документованих особливостей рано чи пізно призводить до втрати знань про їх наявність та методи використання, що призводить до погіршення здатності користувача виконувати свої зобов'язання при використанні програмного забезпечення.

Окремий вид документації, що рекомендує можливі варіанти для реалізації процесів створення документації та її ведення – це методології и стандарти якості. Ніякий стандарт і ніяка модель менеджменту процесів проекту або модель якості не вимагають, щоб проектні документи мали ту чи іншу структуру, форму, назву або контент. Вони лише пропонують найбільш очевидні рішення, які не є обов'язковими. Вимоги стандартів і моделей полягають в тому, що інформація певного роду повинна бути зафіксована в тому чи іншому вигляді, при цьому не обов'язково в окремому документі.

Форма і структура представлення проектної документації для програмного забезпечення, найбільш повно відображають специфіку проектів. Розробка оптимальної форми і структури документації - одна зі складових діяльності щодо поліпшення процесів і один з предметів консалтингу будь-якої організації з розробки програмного забезпечення.

Що ж стосується відмінності «класичних», «полегшених» і agile-методик з точки зору управління документацією, представляється, то воно пов'язане лише з її формами, повнотою і детальністю, а також підходами до її еволюції і верифікації. Склад документації, що поставляється замовнику залишається практично незмінним, незалежно від методології оскільки він багато в чому визначається його потребами і звичками, так само як і її форма і зміст.

Крім проектної документації, технічного завдання та супроводжуючих розробку записів, існують також і інші програмні документи, що описують функції програмного забезпечення та принципи його використання. Всього існує чотири основних типи документації на програмне забезпечення (рис. 5.5).

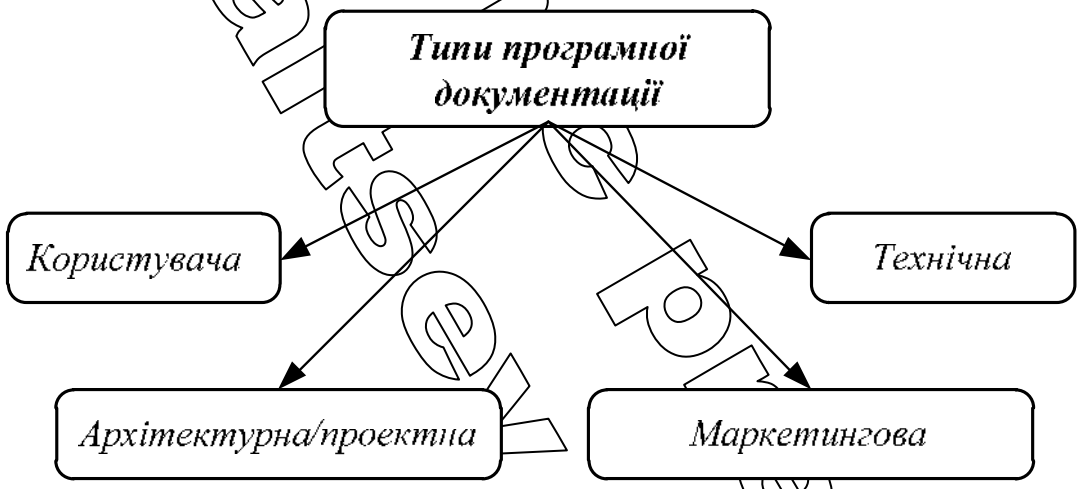


Рис. 5.5 – Типи програмної документації

1. Архітектурна або проектна – ця документація зазвичай описує продукт в загальних рисах та включає опис робочого середовища і принципів, які використані при реалізації програмного проекту. В проектному документі розробник приводить обґрунтування того, чому структури даних організовані саме таким чином. Описуються причини, чому який-небудь клас сконструйований певним чином, при необхідності приводяться рекомендації про виконання

еволюції програмного забезпечення в подальшому.

2. Технічна документація – це документація на код, алгоритми, інтерфейси, API та тощо. При створенні програми, одного лише коду, як правило, недостатньо. Повинен бути наданий певний текст, що описує різні аспекти того, що саме робить код. Така документація часто включається безпосередньо в вихідний код або надається разом з ним. Подібна документація має сильно виражений технічних характер і в основному використовується для визначення і опису API, структур даних і алгоритмів. При складанні технічної документації використовуються автоматизовані засоби - генератори документації. Вони отримують інформацію з спеціальним чином оформлених коментарів у вихідному коді, і створюють довідкові керівництва в будь-якому форматі, наприклад, у вигляді тексту або HTML. Використання генераторів документації та документують коментарів багатьма розробниками визнається зручним засобом, з різних причин. Зокрема, при такому підході документація є частиною вихідного коду, і одні і ті ж інструменти можуть використовуватися для складання програми і одночасної збірки документації до неї. Це також спрощує підтримку документації в актуальному стані.

3. Документація користувача – ця документація призначена для користувачів, адміністраторів системи та іншого персоналу. На відміну від технічної документації, сфокусованої на коді і тому як він працює, призначена для користувача документація описує лише те, як використовувати програмне забезпечення. Зазвичай, призначена для користувача документація вдає із себе керівництво користувача, яке описує кожну функцію програми, а також кроки, які потрібно виконати для використання цієї функції. Деяка документація надає інструкції про те що робити в разі виникнення проблем. Дуже

важливо, щоб документація була актуальною. Керівництво повинно мати чітку структуру; дуже корисно, якщо є наскрізний предметний покажчик. Логічна зв'язність і простота також мають велике значення.

Користувацька документація відображає необхідну зовнішню якість і якість у використанні, а також можливість освоєння й ефективного застосування ПЗ достатньо кваліфікованими фахівцями. Вона застосовується безпосередніми користувачами відповідно до функціонального призначення ПЗ, а також замовниками, покупцями і постачальниками програмних продуктів. Склад цієї документації формується з використанням частини технологічних документів з урахуванням вимог замовників або потенційних користувачів ПЗ.

Документація операторів-користувачів повинна забезпечувати коректну та кваліфіковану експлуатацію комплексу ПЗ в усьому діапазоні його характеристик, приписаних вимог замовника і зафіксованих метриками у використанні. Об'єктами розробки та оцінювання є документи на процедури і компоненти інтерфейсу із зовнішнім середовищем і з користувачами, що визначають ініціалізацію відповідних операцій, їх хід і результати, а також комфортність їх виконання. Повинно бути передбачено достатню якість ідентифікації помилкових дій і ситуацій, а також стандартизованої форми повідомлень про помилки користувачів.

Придбання, постачання, розробка, функціонування та супроводження програмних засобів в значній мірі залежить від кваліфікації фахівців-розробників. Тому експлуатаційної документацією обов'язково повинно підтримуватися ефективне навчання персоналу з метою його підготовки до придбання, постачання, застосуванню і супроводження програмного засобу. Процес навчання фахівців, контроль і

облік результатів навчання з оцінюванням досягнутої ними кваліфікації повинен гарантувати, що відповідні категорії навченого персоналу готові для виконання запланованих дій і рішення задач з певним програмним засобом.

Існує три підходи до організації користувальницької документації. Довідник з основних функцій (англ. Tutorial), найбільш корисні для нових користувачів та дозволяє послідовно навчати користувачів по ряду кроків виконанню будь-яких типових задач. Тематичний підхід, при якому кожна глава керівництва присвячена якійсь окремій темі. В останньому, третьому підході, команди або завдання організовані у вигляді алфавітного довідника.

У багатьох випадках розробники програмного продукту обмежують набір користувальницької документації лише вбудованою системою допомоги (англ. Online help), що містить довідкову інформацію про команди або пунктах меню. Робота з навчання нових користувачів перекладається на приватних видавців.

4. Маркетингова документація – це документація, що найчастіше містить план рекламної роботи; стимулювання продажів; визначення концепції рекламного звернення та окремих його тем; вибір засобів і носіїв реклами; конкретизації завдань рекламних звернень з урахуванням вибраних засобів і носіїв реклами; вибір жанрів і форм рекламних звернень; розроблення загального бізнес-плану комунікацій та його невід'ємної частини – рекламної діяльності при просуванні програмного продукту; створення рекламних звернень (визначення творчих підходів, написання тексту або сценарію, розроблення оригінал-макета); розміщення рекламних звернень у засобах масової інформації (у вибраних засобах і носіях реклами); контроль за перебігом показу рекламного звернення цільовій аудиторії, вимірювання

ефективності цього показу та (за необхідності) оперативне коригування його перебігу та тощо.

При просуванні на ринок багато програмних додатків потребують проведення рекламних заходів та створення супроводжуючих рекламних матеріалів, з тим щоб зацікавити людей, звернувши їхню увагу на програмний продукт. Така форма документації має на меті:

– зацікавити потенційних користувачів;

– інформувати для чого призначено програмне забезпечення;

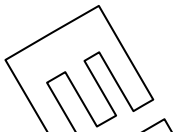
– інформувати, що і як робить програмне забезпечення;

– пояснити переваги використання програмного продукту в порівнянні з існуючими рішеннями.

Дуже часто буває так, що коробка в якій постачається програмне забезпечення з необхідною програмною документацією та іншими маркетинговими матеріалами дають більш ясну картину про можливості та способи використання програмного забезпечення, ніж всі інші типи документації.

Таким чином, для всіх етапів життєвого циклу існують свої ***типи та види програмної документації***, що забезпечує потреби розробників і користувачів, залучених до створення та використання програмного забезпечення необхідною та самодостатньою інформацією про функціональність та склад програмного забезпечення.

5.5. ОНОВЛЕННЯ ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ



Необхідність оновлення програмної документації (версії програмної документації) пов’язана з внесенням змін у вимоги до програмного продукту. Такі зміни можуть виникати через невиявлені вимоги на етапі початкового проектування програмної системи, не чітко сформовані вимоги до програмної системи, а також через зміну вимог в процесі розробки програмної системи.

У зв’язку з тим, що програмна документація складається з ряду пов'язаних документів, зміни, виконані в одному з документів, можуть бути причиною корекції інших документів або різних частин одного і того ж документа. Процедура оновлення документації включає вирішення задачі з пошуку місць документа, що потребують корекції і корекцію інформації в відповідних місцях. Для полегшення виконання завдання пошуку при створенні документації рекомендується виконувати наступні правила створення документа:

1) всередині кожної секції і (або) розділу матеріал повинен бути згрупований відповідно до описаними елементами. У групі елементи повинні розташовуватися відповідно до прийнятих в команді розробників/організації з розробки програмного забезпечення принців впорядкованості, наприклад відповідно до алфавітного порядку імен модулів програмного забезпечення або у відповідності зі схемою їх функціонування, або архітектурою та тощо;

2) всередині документа і кожної його секції повинна витримуватися єдина структура нумерації, яка полегшує пошук;

3) назва секції, номер версії документа і дата його останнього затвердження повинні бути присутніми на кожній сторінці програмної документації;

4) структура і оформлення всіх документів повинні бути однотипними.

Виконання цих правил дозволяє різним користувачам створювати документацію, яка підходить для них та може бути легко передана наступним розробникам.

В ході реалізації програмного проекту, зазвичай, документація модифікується в декілька кроків. Розробник починає свою роботу з вимог користувача до майбутньої системи ПЗ. На основі зібраних вимог розробляються цілі та специфікація вимог як документ, що може використовуватися для зовнішнього проектування ПЗ. Наступним документом може бути опис архітектури системи, що визначає логічні деталі різних компонент і всієї системи в цілому. Кінцевим документом є технічне завдання або аналогічний йому документ, що відображає структуру програмного забезпечення і дозволяє приступити до створення програмного забезпечення.

Кожен крок у процесі розробки практично вимагає перегляду документів попереднього кроку, так як кожний крок з просування розробки над програмним проектом веде до деталізації проекту та приводить до необхідності в несенні змін до програмної документації. Всі зміни, що вносяться в документ повинні знайти відображення в попередніх документах. Отже, процес буде повторюватися кілька разів за хід виконання проекту. Кожне таке повторення потребує управління програмними документами. Для управління програмними документами застосовують декілька методів, серед яких:

• Збір документів: здатність ефективно зберігати електронні та паперові документи різних форматів в центральному сховищі. Збір документів - це не тільки збереження інформації організованим чином, але і здатність легко витягувати істотну інформацію з документів і з архівних

даних за минулі періоди.

• Контроль версій: можливість надати параметри перевірки на вході і перевірки на виході і надати різні рівні безпеки, такі як доступ на читання і на запис, щоб гарантувати цілісність даних, що знаходяться в збережених документах.

• Технологічні процеси: можливість створювати і застосовувати настроюються технологічні процеси, відповідні бізнес-процесів і затвердженим процесу документообігу у вашій організації.

• Звітність і аналіз: можливість обмінюватися інформацією між документами, а також узагальнювати дані в безлічі документів з метою складання звітів і проведення аналізу, щоб забезпечити краще розуміння (ясність) у вашій організації.

• Співпраця: здатність розподіляти документи між відповідними зацікавленими особами, а також обмежувати доступ до документів тим людям, які не повинні мати до них доступ.

При завершенні проектування, всі позначення і посилання між документами повинні бути перевірені. Крім того, необхідно переконатися у відповідності функціональних і програмних описів. Всі розбіжності повинні бути усунені в документації та в зв’язках між документацією. Так як документування складної системи ПЗ і його оновлення вимагають величезних зусиль та в багатьох проектах для автоматизації роботи над програмною документацією використовують спеціалізоване програмне забезпечення. Останнє призначене для виконання операцій з запам'ятовування документації, вибіркового пошуку інформації з видачею за запитами, підтримки ведення версій змін у документації та тощо.

За допомогою спеціалізованого програмного забезпечення можливо згенерувати перехресні посилання між

будь-якими іменами або позначеннями, які можуть бути надані користувачеві. Наприклад, користувач може швидко знайти, якими модулями використовуються деякі набори даних або підпрограм. Ця можливість може бути особливо корисна, якщо над розробкою системи ПЗ працювала велика група програмістів з віддаленим доступом до проекту. При зберіганні в пам'яті імена розробників модулів, то можна досить швидко встановити список модулів за певним розробником, порушення в змінах, і визначити завдання на корекцію документації для конкретного виконавця.

Таким чином, ***основною ціллю оновлення документації до програмного забезпечення*** – забезпечення створення програмної документації, що відображає актуальний функціонал програмного забезпечення і дозволяє забезпечення потреб розробників і користувачів, залучених до створення та використання ПЗ необхідною та самодостатньою інформацією потрібною для розробки, виготовлення, експлуатації та супроводу ПЗ.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Які основні цілі документування.
2. Навести характеристику цілей та завдань документування проектів, пов'язаних з розробкою програмного забезпечення.
3. Проаналізуйте життєвий цикл документа з прив’язкою до стандартних типів життєвих циклів розробки програмного забезпечення.
4. Який типовий склад та зміст документації до готової програмної системи?

ГЛОСАРІЙ

.NET платформа (Microsoft NET Framework) – програмна технологія, призначена для створення як звичайних програм, так і веб-додатків (у якості платформи для розробок запропонована фірмою Microsoft).

ADL (Architecture Description Language) – мова описання архітектури.

ANSI (American National Standards Institute) – Американський національний інститут стандартів.

AOP (Aspect-Oriented Programming) – це методика програмування, що базується на понятті аспекту – блоку коду, що інкапсулює наскрізну поведінку у складі класів та модулів, що повторно використовуються. Вона надає засоби для виділення наскрізної функціональності в окремі модулі, що полегшує роботу із компонентами програмної системи та знижує складність системи в цілому. Аспектно-орієнтоване програмування (АОП) є однією з концепцій програмування, яка продовжує розвиток процедурного та об`єктно-орієнтованого програмування.

AOSD (Aspect-Oriented Software Development) – технологія, що може застосовуватися для поліпшення модульності та адаптації програмної системи в складних та великомасштабних розподілених системах.

API (Application Programming Interface) – програмний інтерфейс додатка.

AR (Acceptance Review) – огляд результатів приймальних випробувань.

AutoCAD (Auto Computer-Aided Design) – двох- і трьох вимірна система автоматизованого проектування.

CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing) – система автоматизованого проектування/система автоматизованого виробництва.

CASE (Computer-Aided Software Engineering) – система автоматизованої розробки програм.

CASE-засоби – програмні засоби, що підтримують процеси створення і супроводу програмних продуктів, включаючи аналіз і формулювання вимог, проектування

продукту і баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне управління і управління проектом, а також інші процеси. CASE-засоби разом із системним програмним забезпеченням і технічними засобами утворюють повну середу розробки програмних систем.

CASE-технологія (Computer Aided Software Engineering) - технологія, що представляє собою методологію проектування автоматизованих систем, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки і супроводу програмних систем і розробляти додатки у відповідності з інформаційними потребами користувачів.

CCIE (Certified Cisco Internetwork Expert) – сертифікований компанією Cisco експерт з об'єднаних мереж.

CCNP (Cisco Certified Network Professional) – сертифікований професіонал мереж.

CCNA (Cisco Certified Network Associate) – сертифікований фахівець мереж.

CBSD (Component-based software development) – розробка промислового ПЗ, що скерована на побудову великих програмних систем, шляхом інтегрування раніше розроблених програмних компонентів. Підвищуючи гнучкість і надійність систем, такий підхід одночасно дозволяє знижувати вартість розробок програмного забезпечення, прискорення інтеграції кінцевих продуктів і скорочення строків проходження етапів життєвого циклу великих програмних систем на рівнях підтримки й відновлення. З даною технологією пов’язаний процес компонентного проектування ПЗ (component-based software engineering, CBSE), а також комерційні стандартні (off-the-shelf, COTS) програмні компонентні продукти.

CBSE (component-based software engineering) – концепція, що з’явилася в середині 1990-х рр. і полягала в проектуванні і розробці архітектури програмних систем, побудованих на процесі створення й об’єднання

високоякісних компонентів. Базується на моделях COM, DCOM і CORBA.

CDR (Critical Design Review) – критичний огляд проекту.

CNE (Certified Novell Engineer) – сертифікований інженер.

COM (Component Object Model) – об'єктна модель компонентів.

DAOP (Dynamic Aspect-Oriented Platform) – надає послуги розподілених додатків, а також механізми компонування з включенням сторонніх компонентів до компонентів програмного забезпечення під час його виконання.

DCOM (Distributed Component Object Model) – розподілена модель компонентних об'єктів.

DDL (Data Definition Language) – мова визначення даних.

DHCP (Dynamic Host Configuration Protocol) – протокол динамічної конфігурації вузла.

DLL (Dynamic-link Library) – бібліотека динамічного компонування.

DNS (Domain Name System) – система доменних імен.

DSDL (Document Schema Definition Languages) – мови опису схеми документа.

DWH (Data Warehouse) – сховище даних.

EIA (Electronic Industry Association) – асоціація електронної промисловості.

EOSDIS (Earth Observing System Data and Information System) – база даних системи спостереження землі.

EULA (End User License Agreement) – ліцензійна угода кінцевого користувача.

FPP (Full Package Product) – коробкові версії продуктів.

FRR (Flight Readiness Review) – огляд стану готовності до льотних випробувань.

FSF (Free Software Foundation) – фонд вільного програмного забезпечення.

GNU (General Public License) – відкрита ліцензійна угода.

HTTP (HyperText Transfer Protocol) – протокол пересилання гіпертексту.

IEC (International Electrotechnical Commission) – міжнародна комісія по електротехніці.

IEEE (Institute of Electrical and Electronic Engineers) – інститут інженерів по електротехніці та електроніці.

IFIP (International Federation of Information Processing) – міжнародна федерація з обробки інформації.

ISBL (Information Systems Base Language) – базова мова інформаційних систем.

ISO (International Standards Organization) – міжнародна організація зі стандартизації.

IT (Information Technology) – інформаційні технології.

ITIL (IT Infrastructure Library) – бібліотека інфраструктури інформаційних технологій.

ITSM (IT Service Management) – управління послугами IT.

MCP (Microsoft Certified Partner) – сертифікований партнер Microsoft.

MCPD (Microsoft Certified Professional Developer) – сертифікований професійний розробник Microsoft.

MCSE (Microsoft Certified Systems Engineer) – сертифікований системний інженер Microsoft.

MCTS (Microsoft Certified Technology Specialist) – сертифікований технічний спеціаліст Microsoft.

MDR (Mission Definition Review) – огляд визначення програми запуску.

MFC (Microsoft Foundatlion Classes) – бібліотека класів.

MSDN (Microsoft Developer Network) – підрозділ компанії Майкрософт, яка відповідає за взаємодію фірми з розробниками, які цікавляться операційною системою, а також розробники, що використовують програмні інтерфейси операційної системи і скриптові мови різних додатків, розроблених Microsoft. Така взаємодія з розробниками має кілька форм: веб-сайти, новинні розсилки, конференції розробників, блоги, розсилка CD / DVD. Життєвий цикл взаємодії з розробниками варіюється від підтримки вже застарілих продуктів до поширення інформації про нові продукти та їх можливості.

MSF (Microsoft Solutions Framework) – методологія розробки програмного забезпечення.

MySQL – вільна система управління базами даних.

ODBC (Open Database Connectivity) – відкритий зв'язок з базами даних.

OEM (Original Equipment Manufacture) – фабрика оригінального обладнання.

ORR (Operation Requirement Review) – огляд виконання вимог з експлуатації.

PDR (Preliminary Design Review) – попередній огляд проекту.

PHP (Hypertext Preprocessor) – препроцесор гіпертексту.

PRR (PreliminaryRequirementReview)–попередній огляд вимог.

QR (QualificationReview) – огляд результатів кваліфікаційних випробувань.

RAD (Rapid Application Development) – швидке створення додатків.

RM-ODP (The Reference Model of Open Distributed Processing) – еталонна модель для відкритої розподіленої обробки.

RSI (Repetitive Strain Injury) – травма від повторюваних навантажень м'язів.

SAAM (Software Architecture Analysis Method) – метод оцінки архітектури системи.

SCA (Sun Certified Associate) – сертифікований програміст початкового рівня для платформи J2SE.

SCD (Sun Certified Developer) – сертифікований розробник Sun для платформи Java.

SEI (Software Engineering Institute) – Інститут програмної інженерії.

SOAP (Simple Object Access Protocol) – простий протокол доступу до об'єктів.

SQL (Structured Query Language) – мова структурованих запитів.

SRR (SystemRequirementReview) – огляд виконання системних вимог.

TDD (Test Driven Development) – розробка через тестування.

TDS (Tabular Data Stream) – протокол передачі табличних даних.

UML (Unified Modeling Language) – уніфікована мова об'єктно-орієнтованого моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення

VBA (Visual Basic for Applications) – Visual Basic для додатків.

VPN (Virtual Private Network) – віртуальна приватна мережа.

Windows API (application programming interfaces) - загальназвацілогонаборубазовихфункційінтерфейсівпрограмуваннядодатківопераційнихсистемсімейств Windows і Windows NT корпорації Microsoft. Єнайбільшимпрямимспособомвзаємодіїзастосунківзос Windows.

WINS (Windows Internet Name Service) – служба визначення адрес.

Абстрагування від проблеми – ігнорування ряду подробиць з тим, щоб звести задачу до більш простого завдання.

Абстрактна машина Дейкстри – застосовується в проектуванні архітектури системи, найнижчий рівень абстракції - це рівень апаратури. Кожний рівень реалізує абстрактну машину з дедалі більшими можливостями.

Абстрактний батьківський клас – батьківський клас, який не має примірників об'єктів.

Автоматизована система (АС) – організаційно-технічна система, що забезпечує вироблення рішень на основі автоматизації інформаційних процесів у різних сферах діяльності (управління, проектування, виробництво тощо) або їх поєднаннях, система, яка складається з персоналу та комплексу засобів автоматизації його діяльності, що реалізує інформаційну технологію виконання установлених функцій.

Автономне тестування (тестування модуля) (module testing) – контроль окремого модуля в ізольованому середовищі (наприклад, за допомогою ведучої програми), інспекція тексту модуля на сесії програмістів, яка іноді доповнюється математичним доказом правильності модуля.

Адреса – число, яке однозначно визначає розташування елемента інформації в пам'яті мікрокомп'ютера.

Адрес виконання – фактична адреса інформаційного елемента в пам'яті, який вираховується мікропроцесором в ході виконання команди.

Адресна команда – команда, яка винна звернути до пам'яті мікрокомп'ютера з метою вилучення операції або збереження результату операцій.

Акумулятор – регістр в складі арифметичний-логічного пристрою призначений для тимчасового зберігання операції або результату операції.

Алгоритм – послідовність операцій, які однозначно визначають вирішення певної задачі.

Алгоритм – строго однозначно визначена для виконавця послідовність дій, що призводять до вирішення завдання.

Альфа тестування (системне тестування, лабораторні випробування) - фаза тестування, виконувана розробниками для підтвердження, що всі фрагменти правильно інтегровані в систему, а сама система працює надійно.

Аналого-цифровий перетворювач (АЦП) – це пристрій для конвертації аналогового сигналу в цифровий сигнал для зберігання та подальшої обробки в мікрокомп'ютері.

Апаратне забезпечення – комплекс електронних, електричних і механічних пристроїв, що входять до складу системи або мережі.

Артефакт реалізації – щось, що не можна виявити в постановці розв'язуваної задачі, але необхідне для складання програми.

Архітектура мікрокомп'ютера – сукупність важливих характеристик мікрокомп'ютера: розрядність слова, формати и система команд, режими адресації пам'яті, склад програмно-доступних регістрів, спосіб адресації до зовнішніх пристроїв и т.п.

Архітектура системи – структура об'єднання кількох програмних засобів в одне ціле.

Атестація (certification) – авторитетне підтвердження правильності програми.

База даних (БД) – інформаційна модель, що дозволяє в упорядкованому вигляді зберігати дані про групу об'єктів з однаковим набором властивостей або пойменовану сукупність структурованих даних (пойменована сукупність структурованих даних предметної області).

Байт – послідовність з 8 бітів, який розглядається як один цілісний елемент даних або пам'яті.

Батьківський клас – початковий клас, від якого успадковуються класи-нащадки.

Батько – безпосередній клас-предок, який стоїть біля кореня схеми ієрархії, і від якого породжуються перші нащадки, а від нащадків ще нащадки.

Бета-тестування – це фаза загального тестування, при якій програмний виріб поставляється обмеженому колу кінцевих користувачів для більш жорсткого тестування.

Біт – найпростіший елемент кодупрограми, може складається з одного із двох елементів (0 и 1).

Валідація – підтвердження того, що окремі вимоги, які стосуються певного передбаченого використовування, виконуються, і яке здійснюється шляхом перевіряння та забезпечення об'єктивних доказів [ДСТУ 3918].

Веб-програмування – розділ програмування, що розвивається досить бурхливо, орієнтований на розробку динамічних Internet додатків.

Вектор переривання – ділянку пам'яті мікрокомп'ютера, що містить адресу програми обробки переривання і слово, що характеризує стан процесора (ССП) в даний момент часу.

Велика інтегральна схема (БІС) – надмініатюрна електронна мікросхема, яка реалізована на напівпровідниковій пластині площею менше 1 $см^2$ та містіть сотні й тисячі електронних елементів (резистори, конденсатори, транзистори, діоди и т. и) і виконує певні покладені на неї функції.

Верифікація – підтвердження погодженості результатів, отриманих на кожній стадії розроблення програмного забезпечення, з вимогами, встановленими на попередніх етапах [ДСТУ 3918] або встановлення того, що поведінка програми співпадає з очікуваною поведінкою.

Версія – це різновид елементу, який підлягає ідентифікації [ДСТУ 3918]

Визначення вимог – збір та аналіз вимог замовника виконавцем і подання їх у нотації, яка є зрозумілою як для замовника, так і для виконавця.

Випробування – спроба знайти помилки, виконуючи програму в заданому програмному середовищі.

Відновлюваність програмного забезпечення – властивість, що характеризує можливість пристосовуватися до виявлення помилок та їхнього усунення.

Візуальне моделювання – процес графічного представлення моделі за допомогою деякого стандартного набору графічних елементів або це спосіб створення програм шляхом маніпулювання графічними об'єктами замість написання програмного коду в текстовому вигляді. Мови візуального програмування можуть бути додатково класифіковані в залежності від типу і ступеня візуального вираження.

Вісімкова система числення – система числення з основою 8, в якій для запису чисел використовуються восьмеричні цифри 0,1,2, 3,4, 5, 6, 7.

Властивості ПЗ – об'єктивні властивості ПЗ, які виявляються у процесі створення або експлуатації і обумовлюють його розбіжність або подібність з будь-якими іншими об’єктами та можуть бути базовими (загальними), властивими даному класу продукції, і специфічними, які характерні тільки для окремого виду програмного забезпечення. За ступенем впливу на якість ПЗ розрізняють істотні та несуттєві властивості [ДСТУ 2850].

Властивості (property) – це певним чином оформлені методи, призначені як для читання і контрольованої зміни внутрішніх даних об'єкта (полів), так і виконання дій, пов'язаних із поведінкою об'єкта.

ВОІВ – Всесвітня Організація Інтелектуальної Власності.

Впровадження – стадія, після завершення якої, програмну документацію розмножено в потрібній кількості, програму встановлено і вона супроводжується, користувачі навчені.

Гіпермедіа – термін, введенийТедомНельсоном, івикористанийуйогороботі Complex information processing: a file structure for the complex, the changing and the indeterminate (1965 рік). Гіпермедіа - це гіпертекст, в який включено графіку, звук, відео, текст і посилання, для того щоб створити основу нелінійного середовища інформації. Гіпермедіа співвідноситься з визначенням мультимедіа, яке використовується, щоб описати неінтерактивні послідовні дані так само як і гіпермедіа.

ГСТУ – Галузевий Стандарт України.

Дані – цифрова інформація, з якою має справу комп'ютер (число, код символу і т.п.)

Двійкова система числення – система числення з основою 2, в якій для запису чисел використовуються двійкові цифри 0 і 1.

Декомпіляція – декодування програми таким чином, щоб получити похідний код в вигляді мов високого рівня.

Деструкція – особливий метод самого об'єкта, що забезпечує знищення даного об'єкта.

Динамічна змінна – це так би мовити статична змінна, але така, що розміщується в особливій області пам'яті поза кодом програми. У будь-який момент часу пам'ять для розміщення динамічних змінних може, як виділятися, так і звільнятися.

Динамічне зв'язування – асоціація запиту з об'єктом і однією з його операцій під час виконання.

Дисплей – зовнішній пристрій для візуального відображення інформації з пам'яті мікрокомп'ютера на екрані, панелі і інших фізичних засобах.

Довжина слова – число бітів в машинному слові (зазвичай кратне 8),

Додатковий код – форма представлення негативних чисел в мікрокомп'ютері.

Доказ (proof) – спроби знайти в програмі помилки шляхом доказів на основі математичних теорем про правильність програми, безвідносно до зовнішнього програмного середовища.

Документ – документ, виконаний за заданою формою, в якому представлено будь-яке проектне рішення.

ДСТУ – Державний Стандарт України.

Експертна система – це методологія адаптації алгоритму успішних рішень одної сфери науково-практично\ діяльності в іншу. З поширенням компютерних технологій це тотожна (подібна, основана на оптимізуючому алгоритмі) інтелектуальна комп'ютерна програма, що містить знання та аналітичні здібності одного або кількох експертів у відношенні до деякої галузі застосування і здатна робити логічні висновки на основі цих знань, тим самим забезпечуючи вирішення специфічних завдань (консультування, навчання, діагностика, тестування, проектування тощо) без присутності експерта (спеціаліста в конкретній проблемній галузі). Також визначається як система, яка використовує базу знань для вирішення завдань (видачі рекомендацій) в деякій предметній галузі.

Експлуатаційна документація – частина робочої документації на програму, призначена для використання при експлуатації програми, яка визначає правила дії персоналу і користувачів програми при її функціонуванні, перевірці та забезпеченні її працездатності.

Екстремальне програмування (extreme programming) (XP) - адаптивний інженерний підхід, раціональне поєднання відомих методів та їх сукупного використання дає істотні результати й успішно виконані проекти при розробці невеликих систем, вимоги до яких чітко не визначені й цілком можуть змінитися.

Енергозалежна пам'ять – запам'ятовуючий пристрій, що втрачає данні при відключенні живлення.

Ескізний проект (ЕП) – комплект проектних документів на програму, який розробляється на стадії "Ескізний проект", затверджений в установленому порядку, що містить опис декількох альтернативних варіантів реалізації майбутнього

виробу й уточнені вимоги на основі їх аналізу. Ступінь опрацювання при цьому повинна бути достатньою лише для досягнення можливості порівняння варіантів.

Етап проекту – зазвичай частина стадії проекту, виділена з міркувань єдності характеру робіт та (або) завершального результату або спеціалізації виконавців.

Єдина система програмної документації (ЕСПД) – комплекс державних стандартів, що встановлює взаємопов'язані правила розробки, оформлення та обігу програм і програмної документації.

ЄЕС – Європейське Економічне Співтовариство.

Ємність пам'яті – найбільший обсяг даних, виражений в машинних одиницях інформації (бітах, байтах або словах), який може зберігатися в запам'ятовуючому пристрої комп'ютера.

Життєвий цикл ПЗ – весь період існування ПЗ з початку розроблення до завершення його використовування.

Життєвий цикл (ЖЦ) – база для природної систематизації інструментів і методів, ресурсів і результатів на різних етапах розробки та використання програмних систем. Сукупність взаємопов'язаних процесів створення та послідовної зміни стану продукції від формування до неї вихідних вимог до закінчення її експлуатації або споживання.

Забезпечення якості – всі дії, які планують і регулярно здійснюють в рамках системи якості та належним чином демонструють з метою забезпечення достатньої впевненості у тому, що об'єкт буде задовольняти вимогам, що виставляються.

Заглушка – макет ще не реалізованого модуля, необхідний при низхідній реалізації, являє собою найпростішу підпрограму або без дій, або з діями виведення вхідних даних, або повертає до вищестоящих модулів тестові дані (які зазвичай присвоюються всередині заглушки), або містить комбінацію цих дій.

Замовник – організація, яка замовляє або закуповує систему, програмний продукт або послуги від постачальника.

Зв'язність – особливо продуманий логічний пристрій збереження цілісності структури даних, елементи якої можуть

перебувати в довільних, несуміжних, неконтрольованих за адресацією ділянках динамічно розподіленої пам'яті поза кодом програми.

Знак відповідності - зареєстрований в законодавчому порядку сертифікаційний знак, що використовується у встановленому порядку третьою стороною для продукції (послуги), яка відповідає вимогам нормативного документу, який застосовується при сертифікації.

Ієрархія - підпорядкованість.

Інженер (від лат. ingenium - природний розум, винахідливість) - фахівець із вищою технічною освітою, творець інформації про архітектуру матеріального засобу досягнення мети чи способу виготовлення цього засобу (продукту) і здійснюючий керівництво і контроль за виготовленням продукту.

Інженерія програмування (англ. software engineering - розробка програмного забезпечення) – інженерна справа, творча технічна діяльність. Інженерія спирається на специфічні методи та методики, у тому числі евристичні. Інженерія вивчає різні методи та інструментальні засоби з точки зору певних цілей, тобто має очевидну практичну спрямованість. Основна ідея інженерії програмування в тому, що розробка програмного забезпечення є формальним процесом, який можна вивчати, висловлювати в методиках і вдосконалювати. Головна відмінність між технологією програмування і програмною інженерією полягає у способі розгляду та систематизації матеріалу. У програмній інженерії вивчаються, перш за все, методи та інструментальні засоби розробки програм з точки зору досягнення певних цілей - вони можуть використовуватися в різних технологічних процесах (і в різних технологіях програмування).

Інженер-програміст – найменування посади відповідно до кваліфікаційного довідника посад керівників, спеціалістів та інших службовців, фахівець зі створення та експлуатації програм.

Інженер-системотехнік – найменування посади відповідно до кваліфікаційного довідника посад керівників, спеціалістів та інших службовців, інженер інженерів, фахівець

із вирішення проектних завдань створення таких особливо складних штучних систем як автоматизовані системи.

Інкапсуляція – це механізм, що об'єднує дані й код, який маніпулює цими даними, а також захищає і те, й інше від зовнішнього втручання або неправильного використання.

Інтегроване середовище розробки – це комп'ютерна програма, що допомагає програмістові розробляти нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче

Інтегровані структури даних – структури даних, складовими частинами яких є інші структури даних - прості або у свою чергу інтегровані. Інтегровані структури даних конструюються програмістом з використанням засобів інтеграції даних, що надаються мовами програмування.

Інтерпретатор – програма чи технічні засоби, необхідні для виконання інших програм, вид транслятора, який здійснює пооператорну (покомандну) обробку, перетворення в машинні коди та виконання програми або запиту (на відміну від компілятора, який транслює в машинні коди всю програму без її виконання).

Інтерфейс – сукупність апаратних і програмних засобів для з'єднання двох і більше пристроїв з метою обміну даними між ними або це набір форматів допустимих повідомлень. Для виключення можливих, але неприпустимих повідомлень, використовується механізм приховування інформації.

Інформаційна система – сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.

Кваліфікація – процес демонстрації придатності об'єкта виконувати задані вимоги.

Клас – представляє собою об'єднуючу концепцію набору об'єктів, що мають спільні характеристики

Код операції – частина машинної мови, яка називається інструкцією та визначає операцію, яка повинна бути виконана процесором.

Код – форма подання інформації в пам'яті мікрокомп'ютера.

Кодувальник програм – програміст, який пише і автономно тестує код компонентів програм.

Команда (інструкція) – послідовність бітів, яка визначає крок виконання програми і містить код операції і, можливо, один або декілька операндів.

Компілятор – комп'ютерна програма (або набір комп'ютерних програм), що перетворює (компілює) програмний код, написаний певною мовою програмування, на семантично еквівалентний код в іншій мові програмування, який, як правило, необхідний для виконання програми машиною, наприклад, комп'ютером.

Комплексне тестування (system testing) – контроль та/або випробування системи по відношенню до вихідних цілей. Є процесом контролю, якщо воно виконується в середовищі, яке моделюється, і процесом випробування при виконанні в реальному середовищі.

Контракт – двостороння угода, якій, як правило, надано юридичну силу, або подібна внутрішня угода.

Контролер – пристрій для управління передачею даних між мікропроцесором і зовнішнім пристроєм.

Контроль (verification) – спроба знайти помилки, виконуючи програму в тестовому середовищі або в середовищі, що моделюється .

Коректність програмного забезпечення – властивість безпомилкової реалізації необхідного алгоритму, при відсутності таких чинників як: помилки вхідних даних, помилки операторів ЕОМ (людей), збоїв і відмов ЕОМ.

КПК – кишеньковий комп'ютер.

Купа – це сховище пам'яті, що розташоване в ОЗП та допускає динамічне виділення пам'яті.

Логічна структура даних – розгляд структури даних без урахування її подання до машинної пам'яті.

Локальна мережа – мережа з'єднаних між собою комп'ютерів, розгорнута на обмеженій території (наприклад, в межах промислового підприємства або одного цеху).

Магістраль – набір провідників, що використовуються в якості каналу зв'язку між декількома компонентами мікрокомп'ютера.

Маркер або курсор – особливий рухливий знак на екрані дисплея, який вказує конкретну позицію для відображення або заміни символу. Положення маркера на екрані дисплея можна змінювати за допомогою клавіш клавіатури, а також програмним шляхом.

Маскування переривання – тимчасове або постійне заборона переривання шляхом установки відповідного біта в певному регістрі.

Машинна мова (машинний код) – мова, алфавітом якого є лише виконавчі цифри 0 і 1. Ця мова використовується для написання програм у вигляді послідовності двійкових кодів. Для деякого полегшення програмування замість довічного нерідко використовується восьмеричний або шістнадцятковий код.

Машинне слово – одиниця інформації в мікрокомп'ютері, що складається зазвичай з одного, двох або більше байтів (в залежності від типу мікрокомп'ютера) і розглядається як одне ціле.

Меню – це набір елементів управління, у деяких випадках вкладений (тобто що має більш одного рівня). Цей набір звичайно відповідає функціям, виконання яких може знадобитися користувачу при роботі з поточним матеріалом.

Метод – спосіб практичного здійснення чого-небудь.

Методи об'єкта (methods, member functions) – підпрограми, що реалізують дії (виконання алгоритмів) у відповідь на їх виклик у вигляді відданого повідомлення;

Методика – сукупність методів практичного виконання чого-небудь.

Методологія (від грец.Methodos і logos - слово, вчення про методи) – система принципів і способів організації та побудови теоретичної і практичної діяльності, а також вчення про цю систему. Методологія програмування вивчає методи з точки зору основ побудови. Це об'єднана єдиним філософським підходом сукупність методів, що застосовуються в процесі розробки програмних продуктів. Будь-яка методологія створюється на основі вже накопичених у предметній області емпіричних фактів і практичних результатів.

Механізм приховування інформації – механізм, який використовується для виключення можливих, але неприпустимих повідомлень об'єктам.

Міграція програмного забезпечення – заміна існуючого програмного забезпечення аналогами з метою підвищення безпеки та зниження залежності від виробника (розробника), оптимального вибору і настроювання ПЗ для вирішення конкретних задач.

Мікрокомп'ютер (мікроЕОМ, мікропроцесорна система) – мініатюрна обчислювальна машина, що складається змікропроцесора, пам'яті, пристрої введення-виведення і блоків з'єднання з пристроями введення-виведення (контролерів).

Мікропроцесор – програмно-керований електронний цифровий пристрій, призначений для обробки інформації, представленої в цифровому вигляді, і побудоване на одній або декількох інтегральних мікросхемах.

Мікропроцесорний комплект – сукупність інтегральних мікросхем і інших мікросхем, сумісних між собою по конструктивному і технологічному виконанню та призначених для спільного застосування.

Мнемокод – символічний запис машинних команд мікрокомп'ютера.

Множинне успадкування класів – успадкування, при якому кожен клас може, в принципі, породжуватися від одного або відразу від декількох батьківських класів, наслідуючи поведінку всіх своїх предків.

Мова асемблера – низькорівнева мова програмування, яка використовує мнемоніку, інструкції та операнди для представлення машинного коду.

Мова асемблера – низькорівнева мова програмування, яка використовує мнемоніку, інструкції та операнди для представлення машинного коду.

Мова програмування – мова для запису програм у вигляді послідовності операторів або формальна знакова система, призначена для запису комп'ютерних програм. Мова програмування визначає набір лексичних, синтаксичних і

семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням.

Мови веб-програмування – це відповідно мови, які в основному призначені для роботи з Інтернет-технологіями. Мови веб-програмування діляться на дві групи: клієнтські та серверні.

Модель життєвого циклу – концептуальна структура, яка включає процеси, дії і задачі щодо розроблення, експлуатації та супроводу програмного забезпечення, і яка охоплює життєвий цикл системи, починаючи з визначення вимог до неї і закінчуючи завершенням її використовування.

Модель компонентних об'єктів (Component Object Model (COM) - специфікація методу створення компонентів і побудови з них програм.

Модель – один об'єкт або система може виступати в ролі моделі іншого об'єкта або системи, якщо між ними встановлено схожість в якомусь сенсі.

Модуль – фундаментальне поняття і функціональний елемент технології структурного програмування, підпрограма, але оформлена у відповідності з особливими правилами, файл (unit) з описами споріднених класів.

Модульність програм – основний принцип технології структурного програмування, характеризується тим, що вся програма складається з модулів.

Мультимедіа (multimedia) – це взаємодія візуальних і аудіоефектів під управлінням інтерактивного програмного забезпечення.

Налагодження (debugging) – є засобом встановлення точної природи помилок.

Наскрізний структурний контроль – використання на багатьох етапах проекту контролю коректності специфікації зв'язків частин програми.

Нащадок – клас, який використовує характеристики іншого класу за допомогою наслідування.

Незалежна пам'ять – запам'ятовуючий пристрій, що зберігає без зміни данні при відключенні живлення.

Непрямий адрес – адреса, що вказує на комірку пам'яті, яка містить виконавчий (фактична) адресоперанда або інший адрес.

Низхідна реалізація програми – у технології структурного програмування первинна реалізація групи модулів верхніх рівнів, які називаються ядром програми, і, далі, поступово відповідно до плану, реалізуються модулі нижніх рівнів. Необхідні для компонування програми, відсутні модулі імітуються заглушками.

Низхідне проектування – один з головних принципів технології структурного програмування, згідно з яким, при розробці ієрархії модулів програм виділяються спочатку модулі самого верхнього рівня ієрархії, а потім підлеглі модулі.

Об'єкт – логічна одиниця, що містить всю інформацію про деякий фізичний предмет або поняття, яке реалізується в програмі, структурована змінна типу клас, яка містить поля даних і методи з кодом алгоритму.

Об'єктна модель – модель, що описує структуру об'єктів, які становлять систему, їх атрибути, операції, взаємозв'язок з іншими об'єктами. В об'єктній моделі мають бути відображені ті поняття й об'єкти реального світу, які важливі для системи, що розробляється.

Об'єктно-орієнтоване програмування (ООП) (object-oriented programming) – це процес реалізації програм, заснований на представленні програми у вигляді сукупності об'єктів. Де об'єкт - деяка динамічна структура - має деякий набір властивостей.

Об'єктно-орієнтоване проектування (ООПр) (object-oriented design, OOD) – методологія проектування, що сполучає в собі процес об'єктної декомпозиції і прийоми подання логічної і фізичної, а також статичної та динамічної моделей проектованої системи.

Об'єктно-орієнтований аналіз (ООА) (object-oriented analysis) – методологія, при якій вимоги до системи сприймаються з точки зору класів та об'єктів, прагматично виявлених у предметній області.

ОЗП – Оперативний Запам'ятовуючий Пристрій.

ОКХ – Освітньо-Кваліфікаційна Характеристика.

Операнд – елемент інформації, що бере участь у виконанні операції.

Оперативна пам'ять – запам'ятовуючий пристрій, призначений для високошвидкісного читання і запису інформації в мікрокомп'ютері.

Оператор мови програмування – запис, що виражає певну закінчену дію в програмі (наприклад, множення двох величин і присвоювання результату третьої величиною).

Операції над структурами даних – над усіма структурами даних можуть виконуватися п'ять операцій: створення, знищення, вибір (доступ), оновлення, копіювання.

Операційна система (ОС) – це базовий комплекс програмного забезпечення, що виконує управління апаратним забезпеченням комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організує взаємодію з користувачем.

Операційний підхід до складання алгоритмів – відповідно до цього підходу, операції (алгоритмічні дії) виділяються послідовно на шляху обчислень при якихось наборах даних.

Оптимізація розробки програм – знаходження розумного компромісу між метою, що повинна бути досягнутою, і затраченими на це ресурсами.

Організованість даних – продуманий пристрій з метою раціонального використання за призначенням.

Основна пам'ять мікрокомп'ютера – пам'ять, що включає в себе оперативний пристрій (ОЗП) і постійний запам'ятовуючий пристрій (ПЗП) будь-якого типу.

Оцінювання – систематичне визначення ступеня відповідності об'єкта заданим критеріям.

Переривання – припинення виконання поточної програми з метою виконання іншої, більш термінової програми, яка називається програмою обробки переривання.

Підпрограма – послідовність машинних команд або операторів мови програмування, що реалізує певну функцію і допускає багаторазове використання іншими програмами шляхом виклику (або звернення).

Підпрограма – деяка послідовність інструкцій, яку можна викликати у кількох місцях програми, програмна одиниця, компільована незалежно від інших частин програми. У Об'єктно-орієнтованому програмуванні відповідає методу.

Покажчик стека – особливий регістр в складі мікропроцесора, що містить адресу стека, тобто адреса, що використовується для занесення чергового даного в стек або вилучення даного з стека.

Показники якості (критерії) – величини, властивості, поняття, що характеризують систему з точки зору суб'єкта, які дозволяють оцінити ступінь задоволення його потреб.

Поліморфізм – це засіб для надання різних значень одній і тій же події в залежності від типу оброблюваних даних. Тобто поліморфізм визначає різні форми реалізації однойменної дії.

Порт – програмний механізм накопичення та верифікації як вхідних, так і вихідних даних у відповідних чергах.

Портфоліо – зібрання зразків робіт, що дають уявлення про пропоновані послуги організації (фірми) або спеціаліста.

Постачальник – організація, яка має контракт із замовником на постачання системи, програмного продукту або програмної послуги відповідно до умов контракту.

Постійний запам'ятовуючий пристрій (ПЗП) – пристрій, що запам'ятовує і призначений тільки для читання інформації, яка один раз була занесена в нього.

Прапори (коди умов) – біти в регістрі стану процесора, що позначають наявність або відсутність деяких умов після виконання чергової команди (наприклад, прапор негативного результату).

Предикат – функція, визначена на деякій предметній області змінних і приймаюча значення в галузі істинносних значень.

Предок – це клас, що надає свої можливості та характеристики іншим класам.

Прикладне програмне забезпечення – сукупність програм для вирішення завдань, що мають прикладний характер (наприклад, завдань управління роботом). Прикладне

програмне забезпечення може створюватися в ході експлуатації програмного забезпечення та комп'ютера.

Прикладне програмування (application programming) – розробка та налагодження програм для кінцевих користувачів, наприклад бухгалтерських, обробки текстів і т.п.

Програма – закінчена послідовність машинних команд або операторів мови програмування, що визначає порядок дій для вирішення деякої задачі обробки даних.

Програмна послуга – це виконання діяльності, роботи або обов'язків, пов'язаних із програмним продуктом, наприклад таких, як його розроблення, супровід та експлуатація.

Програмна специфікація (program specification) – точний опис того результату, якого потрібно досягти за допомогою програми. Цей опис має точно встановлювати, що повинна робити програма, не вказуючи, як вона повинна це робити.

Програмне забезпечення автоматизованих систем – сукупність програм на носіях даних і програмних документів, призначена для налагодження, функціонування і перевірки працездатності автоматизованих систем.

Програмне забезпечення (англ. software) – всі програми, якими забезпечена комп'ютерна система; розрізняють системне програмне забезпечення (зокрема, операційна система, транслятори, редактори, графічний інтерфейс користувача) та прикладне програмне забезпечення, що використовується для виконання конкретних завдань; програми (тобто набір упорядкованих команд), дані, правила та будь-яка відповідна документація, що відноситься до роботи комп'ютерної системи; складова частина обчислювальної техніки, сукупність програм з даними і документації на них, що забезпечує її функціонування.

Програмний виріб – програма на носії даних, що є продуктом промислового виробництва.

Програмний документ – документ, який містить відомості, необхідні для розробки, виготовлення, експлуатації та супроводу програмного виробу.

Програмний лічильник – регістр в складі мікропроцесора, який зберігає адресу чергової команди або компонента команди.

Програмний продукт – функціонально визначений набір комп'ютерних програм, процедур і пов'язаних з ними документації та даних, які можна запускати, тестувати, виправляти і розвивати. Така програма повинна бути написана в єдиному стилі, ретельно протестована до необхідного рівня надійності, супроводжена докладною документацією та підготовлена для тиражування. Стандартний термін - програмний виріб.

Програмно-апаратні засоби – об'єднання апаратної приставки та машинних інструкцій або комп'ютерних даних, які розміщуються як програмний засіб типу «тільки для читання» у апаратній приставці для забезпечення її функціонування. Програмний засіб не можна швидко замінити програмним шляхом.

Програмно-доступний регістр – регістр мікропроцесора або контролера зовнішнього пристрою, який може бути явно задіяний в програмі, написаній, як правило, на мові асемблера.

Програмно-технічний комплекс – сукупність технічних засобів автоматизації, що поставляється в комплекті з програмним забезпеченням, необхідним сервісним устаткуванням і експлуатаційною документацією, що з'єднується на місті експлуатації з периферійним устаткуванням і/або з іншим програмно-технічним комплексом для виконання усіх або частини функцій контролю і керування в складі конкретної інформаційної або керуючої системи.

Програмування – процес і мистецтво створення комп'ютерних програм та/або програмного забезпечення за допомогою мов програмування. Програмування включає в себе як елементи мистецтва, так і фундаментальні науки (у першу чергу математику та інженерію).

Проект (від лат. Projectus - кинутий вперед) – сукупність проектних документів відповідно до встановленого переліку, яка представляє результат проектування.

Проектне завдання (англ. Engineering Task) – характеризується невизначеністю апріорі інформації: що потрібно одержати, що задано. Більше того, спосіб вирішення задачі є об'єктом проектування. І, нарешті, рішення проектної задачі повинно бути знайдено в рамках обмежень зовнішнього середовища проектування: доступних грошових коштів, заздалегідь заданих термінів, можливостями технічних засобів та інструментарію програмування, наукових знань, програмних заділів тощо.

Проектне рішення – описання в заданій формі об'єкта проектування або його частини, необхідне і достатнє для визначення подальшого напрямку проектування.

Проектний документ – документ, виконаний за заданою формою, в якому представлено будь-яке проектне рішення. У програмуванні проектні рішення оформляються у вигляді програмної документації. Розрізняють зовнішню програмну документацію, яка узгоджується із замовником, і внутрішню проміжну документацію проекту, яка необхідна самим програмістам для їх роботи.

Проектування – це розробка проекту, процес створення специфікації, необхідної для побудови в заданих умовах ще неіснуючого об'єкта на основі первинного опису цього об'єкта. Результатом проектування є проектне рішення або сукупність проектних рішень, що задовольняють заданим вимогам (задані вимоги обов'язково повинні включати форму подання рішення) або це процес перетворення вимог до розробки в послідовність проектних рішень щодо способів реалізації вимог: формування загальної архітектури програмної системи та принципів її прив’язки до конкретного середовища функціонування; визначення детального складу модулів кожної з архітектурних компонент.

Профайлер – інструмент аналізу продуктивності, який вимірює поведінку програми коли вона працює, особливо частоту і тривалість звернень до функцій.

Процес – сукупність взаємозалежних дій, які перетворюють «входи» на «виходи» (вхідні дані на вихідні) [ДСТУ 3918].

Реалізація – перетворення проектних рішень на програмну систему, яка реалізує такі рішення.

Регістр загального призначення – регістр в складі мікропроцесора, який може бути задіяний в програмі.

Регістр стану процесора – доступний регістр, призначений для зберігання значення, що відповідає стану процесора до, після або під час виконання програми.

Регістр – послідовний або паралельний логічний пристрій, що використовується для зберігання n-розрядних двійкових чисел і виконання перетворень над ними, що можуть розглядатися як один електронний елемент.

Режим адресації – метод, або схема, формування виконавчого адреси команди або операнда.

Резидентна програма – програма, що постійно знаходиться в оперативній пам'яті ЕОМ і не видаляється ОС.

Резюме – документ, що містить інформацію про навички, досвід роботи, освіту та іншої інформації, яка відноситься до справи, зазвичай потрібний при розгляді кандидатури людини для найму на роботу.

Рефакторинг – перетворення програмного коду, зміна внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи.

Робота в масштабі реального часу – режим, в якому мікрокомп'ютер обробляє дані відразу в міру їх надходження (наприклад, з датчиків) і формує керуючий вплив в масштабі часу роботи керованого обладнання (наприклад, робота).

Робочий проект (РП) – найменування стадії та програмний документ, що містить опис реалізованого виробу.

Розробник – організація, яка в рамках життєвого циклу програмного забезпечення виконує дії з його розроблення, зокрема й аналізування вимог, проектування, випробування під час прийняття [ДСТУ 3918].

Розрядність слова (довжина розрядної сітки) – число двійкових розрядів (бітів) в машинному слові.

Сертифікація персоналу – встановлення якісних характеристик персоналу вимогам вітчизняних та/чи міжнародних стандартів.

Сертифікація – діяльність третьої сторони, незалежної від виробника (продавця) і споживача продукції, по затвердженню відповідності продукції встановленим вимогам.

Сесія програмістів – зустріч кодувальників для проведення взаємної інспекції текстів програм і набору використаних тестів.

Система команд – сукупність всіх типів команд, які може виконувати мікропроцесор в даному мікрокомп'ютері.

Система програмування – система для розробки нових програм на конкретній мові програмування.

Система управління базою даних (СУБД) – це спеціалізована програма (частіше комплекс програм), призначена для організації та ведення бази даних.

Система числення – способи запису чисел і виконання над ними арифметичних операцій.

Система – безліч елементів, що знаходяться у відносинах і зв'язках один з одним, яка утворює певну цілісність, єдність.

Система – інтегрована структура, яка складається з одного або кількох процесів, компонентів апаратного забезпечення, компонентів програмного забезпечення, засобів та персоналу, що забезпечує можливість задоволення встановленої потреби або цільової функції.

Системне (базове) ПЗ – програмне забезпечення, що включає в себе операційні системи, мережеве ПЗ, сервісні програми, а також засоби розробки програм (транслятори, редактори зв'язків, відлагоджувачі тощо.).

Системне програмування (або програмування систем) – рід діяльності, що полягає в роботі над системним програмним забезпеченням. Основна відмінна риса системного програмування втому, що його результатом є випуск програмного забезпечення, який пропонує сервіси по взаємодії з апаратним забезпеченням (наприклад, дефрагментація жорсткого диска), що має на увазі сильну залежність таких програм від апаратної частини.

Системний аналітик – програміст, який розробляє проект від вимог до внутрішньої структури програми та бере

участь у тестуванні як при інтеграції компонентів у ядро, так і в комплексному тестуванні ПЗ.

Системний підхід (англ. Systems thinking - системне мислення) – напрям методології досліджень, який полягає в дослідженні об'єкта як цілісної множини елементів в сукупності відношень і зв'язків між ними, тобто розгляд об'єкта як системи.

Службове завдання – це завдання роботодавця, в якому він доручає працівникові виконати ті чи інші обов'язки. Наприклад, службове завдання вдосконалити комп'ютерну програму з розрахунку собівартість виробництва по одному з продуктом роботодавця.

Службовий твір – це твір, створений автором у порядку виконання службових обов'язків згідно службового завдання чи трудового договору (контракту) між ним і роботодавцем.

Службові обов'язки – це обов'язки працівника, визначені в трудовому договорі, посадових інструкціях, які передбачають виконання певних робіт. Наприклад, вести бухгалтерський облік, писати комп'ютерну програму, вести листування.

Спадкування – це визначення класу і потім використання його для побудови ієрархії класів-нащадків, причому кожен нащадок успадковує доступ до коду і даних усіх своїх класів прабатьків.

Специфікація – у сфері проектної діяльності це будь-яке описання в точних термінах.

Стадія проекту – одна з частин процесу створення програми, встановлена нормативними документами і закінчується випуском проектної документації, що містить опис повної, в рамках заданих вимог, моделі програми на заданому для цієї стадії рівні, або виготовленням програм. Після досягнення стадії замовник має можливість розглянути стан проекту і прийняти рішення щодо подальшого продовження проектних робіт.

Статистичне моделювання – спосіб вивчення поведінки стохастичних систем в умовах, коли внутрішні взаємодії в цих системах не визначені.

Стек – область пам'яті, відведена для тимчасового запам'ятовування даних і використовувана за принципом LIFO "LastIn, FirstOut" ("останній елемент даних занесений - перший витягується").

Стійкість програмного забезпечення – властивість здійснювати необхідне перетворення інформації при збереженні вихідних рішень програми в межах допусків, встановлених специфікацією при впливі на програми таких факторів нестійкості, як помилки операторів ЕОМ, а також не виявлених помилок програми.

Стохастичне програмування – напрямок у математичному програмуванні, комплекс методів вирішення оптимізаційних завдань стохастичного характеру. Це говорить про те, що або умови задачі, або параметри цільової функції, або й ті й інші являють собою випадкові величини.

Стратегія (від грец.Stratos військо і ago веду) – наука, мистецтво генерації найбільш істотних спільних довгострокових цілей і найбільш загального плану досягнення переваги, курсу дій і розподілу ресурсів ще до виконання реальних дій. Стратегія охоплює теорію і практику підготовки до виконання проекту, а також найбільш загальне планування тактик ведення проектів. Стратегія визначає, куди, у якому напрямку рухатися, куди тримати курс ще до початку проекту. А тактика визначає, як, яким способом рухатися, які конкретні дії робити при труднощах у ході виконання проекту.

Структура даних програми – безліч елементів даних, безліч зв'язків між ними, а також характер їхньої організованості.

Структура програми – штучно виділені програмістом взаємодіючі частини програми.

Структурне кодування модулів програм – основний принцип технології структурного програмування, сприйнятий технологією об'єктно-орієнтованого програмування, який полягає в особливому оформленні текстів модулів (методів). У модуля повинен бути легко помітний заголовок з коментарем, що пояснює функціональне призначення модуля. Імена змінних повинні бути мнемонічними. Суть змінних та порядок розміщення в них інформації повинен бути пояснений

коментарями. Код повинен бути закодований з використанням типових алгоритмічних структур.

Структурне програмування – методологія розробки програмного забезпечення, в основі якої лежить подання програми у вигляді ієрархічної структури блоків.

Структурний аналіз – виявлення елементів об'єкта і зв'язків між ними.

Структурний підхід – набір принципів, що характеризує технологію структурного програмування: модульність програм; структурне кодування модулів програм; спадне проектування раціональної ієрархії модулів програм; спадна реалізація програми з використанням заглушок; здійснення планування на всіх стадіях проекту; наскрізний структурний контроль програмних комплексів у цілому та їхніх складових модулів."

Супровід – діяльність з надання послуг, необхідних для забезпечення сталого функціонування розвитку програмного виробу, включає аналіз функціонування, розвиток і вдосконалення програми, а також внесення змін до неї з метою усунення помилок.

Супровідник – організація, що виконує дії щодо супроводу продукту.

Схема алгоритму – діаграма, що складається з з'єднаних між собою блоків різного типу і відображає основні етапи виконання алгоритму або програми.

Схема ієрархії програми - використовується в технології структурного програмування, відображає тільки підпорядкованість модулів (підпрограм), але не порядок їхнього виклику або функціонування програми.

Сценарій діалогу програми – послідовність введення і виведення інформації в діалоговому режимі роботи програми.

Сценарій – послідовність подій, яка може мати місце при конкретному виконанні системи.

Таймер – компонент мікрокомп'ютера, що генерує періодичні сигнали для відліку поточного часу в програмах. Кожен такий сигнал зазвичай викликає переривання.

Теорія ігор – напрям у прикладній математиці, в якому розглядаються математичні моделі конфліктних ситуацій

(таких ситуацій, за яких інтереси учасників протилежні ("антагоністичні ігри"), або не однакові, хоча і не протилежні ("ігри з непротилежними інтересами")).

Термінал – пристрій для роботи з віддаленим комп'ютером, зазвичай включає в себе клавіатуру і дисплей.

Тестувальник – програміст, який готує набори тестів для налагодження розроблюваного програмного виробу.

Тестування – перевірка кожного з модулів та способів їхньої інтеграції; тестування програмного продукту в цілому (так звана верифікація); тестування відповідності функцій працюючої програмної системи вимогам (requirements), поставленим до неї замовником (так звана валідація).

Тестування (testing) – процес виконання програми з наміром знайти помилки. Може здійснюватися як з ЕОМ, так і без ЕОМ.

Технічне завдання (ТЗ) – документ, оформлений в установленому порядку, визначає цілі створення програми, вимоги до програми і основні вихідні дані, необхідні для її розробки, а також план-графік створення програми.

Технічне обслуговування ПЗ – сукупність усіх технічних і адміністративних дій, включаючи нагляд, призначених для підтримки ПЗ у робочому стані.

Технічний проект (ТП) – комплект проектних документів на програму, що розробляється на стадії "Технічний проект", затверджений у встановленому порядку, який містить основні проектні рішення щодо програми в цілому, її функцій і достатній для розробки робочого проекту.

Технологія візуального програмування – популярна парадигма програмування, що складається в автоматизованій розробці програм з використанням особливої діалогової оболонки.

Технологія об'єктно-орієнтованого програмування – технологія, орієнтована на отримання програм, що складаються з об'єктів.

Технологія програмування Дейкстри, заснована на абстракції даних – це технологія проектування, в якій на чолі ставляться дані, спочатку дуже ретельно специфікується вихід, вхід, проміжні дані, велика увага приділяється типізації

даних з використанням структур задля об'єднання близької за змістом інформації в єдині дані.

Технологія програмування – сукупність методів, прийомів і засобів для скорочення вартості та підвищення якості розробки програмних систем.

Технологія структурного програмування – технологія заснована на структурному підході.

Технологія (від грец.Techne - мистецтво, майстерність, уміння і logos - слово, вчення) – сукупність виробничих процесів у певній галузі виробництва, а також науковий опис способів виробництва, сукупність прийомів, застосовуваних в якій-небудь справі, майстерності, мистецтві. Сучасна методологія проектування дозволила довести методи проектування до технологій з набором методик.

Типізація – спосіб захиститися від використання об'єктів одного класу замість іншого, або, принаймні, керувати таким використанням. Типізація змушує висловлювати абстракції так, щоб мова програмування, що використовується в реалізації, підтримувала дотримання прийнятих проектних рішень.

Транспортна задача – одне з найбільш часто використовуваних завдань (як правило - лінійного) програмування. У загальному вигляді її можна сформулювати наступним чином: необхідно визначити такий план доставки вантажів від постачальників до споживачів, щоб витрати на перевезення (або загальна дальність або обсяг транспортної роботи) були мінімальними.

Тригер – електронний елемент, який може знаходитися в одному з двох станів. Використовується як елемент пам'яті для зберігання одного біта інформації.

Трудовий договір – угода між працівником і власником підприємства, установи чи організації (або уповноваженим ним органом) або фізичною особою, за якою працівник зобов'язується виконувати роботу, визначену цією угодою, а власник підприємства, установи чи організації (або уповноваженим ним органом) чи фізична особа зобов'язується виплачувати працівникові заробітну плату і забезпечувати умови праці, необхідні для виконання роботи.

Угода – визначення строків та термінів умов, за яких будуть здійснюватися робочі стосунки.

Управління розробкою програмних систем (software management) – діяльність, спрямована на забезпечення необхідних умов для роботи колективу розробників програмного забезпечення, на планування і контроль діяльності цього колективу з метою забезпечення необхідної якості ПЗ, виконання строків і бюджету розробки ПЗ.

Успадкування – метод утворення нових класів, що використовується в об'єктно-орієнтованому програмуванні, на основі використання вже існуючих класів.

Фаза (етап) життєвого циклу ПЗ – сукупність функціонально або за типовістю чи спільністю організацій пов'язаних дій або задач, у процесі виконання яких реалізують розроблення, виробництво та експлуатація продукту, що постачається.

Фізична структура даних – спосіб фізичного представлення даних у пам'яті машини, називається ще структурою зберігання, внутрішньою структурою, структурою пам'яті або дампом.

Філософія моделей – визначення оптимальної кількості та характеристик фізичних моделей, необхідних для досягнення високого рівня впевненості в результатах верифікації продукту за умови мінімально можливого часу планування та прийнятної комбінації вартості й ризику.

Форма – візуальний компонент, що володіє властивістю вікна Windows.

Формат команди – сукупність відомостей про довжину, склад, призначення і взаємне розташування частин (полів) команди.

Функція – навмисний вплив або дія системи, підсистеми, виробу або будь-якої частини, деталі.

Характеристика, підхарактеристика – набір властивостей програмного забезпечення, за допомогою якого він може бути описаний та оцінений. Характеристика може зводитися до кількох рівнів підхарактристик, які спираються на її спроможності задовольняти заявлені потреби або потреби, які можуть виникнути.

Центр сертифікації – юридична особа, уповноважена одночасно виконувати функції органу із сертифікації та випробувальної лабораторії.

Час доступу – характеристика швидкодії пам'яті комп'ютера, яка вимірюється від моменту звернення до пам'яті до моменту завершення запису або читання даних з пам'яті. Значення цієї характеристики залежить від типу пам'яті.

Шаблони проектування (паттерн, англ. design pattern) – це багато разів застосовувана архітектурна конструкція, що надає рішення загальної проблеми проектування в рамках конкретного контексту й описує значимість цього рішення.

Швидкість передачі даних – обсяг інформації, що передається між основною пам'яттю мікрокомп'ютера і пристроєм зовнішньої пам'яті за одиницю часу.

Шістнадцяткова система числення – система числення з основою 16, в якій в якості цифр використовуються символи 0, 1, 2, 3, 4, 5,6, 7,8, 9, A, B, C, D, E, F.

Ядро – вже реалізована частина програми, яка постійно збільшується.

Якість програмного забезпечення – сукупність властивостей ПЗ, які обумовлюють його придатність задовольняти деякі потреби відповідно до призначення.

Додаток А

Перелік національних, міждержавних та міжнародних стандартів щодо створення, впровадження та супроводження автоматизованих і інформаційних систем та їх програмного заьезпечення.

ДСТУ 3008-95 „Документація. Звіти у сфері науки і техніки. Структура і правила оформлення”;

- ДСТУ 3973-2000„Правила виконання науково-дослідних робіт. Загальні положення”;
- ДСТУ 3974-2000 „Правила виконання дослідно-конструкторських робіт. Загальні положення”;
- ДСТУ 3396.0-96 Захист інформації. Технічний захист інформації. Основні положення;
- ДСТУ 3396.1-96 Захист інформації. Технічний захист інформації. Порядок проведення робіт;
- ДСТУ 3396.2-97 Захист інформації. Технічний захист інформації. Терміни та визначення;
- ДСТУ 2844-94 Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення;
- ДСТУ 2873-94 Системи оброблення інформації. Програмування. Терміни та визначення;
- ДСТУ 2941-94 Системи оброблення інформації. Розроблення систем. Терміни та визначення;
- ДСТУ ISO/IEC 2382-15:2005 Інформаційні технології. Словник термінів. Частина 15. Мови програмування;
- ДСТУ ISO/IEC 2382-5:2005 Інформаційні технології. Словник термінів. Частина 5. Подання даних;
- ДСТУ ISO/IEC 2382-4:2005 Інформаційні технології. Словник термінів. Частина 4. Організація даних;
- ДСТУ ISO/IEC 2382-17:2005 Інформаційні технології. Словник термінів. Частина 17. Бази даних;

- ДСТУ ISO/IEC 2382-18:2005 Інформаційні технології. Словник термінів. Частина 18. Розподілене оброблення даних;
- ДСТУ ISO/IEC 2382-9:2005 Інформаційні технології. Словник термінів. Частина 9: Обмін даними;
- ДСТУ ISO/IEC 2382-14:2005 Інформаційні технології. Словник термінів. Частина 14. Безвідмовність, ремонтопридатність і готовність;
- ДСТУ ISO/IEC 90003:2006 Програмна інженерія. Настанови щодо застосування ISO 9001:2000 до програмного забезпечення (ISO/IEC 9003:2004, IDT);
- ДСТУ 4071–2002 Інформаційні технології. Архітектура відкритого розподіленого керування та підтримка загальної архітектури брокера об'єктних запитів (CORBA);
- ДСТУ 4072–2002 Інформаційні технології. Мови програмування, їхні середовище і системний інтерфейс. Незалежний від мов виклик процедур (LIPC);
- ДСТУ ISO/IEC TR 14369:2003 Інформаційні технології. Мови програмування, їхні середовище і системний інтерфейс. Настанова щодо підготовки незалежних від мов специфікацій сервісу (LISS);
- ДСТУ 4249:03 Інформаційні технології. Настанова щодо POSIX-сумісних середовищ відкритих систем (POSIX-OSE) (ISO/IEC TR 14252:1996, MOD);
- ДСТУ 2850-94 Програмні засоби ЕОМ. Показники і методи оцінювання якості;
- ДСТУ 2851-94 Програмні засоби ЕОМ. Документування результатів випробувань;
- ДСТУ 2853-94 Програмні засоби ЕОМ. Підготовлення і проведення випробувань;
- ДСТУ 3918-99 (ISO/IEC 12207:1995) Інформаційні технології. Процеси життєвого циклу програмного забезпечення;

- ДСТУ 3919-99 (ISO/IEC 14102:1995) Інформаційні технології. Основні напрямки оцінювання та відбору CASE-інструментів;
- ДСТУ 4302:2004 Інформаційні технології. Настанови щодо документування комп`ютерних програм (ISO/IEC 6592:2000, MOD) ;
- ДСТУ ISO/IEC TR 12182:2004 Інформаційні технології. Класифікація програмних засобів (ISO/IEC TR 12182:1998, IDT) ;
- ДСТУ ISO/IEC 14598-1:2004 Інформаційні технології. Оцінювання програмного продукту. Частина 1. Загальний огляд (ISO/IEC 14598-1:1999, IDT) ;
- ДСТУ ISO/IEC 14598-2:2005 Інформаційні технології. Оцінювання програмного продукту. Частина 2. Планування та керування (ISO/IEC 14598-2:2000, IDT) ;
- ДСТУ ISO/IEC 14598-3:2005 Інформаційні технології. Оцінювання програмного продукту. Частина 3. Процес для розробників (ISO/IEC 14598-3:2000, IDT) ;
- ДСТУ ISO/IEC 14598-4:2005 Інформаційні технології. Оцінювання програмного продукту. Частина 4. Процес для замовників (ISO/IEC 14598-4:1999, IDT) ;
- ДСТУ ISO/IEC 14598-5:2005 Інформаційні технології. Оцінювання програмного продукту. Частина 5. Процес для оцінювачів (ISO/IEC 14598-5:1998, IDT) ;
- ДСТУ ISO/IEC 14598-6:2005 Інформаційні технології. Оцінювання програмного продукту. Частина 6. Документація модулів оцінювання (ISO/IEC 14598-6:2001, IDT);
- ДСТУ ISO/IEC 14764-2002 Інформаційні технології. Супровід програмного забезпечення (ISO/IEC 14764:1999, IDT);
- ДСТУ ISO/IEC 15288:2005 Інформаційні технології. Процеси життєвого циклу системи (ISO/IEC 15288:2002, IDT) ;

- ДСТУ ISO/IEC TR 15504-1-2002 Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 1. Концепції та вступна настанова (ISO/IEC TR 15504-1:1998, IDT) ;
- ДСТУ ISO 9735-1:2006 Електронний обмін даними для адміністрування, у торгівлі і на транспорті (EDIFACT). Правила синтаксису прикладного рівня (номер версії синтаксису: 4, номер редакції синтаксису: 1); у 10 частинах.
- ДСТУ ISO/TS 20625:2007 Обмін електронними даними для управління, торгівлі і транспорту (EDIFACT). Правила генерації файлів XML-схем (XSD) на основі настанови з реалізації EDI(FACT);
- ДСТУ 4145:2002 Інформаційні технології. Криптографічний захист інформації. Електронний цифровий підпис, що ґрунтується на еліптичних кривих;
- ДСТУ ISO/IEC 13888–2002 Інформаційні технології. Методи захисту. Неспростовність»: Частина 1. Загальні положення;
- ДСТУ ISO/IEC 13888–2002 Інформаційні технології. Методи захисту. Неспростовність»: Частина 3. Механізми з використанням асиметричних методів
- ДСТУ ISO/IEC 14888–1:2002 Інформаційні технології. Методи захисту. Цифрові підписи з доповненням» Частина 1. Загальні положення
- ДСТУ ISO/IEC 14888–2:2002 Інформаційні технології. Методи захисту. Цифрові підписи з доповненням» Частина 2. Механізми на основі ідентифікаторів
- ДСТУ ISO/IEC 14888–3:2002 Інформаційні технології. Методи захисту. Цифрові підписи з доповненням» Частина 3. Механізми на основі сертифікатів
- ДСТУ ISO/IEC 10118-1:2000 Інформаційні технології. Методи захисту. Геш функції. Частина 1. Загальні положення

- ДСТУ ISO/IEC 10118-2:2000 Інформаційні технології. Методи захисту. Геш функції. Частина 2. Геш функції, що використовують n-бітний блоковий шифр
- ДСТУ ISO/IEC 10118-3:2004 Інформаційні технології. Методи захисту. Геш функції. Частина 3. Спеціалізовані геш функції
- ДСТУ ISO/IEC 13335-1:2004 Інформаційні технології. Методи захисту. Керування інформацією й безпекою технології комунікацій. Частина 1. Поняття й моделі для інформації й керування безпекою технології комунікацій
- ДСТУ ISO/IEC 15946-1:2008 Інформаційні технології. Методи захисту. Криптогра¬фічні методи, засновані на еліптичних кривих. Частина 1. Загальні положення
- ДСТУ ISO/IEC 18014-1:2002 Інформаційні технології. Методи захисту .Послуги штемпелювання часу - Частина 1: Структура
- ДСТУ ISO/IEC 18014-2:2002 Інформаційні технології. Методи захисту. Послуги штемпелювання часу. Частина 2. Механізми, що генерують незалежні токени
- ДСТУ ISO/IEC 9798-1:1997 Інформаційні технології. Методи захисту. Автентифікація сутності. Частина 1. Загальні положення
- ДСТУ ISO/IEC 9798-3:1998 Інформаційні технології. Методи захисту. Автентифікація сутності. Частина 3. Механізми , що використовують методи цифрового підпису
- ДСТУ ISO/IEC TR 13335-1:2001 Інформаційні технології. Настанова для керування IT безпекою. Частина 5. Настанова керування безпекою мережі
- ДСТУ-П CWA 14172-1:2008 Настанова EESSI з оцінювання відповідності. Частина 1: Загальні положення
- ДСТУ-П CWA 14172-2:2008 Настанова EESSI з оцінювання відповідності. Частина 2. Послуги та процеси органу сертифікації

- ДСТУ-П CWA 14172-3:2008 Настанова EESSI з оцінювання відповідності. Частина 3. Надійні системи, що управляють сертифікатами для електронних підписів
- ДСТУ-П CWA 14172-4:2008 Настанова EESSI з оцінювання відповідності. Частина 4. Застосовування для накладання підпису та загальні настанови з перевірки електронного підпису
- ДСТУ-П CWA 14172-5:2008 Настанова EESSI з оцінювання відповідності. Частина 5. Безпечні засоби створення підпису
- ДСТУ-П CWA 14172-6:2008 Настанова EESSI з оцінювання відповідності. Частина 6. Засіб створення підписів, що підтримує підписи, крім кваліфікованих
- ДСТУ-П CWA 14172-7:2008 Настанова EESSI з оцінювання відповідності. Частина 7. Криптографічні модулі, використовувані провайдерами послуг сертифікації для операцій підписування та послуг генерування ключів
- ДСТУ-П CWA 14172-8:2008 Настанова EESSI з оцінювання відповідності. Частина 8. Послуги та процеси органу штемпелювання часу
- ДСТУ CWA 14365-1:2008 Настанова з використання електронних підписів. Частина 1. Юридичні та технічні аспекти
- ДСТУ ISO/IEC 8824-1:2008 Інформаційні технології. Нотація абстрактного синтаксису (ASN.1) Частина 1: Специфікація базової нотації
- ДСТУ ISO/IEC 8824-2:2008 Інформаційні технології. Нотація абстрактного синтаксису 1 (ASN.1). Частина 2. Специфікація інформаційного об’єкту
- ДСТУ ISO/IEC 8824-3:2008 Інформаційні технології. Нотація абстрактного синтаксису 1 (ASN.1) Частина 3. Специфікація обмежень

- ДСТУ ISO/IEC 8824-4:2008 Інформаційні технології Нотація абстрактного синтаксису 1 (ASN.1) Частина 4: Параметризація специфікацій ASN.1
- ДСТУ CWA 14167-3:2008 Криптографічний модуль для послуг генерування ключів провайдером послуг сертифікації. Профіль захисту CMCKG-PP
- ДСТУ ETSI TS 101 733:2009 Електронні підписи та інфраструктури (ESI). CMS-розширені електронні підписи (CAdES)
- ДСТУ ETSI TS 102 734:2009 Електронні підписи й інфраструктури; Профілі CMS розширених електронних підписів, що ґрунтуються на TS 101 733 (CAdES)
- ДСТУ ETSI TS 101 903:2009 XML-розширені електронні підписи (XAdES)
- ДСТУ ETSI TS 102 904:2009 Електронні підписи й інфраструктури. Профілі розширених електронних підписів XML, що ґрунтуються на TS 101 903 (XAdES)
- ДСТУ ETSI TS 101 862:2009 Профіль посилених сертифікатів
- ДСТУ ETSI TS 101 861: 2009 Профіль штемпелювання часу
- ДСТУ ETSI TS 102 176-1:2009 Електронні підписи й інфраструктури (ESI). Алгоритми й параметри для безпечних електронних підписів Частина 1. Геш-Функції й асиметричні алгоритми
- ДСТУ ETSI TS 102 176-2:2009 Електронні підписи й інфраструктури (ESI). Алгоритми та параметри для безпечних електронних підписів. Частина 2. Протоколи безпечного каналу й алгоритми для засобів накладання підпису
- ДСТУ ETSI TS 102 023:2009 Електронні підписи й інфраструктури (ESI). Вимоги політики для органів штемпелювання часу

- ДСТУ ETSI TS 102 047:2009 Міжнародна гармонізація форматів електронних підписів
- ДСТУ ETSI TS 102 045:2009 Електронні підписи й інфраструктури (ESI). Політика підписів для розширеної бізнес-моделі
- ДСТУ 4353-5:2004 Інформаційні технології. Восьмибітні однобайтні набори кодованих графічних символів. Частина 5: Латиниця/кирилиця (ISO/IEC 8859-5:1999)
- ДСТУ 4354-1:2004 Інформаційні технології. Універсальний мультиоктетний набір кодованих символів (UCS). Частина 1: Архітектура і базова мультилінгвістична плата (ISO/IEC 10646-1:2000)
- ДСТУ 4355-2004 Інформаційні технології. Процедура реєстрації ESCAPE-послідовностей і наборів кодованих символів (ISO/IEC 2375: 2003)
- ДСТУ 4356-2004 Інформаційні технології. Міжнародне впорядкування і зіставлення рядків. Метод порівняння символьних рядків і опис порядку підгонки загальних шаблонів (ISO/IEC 14651:2001)
- ДСТУ 4358-2004 Інформаційні технології. Процедури реєстрації культурних елементів (ISO/IEC 15897:1999)
- ДСТУ ISO/IEC TR 11017:2004 Інформаційні технології. Середовище інтернаціоналізації (ISO/IEC TR 11017:1998)
- ДСТУ 3986:2000 (ISO 8879:1986) Інформаційні технології. Електронний доку-ментообіг. Стандартна мова узагальненої розмітки (SGML)
- ДСТУ 3719:1998 (ISO/IEC 8613:1989) Інформаційні технології. Електронний документообіг. Архітектура службових документів (ODA) та обмінний формат. Частини 1-4
- ГОСТ 19.001-77. Єдина система програмної документації. Загальні положення;

- ГОСТ 19.005-85. Єдина система програмної документації. Р-схеми алгоритмів та програм. Позначення умовні графічні та правила виконання;
- ГОСТ 19.101-77 (СТ СЗВ 1626-79). Єдина система програмної документації. Види програм і програмних документів;
- ГОСТ 19.102-77. Єдина система програмної документації. Стадії розробки;
- ГОСТ 19.103-77. Єдина система програмної документації. Позначення програм програмних документів;
- ГОСТ 19.104-78 (СТ СЗВ 2088-80). Єдина система програмної документації. Основні написи;
- ГОСТ 19.105-78 (СТ СЗВ 2088-80). Єдина система програмної документації. Загальні вимоги до текстових програмних документів;
- ГОСТ 19.106-78 (СТ СЗВ 2088-80). Єдина система програмної документації. Вимоги до програмних документів, що виконані друкованим способом;
- ГОСТ 19.201-78 (СТ СЗВ 1627-79). Єдина система програмної документації. Технічне завдання. Вимоги до змісту та оформлення;
- ГОСТ 19.202-78 (СТ СЗВ 2090-80). Єдина система програмної документації. Специфікація. Вимоги до змісту та оформлення;
- ГОСТ 19.301-79 (СТ СЗВ 3747-82). Єдина система програмної документації. Програма та методика випробувань. Вимоги до змісту та оформлення;
- ГОСТ 19.401-78 (СТ СЗВ 3746-82). Єдина система програмної документації. Текст програми. Вимоги до змісту та оформлення;
- ГОСТ 19.402-78 (СТ СЗВ 2092-80). Єдина система програмної документації. Опис програми;

- ГОСТ 19.403-79. Єдина система програмної документації. Відомість утримувачів оригіналів;
- ГОСТ 19.404-79. Єдина система програмної документації. Пояснювальна записка. Вимоги до змісту та оформлення;
- ГОСТ 19.501-78. Єдина система програмної документації. Формуляр. Вимоги до змісту та оформлення;
- ГОСТ 19.502-78 (СТ СЗВ 2093-80). Єдина система програмної документації. Опис застосування. Вимоги до змісту та оформлення;
- ГОСТ 19.503-79 (СТ СЗВ 2094-80). Єдина система програмної документації. Нас танова системного програміста. Вимоги до змісту та оформлення;
- ГОСТ 19.504-79 (СТ СЗВ 2095-80). Єдина система програмної документації. Настанова програміста. Вимоги до змісту та оформлення;
- ГОСТ 19.505-79 (СТ СЗВ 2096-80). Єдина система програмної документації. Настанова оператора. Вимоги до змісту та оформлення;
- ГОСТ 19.506-79 (СТ СЗВ 2097-80). Єдина система програмної документації. Опис мови. Вимоги до змісту та оформлення;
- ГОСТ 19.507-79 (СТ СЗВ 2091-80). Єдина система програмної документації. Відомість експлуатаційних документів;
- ГОСТ 19.508-79. Єдина система програмної документації. Посібник з технічного обслуговування. Вимоги до змісту та оформлення;
- ГОСТ 19.602-78. Єдина система програмної документації. Правила дублювання, обліку та зберігання програмних документів, що виконані друкарським способом;
- ГОСТ 19.603-78 (СТ СЗВ 2089-80). Єдина система програмної документації. Загальні правила внесення змін;

- ГОСТ 19.604-78 (СТ СЭВ 2089-80). Єдина система програмної документації. Правила внесення змін до програмних документів, що виконані друкарським способом;
- ГОСТ 19.701-90 (ИСО 5807-85). Єдина система програмної документації. Схеми алгоритмів, програм, даних та систем;
- ГОСТ 19781-90 Програмне забезпечення систем обробки інформації. Терміни та визначення;
- ГОСТ 28195-89. Оцінка якості програмних засобів. Загальні положення. ГОСТ 28806-90. Якість програмних засобів. Терміни та визначення;
- ГОСТ 34.003-90. Інформаційна технологія. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Терміни та визначення;
- ГОСТ 34.201-89. Інформаційна технологія. Комплекс стандартів на автоматизовані системи. Види, комплектність і позначення документів при створенні автоматизованих систем;
- ГОСТ 34.601-90. Інформаційна технологія. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ГОСТ 34.602-89. Інформаційна технологія. Комплекс стандартів на автоматизовані системи. Технічне завдання на створення автоматизованої системи;
- ГОСТ 34.603-92. Інформаційна технологія. Види випробувань автоматизованих систем;
- РД 50-34.698-90. Методичні вказівки. Інформаційна технологія. Комплекс стандартів і керівних документів на автоматизовані системи. Автоматизовані системи. Вимоги до змісту документів;
- РД 50-682-89. Методичні вказівки. Інформаційна технологія. Комплекс стандартів і керівних документів на автоматизовані системи. Загальні положення;

- ГОСТ Р ИСО/ МЭК ТО 9274-93. Інформаційна технологія. Настанова з управління документуванням програмного забезпечення;
- ГОСТ Р ИСО/МЭК ТО 10000-1-93. Інформаційна технологія. Основи та таксономія функціональних стандартів. Частина 1. Основи.
- Постанова Кабінету Міністрів України N 869 від 12 серпня 2009 р. Про затвердження загальних вимог до програмних продуктів, які закуповуються та створюються на замовлення державних органів.

Додаток Б

Перелік можливих тем рефератів

1. Еволюція методів розробки ПЗ.
2. Методи документування архітектури ПЗ.
3. Процеси керування в процесі розробки ПЗ.
4. CASE технології розробки ПЗ.
5. Побудова процесу розробки ПЗ.
6. Людський фактор в розробці ПЗ.
7. Моделі і методи оцінки особистісних характеристик виконавців і команди в цілому.
8. Кількісні методики оцінки ризиків ПЗ.
9. Метричні показники в оцінці ПЗ.
10. Моделі структурного аналізу ПЗ.
11. Порівняльний аналіз інструментів моделювання і трасування програмних вимог.
12. Порівняльний аналіз інструментів верифікації ПЗ.
13. Порівняльний аналіз інструментів оптимізації ПЗ.
14. Порівняльний аналіз інструментів тестування програмного забезпечення (генератори тестів, схеми виконання тестів, оцінка тестів, управління тестами).
15. Порівняльний аналіз інструментів супроводу програмного забезпечення.
16. Системи моделювання процесів розробки програмного забезпечення.
17. Порівняльний аналіз інструментів забезпечення якості програмного забезпечення.
18. Порівняльний аналіз інструментів управління конфігурацією програмного забезпечення.
19. Інструменти планування і відстеження програмних проектів
20. Інструменти, що реалізують підтримку інфраструктури розробки.
21. Методи і технології балансування навантаження.
22. Бібліотеки інформаційних ресурсів з програмної інженерії.
23. Проблеми IT аутсорсингу.

24. Краудсорсінг.
25. Використання паралелізму в програмах.
26. Метрики програмного забезпечення.
27. Програмне забезпечення в системах "розумний дім".
28. Моделювання ризиків при розробці ПЗ.
29. Парадигми програмування.
30. Концепція життя в оточенні цифрових пристроїв.
31. Методи і технології профілювання користувачів.
32. Моделі життєвих циклів програмних систем.
33. Основи сучасних технологій програмної інженерії.
34. Стандарти адміністративного управління якістю ПЗ.
35. Стандарти відкритих систем, що регламентують структуру і інтерфейси програмного забезпечення.
36. Процеси системного проектування ПЗ.
37. Особливості проектування модулів і компонентів ПЗ.
38. Розробка вимог до програмних систем.
39. Структура документів, що відображають вимоги до ПЗ.
40. Планування життєвого циклу ПЗ.
41. Планування процесів управління якістю ПЗ.
42. Ризики в життєвому циклі ПЗ.
43. Ризики при формуванні вимог до характеристик ПЗ.
44. Фактори, що визначають якість ПЗ.
45. Організація і методи супроводу ПЗ.
47. Процеси управління конфігурацією ПЗ.
48. Організація документування ПЗ.
49. Формування вимог до документації ПЗ.
50. Планування документування проектів ПЗ.

Перелік рекомендованих джерел

1 Монографії, підручники, навчальні посібники, нормативні документи, стандарти

1. Knuth D.E. The art of computer programming. Fundamentals algorithms / D.E. Knuth. – Massachusetts: Addison-Wesley, 1997. – 650 s.
2. Knuth D.E. The art of computer programming. Seminumerical Algorithms/ D.E. Knuth. – London: Addison-Wesley, 1997. – 762 s.
3. Knuth D.E. The art of computer programming. Sorting and Searching/ D.E. Knuth. – London: Addison-Wesley, 1997. – 780 s.
4. Knuth D.E. The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions / D.E. Knuth. – London: Addison-Wesley, 2008. – 240 s.
5. Knuth D.E. The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams / D.E. Knuth. – London: Addison-Wesley, 2009. – 272 s.
6. Knuth D.E. The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations/D.E. Knuth. – Massachusetts: Addison-Wesley, 2005. – 144 s.
7. SWEBOK V3.0. The Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Professional Practices Committee (Керівництво з областей знань програмної інженерії). – Tokio: OÌ"musha, 2014. – 335 с.
8. Sommerville I. Software Engineering / I. Sommerville. – London: Addison-Wesley Publishers Limited, 2002. – 624.
9. Шеховцов В.А. Операційні системи: підручник / В.А. Шеховцов. – К.: Видавнича група BHV, 2005. – 576 с.
10. Струбицький П.Р. Програмна інженерія: навчальний посібник/ П.Р.Струбицький, О.В.Бондар – Тернопіль: ТАНГ, 2004 – 78 с.
11. Сидоров М.О. Вступ до інженерії програмного забезпечення / М. О. Сидоров. – К.: НАУ, 2010. – 112 с.

12. Сидоров М.О. Групова динаміка і комунікації: курс лекцій. – К.: НАУ, 2008 – 74 с.
13. Томашевський В.М. Моделювання систем: підручник/ В.М. Томашевський – К.: BHV group, 2005. – 352 с.
14. Мінухін С.В. Методи і моделі проектування на основі сучасних CASE-засобів: навч. посіб./ С.В. Мінухін, О.М. Беседовський, С.В. Знахур – Харків:ХНЕУ, 2008. – 272 с.
15. Мещанінов О. П. Моделювання систем: навчальний посібник/ О.П. Мещанінов. – Миколаїв: МФНаУКМА, 2001. – 268 с.
16. Ноздріна Л. В. Управління проектами: підручник / Л. В. Ноздріна. – К.: Центр учбової літератури, 2010. – 432 с.
17. Управління проектами: навч. посіб. /за ред. О. В. Ульянченка - Х. : ХНАУ ім. В. В. Докучаєва, 2010. - 522 с.
18. Калач Г. М. Управління проектами : навч. посіб. / Г. М. Калач – Ірпінь: Нац. ун-т держ. податк. Служби України, 2010. – 333 с.
19. Інформаційні системи і технології в економіці: посібник / за ред. В.С. Пономаренка. – К.: Видавничий центр «Академія», 2002 – 544с.
20. Проектування інформаційних систем: посібник / за ред. В.С. Пономаренка – К.:Видавничий центр «Академія», 2002 – 488с.
21. Денісова О. О. Автоматизоване проектування інформаційних систем: навч. посіб. / О. О. Денісова – К. : КНЕУ, 2011. – 413 с.
22. Тлумачний словник з інформатики / Г.Г. Півняк, Б.С. Бусигін, М.М. Дівізінюк та ін. – Донецьк: Нац. гірнич. ун-т, 2010. – 600 с.
23. Пупена О.М. Проектування комп'ютерно-інтегрованих систем / О.М. Пупена – К.: НУХТ, 2013. – 45 с.
24. Воронкова, В.Г. Управління людськими ресурсами: філософські засади: навчальний посібник / В.Г. Воронкової – К.:ВД «Професіонал», 2006. – 576 с.
25. Лавріщева К.М. Програмна інженерія / К.М. Лавріщева – К.: Т-во «Знання», 2008. – 319 с.
26. Бабенко Л.П. Основи програмної інженерії / Л.П.Бабенко, К.М. Лавріщева – К.:Т-во «Знання», 2001. – 269с.

27. Баженов В.А. Інформатика. Комп'ютерна техніка. Комп'ютерні технології/ В.А. Баженов [та ін.] – К.: Каравела, 2003 – 592 с

28. Дибкова Л.М. Інформатика та комп'ютерна техніка/ Л.М. Дибкова – К.: Академвидав, 2007 – 320 с.

29. PMBOK A Guide to the Project Management Body of Knowledge. – Pennsylvania: Project Management Institute, 2009. – 241 s.

30. Dijkstra E.W. Software Engineering Techniques. Structured Programming/ E.W. Dijkstra, J.N. Buxton, B. Randell – Brussels: NATO Science Committee, 1969. – 130 s.

31. Meyer B. Object-Oriented Software Construction / B. Meyer – New Jersey: Prentice Hall, 1997. – 1370 s.

32. Wilkinson B. Parallel programming: techniques and applications using networked workstations and parallel computers / B. Wilkinson, M. Allen – NewYork: Prentice Hall, 2005. – 488 s.

33. Morgan C. Programming from specifications / C. Morgan – New York: Prentice Hall, 1994.– 260 s.

34. Bass L. Software Architecture in Practice / L. Bass, R. Kazman. – London: Addison-Wesley Professional, 2006. – 640 s.

35. Walker R. Software Project Management / R. Walker. – London: Addison-Wesley Professional, 1998. – 448 s.

36. Booch G. The Unified Modeling Language: Users Guide/ G. Booch. – London: Pearson Education, 1999. – 482 s.

37. Larry L.C. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design/ L.C. Larry, A.D Lockwood – London: Addison-Wesley Professional, 1999. – 608 s.

38. Braude E.J. Software Design: From Programming to Architecture/ E.J. Braude – New Jersey:John Wiley & Sons, 2004. – 550 s.

39. Leffingwell D. Managing Software Requirements/ D. Leffingwell, D. Widrig. – London: Addison Wesley, 1999. – 448 s.

40. Beck K. Test-driven Development: By Example/ K. Beck – London: Addison-WesleyProfessional, 2003.– 220 s.

41. Hunt A. Pragmatic Programmer, The: From Journeyman to Master/ A. Hunt D. Thomas. – Addison Wesley, 1999. – 352 s.

42. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software/ Gamma E., Johnson R., Vlissides J., Helm R. – London:Addison-Wesley Professional, 1994. – 416 s.
43. Patterns of Enterprise Application Architecture/ M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. – London:Addison-Wesley Professional, 2002. – 533 s.
44. Monson-Haefel R. Enterprise JavaBeans/ R. Monson-Haefel, B. Burke – O'Reilly Media, Inc. 2006. – 768 s.
45. Writing Effective Use Cases/ Cockburn A. – London: Addison-Wesley, 2001. – 113 s.
46. Characteristics of Software Quality/ B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod and M. J. Merritt. – Lincoln: University of Nebraska-Lincoln, 2013. –206 s.
47. Brooks F.P. The Mythical Man-Month: Essays on Software Engineering/ F.P. Brooks – London: Addison-Wesley Professional, 1995. – 322 s.
48. Lister T. Peopleware: Productive Projects and Teams/ T. Lister, T. DeMarco – London : Addison-Wesley Professional, 2013. – 272 s.
49. Blanchard K. The One Minute Manager Meets The Monkey/ K. Blanchard – William Morrow, 1989. – 144 s.
50. Liskov B. Abstraction and Specification in Program Development/ B. Liskov, J.V Guttag – Cambridge :MIT Press, 1986 – 469 s.
51. Tanenbaum S. Distributed systems: principles and paradigms/S. Tanenbaum, M. Steen. – New York: Prentice Hall, 2006 – 686 s.
52. Tanenbaum S. Modern Operating Systems/ S. Tanenbaum, H. Bos - New York: Prentice Hall, 2001 – 976 s.
53. Tanenbaum S. Computer Networks/ S. Tanenbaum - New York : Prentice Hall, 2002 - 891 s.
54. Cormen T.H. Introduction to Algorithms/ T.H. Cormen, C.E. Leiserson, R. L. Rivest, Clifford Stein – Cambridge :MIT Press, 2001 – 1180 s.
55. Clarke E.M. Model Checking/ E.M. Clarke – Cambridge:Mit Press, 1999 – 330 s.

56. Newcomer E. Understanding Web Services: XML, WSDL, SOAP and UDDI / E. Newcomer – London:Addison-Wesley Professional, 2002 - 368 s.

57. Norman K.L. Cyberpsychology: An Introduction to Human–Computer Interaction/ K.L. Norman – Cambridge: Cambridge university press, 2017. – 437 s.

2 Статті та матеріали конференцій

58. Yukselturk E., Altiok S. An investigation of the effects of programming with Scratch on the preservice IT teachers' self-efficacy perceptions and attitudes towards computer programming //British Journal of Educational Technology. – 2017. – vol. 48. – №. 3. – С. 789-801.

59. Грицюк Ю. І. Особливості визначення вимог до програмного забезпечення та проблеми їх аналізу / Ю. І. Грицюк, І. Ф. Лешкевич // Науковий вісник НЛТУ України. – 2017. – Вип. 27.4. – С. 148-158.

60. Зайцев Є.О., Сидорчук В.Є. Економічні аспекти виробництва програмних продуктів // Всеукраїнська науково-практична конференція: «Конкурентоспроможність національної економіки та освіти: пошук ефективних рішень» - Україна, Вінниця, 15 квітня 2015 – Вінниця: Вінницький торговельно-економічний інститут, 2015. – с. 121 – 122.

61. Blair W., Xi H. Dependent Types for Multi-Rate Flowsin Synchronous Programming (System Description) //Electronic proceedingsin theoretical computerscience. – 2017. – №. 241. – С. 36-44.

62. Зайцев Є.О., Сидорчук В.Є. Застосування середовища LabVIEW в сфері технічного аналізу фінансових ринків // Збірник наукових публікацій за матеріалами міжнародної науково-практичної конференції: «Наука в епоху дисбалансів», частина 2 – 25 січня 2016 р, м. Київ, Україна. – К.: Центр наукових публікацій, 2016. – с. 30-35.

63. Harada T., Takadama K. Machine-Code Program Evolution by Genetic Programming Using Asynchronous Reference-Based Evaluation Through Single-Event Upsetin On-Board Computer (SpecialIssueonAI, Robotics, and Automationin

Space) //Journal of robotics and mechatronics. – 2017. – vol. 29. – №. 5. – С. 808-818.

64. Зайцев Є.О., Сидорчук В.Є. Економіка розробки програмного забезпечення// Всеукраїнська науково-практична конференція: «Шляхи активізації інноваційної діяльності в освіті, науці, економіці» - Україна, Вінниця, 12 квітня 2016 – К.: , 2016. – с.115–117.

65. Зайцев Є.О., Криворучко О.В., Рассамакин В.Я Концептуальні засади створення та розвитку програмного забезпечення для економічних систем // Тези доповідей міжнародної науково-практичної конференції "Глобалізаційні виклики розвитку національних економік" присвяченої 70-річчю Київського національного торговельно-економічного університету 19-20 жовтня 2016 року, м. Київ, Україна – К.: ЦПНМВ КНТЕУ, 2016 – с. 678-690

66. Зайцев Є.О., Сидорчук В.Є. Забезпечення інформаційної безпеки при використанні програмного забезпечення аналізу стану фінансових ринків // Тези доповідей IV Всеукраїнської науково-практичної Інтернет-конференції «Розвиток фінансового ринку в Україні: проблеми та перспективи» – 10 листопада 2016 року, м. Полтава, Україна. – Полтава: ПолтНТУ, 2016. – с. 116-119.

67. Зайцев Є.О., Сидорчук В.Є. Етапи створення та розвитку програмного забезпечення інформаційних систем збору, обробки та аналізу даних // Сборник статей научно-информационного центра «Знание» по материалам XIX международнойзаочнойнаучно-практическойконференции: «Развитие науки в XXI веке», г. Харьков, 14 ноября 2016 г. – Х. : научно-информационный центр «Знание», 2016. – с. 71-79.

68. Shah P., Berges M., Hubwieser P. Qualitative Content Analysis of Programming Errors //Proceedings of the 5th International Conference on Information and Education Technology. – ACM, 2017. – С. 161-166.

69. Зайцев Є.О., Яремич В.Р. Управління ризиками при створенні програмних додатків в контексті SMART-освіти // Тези доповідей Міжнародної науково-методичної конференції Smart-КНТЕУ 24 листопада 2016 року, м. Київ, Україна – К.: ЦПНМВ КНТЕУ, 2016 – с. 156–157.

70. Зайцев Є.О., Сидорчук В.Є. Використанням технологій smart-освіти в процесі проектуванні програмних додатків//Тези доповідей Міжнародної науково-методичної конференції Smart-КНТЕУ 24.11.2016 року, м. Київ, Україна – К.: ЦПНМВ КНТЕУ, 2016 – с. 311–314.

71. McGee S. etal. Does a taste of Computing Increase Computer science enrollment ? //Computing in Science & Engineering. – 2017. – vol. 19. – №. 3. – С. 8-18.

72. Зайцев Е.А., Сидорчук В.Е. Деякі аспекти інформаційної безпеки Украйни //Збірник тез Міжнародної наукової інтернет-конференції "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення" (випуск 17) – 2 березня 2017 року, м. Тернопіль, Україна – Тернопіль, 2017. – С.14–15.

73. Wang X. M. etal. Enhancing Students' Computer Programming Performances, Critical Thinking A wareness and Attitudes to wards Programming: An Online Peer-Assessment Attempt //Educational Technology&Society. – 2017. – vol. 20. – №. 4. – С. 58-68.

74. Goodwin S. etal. What do constraint programming users want to see? Exploring the role of visualisation inprofilingof models and search //IEEE transactionson visualization and computer graphics. – 2017. – Т. 23. – №. 1. – С. 281-290.

75. Kelter T., Marwedel P. Parallelismanalysis: Precise WCETvalues for complex multi-core systems //Science of Computer Programming. – 2017. – Т. 133. – С. 175-193.

76. Поморова, О. В. Аналіз опрацювання метрик якості программного забезпечення на етапі проектування / О. В. Поморова, Т. О. Говорущенко, С. Я. Тарасек // Вісник Хмельницького національного університету. Технічні науки. – 2010. – № 1. – С. 54-62.

3 Internet-ресурси

77. Computer Engineering Curricula 2016 Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering – режим доступу: https://www.computer.org/cms/

Computer.org/professiona-education/curricula/Computer EngineeringCurricula2016.pdf

78. Безопасность на прикладном уровне: PGP и S/MIME. – режим доступу: http://www.intuit.io/studies/courses/553/409/lecture/9375.

79. Інформаційні системи та технології. – режим доступу: https://sites.google.com/site/khomoshyura/elektronni-posibniki-ta-pidrucniki

80. Керування вимогами – режим доступу: https://www.ibm.com /developerworks/ru/library/r-requirements/.

81. What is fundamental test process in software testing? – режим доступу: http://istqbexamcertification.com/what-is-fundamental-test-process-in-software-testing/.

82. Методи збору та виявлення вимог – режим доступу:http://studopedia.com.ua/1_13240_lektsiya--metodi-zboru-ta-viyavlennya-vimog.html.

83. Методи збору вимог або «Як зрозуміти, що хоче замовник?»– режим доступу: http://it-ua.info/news/2016/08/17/metodi-zboru-vimog-abo-yak-zrozumti-scho-hoche-zamovnik.html.

84. На шляху до вдалого проекту: 11 порад для ефективного спілкування з клієнтом і командою – режим доступу: http://it-ua.info/news/2016/10/27/na-shlyahu-do-vdalogo-proektu-11-porad-dlya-efektivnogo-splkuvannya-z-klntom-komandoyu.html.

85. How to communicate effectively in it projects – режим доступу: https://www.smashingmagazine.com/2014/06/communicating-effectively-in-projects.

86. Lammers M. An Application Programming Interface for Synthetic Snowflake Particle Structure and Scattering Data – режим доступу: https://ntrs.nasa.gov/search.jsp?R=20170006242.