МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В.Я. Юрчишин

Проєктування сучасних високопродуктивних обчислювальних систем

ЛЕКЦІЇ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*»)

Київ
«КПІ ім. Ігоря Сікорського»
2022

Рецензенти: *Зубчукчук Віктор Іванович, канд. техн. наук, доц.*
Тимошин Юрій Афанасійович, канд. техн. наук, доц.

Відповідальний редактор *Заболотня Т.М., канд. техн. наук, доц.*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол №1 від 02.09.2022 р.) за поданням Вченої ради факультету прикладної математики (протокол № 1 від 01.09.2022 р.)

Електронне мережне навчальне видання

Юрчишин Василь Якович, к. т. н, доц.

Проєктування сучасних високопродуктивних обчислювальних систем

ЛЕКЦІЇ

Проєктування сучасних високопродуктивних обчислювальних систем: лекції [Електронний ресурс]: навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*»)/ В.Я.Юрчишин; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 8,543Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 279 с.

Навчальний посібник розроблено для вивчення студентами теоретичних відомостей та ознайомлення з практичними прийомами роботи при розробці сучасних високопродуктивних обчислювальних систем.

Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*») освітньо-кваліфікаційного рівня Магістр факультету прикладної математики НТУУ КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Проектування високопродуктивних обчислювальних систем неможливе без застосування сучасних інформаційних технологій, що сприяє розробці сучасного програмного забезпечення, прикладом якого можуть виступати широкомасштабні системи моніторингу, керування та аналізу з глобально розподіленими джерелами даних. Наприклад, значення хмарних та грід-технологій є одними з найважливішими в світі, завдяки їм границі країн, відстані та фізичні обмеження перестають мати минулі значення.

Програма вивчення навчальної дисципліни «Проектування сучасних високопродуктивних обчислювальних систем» складена відповідно до освітньо-професійної програми підготовки здобувачів другого (магістерського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення»

Предметом вивчення навчальної дисципліни є сучасні методи та технології проектування сучасних високопродуктивних обчислювальних систем.

Навчальна дисципліна «Проектування сучасних високопродуктивних обчислювальних систем» є вибірковою дисципліною підготовки магістрів зі спеціальності 121 «Інженерія програмного забезпечення».

Мета та завдання навчальної дисципліни

Мета – формування представлень про сучасну різноманітність методів проектування сучасних високопродуктивних обчислювальних систем, їх реалізації із застосуванням сучасних інформаційних технологій та інструментальних засобів реалізації програмного забезпечення високопродуктивних обчислювальних систем.

Метою лекцій є отримання необхідного рівня знань і придбання практичних умінь і навичок з використання розподілених обчислень, віртуалізації серверних систем, створення програмних продуктів для проведення складних обчислень в локальних та глобальних мережах.

Курс лекцій розрахований на 36 академічних годин аудиторних занять. Посібник складається з 18 лекцій.

1 Технології, методи і засоби проектування високопродуктивних обчислювальних систем.

1.1 Вступ до практичного інженерного проектування.

Все своє свідоме життя я працюю в IT- сфері, причому не простим виконавцем, а з 1967 і до кінця 2014 року приймав активну участь в розробці засобів обчислювальної техніки, перебуваючи переважно більшу частину цього часу в статусі головного конструктора чи керівника теми розроблюваних засобів обчислювальної техніки. Це були досить масштабні проекти, всі вони виконувалися по відповідальних постановах: Постановах держкомітету по науці і техніці, Постановах військово-промислового комплексу, Постановах ради міністрів СРСР, Постановах ЦК КПРС. Це не були звіти, бумажні роботи, пропозиції чи рекомендації комусь – це були проектні роботи, які закінчувалися створенням робочих засобів і впровадженням їх в виробництво. Тому те, про що розповідається – це не запозичене з інших джерел, це те, через що я пройшов. На протязі всього мого розроблювального періоду поряд зі мною сидів програміст і ми фактично разом створювали щось нове (правда, він тоді називався не програміст, а математик). Зараз нічого не змінилося. Ви зможите самостійно створити сайт, якийсь додаток, але Ви самостійно не створите програмне забезпечення для Боінг-737 чи програмне забезпечення керування металоплавильним комплексом, чи атомною електростанцією.

Тому наш курс не про розробку програмного забезпечення як такого. Це Ви робите вже п’ятий рік і будете робити ще рік. Наш курс про розробку комп’ютерних інформаційних систем, в яких програмний продукт займає дуже суттєве місце.

Слід зазначити, що значна кількість викладачів університету і фахівців академічних інститутів АН України не дуже обізнані як в проведенні проктування в традиційному плані, так і важливості дотримання нормативних вимог (дотримання державних стандартів) при інженерному проектуванні і, наприклад, пропагують та використовують для створення програмної документації державний стандарт України ДСТУ 3008-95, який має назву «Документація. **Звіти у сфері науки і техніки**. Структура і правила оформлення.», що не може бути прийнятним для розробки проектної документації. Звітна і проектна документація – це навіть не спорідненні

поняття. Інженерне проектування регламентується діючими в Україні державними стандартами і не може виконуватися по нормативній документації для виконання **звітів**. Інженер (а саме таке звання Ви отримуєте після закінчення вузу) повинен добре розуміти різницю між звітною і проектною документацією.

Що стосується нормативної бази та стандартизації, то майбутній професіонал, особливо якщо він отримає інженерну посаду, повинен чітко розуміти важливість дотримування стандартів в інженерній діяльності, що все матеріальне виробництво пов'язане з конструкторскою документацією (КД), яка є основним елементом інженерної діяльності, і що програмна документація, виступаючи навіть головним компонентом, є складовою конструкторської документації. Просто так все влаштовано. До речі, в Радянському Союзі відділ стандартизації не перебував в структурі інших підрозділів, а безпосередньо був в підлеглості директора підприємства чи головного інженера. Враховуючи те, що Україна приречена відродити свої втрачені індустріальні здобутки і що вища технічні освіта має безпосереднє відношення до цього, студенти повинні навчитися працювати з нормативною технічною документацією, добре ознайомитися як створюється КД, де і як вона зберігається, як вона розмножується, ознайомитися з супроводом розробленої КД, з правилами внесення змін в розроблену КД згідно діючим стандартам. Це неможливо засвоїти прочитанням кількох абзаців, чи навіть сторінок. Таке засвоєння приходить під час певного часу власної проектної діяльності. Далі ми розглянемо початкові відомості з цих питань. Це стисла і неповна інформація, яка не охоплює всіх можливих питань, які можуть постати перед Вами під час професійної діяльності, але зможуть підказати напрямок їх вирішення.

Ви повинні знати, що кожний окремий лист КД обов'язково повинен мати децимальний номер, який є ідентифікатором документу, до якого він відноситься, і при випадковій зміні місцезнаходження якогось листа (загубленні, знищенні, помилкових діях при згрупуванні окремих документів КД) забезпечує його відновлення і безпомилкове повернення в первинний документ на початкове місце. При необхідності внесення змін в розроблений документ вказані зміни мають бути внесені з зазначенням хто, коли і на якій підставі вніс зміни в документ. Такі записи вносяться в відповідні графи штампів (таблички), які є в кожному конструкторському документі. Хай Вас не лякає вираз «конструкторський документ». Всі Ваші програмні напрацювання на інженерній мові є теж конструкторськими документами і відрізняються (в позначенні) від незвичних Вам конструкторських документів всього кількома буковками в кінці позначення документу. Розробка програмної документації регламентувалася загальним стандартом ГОСТ 19, який складався з досить

великої кількості окремих стандартів. В 2019 році розпочалась заміна використання в Україні ГОСТ 19 як застарілих з введенням оновлених, гармонізованих з міжнародними стандартами. Це дуже важливі питання державного значення. Непідпідставно вважається, що державні стандарти як і правила дорожнього руху написані кров'ю. Про це треба пам'ятати!

Дотримання стандартів має дуже велике значення, особливо для програмістів, адже без їх продукції сьогодні неможлива реалізація наймаштабніших проектів: керування транспортом, електрогенеруючими комплексами і особливо атомними електростанціями, системами енергопостачання, прокатними станами, металоплавильними комплексами та іншими енергонебезпечними системами. При цьому слід зазначити, що саме недотримання нормативів привело до найбільшої катастрофи не тільки в Україні, а і в світі – аварії на Чорнобельській АЕС 1986р.

Від виникнення ідеї про створення чогось нового і до її матеріалізації пролягає довга і нелегка дорога, яка починається з пошукової науково-дослідної роботи, де досліджується сама можливість реалізації певної ідеї. Початком цієї роботи є дуже тривалий і детальний пошук існуючих рішень по патентній і науково-технічній літературі. Глибина цього пошуку перш за все напралена не стільки на убезпечення себе від повторних марних спроб створення власного «велосипеду», скільки від створення існуючого і запатентованого іншими, бо це може створити для Вас неабиякі фінансові проблеми. При цьому слід пам'ятати, що патентний пошук треба виконувати не тільки в своєму класі, а при можливості по всіх класах. Класифікація зроблена для відокремлення «мух» від «котлет»: макарони і інтегральні схеми знаходяться в різних класах. Не забувайте, що Зінгер в свій час запантував не швейну машину як таку, а голку з отвором не в її хвості, а біля острія. Коли ми в свій час робили патентний пошук, то нас в вашому віці керівники не раз запитували: «а в класі гробів ви пошук робили?», підкреслюючи цим те, що пошук має бути надзвичайно широким.

За пошуковою науково-дослідною роботою іде так звана приладна науково-дослідна робота з відтворенням відповідних макетних зразків і дослідження їх працездатнисті. Позитивне завершення приладної науково-дослідної роботи (коли макетний зразок витримав необхідні випробування) підтверджує працездатність первинної ідеї. Після цього можна приступати до розробки так званої робочої документації для виготовлення та випробування дослідного зразку. Цей великий етап має назву ДОСЛІДНО-КОНСТРУКТОРСЬКА РОБОТА (ДКР) і складається з кількох етапів.

ДКР складається з таких трьох основних послідовних етапів:

1 – ескізний проект, на якому розроблюється КД для виготовлення макетного зразку, в результаті чого виготовлюється і випробовується макетний зразок виробу та корегується КД на макетний зразок за результатами його випробування. Це зовсім не такий макетний зразок, як на етапах НДР- це перша спроба реалізації кінцевого виробу. На етапі ескізного проекту розроблювальній КД присвоюється літера «Е». Ця літера вписується в основний штамп креслень КД ескізного проекту. Ескізний проект, як правило, виноситься на захист і захищається на науково-технічній раді підприємства.

2 – технічний проект, на якому розроблюється КД на окремі важливі вузли виробу, виготовлюються та випробовуються окремі найважливіші вузли і вносяться зміни (при необхідності) в КД за результатами їх випробувань. На етапі технічного проекту КД присвоюється літера «Т». Ця літера вписується в основний штамп конструкторської документації технічного проекту. Технічний проект теж, як правило, виноситься на захист і захищається на науково-технічній раді підприємства.

3 – розробка КД дослідного зразку, на якому розроблюється КД з літерою «О», по якій виготовлюється дослідний зразок та проводяться його випробування.

Спочатку проводяться попередні, або так звані лабораторні випробування. За їх результатами вносяться зміни в КД дослідного зразку (при необхідності) та доопрацьовується сам дослідний зразок (теж при необхідності), після чого проводяться його приймальні випробування, які в можуть бути відомчими, міжвідомчими, державними або міждержавними (в залежності від важливості розробки). За результатами приймальних випробувань вносяться зміни в КД дослідного зразку і цій КД присвоюється літера «О1». Всі ці перераховані роботи виконуються безпосередньо розробником виробу. Після цього КД з літерою О1 передається (також розробником) на виробниче підприємство для організації серійного виробництва, яке теж відбувається при безпосередній участі розробника КД дослідного зразку.

Саме випробування дослідного зразку дуже важлива і відповідальна процедура. Виконується воно кваліфікованою комісією найвищого рангу. Для проведення випробувань розроблюється (розробником виробу) і погоджується з необхідними інстанціями, передбаченими нормативними документами (обов’язково з замовноком розробки), програма випробування і методика випробування. Програма випробування – це дуже детальний опис що треба

зробити під час випробувань, а методика випробувань – детальний опис як саме це зробити.

Випробування бувають внутрівідомчими – якщо виріб розроблено для використання (виготовлення) в якісь одній структурі (міністерстві, об'єднанні, підприємстві) і приймальна комісія призначається вищестоящою організацією з авторитетних фахівців цієї структури.

Якщо розробка виконується для використання (виготовлення) в різних відомчих структурах (міністерствах, об'єднаннях чи підприємствах різних галузей), то випробування стають міжвідомчими і приймальна комісія призначається з найавторитетніших фахівців цих відомств.

При розробці державного значення випробування стають державними і приймальна комісія призначається з найавторитетніших фахівців держави. При міждержавній розробці випробування проводяться міждержавною комісією при залученні фахівців з цієї галузі з різних країн.

Головою приймальної комісії, або його заступником призначається представник замовника розробки.

Організація серійного виробництва розпочинається з розробки технології виготовлення виробу, детальних технологічних процесів, великої кількості технологічних операцій та пооперaційного контролю, розробки та виготовлення необхідної оснастки для виготовлення та контролю якості виробу.

Розпочинається серійне виробництво з невеликої кількості виробів (сотні – тисячі одиниць) – так званої першої промислової партії (ППП) – її ще називають установочною партією , після чого знову проводять випробування виготовлених зразків ППП майже за такою ж програмою, як і дослідного зразку на приймальних випробуваннях. За результатами цих випробувань проводять корегування КД виробу (при необхідності) та присвоюють відкорегованій КД літеру «А». Тільки після цього розроблена КД стає документацією серійного виробництва. Це є хрестоматійними положеннями для випускника технічного ВУЗу і їх треба пам'ятати як таблицю множення. Після цього розпочинається серійне виробництво.

Слід пам'ятати, що і на етапі серійного виробництва за розробником залишається автрский супровід розробки. Таке детальне описування розробки та впровадження в виробництво нової продукції може показатись перебільшенням, особливо для програмістів, але воно продиктоване власним життєвим досвідом та власними наступаннями на "граблі" на початковому етапі фахового зросту. Знання цих питань є важливо необхідними в повсякденній

практиці фахівця, але в навчальному процесі цьому не приділяється належної уваги. Слід пам'ятати, що сьогоднішні програмісти завтра можуть стати керівниками великих комплексних підрозділів і що сама програмна продукція не є самостійним об'єктом і не може існувати сама по собі в відриві від матеріальних об'єктів.

Програмна документація може розроблюватися як для існуючих об'єктів, так і для новосворюванних, але незалежно від цього вона повинна пройти повний обсяг випробувань, бо мали місце випадки, коли неповне випробовування новоствореної програмної документації для існуючих об'єктів приводили до жахливих наслідків, наприклад, до «заморожування» гігантських металопрокатних станів (стан 2500). В цьому Ви можите пересвідчитися у викладача сусідньої кафедри СКС проф.Зайцева. Він може Вам розповісти як вони розробили в Київському інституті автоматики програму керування прокатним станом 2500, провели випробування в тому обсязі, який вважався їм достатнім, але насправді був неповним (в неповному обсязі) і як через деякий час при прокаті нержавіючого листа (ширина 2,5 м) він (прокатний стан) зупинився і «заморозився». Він дуже гарно розповідає як вони потім його різали автогеном і куди їм (програмістам) після цього вішали ордени і медалі.

Неповне випробування може диктуватися різними обставинами як технічного, так і чисто суб'єктивного характеру, в тім рахунку рішенням осіб, в підлеглості яких знаходиться розробник. Такі особи не будуть нести ніякої відповідальності при виникнені негативних наслідків в майбутньому, вся відповідальність залишається за розробником – про це не можна ніколи забувати. В таких випадках потрібно обов'язково в письмовій формі вказувати свою незгоду з неповним випробуванням в вигляді власної думки. І робити це краще на титульній сторінці такого документу як «Програма і методика випробування» виробу, який є обов'язковим для завершення розробки і тривалий час зберігається після її завершення (здається в архів для зберігання). Такий запис убереже Вас від негараздів, які можуть бути пов'язані з випробуваннями в неповному обсязі.

Слід іще зазначити, що тестуванням не можна замінити випробування.

Тестування може бути елементом (складовою часткою) випробування.

Випробування повинно включати відпрацювання всіх можливих впливових факторів (температурних, кліматичних, механічних, електричних, магнітних, електромагнітних, радіаційних та інших).

Наша дисципліна стосуєься проектування інформаційних систем.

Що таке інформаційна система? Це поняття зараз має не менше визначень, ніж поняття інформації, яке ми з вами безуспішно намагалися визначити 5 років назад на першому курсі в рамках дисципліни «Теорія інформації та кодування». Кардинальна відмінність минулого моменту від сьогоднішнього в тому, що там ви виступали звичайними завсідниками, а тут ви являєтесь центральною (основною) ланкою системи.

Визначення інформаційної системи ви знайдете в будь-якому підручнику і воно приблизно таке:

Інформаційна система – взаємозв'язана сукупність засобів, методів і персоналу, використовуваних для зберігання, обробки і видачі інформації. Все вірно, але далеко не точно як для вас.

Автомобіль – це не набір: колеса, кузов, двері, мотор – це перш за все практична реалізація задуму конструктора автомобіля і тільки потім це транспортний засіб для вас. Як і людина – це не руки, ноги, голова, туловище, ніс, рот, очі, вуха – це важливо, але не найголовніше, бо якщо підійти ззаду з молотком, то все це можна перетворити навіть в щось рухаючесь, але не відображаюче первинний задум проектанта.

Звичайно, технічне втілення інформаційної системи саме по собі нічого не означатиме, якщо не врахована роль людини, для якої призначена вироблена інформація і без якої неможливе її отримання і представлення. Але я не бачу тут творця. А саме творець визначає тут все і в проектуванні інформаційних систем таким являєтесь ви - програмісти, бо всі сучасні інформаційні системи є програмними ситемами.

Пам'ятаєте, ще на першому курсі я вам розповідав, що ваша спеціальність є найнебезпечнішою. Я впевнений, що за 5 років навчання вам не загострювали увагу на цьому питанню, бо там була зовсім інша специфіка і не кожний мав нагоду практично все життя приймати безпосередню участь в створені засобів і систем обчислювальної техніки в промислових маштабах. Я той, хто ще пам'ятає пульт процесора, як він виглядає і як за ним працювати, а також пам'ятаю паперові перфострічки і картонні перфокарти. Вам мабуть навіть важко уявити, що це таке. Мені випало пройти не тільки через створення великих машин і систем, а й бути щільно задіяним в створенні майже повної гами відсутніх тоді вічизняних периферійних пристроїв ЕОМ від відображаючих (в тім рахунку лазерних на великий екран), друкуючих (послідовних, паралельних) пристроїв і до дискових магнітних накопичувачів. І було це не в якості «піднощика патронів», а на такому передньому краю, що більш переднього вже не було. Було нелегко, працювали на зношення, але це

були найкращі часи мого життя. Тому наше опанування буде відбуватися по принципу: «що було, що є і на чому серце заспокоється».

Зараз я хочу повернутися в далекий 1967р. Тоді ми виготовляли ЕОМ Дніпро-2 на Київському заводі обчислювальних машин (завод ВУМ , в подальшому НВО Електронмаш на вул. Велика Окружна 4). Це був повний функціональний аналог американської системи IBM-360, з якої розпочалася сучасна ера обчислювальної техніки. (про запуск в виробництво системи IBM-360 американці об'явили в 4 кварталі 1965 р).

Повертаючись до Дн-2, на заводі ВУМ працювали 10 тисяч працівників, які виробляли 20 машин протягом року і гостро стояло питання необхідності збільшення виробництва. Всі основні елементи тодішнього комп'ютера – ключі, інвертори, логічні елементи, тригери, лічильники, дешифратори, суматори – виконувалися на окремих дискретних транзисторах, діодах, резисторах, конденсаторах.

Основним функціональним елементом був електронний ключ – він же інвертор І-НЕ, АБО-НЕ, або повторювач І, АБО. Ви маєте мати про це уяву з комп'ютерної логіки.

Такий ключ мав в своєму складі 4 – 6 транзисторів, до 10 діодів, стільки ж резисторів і кілька конденсаторів. Ці елементи ключа розміщували на текстолітову плату розміром в сірникову коробку, яку ми називали «модуль». Висота такого модуля теж була як сірникова коробка. 20 таких модулів розміщували на текстолітовій платі, реалізовуючи тригери, регістри, лічильники, суматори та ін.

Платами з модулями набивали шафи процесору, пам'яті, каналів вводу-виведення, великої кількості зовнішних пристроїв обчислювальної машини. При цьому зазначу, що шафа процесору мала розміри: широна – 1,5 м, висота – 1,7 м, завтовшки – 0,5 м. Оперативна пам'ять базової моделі машини займала 2 таких шафи, канали вводу-виведення теж 2 шафи, зовнішня пам'ять на магнітних стрічках займала 9 таких шаф. Одна ЕОМ нараховувала близько 20 таких шаф. Цим і обумовлювалася необхідність такої великої кількості робочої сили і дуже обмежений обсяг випуску.

На базі перших же машин на заводі почали створюватися автоматизовані системи керування (так звані АСУ) підприємством, а саме по розрахунку заробітнної плати, керування виробництвом (відсліджування руху комплектуючих, матеріалів, виготовлення деталей). Це були перші і власні

проекти, але великих проблем по їх створенню не було. Вже 1968 р були створено АСУ Кадри, АСУ Зарплата і АСУ Виробництво.

Значною проблемою стала система автоматизації проектування.

Необхідність в цій системі була надзвичайно велика. Динаміка зростання в ІТ сфері вже тоді була така велика, що ми практично кожний рік створювали нову ЕОМ. Так в 1969 році була спроектована ЕОМ класу IBM-360 АСВТ М-3000, в 1971 році запускалася в виробництво ЕОМ М-4000, в 1973 році в 3-му кварталі розпочато серійний випуск (постачання замовнику) ЕОМ М-4030, далі М-4030-1. Все це великі машини класу IBM-360, IBM-370. Так звані мейнфрейми. Крім них створювалися і запроваджувалися в виробництво ціла гама менших машин: М-6000, СМ-3, СМ-4, СМ-1420, СМ-1700, СМ-1810, машини серії МІР (машина інженерних розрахунків) МІР-1, МІР-2, МІР-3, МРТ, Каштан.

На заводі був досить великий склад розробників, конструкторів і технологів. Так, процесор ЕОМ класу IBM створював відділ в складі 4-х лабораторій, оперативну пам'ять – теж відділ в складі 4-х лабораторій, канали вводу-виведення – теж відділ в складі 4-х лабораторій, далі відділи зовнішніх пристроїв … Розробники і конструктори функціонально працювали в спеціальному конструкторському бюро (СКБ) в загальній кількості 2 тисячі співробітників. Робота розробників закінчувалась створенням електричних схем. Ці схеми потрібно було перевести в конструкторську документацію ЕОМ, основний обсяг якої складала документація друкованих плат з електронними елементами . Цим займалися конструкторські відділи, які по отриманих електричних схемах креслили на папері фотошаблони майбутніх друкованих плат і передавали далі свої креслення в технологічні відділи для створення з цих креслень реалних фотошаблонів, за допомогою яких виготовлювалися в виробництві друковані плати. (Кількість працюючих в технологічних відділах перевищувала кількість працюючих в СКБ.) Це надзвичайно трудомістка і рутинна праця. Крім того, якщо вкрадалася помилка на початковому етапі, то виявлялася вона при виробництві і весь ланцюг мав бути повторений. Вкрай потрібна була система для підвищення продуктивності такої розробки, потрібна була її автоматизація.

Створення такої автоматизованої системи проектування – це програмний продукт, але створити такий програмний продукт самотужки програміст не зможе. І не тільки тому, що програмісту навіть важко було осягнути той величезний обсяг робіт, а і тому, що необхідних технічних засобів тоді ще не

було. Да і зараз при створенні нової складної системи не все є і дуже рідко може бути повна уява що і як потрібно робити.

Отже, створення такої автоматизованої системи проектування – це програмний продукт, але створити такий програмний продукт самотужки програміст не зможе. Програмісту мають підпорядковуватися розробник, конструктор, технолог і виробник. Але повертаючись до нашої системи маємо сказати, що це ще далеко не все. В автоматизованій системі фотошаблон має створювати комп'ютер. Для цього потрібно було як мінімум мати графічний екран. В той час були тільки алфавітно-цифрові дисплеї на електронно-променевих трубках досить невеликих розмірів з недостатньою для створення фотошаблонів роздільною здатністтю (неможливість отримання необхідного розміру п'ятна). Подивіться в Інтернеті коли з'явилися монітори VGA, не кажучи вже про супер-VGA.

Тому для створення такої системи виникла необхіднсть створення широкоформатного графічного екранного пульту (він мав скорочену назву ШГ ЕПП), що потребувало не тільки розробки нових методів точного керування електронним променем на неосвоєних раніше площах екрану, але і створення нової електронно-променевої трубки значно більших розмірів з іншим люмінофором (зеленим) і значно меншим діаметром сліду (не більше 0,3 мм). Це була дуже складна задача, яка була успішно вирішена і впроваджена на заводі ВУМ. Я це добре знаю, бо брав в цьому безпосередню участь. Для її вирішення було залучено 2 міністерства: Мінприлад і МЕП (міністерство електронної промисловості). А організаторами, кураторами, контролерами, поганялами, випробувачами, приймальниками робіт (які виконувались в цих міністерствах) були кілька хлопців вашого віку з заводу ВУМ, які потім на основі цього створювали вказаний широкоформатний графічний екранний пульт. Вам прийдеться робити те саме. Саме тому я так докладно про це розповідаю.

Як я вже казав, автоматизована інформаційна система – це програмний продукт і керувати цим проектом і вести його розробку має програміст, але в складній програмній системі левовою часткою робіт може бути і часто буває апаратна частина. І розробник – а в цьому випадку це програміст – повинен чітко розуміти, що це таке, як його можна зробити при його відсутності, хто це зможе зробити, в які терміни і скільки це може коштувати. Саме цей розробник має розробити всеосяжне зрозуміле технічне завдання на необхідний відсутній виріб. Щоб це зробити одного знання програмування тут явно недостатньо. Треба мати ерудицію в багатьох областях, в тім рахунку і в області

конструювання, добре знати математику, фізику, хімію, матеріалознавство та ін.

Для Вас зараз я добавив би сюди і біології, бо біологічні системи будуть відігравати зростаючу і зростаючу роль. Вказані галузі знань вам потрібно опановувати самотужки і інтенсивно, бо якщо вам дали непогані знання з математики, то фізики вважайте в вас не було зовсім, не кажучи вже про інші галузі. І відповідати собі потрібно не на запитання «ЯК ?», а на запитання «ЧОМУ ?», бо не знаходячи відповіді на запитання «ЧОМУ ?», ми відповідаємо собі на запитання «ЯК ?» і заспокоюємося, вважаючи проблему вирішеною. Але це не так, просто проблема віддаляється на невизначений термін.

Ви можете не погоджуватися з цим, але по цьому поводу можу сказати наступне. Не одну сотню років людство знайоме з електрикою. Переважна більшість викладачів нашого вишу мають дипломи інженера-електрика. Такий диплом отримували не тільки випускники енергетичних факультетів, а і всі випускники ІТ-спеціальностуй до 1995 року. Але жодний із них, навіть з академічними званнями, не знають що таке електрика і не тільки вони. Стверджуючи це, я не маю на увазі закони Ома, Кулона, Фарадея. Я маю на увазі електрику як явище. І це тому, що замість відповіді на запитання «ЧОМУ ?», стали відповідвти на запитання «ЯК ?».

Знаєте, ще будучи студентом я намагався дізнатися в свого викладача-фізика, що таке електрика. Отримував відповідь: це електрони. На моє наступне запитання, а що таке електрони, відповідь була така: « ну ти взагалі. На це питання відповідь дав Дж. Дж.Томсон ще в 1900 році». Я перечитав Дж. Дж.Томсона і в оригіналі і в перекладах, але відповіді так і не знайшов. Її нема. Якби ми її знали, то були би зараз на значно вищому щаблі розвитку. І як мінімум би не стверджували, що ефіру нема (в крайньому разі на академічному рівні).

Все це я кажу для того, щоб розвіяти в вас уяву, що вже через рік ваші «університети» закінчуться і можна буде «почивать на лаврах». Через рік все має розпочатися з більшою інтенсивністю. Правда, не для всіх. Кого влаштує роль сьогоднішних наших заробітчан – ті можуть і на диван.

Отже, вам прийдеться вчитися, вчитися і ще раз вчитися, самотужки опановуючи те, чого вас не вчили і постійно підвищуючи свій рівень в тих питаннях, які ви вчили. Така вона сиром’яжна правда життя. Для цього в вас незрівнянні з нами можливості. Щоб щось добре робити потрібно розбиратися в суті.

При вивчені нашої дисципліни ми не будемо зосереджуватися тільки на програмування як такому. Цим ви інтенсивно займалися протягом останніх 5 років. Більше уваги будемо приділяти тим питанням проектування, які ви не знаєте, але їх потрібно конче знати фахівцю вашого рівня. Саме тому ми починаємо розгляд нашого предмету не з програмування (хотя загально це програмний продукт), бо програмування інформаційних систем Ви вивчали на іншому предметі.

Чому я це так детально розповідаю – тому, що те, що вам прийдеться робити найближчим часом – це щось подібне нашим АСУ Кадри і АСУ Зарплата в 1968р. З цим проблем у вас не буде. А от далі будуть з'являтися проблеми, аналогічні нашим при створенні автоматизації проектування. І вам – програмістам прийдеться не тільки створювати концепцію системи і її програмування, а і як нам в свій час, визначати відсутні технічні засоби, їх параметри, способи їх реалізації, розроблювати технічне завдання на їх створення і відшукувати їх розробників і виготовлювачів. А це все те, що ми дуже коротко розглянули і далі будемо розглядати. Це те, чого ви не знаєте, але повинні знати, бо стаєте головним творцем. Нам було нелегко, бо приходилось поєднувати апаратну і програмну частини, а вам без цих знань буде ще важче, бо програмна і апаратна частини зараз розділилися і розійшлися на значну відстань. Саме тому я так детально розповідав як відбувалося проектування раніше. Бо обізнаний – значить озброєний.

1.2. *Види проектних робіт. Методи, технології та засоби проектування високопродуктивних інформаційних систем*

Проектування – це розробка проекту. Створенню будь-якого об'єкту передує процес проектування. Розробка проекту є розробка комплекту документів, необхідних і достатніх для створення заданого об'єкту.

До цього комплекту входять багато документів, обов'язковими із яких є:
- робочі креслення (найбільш важлива частина проектної документації);
- специфікації (обладнання, матеріалів, тощо);
- калькуляції;
- пояснювальна записка.

Робочі креслення – це комплект креслень, який містить усю інформацію, необхідну для виготовлення об'єкту.

Специфікація – перелік окремих пристроїв, вузлів і деталей, елементів та матеріалів, необхідних дл я реалізації проекту.

Калькуляція є розрахунком вартості реалізації проекту. Вона включає перелік усіх елементів та матеріалів, їх кількостей та цін, обсягів та вартостей робіт, необхідних для реалізації проектованого об'єкту.

Пояснювальна записка містить дані про:
- мету та підстави виконання проекту;
- дані досліджень, щодо умов та вихідних даних, необхідних для виконання проекту;
- вибір та обґрунтування принципів дії проектованого об'єкту;
- вибір обладнання та матеріалів;
- опис принципів дії об'єкту;
- опис порядку виконання робіт по побудові об'єкту.

Розробка проекту включає в себе такі етапи:
- Розробку технічного завдання (ТЗ); (магістерська дисертація теж розпочинається з технічного завдання)
- Розробку технічної документації;
- Виготовлення та випробування зразків;
- Приймання результатів розробки.

ТЗ – є головним вихідним документом для створення продукту. В ТЗ визначається повний перелік вимог до створюваного об'єкту, перелік розроблюваних документів та порядок приймання роботи. ТЗ є офіційним документом, на основі якого створюється об'єкт та перевіряється виконання роботи.

Процес проектування регламентується державними, міжнародними та відомчими стандартами.

Головними учасниками процесу проектування є замовник та виконавець проекту. (в вашій роботі замовник – декан, виконавець – ви)

Розробка інформаційної системи (ІС) – від початкової фази до розгортання – складається з трьох послідовних і поступальних етапів: аналізу, проектування і реалізації. Проектування ІС – логічно складна, трудомістка і тривала робота, яка вимагає високої кваліфікації фахівців, які беруть участь в ній. Проте до теперішнього часу проектування ІС нерідко виконується на інтуїтивному рівні неформалізованими методами, які включають практичний досвід, експертні оцінки і дорогі експериментальні перевірки якості функціонування ІС. Істотним є те, що ІС є програмними системами.

Основна доля трудовитрат при створенні ІС доводиться на прикладне програмне забезпечення (ПЗ) і бази даних (БД).

Найістотнішою і невід'ємнішою властивістю програмних систем є їх складність. Завдяки унікальності і несхожості своїх складових частин програмні системи принципово відрізняються від технічних систем (наприклад,

комп’ютерів), в яких переважають елементи, які повторюються.

Самі комп’ютери складніші, ніж більшість продуктів людської діяльності. Кількість їх можливих станів дуже велика, тому їх важко розуміти, описувати і тестувати. У програмних систем кількість можливих станів на порядок величин перевищує кількість станів комп’ютерів. Це потрібно пам’ятати і враховувати.

Складність ПЗ є істотною, а не другорядною властивістю. Багато проблем розробки ПЗ виходять з цієї складності і її нелінійного росту при збільшенні розміру. Складність є причиною утруднень, які виникають в процесі спілкування між розробниками, що веде до помилок в продукті, перевищення вартості розробки, затягування виконання графіків робіт. Складність викликає труднощі розуміння усіх можливих станів програм, що призводить до зниження їх надійності. Складність структури стримує розвиток ПЗ і можливості додавання нових функцій.

Для успішної реалізації проекту об’єкт проектування має бути передусім адекватно описаний, тобто мають бути побудовані повні і несуперечливі моделі архітектури ПЗ, що обумовлює сукупність структурних елементів системи і зв’язків між ними, поведінку елементів системи в процесі їх взаємодії, а також ієрархію підсистем, що об’єднують структурні елементи. Під моделлю розуміється повний опис системи ПЗ з певної точки зору. Моделі є засобами для візуалізації, опису, проектування і документування архітектури системи. Моделі будуються для того,
щоб зрозуміти і осмислити структуру і поведінку майбутньої системи, полегшити керування процесом її створення і зменшити можливий ризик.

Кінцева мета розробки ПЗ – це не моделювання, а отримання працюючих застосувань (коду). Діаграми – це лише наочні зображення, тому, використовуючи графічні мови моделювання, важливо розуміти, чим вони допоможуть при написанні коду програм.

Основні поняття технології проектування ІС.

Аналіз сучасного стану ринку ІС показує стійку тенденцію росту попиту на інформаційні системи організаційного керування, причому попит продовжує рости саме на інтегровані системи у керування.

Сучасні проекти ІС характеризуються, як правило, наступними особливостями:
– складність опису (досить велика кількість функцій, процесів, елементів даних і складні взаємозв’язки між ними), що вимагає ретельного моделювання і аналізу даних і процесів;
– наявність сукупності тісно взаємодіючих компонентів (підсистем), які мають свої локальні завдання і цілі функціонування, використовують нерегламентовані запити до даних великого об’єму);

-відсутність прямих аналогів, що обмежує можливість використання яких-небудь типових проектних рішень і прикладних систем;
– необхідність інтеграції (композиції) існуючих і розроблюваних додатків;
– функціонування в неоднорідному середовищі на декількох апаратних платформах;
– роз'єднаність і різнорідненість окремих груп розробників по рівню кваліфікації і традиціям використання тих або інших інструментальних засобів, що склалися.

Для успішної реалізації ІС об'єкт проектування має бути адекватно описаний, мають бути побудовані повні і несуперечливі функціональні і інформаційні моделі *архітектури* ІС, які обумовлює сукупність структурних елементів системи і зв'язків між ними, поведінку елементів системи в процесі їх взаємодії, а також ієрархію підсистем, що об'єднують структурні елементи.

Поняття інформаційної системи

Під *системою* розуміють будь-який об'єкт, який одночасно розглядається і як єдине ціле, і як об'єднана в інтересах досягнення поставлених цілей сукупність різнорідних елементів. Тобто, ***система*** – це сукупність взаємозв'язаних між собою елементів, пов'язаних єдиною метою, щоб могла реалізовуватися функція системи.

Інформаційна система – взаємозв'язана сукупність засобів, методів і персоналу, використовуваних для зберігання, обробки і видачі інформації в інтересах досягнення поставленої мети.

Сучасне розуміння інформаційної системи припускає використання як технічного засобу переробки інформації і керування персонального комп'ютеру, великої ЕОМ (мейнфрейм), суперЕОМ, розподіленої системи, Grid-систем та «хмари».

Існують різні інструментальні середовища розробки і супроводу ІС.

Хоча інформаційні системи є звичайним програмним продуктом, вони мають ряд істотних відмінностей від стандартних застосовних програм і систем. Залежно від предметної області ІС можуть сильно розрізнятися по своїх функціях, архітектурі, реалізації:

- якщо ІС призначені для збору, зберігання і обробки інформації, то в основі будь-якої з них лежить середовище зберігання і доступу до даних;
- якщо ІС орієнтуються на кінцевого користувача, який не має високої кваліфікації в сфері застосування обчислювальної техніки, то клієнтські застосування ІС повинні мати простий, зручний, легко освоюваний інтерфейс, який надає кінцевому користувачеві усі необхідні для роботи функції, але в той же час не дає йому можливість виконувати які-небудь зайві дії. Таким чином, при розробці інформаційної системи доводиться вирішувати різні задачі, в даному випадку дві основні:

– завдання розробки БД, призначеної для зберігання інформації;

– завдання розробки графічного інтерфейсу користувача клієнтських застосувань.

Приклади деяких програмних засобів, які є ІС:
1. 1С-Бухгалтерія 8.0. Використовується з метою формування бухгалтерської звітності підприємства перед податковими органами. Є інформаційною системою.

2. Книга MS Excel, яка містить відомості про штатний розклад, працівників підприємства та оснащена макросами, що дозволяють розраховувати заробітну плату і формувати платіжні відомості. Є інформаційною системою.
3. Система комплексної автоматизації діяльності мережі роздрібних магазинів. Є інформаційною системою.
4. Реляційна СУБД DB-2 фірми IBM. Не є інформаційною системою.

Залежно від об'єму вирішуваних завдань, використовуваних технічних засобів, організації функціонування, ІС діляться на ряд груп (класів):

Інформаційні системи організаційного керування – призначені для автоматизації функцій управлінського персоналу як промислових підприємств, так і непромислових об'єктів (готелів, банків, магазинів). Основними функціями подібних систем є: оперативний контроль і регулювання, оперативний облік і аналіз, перспективне і оперативне планування, бухгалтерський облік.

ІС керування технологічними процесами (ТП) – служать для автоматизації функцій виробничого персоналу по контролю і керуванню виробничими процесами та операціями.

ІС автоматизованого проектування (САПР) – призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів при створенні нової техніки або технології.

Розробка інформаційних систем – це циклічний процес створення унікального продукту, який удосконалюється від версії до версії; різні, не завжди послідовні етапи можуть впливати один на одного. По суті, створенню програмного продукту більше відповідає не модель керування проектом з чітким початком і кінцем, а модель керування усім життєвим циклом продукту. Циклічному ходу розробки більше підходять ітеративні методики, що мають на увазі постійну співпрацю між різними функціональними групами в команді програмного проекту і кінцевими користувачами, і тісний взаємозв'язок між різними етапами життєвого циклу системи.

Основні процеси життєвого циклу ІС

Основні процеси життєвого циклу ІС складаються з п'яти процесів, які

реалізуються під керуванням основних сторін, залучених в життєвий цикл ПЗ.Основними сторонами є замовник, постачальник, розробник.

Основними процесами вважають:
1. *Процес замовлення*. Визначає роботи замовника.
2. *Процес постачання*. Визначає роботи постачальника.
3. *Процес розробки*. Визначає роботи розробника.
4. *Процес експлуатації*. Визначає роботи оператора.
5. *Процес супроводу*. Визначає роботи супроводжуючої організації, яка надає послуги з супроводу програмного продукту.

Одним з базових понять методології проектування ІС являється поняття життєвого циклу її програмного забезпечення (ЖЦПЗ). ЖЦПЗ – це безперервний процес, який починається з моменту ухвалення рішення про необхідність його створення і закінчується у момент його повного вилучення з експлуатації.

В ході життєвого циклу створення ІС проходить через аналіз предметної області, збір вимог, проектування, кодування, тестування, супровід і інші види діяльності.

При розробці ІС створюються і перероблюються різного роду ***артефакти*** – створювані людиною інформаційні сутності, документи в досить загальному сенсі, які беруть участь в якості вхідних даних і виходять в якості результату. Прикладами артефактів є: модель предметної області, опис вимог, технічне завдання, архітектура системи, проектна документація на систему в цілому і на її компоненти, прототипи системи і компонентів, власне, початковий код, призначена для користувача документація, документація адміністратора системи та ін.

Структура ЖЦПЗ за стандартом ISO/IEC 12207 базується на трьох групах процесів:
– основні процеси ЖЦПЗ (придбання, постачання, розробка, експлуатація, супровід);
– допоміжні процеси, які забезпечують виконання основних процесів (документування, керування конфігурацією, забезпечення якості, верифікація, атестація, оцінка, аудит, рішення проблем);
– організаційні процеси (керування проектами, визначення, оцінка і поліпшення самого ЖЦ, навчання).

Розробка включає усі роботи із створення ІС і її компонентів відповідно до заданих вимог, включаючи оформлення проектної і експлуатаційної документації.

Розробка ІС включає, як правило:

*1.Планування і оцінка проекту***.** Під час цієї фази виявляються властивості, які повинна мати готова система, тобто, створюється задум

системи в остаточному варіанті. Під час цієї фази приймаються рішення відносно розмірів, часу відгуку і інших параметрів системи. Визначення здійснюється за допомогою двох основних категорій – вимог і *специфікацій.*
2. Аналіз системних і програмних вимог. Системний аналіз задає роль кожного елементу в комп'ютерній системі, взаємодію елементів один з одним. *Аналіз вимог* відноситься до програмного елементу – програмного забезпечення. Уточнюються і деталізуються його функції,
характеристики і інтерфейс.
3. Проектування алгоритмів і програмних структур.
Проектування полягає в створенні представлень:
– архітектури ПЗ і модульної структури ПЗ;
– алгоритмічної структури ПЗ і структури даних;
– вхідного і вихідного інтерфейсу (вхідних/вихідних форм даних).
4. Реалізація (кодування) полягає в перекладі результатів проектування в текст на мові програмування.
5. Тестування відповідає за виконання програми для виявлення дефектів у функціях, логіці і формі реалізації програмного продукту.
*6. Введення в дію***–** це внесення змін до експлуатованої ІС. Цілі змін:
– виявлення та виправлення помилок;
– адаптація до змін зовнішнього для ІС середовища;
– удосконалення ІС за вимогами замовника.
7. Експлуатація включає роботи по впровадженню компонентів ІС в експлуатацію, в тім рахунку:
– супровід;
– проведення навчання персоналу;
– модифікацію (модернізацію) ІС.

Вимоги – ця властивість, необхідна для вирішення проблеми або досягнення мети. При описі вимог використовуються такі поняття, як якість, можливості або інші характеристики системи, передбачувані її використання або середовища. Найкращою мовою опису вимог, що забезпечує строге, точне і всеосяжне експертне їх вираження, є професійно-природна мова експертів у сфері діяльності організації користувача.

Специфікація – цей опис на мові розробника зовні відомих характерних особливостей поведінки системи.

Верифікація **–** це процес визначення того, чи відповідає поточний стан розробки, досягнутий на цьому етапі, вимогам цього етапу. Перевірка дозволяє оцінити відповідність параметрів розробки з початковими вимогами. На кожному етапі слід переконуватися, що зробили саме те, що планували, і це відповідає загальній логіці розробці – *«Ми створюємо систему правильно».*

Валідація (перевірка правильності) – процес перевірки того, що реалізована система задовольняє пред'явленим вимогам і працює так, як

передбачалося – *«Ми створюємо (створили) правильну систему».*

Стратегії конструювання ІС

Використовують три основні стратегії конструювання ІС:

1. ***Одноразовий прохід, або каскадна*** (водоспадна) стратегія - «модель водоспаду» або «waterfall» – лінійна послідовність етапів конструювання.
2. ***Спіральна (чи ітеративна) стратегія***. *Передбачає* ***ітераційний*** *процес розробки ІС.*

3. ***Еволюційна стратегія***. Система також будується у вигляді послідовності версій, але на початку процесу визначені не всі вимоги. Вимоги уточнюються в результаті розробки версій.

При каскадній розробка розбивається на етапи, перехід до наступного етапу відбувається після повного завершення всіх робіт попереднього етапу. Кожен етап завершується оформленням повного комплекту документації. Склад і зміст цієї документації передбачає, що реалізація проекту може бути продовжена іншою командою розробників. (Рис. 1)

Основним елементом концепції спіральної моделі є ітерація. Кожна ітерація є завершеним циклом розробки, що призводить до випуску діючої версії виробу (або певної його частини). В подальшому від ітерації до ітерації цей продукт вдосконалюється й наприкінці перетворюється у завершену систему (рис. 2)

Рисунок 1 – Каскадна модель

Рисунок 2 – Спіральна модель ІС

Кожне коло спіралі відповідає за створення фрагменту (частки) або версії програмного продукту, на цьому ж колі уточнюються мета й характеристики проекту, вимоги до якості, плануються роботи для наступного витку спіралі. На кожній ітерації послідовно конкретизуються деталі проекту, внаслідок чого вибирається обґрунтований варіант. В подальшому цей варіант доводиться до остаточної реалізації. Використання спіральної моделі дає змогу здійснювати перехід до наступного етапу реалізації проекту, не чекаючи повного завершення поточного етапу робіт, – їх можна буде виконати у наступній ітерації.

Завдання кожної ітерації – якнайшвидше отримати діючий продукт, який можна показати користувачам системи. Ця обставина робить простішим процес внесення уточнень і доповнень до проекту.

Розвиток каскадної та спіральної моделей призвів до їхнього природнього зближення. Результатом такого зближення стала поява сучасного ітераційного підходу, який фактично становить раціональне поєднання цих двох моделей. Різні варіанти ітераційного підходу реалізовані в більшості сучасних методів: Rational Unified Process (RUP), Framework Microsoft Solutions (MSF), XR тощо.

Методи і засоби проектування

Поєднання різних ознак класифікації методів проектування обумовлює характер використовуваної технології проектування ІС, серед яких виділяються два основних види технологій: канонічна й індустріальна. Індустріальна технологія проектування, у свою чергу, розбивається на автоматизоване (використання CASE-технологій) і типове (параметрично-орієнтоване чи модельно-орієнтоване) проектування. Використання індустріальних технологій проектування не виключає використання в окремих випадках канонічної технології.

Характеристика технологій проектування.

Для конкретних видів технологій проектування властиве застосування певних засобів розробки ІС, що підтримують виконання як окремих проектних робіт, етапів, так і їхніх сукупностей. Тому перед розробниками ІС, як правило, постає завдання вибору засобів проектування, що за своїми характеристиками найбільшою мірою відповідають вимогам конкретного підприємства. Засоби проектування повинні бути:

– інваріантними до об’єкту проектування;
– охоплювати в сукупності всі етапи життєвого циклу ІС:
– технічно, програмно й інформаційно сумісними;
– простими в освоєнні та застосуванні;
– економічно доцільними.

Засоби проектування ІС можна розподілити на ручні й комп’ютерні (рис. 3).

Ручні засоби проектування застосовуються на всіх стадіях і етапах проектування ІС. Як правило, це засоби організаційно-методичного забезпечення операцій проектування і, в першу чергу, різні стандарти, які регламентують процес проектування систем. Сюди ж відноситься єдина система класифікації і кодування інформації, уніфікована система документації, моделі опису й аналізу потоків інформації і т. д.

Комп’ютерні засоби проектування можуть застосовуватися як на окремих, так і на всіх стадіях та етапах процесу проектування ІС і відповідно підтримують розробку елементів проекту системи, розділів проекту системи, проекту системи в цілому. Всю множину комп’ютерних засобів проектування залежно від їх призначення поділяють таким чином:

– засоби проектування операцій обробки інформації;
– засоби загальносистемного призначення;
– функціональні засоби проектування;
- засоби автоматизації проектування.

До засобів проектування операцій обробки інформації належать алгоритмічні мови, бібліотеки стандартних підпрограм і класів об’єктів, макрогенератори, генератори програм типових операцій обробки даних тощо, а також засоби розширення функцій операційних систем (утиліти).

Засоби проектування зображені на наступному малюнку.

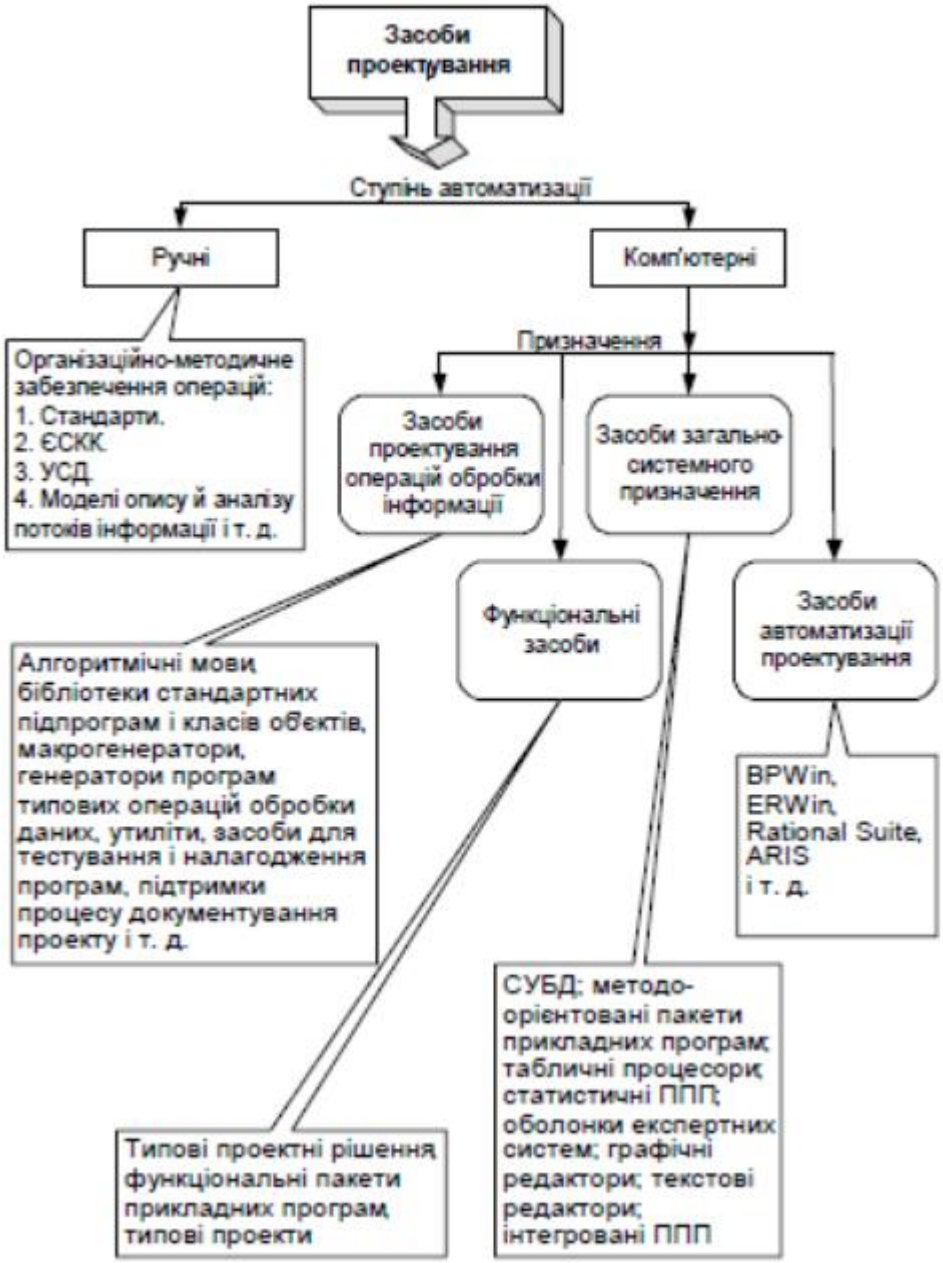


Рисунок 3. Засоби проектування

Сюди включаються також такі найпростіші інструментальні засоби проектування, як засоби для тестування і налагодження програм, підтримки процесу документування проекту і т. д. Особливість останніх програм полягає в тому, що за їх допомогою підвищується продуктивність праці проектувальників, але не розробляється закінчене проектне рішення. Таким чином, засоби даного підкласу підтримують окремі операції проектування ІС і можуть застосовуватися незалежно один від одного.

До засобів загальносистемного призначення відносяться:

– системи керування базами даних;

– методо-орієнтовані пакети прикладних програм (вирішення завдань дискретного програмування, математичної статистики і т. д.);

– табличні процесори;
– статистичні ППП (пакети прикладних програм);
– оболонки експертних систем;
– графічні редактори;
– текстові редактори;

– інтегровані ППП (інтерактивне середовище із вбудованими діалоговими можливостями, що дозволяє інтегрувати перераховані вище програмні засоби).

Для перерахованих засобів проектування характерне їх використання для розробки технологічних підсистем ІС: введення інформації, організації збереження й доступу до даних, обчислень, аналізу і відображення даних, ухвалення рішень.

Різноманітність предметних областей породжує різноманітність даних засобів, орієнтованих на:
– тип організаційної системи (промислова, непромислова сфери);
– рівень керування (наприклад, підприємство, цех, відділ, ділянка, робоче місце);
– функцію керування (планування, облік і т. д.).

До функціональних засобів проектування систем обробки інформації відносяться:
– типові проектні рішення;
– функціональні пакети прикладних програм;
– типові проекти.

Засоби автоматизації проектування (CASE-засоби) підтримують проектування на стадіях і етапах створення ІС. Термін CASE (Computer Aided System/Software Engineering) тлумачиться досить широко – від автоматизації розробки програмного забезпечення до розробки складних ІС в цілому.

Сучасні CASE-засоби, у свою чергу, класифікуються в основному за двома ознаками:
– за охоплюваними етапами процесу розробки ІС;
– за ступенем інтегрованості:
– окремі локальні засоби (tools);
– набір неінтегрованих засобів, які охоплюють більшість етапів розробки ІС (toolkit);
– цілком інтегровані засоби, пов’язані загальною базою проектних даних репозиторієм (workbench).

Огляд CASE-засобів буде зроблений в одній з наступних лекцій.

2 Стандарти проектування та оформлення проектної документації.

2.1. Класифікація та різновиди стандартів для проектування інформаційних систем

Комплекс документів, які регламентують діяльність розробників при проектуванні інформаційних систем, називають нормативно-методичним забезпеченням (НМЗ). Документи, які входять до складу НМЗ – це стандарти, керівні документи, методики, положення, інструкції, шаблони, тощо.

Вказані документи регламентують:
– порядок розроблення, впровадження та супроводу системи;
– загальні вимоги до складу системи, до зв'язків між її компонентами, до якості програмного забезпечення;
– різновиди, склад і зміст проектної та програмної документації.

Донедавна вся розроблювана програмна документація повинна була створюватися з дотриманні вимог Єдиної Системи Програмної Документації (відповідний перелік ГОСТ'ів 19.XXX). В першій лекції ми коротко торкнулися цього питання. Стандарти ЄСПД містять формальні вимоги до складу, змісту та оформлення документів, які описують програму на різних стадіях її життєвого циклу. Також в них є вимоги до порядку зберігання й оновлення документації. Ці стандарти вважаються застарілими (концептуально й за формою), але зараз стандарти ЄСПД продовжують активно використовувати під час розроблення програмної документації.

До сучасної бази НМЗ можна віднести такі міжнародні та вітчизняні стандарти [1, 2]:
- офіційні (public) стандарти:
– міжнародні:
• ISO/IES (ISO – від International Organization of Standardization, міжнародна організація зі стандартизації;
• IEC – від International Electrotechnical Commission, міжнародна комісія з електротехніки);
• ANSI (від American National Standards Institute) американського національного інституту стандартів;
• стандарти міжнародних консорціумів та комітетів зі стандартизації (наприклад консорціум OMG);
• стандарти України (ГОСТ, ДСТУ):

Інші способи класифікації стандартів

1. За об'єктом стандартизації:
– стандарти на продукти й послуги;
– стандарти на процеси й технології;
– стандарти на форми колективної діяльності, або управлінські стандарти.
2. За предметом стандартизації:
   1) функціональні стандарти:
      – стандарти на мови програмування;
      – стандарти на інтерфейс, протоколи тощо;

2) стандарти на організацію життєвого циклу ІС.

До окремої групи прийнято виділяти корпоративні стандарти.

Корпоративні стандарти

Реальне застосування будь-якої технології проектування, розробки й супроводу ІС у певній організації або проекті неможливе без застосування низки стандартів (правил або угод), яких мають дотримуватись усі учасники проекту. Для більшості складних проектів доводиться створювати власні комплекти нормативних і методичних документів, які регламентують процеси, етапи роботи й документи для певних програмних продуктів. Такі стандарти називаються **корпоративними** й становлять угоду щодо єдиних правил організації технології або управління у межах організації. До таких стандартів належать:

– стандарти проектування;

– стандарти на оформлення документації;

– стандарти на інтерфейси користувача. Корпоративні стандарти – це не сучасне створіння Вони існували і широко застосовувалися ще в минулому столітті. Називалися вони тоді стандартами підприємства СТП. Ми працювали в основному по цих СТП. Вони не могли понижувати вимоги діючих тоді державних стандартів – ГОСТів – вони підвищували їх вимоги. Недотримання стандарту тоді переслідувалося законом і тягнуло за собою кримінальну відповідальність. І керівник відділу стандартизації був в підпорядкуванню тільки керівника підприємства, а сам відділ стандартизації не міг входити в якийсь підрозділ. Він був окремим підрозділом з безпрсереднім підпорядкуванням першому керівнику. Так було не тільки в нас, а й у всьому світі і наша модуль була копією західної.

Стандарт проектування має регламентувати:

– набір моделей (діаграм) на кожному етапі проектування системи та рівень їх деталізації;

– правила фіксації проектних рішень на діаграмах, зокрема правила іменування (ідентифікації) об’єктів, набір атрибутів для всіх об’єктів, правила заповнення цих атрибутів на кожній стадії проектування, правила оформлення діаграм, зокрема вимоги до їхньої форми, розмірів, наповнення тощо;

– вимоги до конфігурації робочих місць, налаштування операційної системи, загальні налаштування проекту тощо;

– механізм підтримки спільної роботи над проектом, зокрема: правил інтеграції підсистем проекту, підтримки проекту в актуальному для всіх розробників стані (регламент обміну проектною інформацією, правила та методики синхронізації тощо), правил перевірки проектних рішень на їхню несуперечність тощо.

Стандарт оформлення проектної документації має встановлювати:

– комплектність, склад та структуру документації на кожній стадії проектування;

– вимоги до її оформлення в тім рахунку зміст розділів, підрозділів,

пунктів, таблиць тощо;
– правила підготовки, розгляду, погодження та затвердження документації із вказанням граничних строків виконання кожної стадії;
– вимоги до налаштування технічних засобів підготовки документації;
– вимоги до налаштування CASE-засобів для забезпечення підготовки документації відповідно до встановлених вимог.

Стандарт інтерфейсу користувача має встановлювати:
– правила оформлення екранних форм (шрифти, колір символів та ін.), склад і розташування вікон та елементів керування;
– правила використання засобів вводу (клавіатури, миші тощо);
– правила оформлення текстів довідок;
– перелік повідомлень;
– правила обробки (реакцію) на дії користувача.

За основу корпоративних стандартів можуть прийматися галузеві, національні або міжнародні стандарти. До них можуть належати різного роду методичні матеріали провідних фірм-розробників програмного забезпечення, фірм-консультантів, наукових центрів, консорціумів зі стандартизації тощо.

Стандарти на процеси життєвого циклу інформаційних систем

Міжнародні стандарти проектування інформаційних систем

ISO/IEC 12207 – базовий стандарт на процеси життєвого циклу ІС, орєнтований на різні типи проектів. У стандарті не передбачено конкретних етапів ЖЦ ІС. Замість того визначено тільки низку процесів. Тому стандарт дає змогу реалізувати довільну модель життєвого циклу, і це є його перевагою.

ГОСТ 34.601-90 – поширюється на автоматизовані інформаційні системи (АІС) і регламентує стадії, етапи їхнього створення, містить описання змісту робіт на кожному з етапів. Стандарт орієнтовано на використання каскадної моделі життєвого циклу.

ISO/IEC 12207: 1995-08-01 й супутні стандарти

Міжнародний стандарт **ISO/IEC 12207** (запропонований у 1995 р. технічним комітетом ISO/IEC JTC1 «Інформаційні технології, підкомітет SC7, проектування програмного забезпечення») є найважливішим нормативним документом, який регламентує життєвий цикл програмного забезпечення. Він визначає структуру життєвого циклу, що містить процеси, дії та завдання, які мають бути виконані під час створення ПЗ. Його регламенти є загальними для будь-яких моделей життєвого циклу, методологій і технологій розроблення ПЗ. Способи виконання дій та завдань, включених до перерахованих процесів, можуть бути довільного типу.

Відповідно до базового міжнародного стандарту ISO/IEC 12207 усі процеси життєвого циклу поділяються на три групи:

1 Головні процеси:

1) придбання, – визначає дії підприємства-покупця;
2) постачання, – визначає дії підприємства-постачальника;
3) розробка, – визначає дії підприємства-розробника;
4) функціонування, – визначає дії підприємства-оператора, який забезпечує обслуговування системи загалом (а не тільки програмного

забезпечення) в процесі її функціонування;
5) супровід, – визначає дії персоналу, який забезпечує супроводження програмного продукту, тобто керування, модифікацію, підтримку поточного стану та функціональної придатності; до цього процесу також належить встановлення (інсталяція) ПЗ на обчислювальній системі та його видалення.
2. Допоміжні процеси – процеси, які призначені для підтримки виконання головних процесів, забезпечення якості проекту, організації верифікації, перевірки й тестування ПЗ тощо:
1) процес документування;
2) процес керування конфігурацією;
3) процес забезпечення якості;
4) процес верифікації;
5) процес атестації;
6) процес аудиту;
7) процес спільної оцінки;
8) процес вирішення проблем.
3. Організаційні процеси визначають дії і завдання, які виконують як замовник, так і розробник проекту, а саме:
1) процес керування;
2) процес створення інфраструктури проекту;
3) процес удосконалення;
4) процес навчання.
Характеристики стандарту ISO/IEC 12207:
– динамічність: забезпечується послідовністю виконання процесів, коли один процес у разі потреби активізує інший (повністю або частково); це дозволяє реалізувати довільну модель ЖЦ;
– адаптивність: стандарт ISO 12207 передбачає виключення процесів, різновидів діяльності й завдань, непридатних у певному проекті.

Нижче наведено орієнтовний опис головних процесів життєвого циклу (табл. 1– 2).

Таблиця 1 – Орієнтовний опис процесу придбання. Відношення «Виконавець – Замовник»

Вхід	Дія	Вихід
1	2	3
1 Рішення щодо початку робіт із впровадження ІС. 2 Результати обстеження діяльності замовника.	1 Ініціювання. 2 Підготовка заявочних пропозицій. 3 Підготовка договору. 4 Контроль діяльності постачальника.	1 Техніко-економічне обґрунтування впровадження ІС. 2 Технічне завдання на ІС. 3 Договір на постачання/розроблення.
Вхід	**Дія**	**Вихід**
1	**2**	**3**
3 Результати аналізу ринку ІС. 4 План постачання/ розроблення. 5 Комплексне тестування ІС	5 Приймання ІС	4 Акти приймання етапів роботи. 5 Акт приймально-здавальних випробувань
1 Технічне завдання на ІС. 2 Рішення керівництва про участь у розробці. 3 Результати тендеру. 4 Технічне завдання на ІС. 5 Розроблена ІС і документація.	1 Ініціювання. 2 Відповідь на заявочні пропозиції. 3 Підготовка договору. 4 Планування виконання. 5 Постачання ІС.	1 Рішення щодо участі у розробленні. 2 Комерційні пропозиції/конкурсна заявка. 3 Договір на постачання/розроблення. 4 План управління проектом. 5 Акт приймально-здавальних випробувань.

Таблиця 2 – Орієнтовний опис процесу розробки. Відношення «Виконавець – Розробник ІС»

Вхід	Дія	Вихід
1 Технічне завдання на ІС. 2 Технічне завдання на ІС, модель ЖЦ. 3 Технічне завдання на ІС. Підсистеми ІС. 4 Специфікації вимоги до компонентів ПО. 5 Архітектура ПО. 6 Матеріали докладного проектування ПО. 7 План інтеграції ПО, тести. 8 Архітектура ІС, ПЗ, документація на ІС, тести	1 Підготовка. 2 Аналіз вимог до ІС. 3 Проектування архітектури ІС. 4 Розроблення вимог до програмного забезпечення. 5 Проектування архітектури ПЗ. 6 Докладне проектування ПО. 7 Кодування і тестування програмного забезпечення. 8 Інтеграція ПО і кваліфікаційне тестування ПО. 9 Інтеграція ІС і кваліфікаційне тестування ІС	1 Використовувана модель ЖЦ, стандарти розроблення. 2 План робіт. 3 Склад підсистем, компоненти обладнання. 4 Специфікації вимоги до компонентів ПЗ. 5 Склад компонентів ПО, інтерфейси з БД, план інтеграції ПЗ. 6 Проект БД, специфікації інтерфейсів між компонен-тами ПЗ, вимоги до тестів. 7 Тексти модулів ПЗ, акти автономного тестування. 8 Оцінка відповідності комплексу ПЗ вимогам ТЗ 9 Оцінка відповідності ПО, БД, технічного комплексу та комплекту документації вимогам ТЗ

Для підтримки застосування стандарту ISO/IEC 12207 було розроблено низку технологічних документів, а саме:
– керівництво для ISO/IEC 12207 (ISO/IEC TR 15271 : 1998 Information technology – Guide for ISO/IEC 12207);
– керівництво щодо застосування ISO/IEC 12207 в керуванні проектами (ISO/IEC TR 16326:1999 Software engineering – Guide for the application of ISO/IEC 12207 to project management).
З часом було розроблено стандарт на процеси ЖЦ систем ISO/IEC 15288 (System life cycle processes). Під час створення стандарту було враховано практичний досвід створення систем в урядових, комерційних, військових та академічних організаціях. У розробці стандарту брали участь провідні фахівці з системної інженерії, програмування, керування якістю, людськими ресурсами, безпекою тощо. Стандарт було орієнтовано на широкий клас систем, однак його головне призначення – це підтримка створення інформаційних систем. Згідно стандарту ISO/IEC серії 15288 до структури життєвого циклу рекомендовано включати такі групи процесів:
1. Договірні процеси:
– придбання комплектуючих, програмного забезпечення, контенту тощо (це можуть бути внутрішні рішення або рішення від зовнішнього постачальника);
– постачання (внутрішні рішення або рішення зовнішнього постачальника).

2. Процеси підприємства:
– керування навколишнім середовищем;
– інвестиційне управління;
– керування життєвим циклом ІС;
– керування ресурсами;
– керування якістю.
3. Проектні процеси:
– планування проекту;
– оцінка проекту;
– контроль проекту;
– керування ризиками;
– керування конфігурацією;
– керування інформаційними потоками;
– прийняття рішень.
4. Технічні процеси:
– визначення вимог;
– аналіз вимог;
– розробка архітектури;
– впровадження;
– інтеграція;
– верифікація;
– перехід;
– атестація;
– експлуатація;
– супроводження;
– утилізація.
5. Спеціальні процеси: визначення та встановлення взаємозв'язків відповідно до завдань і цілей проекту.

Стадії створення ІС відповідно до ISO/IEC 15288:
– формування концепції, – аналіз потреб, вибір принципів щодо реалізації проектних рішень;
– розроблення, – проектування самої системи;
– реалізація, – розробка системи на рівні баз даних та програмного коду;
– експлуатація, – введення системи до експлуатації та її подальше використання;
– підтримка, – забезпечення функціонування системи, внесення відповідних змін з метою адаптації до умов зовнішнього середовища;
– зняття з експлуатації, – припинення використання системи, її демонтаж, створення архіву даних тощо.

Канонічне проектування інформаційних систем та його нормативне забезпечення

За своєю сутністю канонічне проектування відображає технологію індивідуального проектування ІС. Серед особливостей канонічного підходу можна виділити такі:
– відображення особливостей «ручної» технології проектування;

– орієнтація на індивідуальне проектування;
– акцент на планування та розподілі робіт на рівні виконавців;
– можливість інтеграції виконання елементарних операцій;
– орієнтація на порівняно невеликі інформаційні проекти;
– використання універсальних інструментальних засобів комп'ютерної підтримки.

Канонічне проектування спрямоване на мінімальне використання типових проектних рішень. Адаптація проектних рішень у канонічному проектуванні здійснюється виключно шляхом перепрограмування відповідних програмних модулів.

Організація канонічного проектування ІС заснована на використанні каскадної моделі життєвого циклу й передбачає низку стадій та етапів. Принцип розподілу процесу проектування на стадії та етапи спрямований на проектування системи «зверху-вниз», тобто на послідовну розробку спочатку узагальнених, а потім деталізованих проектних рішень.
Оскільки об'єкти автоматизації за складністю різні, то набір завдань для локальних рішень у межах певної системи, стадії та етапи робіт для її створення можуть суттєво відрізнятись за трудомісткістю. Тому допустимо об'єднувати послідовні етапи, вилучати певні етапи на довільній стадії проекту, а також починати виконання наступної стадії до закінчення попередньої. Усі стадії та етапи створення ІС фіксуються в договорах й технічних завданнях на виконання робіт.

Канонічне проектування регламентує низка стандартів, а саме:
ГОСТ 34.003 – Терміни та визначення головних понять у сфері автоматизованих систем;
ГОСТ 34.201 – Види, комплектність і позначення документів при створенні автоматизованих систем;
ГОСТ 34.601 – Стадії створення автоматизованих систем;
ГОСТ 34.602 – Технічне завдання на створення ІС;
ГОСТ 34.603 – Види випробувань автоматизованих систем;
ГОСТ 2.105 – Загальні вимоги до текстових документів.

Стосовно проекту розробки ІС можна виділити три узагальнені стадії проектування:
– передпроектну;
– проектну;
– післяпроектну.

Більш детально вказані стадії розглянемо в наступній лекції.

Питання для самоконтролю

1. Що регламентують стандарти, керівні документи, методики, положення, інструкції та шаблони проектування?
2. У чому полягає особливість стандартів ЄСПД?
3. Що таке ISO/IES?
4. Назвіть головні різновиди класифікації стандартів.
5. Які є різновиди стандартів за об'єктом стандартизації?
6. Які є різновиди стандартів за предметом стандартизації?

7. Для чого потрібні корпоративні стандарти? Вкажіть на їх відмінність.
8. Вкажіть різновиди корпоративних стандартів.
9. Що має регламентувати стандарт проектування?
10. Що має встановлювати стандарт оформлення проектної документації? Наведіть приклади.
11. Що має встановлювати стандарт інтерфейсу користувача? Наведіть приклади.
12. Охарактеризуйте головні положення стандарту ISO/IEC 12207.
13. Вкажіть головні процеси життєвого циклу за стандартом
14. Охарактеризуйте стадії та головні етапи канонічного підходу до проектування інформаційних систем.
15. Мета та головні завдання, що має вирішити складання технічного завдання на проектування системи.
16. Охарактеризуйте склад розділів технічного завдання на проектування інформаційної системи.
17. У чому полягає типове проектування інформаційних систем? Охарактеризуйте його особливості, переваги, сферу використання.
18. Вкажіть головні підходи, що використовуються при типовому проектування інформаційних систем.

Література

1. Технології створення програмних продуктів та інформаційних систем : навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 201 7. – 93 с.

2. Шаховська Н. Б., Литвин В.В. Проектування інформаційних систем : навч. посібник / – Львів : Магнолія-2006, 2011. – 380 с.

3. Стадії та етапи проектування.

3.1 *Стадії та етапи проектування інформаційних систем*

Перед тим, як перейти до стадій та етапів проектування інформаційних систем зазначимо, що інформаційний комплекс, або ЕОМ без програмного забезпечення – це купа заліза і пластмаси, а програмне забезпечення без апаратної частини, або ЕОМ теж нічого не значить. І сьогодні склалася практика, що коли розроблюється апаратна частина, то розробка проводиться по її канонах, а коли розроблюється ПЗ, то розробка проводиться по її канонах. Тут ми розглядаємо складну систему з апаратних і програмних засобів, головним розробником якої виступає програміст і при цьому він являється творцем не тільки ПЗ, а всієї системи. І якщо програмна частина має свої стадії

та етапи проектування, а апаратна частина створюється по своїм, то головний розробник зобов'язаний знати як одні, так і інші. Тому не бурчіть, що це засмічує ваші голови, що це не на часі, що до цього треба ще дожити. Потрібно. Така ваша доля. Як кажуть в таких випадках - Бачили очі, що вибирали: їжте – щоб вам повилазило!

Головні етапи робіт (стадії) для ІС можна подати так:

1. Формування вимог до інформаційної системи

Має за мету виконання таких дій:

– обстеження об'єкту та обґрунтування необхідності створення ІС;
– формування вимог користувача до ІС;
– оформлення заявки на розроблення ІС.

2. Розроблення концепції ІС.

Ця стадія складається з таких етапів:

– вивчення об'єкту;
– проведення науково-дослідних робіт (за необхідністю);
– розроблення варіантів концепції ІС відповідно до вимог користувачів;

3. Технічне завдання

Це дуже відповідальний етап, який має за мету розроблення й затвердження ТЗ на створення інформаційної системи. Підготовчим етапом для формування технічного завдання є техніко-економічне обґрунтування проекту. Це спеціальний документ, який фіксує результати визначення стратегії впровадження ІС. У цьому документі має бути чітко визначено результати виконання проекту для замовника, а також вказані графіки проведення робіт і графік фінансування на різних стадіях виконання проекту. Додатково у такому документі вказують терміни, період окупності, очікувані зиски та економічний ефект від реалізації проекту.

Зазвичай, у техніко-економічному обґрунтуванні вказують:

– усі ризики та обмеження, які впливають на успішність проекту;
– умови експлуатації майбутньої системи: архітектурні, програмні, вимоги до апаратних засобів та до компонентів ПЗ і СУБД;
– користувачів системи (можливо, за категоріями);
– функції, які виконуються системою;
– інтерфейси та розподіл функцій між людиною та системою;
– терміни завершення етапів, форма прийому/передачі робіт;
– горизонт проекту;
– можливості подальшого розвитку системи.

За результатами обстежень формується технічне завдання на інформаційну систему.

Технічне завдання – це головний документ, який визначає вимоги та порядок створення (розвитку або модернізації) системи, згідно до яких проводиться її розроблення та приймання під час введення до експлуатації.

Розроблення технічного завдання складається із таких розділів:

– загальні відомості;
– призначення та мета створення (розвитку) ІС;

– характеристика об’єктів автоматизації;
– вимоги до ІС;
– склад і зміст робіт щодо створення ІС;
– порядок контролю та приймання ІС;
– вимоги до складу та змісту робіт щодо підготовки та введення ІС до експлуатації;
– вимоги щодо документування;
– джерела розроблення.

Проектна стадія орієнтована переважно на розробку технічного та робочого проектів. Процес розробки технічного завдання передбачає обстеження об’єкту автоматизації (організації або підрозділу) та його діючої системи керування. Для вирішення завдань інформаційного забезпечення необхідно також проаналізувати інформаційні потоки, форми документації, системи кодування, а також усе, що пов’язано зі структурою баз даних та СУБД, що визначає склад вихідних технологічних вимог.

4. Ескізний проект.

Включає такі етапи робіт:
– розробка попередніх проектних рішень щодо ІС та її складових;
– розробка ескізної документації на систему та на її складові.

Якщо для ІС певного об’єкту автоматизації раніше обрані проектні рішення є очевидними, стадія ескізного проекту може бути виключена з послідовності робіт. Тобто ця стадія не є обов’язковою.
Крім того, на етапі ескізного проекту мають бути визначені:
– цілі та функції ІС та її підсистем;
– склад комплексів задач і окремих завдань;
– концепція та структура інформаційної бази;
– функції СУБД;
– функції та параметри програмних засобів;
– очікуваний ефект від впровадження системи.

Документація з результатами робіт за сукупністю прийнятих проектних рішень погоджується, затверджується і використовується у подальшому проектування для виконання робіт щодо створення ІС.

На підставі технічного завдання (зокрема за наявністю ескізного проекту) розробляється технічний проект.

5. Технічний проект

Включає такі етапи:
– розробка проектних рішень щодо системи та її складових;
– підготовка документації на систему в цілому та на її окремі складові;
– розробка й оформлення документації на постачання комплектуючих до системи і (або) технічних вимог (завдань) на їх розробку;
– розробка завдань на проектування в суміжних частинах проекту об’єкту автоматизації.

На етапі технічного проекту проводяться роботи науково-дослідного та експериментального характеру для вибору головних проектних рішень, а також розраховується економічна ефективність системи. Важливим аспектом розробки технічного проекту є аналіз всієї інформації, що використовується на

предмет таких характеристик, як повнота, відсутність дублювання і надмірності, несуперечливість тощо. Також на цьому етапі визначають форми вихідних документів.

6. Робоча документація (РД)

Передбачає такі дії:

– розроблення РД на систему й окремі її складові;
– розроблення або адаптацію програм.

Один із головних етапів стадії робочого проектування – розроблення РД на інформаційне забезпечення системи. До складу такої документації входять:

– технічний проект ІС;
– описання баз даних;
– перелік вхідних та вихідних даних і документів.

Стадія технічного проектування завершується підготовкою та оформленням документації на постачання та комплектування ІС, визначенням технічних вимог і складанням ТЗ на розроблення системи.

Стадія «Робоча документація» передбачає створення як програмного продукту, так і повного комплекту супроводжувальної документації. Така документація має надавати всі відомості, які забезпечують виконання робіт на стадіях введення ІС до експлуатації, самої експлуатації ІС, а також відомості щодо підтримки рівня якості системи (дотримання експлуатаційних характеристик).

Післяпроектна стадія має за мету реалізацію заходів щодо впровадження системи, підготовку приміщень і технічних засобів, навчання персоналу. Також проводиться експлуатація системи з вирішенням низки завдань, аналізуються результати випробувань, реалізуються заходи щодо супроводу й доопрацювання тощо.

Стадія 7. Введення до експлуатації

Складається із таких пунктів:

– підготовка об’єкту автоматизації до введення ІС у дію;
– підготовка персоналу, за необхідністю – його перекваліфікація;
– комплектація ІС необхідними програмними й технічними засобами, інформаційним забезпеченням тощо;
– проведення будівельно-монтажних робіт (при необхідності);
– проведення пусконалагоджувальних робіт;
– проведення попередніх випробувань;
– проведення дослідної експлуатації;
– здійснення приймальних випробувань.

Головними різновидами випробувань для ІС є такі:

– попередні випробування;
– дослідна експлуатація;
– приймальні випробування, вони можуть бути розширені додатковими випробуваннями ІС і її складових частин.

Впродовж попередніх випробувань (за відповідною програмою та методикою), проводять випробування системи на її працездатність і відповідність ТЗ, усувають недоліки та вносять зміни до проектної та

супровідної документації. Далі проводять дослідну експлуатацію системи, аналізують її результати, здійснюють доробку програмного забезпечення і додаткове налаштування технічних засобів.

На етапі приймальних випробувань увагу концентрують на відповідності ТЗ, аналізують результати комплексних випробувань системи, усувають недоліки, які були виявлені під час цієї роботи. За результатами всіх випробувань оформлюють відповідні акти щодо приймання ІС до дослідної експлуатації, її завершення та приймання системи до постійної експлуатації.

8. Супроводження ІС

Ця стадія передбачає проведення таких дій:

– виконання робіт згідно до гарантійних зобов'язань;

– післягарантійне обслуговування системи.

Головними процесами цієї стадії є здійснення робіт щодо усунення недоліків, виявлених підчас експлуатації системи протягом гарантійного терміну, аналіз роботи системи за реальних умов, виявлення відхилень, з'ясування причини їхнього виникнення, усунення причин відхилень і недоліків, забезпечення стабільної експлуатації та характеристик системи.

Типове проектування інформаційних систем

Згідно із самою назвою, методи типового проектування під час розроблення нової системи зорієнтовані на максимальне використання прототипів (типових проектних рішень або ТПР).

Типове проектне рішення розробляється так, щоб бути придатним до багаторазового використання.

Застосування методів типового проектування має свої особливості. Головною умовою для використання таких методів є можливість декомпозиції системи, яка проектується, на окремі складові (підсистеми, програмні модулі, комплекси виконуваних завдань тощо), для реалізації яких можна знайти типові проектні рішення, які можуть бути адаптовані до потреб певного підприємства.

Крім власне функціональних елементів (програмних, апаратних тощо), типове рішення має включати необхідний комплекс документації з детальним описом ТПР (зокрема процедур налаштування).

За рівнем декомпозиції системи можна виділити такі класи ТПР:

– елементні ТПР – рішення, зроблені для окремого елементу (задачі, різновиду забезпечення тощо);

– підсистемні ТПР – рішення, які розраховані на окремі підсистеми;

– об'єктні ТПР – рішення, розраховані на певний набір підсистем ІС.

Типові рішення кожного із класів мають як певні переваги, так і недоліки. Розглянемо найбільш типові з них.

До переваг елементних ТПР можна віднести реалізацію модульного підходу до проектування ІС. З іншого боку, це призводить до великих витрат на доопрацювання ТПР певних елементів, а також до витрат на об'єднання різних елементів внаслідок їхньої поганої сумісності.

Підсистемні ТПР також дають змогу реалізувати модульний підхід до проектування ІС. Крім того, вони дають змогу здійснювати параметричне

налаштування компонентів на об’єкти різних рівнів керування.

Взаємопов’язані компоненти та високий ступінь інтеграції елементів ІС призводять до мінімізації витрат на проектування і на програмування. Однак у разі декількох розробників програмного забезпечення з’являються проблеми в об’єднанні різних функціональних підсистем. Окрім цього, з погляду безперервного реінжинірингу процесів адаптивність ТПР є недостатньою.

Об’єктні ТПР мають такі переваги, як:

– масштабованість (можливі конфігурації інформаційної системи для різної кількості робочих місць);

– методологічна єдність, тобто загальна методологічна база для всіх компонентів ІС;

– сумісність усіх компонент ІС;

– відкритість архітектури інформаційної системи; це дає змогу розгортати ТПР на платформах різного типу;

– розвинуті можливості щодо конфігурування, здатність використовувати необхідні підмножини компонентів системи.

Щодо недоліків об’єктних ТПР, то серед них можна виділити проблеми реалізації типового проекту на оригінальному об’єкті управління. За певних обставин це призводить до необхідності зміни організаційної структури об’єкта автоматизації.

Під час реалізації типового проектування застосовуються два головні підходи, а саме: параметрично-орієнтоване і модельно-орієнтоване проектування.

Етапами параметрично-орієнтованого проектування є:

– постановка завдань та оцінка можливості використання для їхнього вирішення пакетів прикладних програм (ППП): для оцінки відповідності ППП поставленим завданням використовують спеціальну систему критеріїв;

– аналіз доступних ППП (з огляду на критерії відповідності);

– вибір і придбання необхідних ППП;

– налаштування параметрів придбаних ППП, адаптація та допрограмування функцій.

Серед критеріїв оцінки ППП виділяють такі групи:

– загальне призначення та функціональні можливості пакета;

– відмінні ознаки й властивості пакету;

– вимоги до технічних і програмних засобів;

– документування пакету;

– фінансові фактори, доступність ППП та економічна доцільність його використання;

– особливості інсталяції пакету;

– особливості експлуатації пакету;

– допомога постачальника щодо впровадження та підтримки пакету;

– оцінка якості пакету та досвід його використання;

– перспективи подальшого розвитку та оновлення функцій пакету.

Варто відзначити, що кожна із вказаних груп критеріїв у подальшому може бути деталізована до рівня сукупності приватних показників. Це дає додаткову інформацію для кожного аспекту аналізу обраного ППП. Значення критеріїв

визначаються з використанням методів експертного оцінювання.

На практиці часто використовують ще один підхід до реалізації типового проектування, – так зване модельно-орієнтоване проектування. Сутність цього підходу полягає у прискореній адаптації наявних характеристик типової ІС (з огляду на модель об'єкту автоматизації) з використаннями спеціального програмного інструментарію.

За таким підходом технологія проектування повинна мати кошти як для роботи з моделлю певного підприємства, так і з моделлю типової ІС.

У репозиторії типової інформаційної системи зберігається модель об'єкту автоматизації, яка є основою для конфігурації програмного забезпечення. Крім того, у репозиторії міститься базова модель ІС і типова (або референтна) моделі її певних класів. Базова модель ІС описує бізнес-процеси, організаційну структуру, бізнес-об'єкти, бізнес-функції .

Модель певного підприємства може бути побудована або внаслідок вибору фрагментів типової моделі з огляду на особливості об'єкту автоматизації (накшталт BAAN Enterprise Modeler), або з використанням автоматизованої адаптації цих модулів з огляду на думки експертів (SAP Business Engineering Workbench). Модель підприємства, за якою здійснюється автоматичне конфігурування та налаштування ІС, зберігається в репозиторії. За потреби ця модель може бути відкоригована та адаптована до певних вимог.

Впровадження типової ІС починається з аналізу результатів передпроектного обстеження підприємства, які мають бути оформлені у вигляді вимог до певної ІС. Для оцінки таких вимог може бути використана методика ППП.

Результатом та метою наступного етапу є формування попередньої моделі ІС, яка повинна повною мірою відображати особливості реалізації ІС для певного об'єкту. Попередня модель – це основа для вибору типової моделі системи. Також така модель потрібна, щоб визначити перелік компонентів, для реалізації яких будуть необхідні інші програмні засоби або інструментальні засоби, які є у складі типової ІС. Загалом, під час реалізації типового проекту має місце виконання таких операцій:

– встановлення глобальних параметрів системи;
– визначення структури об'єкту атоматизації;
– визначення структури головних даних;
– завдання переліку реалізованих функцій і процесів;
– описання інтерфейсів;
– описання звітів;
– налаштування авторизації доступу;
– налаштування системи архівування.

Завдяки чисельним перевагам, типове проектування широко представлено й активно використовується в сучасних засобах.

А стосовно апаратної частини:

Стадії розроблення та етапи проектування апаратної частини інформаційної системи (згідно ДСТУ 3944 та 3974-2000):

Якщо будете створювати ІС з існуючими технічними засобами, то наведені перед цим стадії та етапи є зайвими і враховуйте тільки системні.

Питання для самоконтролю

1. Назвіть особливості та головні відмінності канонічного підходу до проектування інформаційних систем.
2. Охарактеризуйте стадії та головні етапи канонічного підходу до проектування інформаційних систем.
3. Мета та головні завдання, які має вирішити складання технічного завдання на проектування системи.
4. Охарактеризуйте склад розділів технічного завдання на проектування інформаційної системи.
5. Які роботи має включати ескізний проект?
6. Які етапи має включати технічний проект?
7. Які дії передбачає етап «Робоча документація»?
8. Які дії передбачає етап «Введення до експлуатації»? Охарактеризуйте їх та наведіть приклади.
9. Які дії передбачає етап «Супровід»? Охарактеризуйте їх та наведіть приклади.
10. У чому полягає типове проектування інформаційних систем? Охарактеризуйте його особливості, переваги, сферу використання.
11. Вкажіть головні підходи, які використовуються при типовому проектуванняі інформаційних систем.

Література

1. Технології створення програмних продуктів та інформаційних систем : навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко ; Харків. нац. університет міського господарства ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 201 7. – 93 с.
2. Шаховська Н. Б., Литвин В.В. Проектування інформаційних систем : навч. посібник / – Львів : Магнолія-2006, 2011. – 380 с

3.2 Процес створення інформаційної системи (ІС).

В цій лекції зробимо загальний огляд робіт по створенню ІС та загальні питання створення ІС і розглянемо це в стислому вигляді, тому що вони дещо перегукуються з розглядами в попередніх лекціях. Приведене *стосується в першу чергу складних інформаційних систем.*

1. Процес створення ІС являє собою сукупність упорядкованих в часі, взаємопов'язаних, об'єднаних робіт, виконання яких необхідне і достатнє для створення ІС, відповідної заданим вимогам.

2. Етапи створення ІС виділяються як частини процесу створення Системи з міркувань раціонального планування і організації робіт, які закінчуються заданим результатом.

3. Роботи з розвитку ІС здійснюють по етапах, застосовуваним для створення ІС.

4. Склад і правила виконання робіт визначають у відповідній документації організацій, які беруть участь у створенні конкретних видів ІС. Залежно від специфіки створюваних ІС і умов їх створення допускається виконувати окремі етапи робіт до завершення попередніх, паралельне в часі виконання етапів робіт, включення нових етапів робіт.

Розробка ІС є досить трудомісткий процес, який вимагає іноді досить тривалого часу. Тому організація розробки, вибір раціональної технології розробки відіграють істотну роль в підвищенні ефективності створення ІС. Сформована в даний час технологія створення ІС являє собою результат великої кількості досліджень і узагальнення значного досвіду розробки і впровадження в різні сфери складних інформаційних систем. В цілому розробка ІС підрозділяється на два важливих підходи: - макропроектування;
- мікропроектування.

Основна мета макропроектування - визначити вигляд майбутньої інформаційної системи, який включає в себе попередній опис мети, підцілі функцій, завдання і методи їх вирішення, контури інформаційного, програмного і технічного забезпечення, організаційного і техніко-економічного обґрунтування доцільності створення системи. Результати макропроектування служать основою подальших робіт зі створення системи.

При мікропроектуванню проводиться формування технічних рішень і створення робочої документації ІС. Стадійність підрозділяється на стадію технічного завдання (ТЗ), технічного і робочого проектування та впровадження.

В процесі реалізації ТЗ виконуються роботи, метою яких є:

- підтвердження доцільності і детальне обґрунтування можливості створення ефективної ІС з функціями і технічними характеристиками, сформульованими у вигляді вихідних технічних вимог до системи;
- планування сукупності всіх науково-дослідних, дослідно-конструкторських робіт, проектних і монтажних робіт, термінів їх виконання і фіксації організацій - виконавців;
- підготовка всіх матеріалів, необхідних для проведення проектних робіт.

При досягненні всіх цілей вихідні технічні вимоги до ІС перероблюються в ході проведених на стадії ТЗ передпроектних досліджень в обгрунтоване, узгоджене і затверджене ТЗ для її створення. ТЗ є основним документом, на відповідність яким здійснюється перевірка системи при її передачі в

промислову експлуатацію і містить план - графік всіх наступних робіт зі створення системи.

Підставою для початку стадії ТЗ є укладення договору між замовником і основним виконавцем на проведення передпроектних, науково-дослідних робіт (НДР), які закінчуються розробкою ТЗ на створення системи. Основними етапами стадії ТЗ є:

- попереднє обстеження об'єкту;
- передпроектні НДР;
- обліковое проектування ІС;
- Розробка ТЗ на створення ІС.

Основна мета етапу "Передпроектні НДР" - визначення і аналіз найбільш складних завдань керування для попереднього вибору способів їх вирішення. При вирішені таких завдань виконавці використовують необхідні математичні моделі процесів.

На етапі "Обліковое проектування" розроблюються основні матеріали, які підтверджують доцільність і можливість створення ІС .

Основна мета робіт етапу "Розробка ТЗ" - складання на базі результатів передпроектних робіт ТЗ на створення ІС . Склад , зміст і порядок оформлення ТЗ на створення ІС регламентуються стандартами, відповідно до яких ТЗ є обов'язковим вихідним документом для проектування системи, на відповідність яким здійснюється її перевірка при передавання в експлуатацію і при здаванні приймальній комісії.

ТЗ включає в себе наступні розділи:

- вступна частина;
- характеристики ІС;
- мета і призначення ІС ;
- техніко-економічні показники;
- вимоги до системи;
- вимоги до замовника з підготовки об'єкта керування;
- перелік робіт зі створення системи;
- порядок впровадження системи в промислову експлуатацію.

Процес дослідження починається з побудови організаційно-технологічної, інформаційної моделі ІС і завершується розробкою її математичної моделі.

Найбільш важливим і першочерговим завданням передпроектного обстеження є системний опис ІС. Системний опис об'єкту дає можливість отримати його організаційну, технологічну, інформаційну та математичну модель і вимоги до кібернетичної моделі системи керування. Однією з найважливіших завдань передпроектного обстеження є формування інформаційної моделі об'єкту керування. Інформаційна модель являє собою сукупність сигналів, даних і повідомлень, які несуть інформацію про об'єкт

керування зовнішньому середовищі. Вона організована за певними правилами. Таким чином, під інформаційною моделлю можна розуміти всю суму відомостей про об'єкт керування і про завдання керування цим об'єктом. В рамках цього поняття в якості моделі можна розглядати схему потоків інформації, що звертаються в процесі керування об'єктом.

Інформаційна модель включає в себе:

- перелік сигналів і показників;
- елементи технологічної моделі;
- елементи організаційної структури;
- характеристики потоків інформації;
- перелік процедур обробки інформації.

Аналіз інформаційних моделей дозволяє:

- оцінити обсяги внутрішньої інформації та інформації, які надходить і переданої;
- зробити висновки про траєкторії руху сигналів;
- оцінити перелік сигналів, повідомлень, за якими приймаються управлінські рішення;
- ступінь використання тих чи інших сигналів;
- перелік надходжувальних сигналів.

Метою робіт, які виконуються на стадії "Технічне проектування (ТП)", є розробка основних технічних рішень по створюваній системі і остаточне визначення її вартості. Зміст цих робіт зводиться до проведення загальносистемного і апаратурного синтезу системи і розробка її спеціального програмного та інформаційного забезпечення.

Підставою для включення в план і виконання робіт на стадії ТП служить затверджене ТЗ на створення системи і договір або інший документ про фінансування робіт. Роботи стадії ТП завершуються розробкою:

- загальносистемних рішень, необхідних і достатніх для випуску на стадії "робочий проект" експлуатаційної документації на систему в цілому;
- проектно-кошторисної документації, яка входить до складу розділу "автоматизація ТП" технічного проекту;
- проектів заявок на розробку нових технічних засобів. Заявки передаються замовнику для подальшого оформлення в установленому порядку;
- документацію спеціального програмного та інформаційного забезпечення, включаючи технічне завдання на програмування.

Основні результати робіт стадії оформлюються у вигляді технічного проекту ІС . Склад і зміст проекту має бути достатнім для розгляду і затвердження на їх підставі на стадії робочого проектування всієї робочої документації, в тім рахунку робочих креслень та інших документів, які використовуються при

виробництві будівельних, монтажних та налагоджувальних робіт. Етапами створення ІС є:

- системотехнічний синтез ІС;
- апаратно-технічний синтез ІС;
- розробка завдань на проектування в суміжних частинах проекту об'єкту автоматизації;
- розробка технічних завдань на оперативно-диспетчерське обладнання, яке не випускається серійно;
- підготовка заявок на розробку нових засобів автоматизації;
- розробка кошторису на ІС;
- розрахунок очікуваної техніко-економічної ефективності ІС;
- технічне проектування ПЗ;
- порівняльний аналіз розроблюваної ІС і її відомих аналогів.

Відповідно до нормативної документації "Робочий проект (РП)" являє собою технічну документацію, затверджену в установленому порядку, яка містить уточнені і деталізовані загальнопроектні рішення, програми та інструкції щодо вирішення завдань, і так само уточнену оцінку економічної ефективності системи і уточнений перелік заходів по підготовці об'єкту до впровадження. Метою робіт, які виконуються на стадії РП, є випуск робочої документації на створювану систему.

Стадія "Впровадження" являє собою завершальну частину роботи зі створення ІС. Метою і головним результатом робіт, виконуваних на даній стадії, є передача діючої системи в експлуатацію.

Стадія "Впровадження" включає в себе наступні етапи:

- підготовка об'єкту до впровадження системи;
- налагодження системи;
- дослідна експлуатація;
- приймально-здавальні випробування;
- здача системи державній, міжвідомчій або відомчої приймальній комісії;
- доопрацювання системи за результатами дослідної експлуатації і випробувань.

Учасниками проектування системи є:

- організація - розробник ІС;
- замовник цієї системи;
- постачальники обчислювальної техніки і готового програмного забезпечення. Можуть залучатися професіонали зі створення баз даних і інформаційних баз знань, локальних обчислювальних мереж.

Стадіями створення ІС є:

-технічне завдання;

- технічне проектування;
- робоче проектування;
- впровадження.

Етап технічного завдання в свою чергу включає:

-стадію передпроектного обстеження об'єкту, на якій встановлюється об'єкт дослідження, програма обстеження, організаційний план обстеження;

Стадія проведення дослідницької роботи включає наступні етапи:

- отримання відомостей про стан, закономірності розвитку і функціонування об'єкту;
- отримання відомостей про можливість раціоналізації функціонування об'єкта;
- отримання вихідних даних для побудови структурної, функціональної, інформаційної та технологічної моделей об'єкту.

Створення ескізу (вигляду ІС) містить:

- розробку попередньої структури, складу автоматизованих функцій, загальний алгоритм функціонування, попередній вибір комплексу завдань, попередню оцінку витрат, попередню оцінку надійності та ефективності майбутньої системи, попередній розподіл завдань між людиною і обчислювальною технікою;
- підготовку та випуск технічного завдання на систему.

При розробці технічного завдання повинні бути використані принципи:

- точність викладу; стислість викладу; лаконічність;
-коректність; конкретність вимог (чисельне вираження вимог);
- наявність вимог до складу системи;
- наявність вимог до її характеристик;
- наявність вимог до показників; облік вимог до нормативно - технічної документації;
- доказ доцільності розробки.

Структура *технічного завдання*:

-загальні відомості, призначення та мета створення системи, характеристика об'єкту автоматизації, вимоги до системи, склад і зміст робіт зі створення системи, порядок контролю і приймання системи, вимоги до складу та змісту робіт з підготовки об'єкту автоматизації до введення системи в дію .

Метою етапу *технічного проектування* є розробка основних технічних рішень по створюваній системі і, можливо, остаточне визначення її вартості (якщо це обумовлено договором). Роботи стадії технічного проекту завершуються розробкою: загальносистемних рішень; проектної документації (для технічного проекту будівництва); переліком необхідних засобів обчислювальної техніки; документації спеціального програмного та інформаційного забезпечення.

Основними стадіями етапу *робочого проектування* є: розробка робочої документації на технічне забезпечення; розробка робочої документації на програмне та інформаційне забезпечення; розробка експлуатаційної документації.

Метою етапу впровадження є передача створеної системи в експлуатацію. Стадії впровадження - підготовка об'єкту до впровадження. На цій стадії виконуються роботи: будівельно - монтажні (за потребою), комплектація системи, забезпечення персоналом.

Налагодження ІС - на цій стадії виконуються: налагодження окремих частин ІС, комплексне налагодження системи, метою яої є перевірка і досягнення правильності виконання алгоритмів функціонування системи як людино - машинного комплексу, проведення випробувань на працездатність.

Дослідна експлуатація системи здійснюється відповідно до розробленої програми і передбачає перевірку технічного стану системи, визначення якісних та кількісних показників виконання функцій системи, перевірку готовності персоналу до експлуатації системи, доопрацювання програмного забезпечення та коригування експлуатаційної документації.

Зміст технічного завдання на ІС

В процесі реалізації ТЗ виконуються роботи, цілями яких є:

- підтвердження доцільності і детальне обґрунтування можливості створення ефективної ІС з функціями і технічними характеристиками, сформульованими у вигляді вихідних технічних вимог до системи;
- планування сукупності всіх науково-дослідних, дослідно-конструкторських робіт, проектних і монтажних робіт, термінів їх виконання і фіксації організацій-виконавців;
- підготовка всіх матеріалів, необхідних для проведення проектних робіт.

При досягненні цих цілей вихідні технічні вимоги перероблюються в ході проведених на стадії ТЗ передпроектних досліджень в обгрунтоване, узгоджене і затверджене ТЗ для створення ІС. ТЗ є основним документом, на відповідність яким здійснюється перевірка системи при її передачі в промислову експлуатацію і містить план-графік всіх наступних робіт зі створення системи.

Підставою для початку стадії ТЗ є укладення договору між замовником і основним виконавцем на проведення передпроектних, науково-дослідних робіт (НДР), що закінчується розробкою ТЗ на створення системи.

Вихідними матеріалами для робіт стадії ТЗ є:

-договір про наміри;

-техніко-економічне обґрунтування створення системи;

- вихідні функціональні і техніко-економічні вимоги замовника до системи;

- вихідні дані, які містяться в матеріалах, розроблених на стадії етапу техніко-економічного обґрунтування (ТЕО).

Основними етапами стадії ТЗ є:

-попереднє обстеження об'єкту, який автоматизується;

-передпроектні НДР;

- обліковое проектування;

-розробка ТЗ на створення.

Основними вихідними документами стадії ТЗ є:

-ТЗ на створення ІС, яке містить технічні вимоги та план-графік робіт, погоджені замовником і основним виконавцем;

-уточнене ТЕО намічених в ТЗ рішень;

- науково-технічні звіти, які містять результати проведених передпроектних досліджень і облікового проектування.

Технічне завдання на створення є обов'язковим вихідним документом для проведення проектних і НДР на стадіях технічного і робочого проектування. Основними принципами розробки ТЗ є точність, лаконічність, коректність, конкретність, облік вимог нормативно-технічної документації, які підтверджують доцільність розробки системи.

Основна мета етапу «Попереднє обстеження автоматизуємого об'єкту », полягає в визначенні видів і обсягів НДР, необхідних основному виконавцю для детального обґрунтування доцільності та можливості створення пропонованої замовником системи. Методично етап зводиться до вивчення вихідних матеріалів, представлених замовником за результатами стадії ТЕО; збору додаткових матеріалів або у формі різних проектних завдань на новий організаційний технологічний процес для проектованої ІС, або шляхом вивчення чинного процесу, або його близьких аналогів; аналізу причин втрат, простоїв і т.п. з метою виявлення можливостей збільшення випуску продукції, підвищення її якості та зниження втрат; оцінкою відомих вітчизняних і зарубіжних технічних рішень, які могли б бути використані для реалізації технічних вимог замовника.

В результаті проведених на даному етапі робіт складається ТЗ на передпроектні НДР, якщо ці роботи виконує основний виконавець. У завданні повинні бути зафіксовані цілі і обсяги НДР, напрямки досліджень, сформульовані на підставі результатів проведеного попереднього обстеження об'єкту та розгляду вимог до системи. Основна мета етапу «Передпроектні НДР» - визначення і аналіз найбільш складних завдань керування для попереднього вибору і способів їх вирішення. При вирішенні таких завдань виконавці широко використовують математичні моделі процесів. Аналіз технологічного процесу як об'єкту керування закінчується складанням змістовного опису ІС. Основні розділи опису ІС повинні відповідати вимогам

нормативних документів. Аналіз інформаційних потоків та формулювання критеріїв керування і обмежень закінчується описом об'єкту, як правило, у формі його структурної схеми, формалізованим описом критеріїв керування і обмежень. Розробка попередніх математичних моделей елементів об'єкту завершується складанням відповідності математичних описів. Залежно від прийнятих припущень описи подаються у формі детермінованих або стохастичних співвідношень. Окремі постійні коефіцієнти цих співвідношень визначаються або аналітичними, або експерементально-статистичними методами. Формулювання постановок функціональних завдань системи полягає у визначенні змісту функцій, які повинна реалізувати система, і в уточненні вимог до їх виконання. Одночасно попередньо оцінюється можливість реалізації цих функцій за допомогою сучасних методів автоматизації. Такі дослідження проводяться в лабораторних умовах або аналітичним шляхом, або методами експериментального моделювання.

На етапі «Обліковое проектування» розроблюються основні матеріали, які підтверджують доцільність і можливість створення ІС. На даному етапі проводиться таке проектування системи, яке включає в себе: попередню розробку функціональної і алгоритмічних структур системи; попередній синтез основних алгоритмів контролю і керування; попередній вибір технічних засобів системи і його обгрунтування; попереднє визначення завдань з модернізації технологічного обладнання; попередній вибір загального програмного забезпечення; попередній вибір загального або придбання алгоритмічних і програмних модулів; пакетів і бібліотек програмних модулів; попередній порівняльний аналіз системи і її відомих аналогів.

При виборі методу синтезу алгоритмів керування зазвичай потрібно встановити розрахунковий інтервал керування і якщо необхідно, виконати декомпозицію задач керування.

Важливу групу робіт на етапі «Облікового проектування» складають попередні розрахунки надійності і метрологічних показників найбільш відповідальних функцій створюваної системи, оцінки необхідної обчислювальної потужності; визначення раціонального завантаження і попередній вибір системи та його обгрунтування. На цьому етапі рекомендується проводити експериментальну перевірку алгоритмів керування на діючих установках за допомогою макетів функціональних блоків і вузлів створюваної системи. Етап може завершуватися коригуванням ТЕО, пов'язаного з уточненням переліку і характеристик реалізованих системою функцій. Порядок і методика проведення розрахунків техніко-економічної ефективності встановлюється діючими нормативними матеріалами. При проведенні розрахунків використовуються вихідні дані, представлені і узгоджені з замовником.

Основна мета робіт етапу «Розробка ТЗ» полягає в складанні на базі результатів передпроектних робіт ТЗ на створення системи. Склад, зміст і порядок оформлення ТЗ регламентуються стандартами, відповідно до яких ТЗ є обов'язковим вихідним документом для проектування системи, на відповідність яким здійснюється її перевірка при передачі в промислову експлуатацію і при здачі приймальній комісії.

ТЗ включає в себе наступні розділи: вступна частина; характеристики ІС; мета і призначення; техніко-економічні показники; вимоги до системи; вимоги до замовника з підготовки об'єкту керування; перелік робіт зі створення системи; порядок впровадження системи в промислову експлуатацію. У вступному повинно міститися: повне найменування системи; підстава для створення системи; терміни початку і закінчення робіт зі створення системи; найменування організації-розробника системи і організацій-співвиконавців - учасників створення системи; повне найменування організації-розробника системи; відомості про джерела і порядок фінансування.

У розділі «Характеристики ОК» повинні міститися такі відомості: необхідна інформація по технологічному обладнанні; дані про біінформаційному і технологічному процесах, регламенті та режимах роботи об'єкту; список використаних енергоресурсів і їх характеристики; характеристики вхідної та вихідної інформації; відомості про умови експлуатації, в тім рахунку про мікроклімат, характеристиках приміщень і особливості ІС і навколишнього середовища (пожежонебезпека, вибухонебезпечність і т.п.).

У розділі «Призначення» повинні бути: мета і критерії керування; перелік функцій системи, які забезпечують досягнення сформульованих цілей; місце системи в загальній системі керування промисловим підприємством або системи надзвичайно ієрархії; планований обсяг і етапи розвитку системи.

У розділі «Техніко-економічні показники системи» містяться очікувані ТЕП із зазначенням максимально допустимої суми одноразових витрат на створення системи, річного економічного ефекту і джерел його виникнення, коефіцієнту економічної ефективності, витрат і т.п.

Розділ «Вимоги» повинен містити такі підрозділи: вимоги до системи в цілому; вимоги до якості виконання окремих функцій; вимоги до складових частин системи.

В підрозділі «Вимоги до системи в цілому» наводяться показники надійності системи і її функцій, вказівки про спосіб обміну інформацією з суміжними системами, вказівки про необхідність уніфікації проектних і технічних рішень, ергономічні вимоги до системи за способами і формою представлення інформації оперативному персоналу в частині реалізації компонування ІС, зручності обслуговування, комфортність диспетчерських і операторських

пунктів, естетичності рішень і форм, вимог до збереження інформації при аваріях в системі енергопостачання.

Підрозділ «Вимоги до якості виконання окремих функцій» повинен містити: по кожній інформаційній функції - періодичність і форми подання інформації в кожному місці контролю, характер використання інформації; по кожній керуючої функції - режими керування і вимоги до якості керування; характеристики необхідної точності і швидкодії виконання кожної функції і вимоги одночасності виконання групи функцій.

В підрозділі «Вимоги до складових частин системи» містяться вимоги до ТО, ПО, ІО з викладом додаткових вимог, а також попередні вимоги до чисельності та кваліфікації оперативного та ремонтного персоналу і режиму його роботи.

В розділі «Вимоги» наводяться всі додаткові вимоги до системи. Розділ «Вимоги до замовника з підготовки об'єкту» включає попередні переліки основних робіт з підготовки об'єкту до впровадження системи і введення її в промислову експлуатацію; заходів, які забезпечують підготовку оперативного персоналу, організацію обслуговування системи, ефективну роботу при її створенні; вимог до технологічного устаткування, пов'язаних зі створенням системи (в тім рахунку по становленню вимірювальної і регулюючої апаратури), виконання яких забезпечує замовник.

У розділі «Склад і зміст робіт» міститься план-графік робіт, в якому вказуються стадії, етапи, зміст робіт, організації-виконавці, терміни виконання етапів і робіт, а також чим закінчуються етапи і роботи, а також порядок подання звітних матеріалів.

Розділ «Порядок введення в промислову експлуатацію» повинен містити відомості про види випробувань системи, а також, при необхідності, додаткові вимоги до порядку проведення приймально-здавальних робіт. Розробка, погодження та затвердження ТЗ на створення ІС реалізується в такий спосіб: ТЗ на створення системи розроблює організація-розробник за участю замовника; ТЗ узгоджується в установленому порядку з замовником і виконавцем; в разі потреби ТЗ узгоджується з усіма регламентуючими організаціями; ТЗ оформлюється відповідно до нормативних документів; підписи осіб, які візують ТЗ перед його узгодженням і затвердженням, і осіб, які беруть участь в його розробці, наводяться на останньому аркуші.

4 . Технології створення ІС

4.1 Технології створення ІС

В цій лекції коротко розглянемо **канонічне проектування ІС** і три сучасні процеси розробки **– уніфікований, процес Rational** (Rational Unified Process

RUP), **екстремальне програмування** (Extreme Programming, XP), **Scrum**-технологію. Усі ці три технології розробки є прикладами ітеративних процесів, але побудовані вони на основі різних припущень про природу розробки ПЗ.

RUP є прикладом так званого «важкого» процесу, детально описаного і припускаючого підтримку власне розробки початкового коду ПЗ великою кількістю допоміжних дій.

Екстремальне програмування, навпаки, представляє так звані «живі» (agile) методи розробки, які роблять упор на використанні хороших розробників, а не на добре відлагоджені процеси розробки. Живі методи уникають фіксації чітких схем дій, щоб забезпечити велику гнучкість в кожному конкретному проекті.

Scrum-технологія (метод Scrum). Достоїнствами XP є велика гнучкість і простота розуміння, проте в повному об'ємі XP не була використана навіть її авторами. Крім того, відомі і успішно впроваджуються окремі практики XP, як, наприклад, парне програмування, колективне володіння кодом, і рефакторинг коду. Тому ми розглянемо поліпшену технологію – Scrum-технологію.

Організація канонічного проектування ІС орієнтована на використання головним чином каскадної моделі життєвого циклу ІС. Стадії і етапи роботи описані в стандарті ГОСТ 34.601.

Ці питання ми докладно і неодноразово розглґнули в попередніх лекціях, але там вони розглядалися як загальнотехнічні питання, а тут розглянемо з програмістським нахилом.

Стадії і етапи створення ІС, виконувані організаціями-учасниками, прописуються в договорах і *технічних завданнях* на виконання робіт:

Стадія 1. Формування вимог до ІС.

Стадія 2. Розробка концепції ІС: вивчення об'єкту автоматизації, розробка варіантів концепції ІС, які задовольняють вимогам користувачів, оформлення звіту і затвердження концепції.

Стадія 3. Технічне завдання: Розробка і затвердження *технічного завдання* на створення ІС.

Стадія 4. Ескізний проект: розробка попередніх проектних рішень по системі і її частинам, розробка ескізної документації на ІС.

Стадія 5. Технічний проект: розробка проектних рішень по системі і її частинам, розробка документації на ІС і її частини.

Стадія 7. Введення в дію: комплектація ІС виробами (програмними і технічними засобами), що поставляються, проведення попередніх випробувань, проведення дослідної експлуатації, проведення приймальних випробувань.

Стадія 8. Супровід ІС: виконання робіт відповідно до гарантійних зобов'язань, післягарантійне обслуговування.

Обстеження об'єкту і обгрунтування створення ІС

Обстеження – це вивчення і діагностичний аналіз організаційної структури підприємства, його діяльності і існуючої системи обробки інформації. Матеріали, отримані в результаті обстеження, використовуються для:

– обгрунтування розробки і поетапного впровадження системи;

– складання *технічного завдання* на розробку системи;

– розробки технічного і робочого проектів системи.

На етапі *обстеження* виділяються дві складові: визначення стратегії впровадження ІС і детальний аналіз діяльності організації.

Основне завдання першого етапу *обстеження* – оцінка реального об'єму проекту, його цілей і завдань на основі виявлених функцій і інформаційних елементів об'єкту, який автоматизується. Ці завдання можуть бути реалізовані або замовником ІС самостійно, або із залученням консалтингових організацій. Етап припускає тісну взаємодію з основними потенційними користувачами системи і бізнес-експертами. Результатом етапу визначення стратегії є документ (*техніко-економічне обгрунтування проекту*), де чітко сформульовано, що отримає замовник, якщо погодиться фінансувати проект, коли він отримає готовий продукт (графік виконання робіт) і скільки це коштуватиме (для великих проектів має бути складений графік фінансування на різних етапах робіт).

На етапі детального аналізу діяльності організації вивчаються завдання, які забезпечують реалізацію функцій керування, організаційна структура, штати і зміст робіт по керуванню підприємством, а також характер підлеглості вищестоящим органам керування. Одній з найбільш трудомістких завдань цього етапу є опис документообігу організації.

При *обстеженні* документообігу складається схема маршруту руху документів, яка повинна відбити:

– кількість документів;

– місце формування показників документу;

– взаємозв'язок документів при їх формуванні;

– маршрут і тривалість руху документу;

– місце використання і зберігання цього документу;

– внутрішні і зовнішні інформаційні зв'язки.

За результатами *обстеження* встановлюється перелік завдань управління, рішення яких доцільно автоматизувати, і черговість їх розробки. Результати *обстеження представляють об'єктивну основу для формування технічного завдання* ІС.

Технічне завдання

Технічне завдання – це документ, який визначає цілі, вимоги і основні початкові дані, необхідні для розробки автоматизованої системи керувння.

При розробці технічного завдання необхідно вирішити наступні завдання:

– встановити мету створення ІС, визначити склад підсистем і функціональних завдань;
– розробити і обгрунтувати вимоги, які пред'являються до підсистем;
– розробити і обгрунтувати вимоги, які пред'являються до інформаційної бази, математичного і програмного забезпечення, комплексу технічних засобів (включаючи засоби зв'язку і передачі даних);
– встановити загальні вимоги до проектованої системи;
– визначити перелік завдань створення системи і виконавців;
– визначити етапи створення системи і терміни їх виконання;
– провести попередній розрахунок витрат на створення системи і визначити рівень економічної ефективності її впровадження.

Приведемо типові вимоги до складу (розділи) і змісту технічного завдання згідно нормативної бази.

1. Загальні відомості:
– повне найменування системи і її умовне позначення;
– шифр теми або шифр (номер) договору;
– найменування підприємств розробника і замовника;
– перелік документів, на підставі яких створюється ІС;
– планові терміни початку і закінчення робіт;
– відомості про джерела і порядок фінансування робіт;
– порядок оформлення і пред'явлення замовникові результатів робіт із створення системи, її частин і окремих засобів.

2. Призначення і цілі створення (розвитку) системи:
– вид діяльності, який автоматизується, перелік об'єктів, на яких передбачається використання системи;
– найменування і необхідні значення технічних, виробничо-економічних та ін. показників об'єкту, які мають бути досягнуті при впровадженні ІС.

3. Характеристика об'єктів автоматизації:
– короткі відомості про об'єкт автоматизації;
– відомості про умови експлуатації і характеристики довкілля.

4. Вимоги до системи.

Вимоги до системи в цілому:
– вимоги до структури і функціонування системи (перелік підсистем, рівні ієрархії, способи інформаційного обміну, взаємодія з суміжними системами, перспективи розвитку системи);
– вимоги до персоналу (чисельність користувачів, кваліфікація, режим роботи, порядок підготовки);
– показники призначення (міра пристосовності системи до змін процесів керування і значень параметрів);
– вимоги до надійності, безпеки, ергономіки, транспортабельності, експлуатації, захисту і збереження інформації, захисту від зовнішніх дій.

Вимоги до функцій (по підсистемах):
– перелік завдань, які підлягають автоматизації;
– часовий регламент реалізації кожної функції;
– вимоги до якості реалізації кожної функції.
Вимоги до видів забезпечення:
– математичного (склад і сфера застосування мат. моделей і методів, типових алгоритмів, які розроблюються);
– інформаційного (склад, структура і організація даних, обмін даними між компонентами системи, сумісність з суміжними системами, СУБД);
– лінгвістичного (мови програмування, мови взаємодії користувачів з системою, системи кодування, мови введення-виводу);
– програмного (незалежність програмних засобів від платформи, якість програмних засобів);
– організаційного (структура і функції експлуатуючих підрозділів, захист від помилкових дій персоналу).
5. Склад і зміст робіт зі створення системи:
– перелік стадій і етапів робіт;
– терміни виконання;
– склад організацій – виконавців робіт.
6. Порядок контролю і приймання системи:
– види, склад, об'єм і методи випробувань системи;
– загальні вимоги до приймання робіт по стадіях.
7. Вимоги до складу і змісту робіт з підготовки об'єкту до введення системи в дію:
– перетворення вхідної інформації до машиночитаємого виду;
– зміни в об'єкті автоматизації;
– терміни і порядок комплектування і навчання персоналу.
8. Вимоги до документування:
– перелік документів, які підлягають розробці;
– перелік документів на машинних носіях.

9. Джерела розробки:
– документи і інформаційні матеріали, на підставі яких розробляється ТЗ і система.

Ескізний проект

Ескізний проект передбачає розробку попередніх проектних рішень по системі і її частинах.

Виконання стадії ескізного проектування не є строго обов'язковим. Якщо основні проектні рішення визначені раніше, або досить очевидні для конкретної ІС і об'єкту автоматизації, то ця стадія може бути виключена із

загальної послідовності робіт.

На етапі ескізного проектування визначаються:

– функції ІС і її підсистем, їх цілі і очікуваний ефект від впровадження;

– склад комплексів завдань і окремих завдань;

– концепція інформаційної бази і її укрупнена структура;

– функції системи керування базою даних;

– склад обчислювальної системи і інших технічних засобів;

– функції і параметри основних програмних засобів.

За результатами виконаної роботи оформляється, узгоджується і затверджується документація в об'ємі, необхідному для опису повної сукупності прийнятих проектних рішень.

На основі *технічного завдання (і ескізного проекту) розробляється технічний проект* ІС.

Технічний проект

Технічний проект системи – це технічна документація, яка містить загальносистемні проектні рішення, алгоритми рішення завдань, а також оцінку економічної ефективності автоматизованої системи керування і перелік заходів по підготовці об'єкту до впровадження. На цьому етапі здійснюється комплекс науково-дослідних і експериментальних робіт для вибору основних проектних рішень і розрахунок економічної ефективності системи.

Робоча документація і випробування ІС

На стадії «*робоча документація*» здійснюється створення програмного продукту і розробка усієї супроводжуючої документації. Документація повинна містити усі необхідні і достатні відомості для забезпечення виконання робіт по введенню ІС в дію і її експлуатації, а також для підтримки рівня експлуатаційних характеристик (якості) системи. Розроблена документація має бути відповідним чином оформлена, погоджена і затверджена.

Для ІС, які є різновидом автоматизованих систем, встановлюють наступні основні види випробувань: *попередні випробування, дослідна експлуатація і приймальні випробування*.

Попередні випробування проводять для визначення працездатності системи і вирішення питання про можливість її приймання в дослідну експлуатацію. Попередні випробування слід виконувати після проведення розробником відладки і тестування програмних і технічних засобів системи і представлення ним відповідних документів про їх готовність до приймальних випробувань.

Дослідну експлуатацію системи проводять з метою визначення фактичних значень кількісних і якісних характеристик системи і готовності персоналу до роботи в умовах її функціонування, а також визначення фактичної ефективності і коригування, при необхідності, документації.

Приймальні випробування проводять для визначення відповідності системи *технічному завданню, оцінки якості дослідної експлуатації* і вирішення

питання про можливість приймання системи в постійну експлуатацію. Далі розглянемо вказані три сучасні процеси розробки

Уніфікований процес Rational (RUP)

Уніфікований процес Rational (Rational Unified Process, RUP) [1] є досить складною, детально пропраццьованною ітеративною моделлю життєвого циклу ПЗ. RUP є програмним продуктом в якості доповнення до мови моделювання UML, розроблений компанією Rational Software, яка нині входить до складу IBM.

Основні принципи RUP

Історично *RUP* є розвитком моделі процесу розробки, прийнятої в компанії Ericsson в 70-х-80-х роках XX століття. Ця модель була створена Джекобсоном (Ivar Jacobson), який у 1987 заснував власну компанію Objectory AB саме для розвитку технологічного процесу розробки ПЗ як окремого продукту, який можна було б переносити в інші організації. Після вливання Objectory в Rational в 1995 розробки Джекобсона були інтегровані з роботами Ройса , Крухтена і Буча, а також з розвивающоюся паралельно універсальною мовою моделювання (Unified Modeling Language, UML) [2].

Основними принципами RUP є:

1. Ітераційний і інкрементний (нарощуваний) підхід до створення ПЗ.
2. Планування і керування проектом на основі функціональних вимог до системи – варіантів використання.
3. Побудова системи на базі архітектури ПЗ.

Перший принцип є визначальний. Відповідно до нього розробка системи виконується у вигляді декількох короткострокових міні-проектів фіксованої тривалості (від 2 до 6 тижнів), які називаються ітераціями. Кожна ітерація включає свої власні етапи аналізу вимог, проектування, реалізації, тестування, інтеграції і завершується створенням працюючої системи.

Фази проекту

З точки зору етапів (фаз) проектування, то вони, загалом, аналогічні «водоспаду». Але RUP виділяє в життєвому циклі 4 основних фази, в рамках кожної з яких можливе проведення декількох ітерацій.

Фаза початку проекту (Inception). Основна мета цієї фази – досягти компромісу між усіма зацікавленими особами відносно завдань проекту і ресурсів, які виділяються на нього. На цій фазі визначаються основні цілі проекту, керівник проекту і бюджет проекту, основні засоби його виконання – технології, інструменти, ключові виконавці, а також відбувається апробація вибраних технологій з метою підтвердження можливості досягти цілей з їх допомогою. На цю фазу може йти близько 10% часу і 5% трудомісткості одного циклу.

Результатами початкової стадії є:

– загальний опис системи: основні вимоги до проекту, його характеристики і обмеження;

– початковий бізнес-план;
– план проекту, який відбиває стадії і ітерації;
– один або декілька прототипів.

Фаза проектування (Elaboration). Основна мета цієї фази – на базі основних, найбільш суттєвих вимог розробити стабільну базову архітектуру продукту, яка дозволяє вирішувати поставлені перед системою завдання і надалі використовується як основа розробки системи. На цю фазу може йти близько 30% часу і 20% трудомісткості одного циклу.

Фаза побудови (Construction). Основна мета цієї фази – детальне прояснення вимог і розробка системи на основі спроектованої раніше архітектури. В результаті повинна вийти система, яка реалізовує усі виділені варіанти використання. На цю фазу йде близько 50% часу і 65% трудомісткості одного циклу.

Результатами стадії розробки є:
– модель варіантів використання (завершена принаймні на 80%), яка визначає функціональні вимоги до системи;
– перелік додаткових вимог, включаючи вимоги нефункціонального характеру;
– опис базової архітектури майбутньої системи, працюючий прототип;
– план розробки усього проекту, що відбиває ітерації і критерії оцінки для кожної ітерації.

Фаза впровадження (Transition). Мета цієї фази – зробити систему повністю доступною кінцевим користувачам. На цій стадії відбувається розгортання системи в її робочому середовищі, бета-тестування, підгонка дрібних деталей під потреби користувачів. На цю фазу може йти близько 10% часу і 10% трудомісткості одного циклу.

Ключові ідеї RUP

Робочі продукти (артефакти), які виробляються в ході проекту, можуть бути представлені в вигляді баз даних і таблиць з інформацією різного типу, різних видів документів, початкового коду і об’єктних модулів, а також моделей, які складаються з окремих елементів.

Найбільш важливі з точки зору RUP артефакти проекту – це моделі, які описують різні аспекти майбутньої системи. Більшість моделей є наборами діаграм UML.

Основна техніка, використовувана в RUP:
1. Вироблення концепції проекту на його початку для чіткої постановки завдань.
2. Керування за планом.
3. Зниження ризиків і відстежування їх наслідків, як можна більше ранній початок робіт по подоланню ризиків.
4. Ретельне економічне обгрунтування усіх дій – робиться тільки те, що треба замовникові.

5. Як можна більше раннє формування базової архітектури.
6. Використання компонентної архітектури.
7. Прототипування, інкрементна розробка і тестування.
8. Регулярні оцінки поточного стану.
9. Керування змінами, постійний відробіток змін ззовні проекту.
10. Націленість на створення продукту, працездатного в реальному оточенні.
11. Націленість на якість.
12. Адаптація процесу під потреби проекту.

Отже, фази в цілому відповідають моделі водоспаду, з тим виключенням, що фаза супроводу не розглядається як окрема. В цілому, при розробці проекту RUP базується на чотирьох ключових ідеях:

1. Ключовою ідеєю процесу є його ітеративність – кожна фаза, починаючи з розвитку, включає декілька ітерацій, на кожній з яких виконується свій шматок аналізу, проектування, реалізації і тестування готового продукту.
2. Увесь хід робіт спрямовується підсумковими цілями проекту (ітераціями), вираженими у вигляді ***прецедентів використання*** або ***варіантів використання (use cases)*** – сценаріїв взаємодії результуючої програмної системи з користувачами або іншими системами. Вимоги аналізуються також як і проект на кожній ітерації і в цілому дуже ретельно, але не до кінця – діє правило 70-80% – розробник повинен уявляти собі вимоги саме настільки, перш ніж починати кодування.
3. На етапі розвитку приймаються ключові рішення, які стосуються ***архітектури*** системи в цілому, її властивостей, функцій, використовуваних технологій і так далі. У кінці цієї фази реалізується 10-30% системи, але усі основні рішення вже прийняті (до 70-80%) і на етапі конструювання в рамках цих рішень система доводиться до кінця. Архітектура встановлює набір компонентів, з яких буде побудовано ПЗ, відповідальність кожного з компонентів (тобто вирішувані ним підзадачі в рамках загальних завдань системи), чітко визначає інтерфейси, через які вони можуть взаємодіяти, а також способи взаємодії компонентів один з одним.

4. Основою процесу розробки є ***плановані і керовані ітерації,*** об’єм яких визначається на основі архітектури (функціональність, яка реалізовується в рамках ітерації, і набір компонентів).

RUP як продукт входить до складу інтегрованого комплексу інструментальних засобів **Rational Suite**, який існує в наступних варіантах:

– Rational Suite AnalystStudio – призначений для визначення і керування повним набором вимог до розроблюваної системи;
– Rational Suite DevelopmentStudio – призначений для проектування і реалізації ПЗ;
– Rational Suite TestStudio – є набір продуктів, призначених для автоматичного тестування додатків;
– Rational Suite Enterprise – забезпечує підтримку повного життєвого циклу ПЗ і

призначений як для менеджерів проекту, так і окремих розробників, які виконують декілька функціональних ролей в команді розробників.

До складу Rational Suite, окрім самої технології RUP як продукту, входять наступні компоненти:

– Rational Rose – засіб візуального моделювання (аналізу і проектування), який використовує мову UML;

– Rational XDE – засіб аналізу і проектування, інтегрований з платформами MS Visual Studio .NET і IBM WebSphere Studio Application Developer;

– Rational Requisite Pro – засіб керування вимогами, призначений для організації спільної роботи групи розробників;

– Rational Rapid Developer – засіб швидкої розробки додатків на платформі Java 2 Enterprise Edition;

– Rational SoDA – засіб автоматичної генерації проектної документації;

– Rational Quantify – засіб кількісного визначення вузьких місць, які впливають на загальну ефективність роботи програми;

– Rational TestManager – засіб планування функціонального навантаження тестування;

– Rational TestFactory – засіб тестування надійності;

– Rational Quality Architect – засіб генерації коду для тестування.

Засоби автоматичної генерації коду, використовуючи інформацію, яка міститься в діаграмах класів і компонентів, формують файли описів класів.

Створюваний таким чином скелет програми може бути уточнений шляхом прямого програмування на відповідній мові (основні мови, підтримувані Rational Rose, – C++ і Java).

Уніфікований процес RUP хоча і є досить складною ітеративною моделлю розробки життєвого циклу ПЗ, все ж може з успіхом застосуються, якщо на деяких етапах (фазах) допускати деякі «вільності».

Серед проблем власне кодування, слід виділити небажання деяких розробників (а також деяких замовників) використати компоненти сторонніх виробників. Щоб не винаходити велосипед, при написанні коду можуть використовуватися готові програмні конструкції (готові рішення), типові рішення, шаблони, так звані «**патерни» (**design patterns – зразки проектування або просто patterns).

Проте які б поліпшення ми не вносили, техніці RUP властиві серйозні обмеження і недоліки:

1. Він уніфікований, тобто підходить для різнорідних процесів, проектів, розробок, що робить його трохи заплутаним і неконкретним (мало конкретних чітких рекомендацій).

2. Він важкуватий для невеликих проектів і колективів розробників, особливо коли бюджет і терміни проектів невеликі.

3. Він вимагає глибокого осмислення вимог, як і традиційний водоспад (хоча і залишає на потім 30%). У реальних ситуаціях і 70% відразу отримати важко.

Екстремальне програмування (XP-процес)

Останнім часом все більшу популярність стали набирати так звані «**гнучкі» (*«живі»)*** методи розробки ПЗ (Agile Software Development). Серед них поширеним являться ***екстремальне програмування (eXtreme Programming, XP)*** – полегшений (рухливий) процес (чи методологія), головний автор якого – Кент Бек (1999) [3].

XP-процес орієнтований на групи малого і середнього розміру, які будують програмне забезпечення в умовах невизначених або швидкозмінюпаних вимог.

Основні принципи «живої» розробки ПЗ зафіксовані в маніфесті «живої» розробки, який з'явився в 2000 році:

1. Люди, які беруть участь в проекті, і їх спілкування важливіші, ніж процеси і інструменти.
2. Працююча програма важливіша, ніж вичерпна документація.
3. Співпраця із замовником важливіша, ніж обговорення деталей контракту.
4. Відробіток змін важливіший, ніж наслідування планів.

«Живі» методи з'явилися як протест проти надмірної бюрократизації розробки ПЗ, великої кількості побічних необхідних для отримання кінцевого результату документів, які доводиться оформляти при проведенні проекту відповідно до більшості «важких» процесів. Велика частина таких робіт і документів не має прямого відношення до розробки ПЗ і забезпечення її якості, а призначена для дотримання формальних пунктів контрактів на розробку, отримання і підтвердження сертифікатів на відповідність різним стандартам.

Основна ідея XP-процесу – усунути високу вартість, характерну для додатків з використанням об'єктів, **патернів** (рішення типових проблем в певному контексті або готові програмні конструкції) і реляційних баз даних. XP-процес має бути високо динамічним процесом. XP-група має справу зі змінами вимог на всьому протязі ітераційного циклу розробки, причому цикл складається з дуже коротких ітерацій.

Ця методика є не стільки наслідуванням якихось загальних схем дій, скільки застосування комбінації відповідної техніки (практик). Кожна техніка важлива, і без її використання розробка вважається такою, що йде не по XP.

1. Гра в планування (Planning game) – швидке визначення зони дії наступної реалізації шляхом об'єднання ділових пріоритетів і технічних оцінок. Замовник формує зону дії, пріоритетність і терміни з точки зору бізнесу, а розробники оцінюють і простежують просування (прогрес).

2. Часта зміна версій (Small releases) – швидкий запуск у виробництво простої системи, тобто найперша працююча версія повинна з'явитися якнайшвидше, і тут же повинна почати використовуватися. Для реалізації нових версій вводяться ще жорсткіші обмеження на тривалість однієї ітерації.

3. Метафора (Metaphor) – уся розробка проводиться на основі простої, загальнодоступної історії про те, як працює вся система. Метафора в досить простому і зрозумілому команді виді повинна описувати основний механізм

роботи системи. Це поняття нагадує архітектуру, але повинне бути набагато простіше, усього у вигляді однієї-двох фраз описувати основну суть прийнятих технічних рішень.

4. Просте проектування (Simple design) – проектування виконується настільки просто, наскільки це можливо в даний момент. Не потрібно додавати функції заздалегідь – тільки після явного прохання про це. Уся зайва складність віддаляється, як тільки виявляється.

5. Тестування (Testing) – безперервне написання тестів для модулів, які повинні виконуватися бездоганно. Розробники спочатку пишуть тести, потім намагаються реалізувати свої модулі так, щоб тести спрацьовували.

6. Реорганізація (рефакторинг, Refactoring) – наведення ладу в коді, переробка окремих файлів, видалення максимальної кількості незрозумілих фрагментів коду, об'єднання класів на основі їх схожої функціональності, корекція коментарів, осмислене перейменування об'єктів. Система реструктурується у зв'язку з додаванням нової функціональності, але її поведінка не змінюється; мета – усунути дублювання, спростити систему.

7. Парне програмування (Pair programming) – увесь код пишеться двома програмістами, працюючими на одному комп'ютері. Об'єднання в пари довільно і міняється від завдання до завдання. Той, в чиїх руках клавіатура, намагається найкращим чином вирішити поточне завдання. Другий програміст аналізує роботу першого і дає поради, обмірковує наслідки тих або інших рішень, нові тести, менш прямі, але гнучкіші рішення.

8. Колективне володіння кодом (Collective ownership) – будь-який розробник може покращувати будь-який код системи у будь-який час. Ніхто не повинен виділяти свою власну область відповідальності, уся команда в цілому відповідає за увесь код.

9. Безперервна інтеграція (Continuous integration) – система інтегрується і будується багато разів в день, у міру завершення кожного завдання. Безперервне регресійне тестування, тобто повторення попередніх тестів, гарантує, що зміни вимог не приведуть до регресу функціональності.

10. 40-годинний тиждень (40 – hour week) – як правило, працюють не більше 40 годин в тиждень. Не можна подвоювати робочий тиждень за рахунок наднормових робіт.

11. Локальний замовник (On – site customer) – в групі увесь час повинен знаходитися представник замовника, дійсно готовий відповідати на питання розробників. Його обов'язком є досить оперативні відповіді на питання будь-якого типу, які стосуються функцій системи, її інтерфейсу, необхідної продуктивності, правильної роботи системи в складних ситуаціях і ін.

12. Стандарти (стилі) кодування (Coding standards) – повинні витримуватися правила, що забезпечують однакове представлення програмного коду в усіх частинах програмної системи. Код розглядається як найважливіший засіб спілкування усередині команди. Ясність коду – один з основних пріоритетів.

13. Відкритий робочий простір (Open workspace). Команда розміщується в одному, досить просторому приміщенні, для спрощення комунікації і можливості проведення колективних обговорень при плануванні і ухваленні важливих технічних рішень.
14. Зміна правил з потреби (just rules). Кожен член команди повинен прийняти перераховані правила, але при виникненні необхідності команда може поміняти їх, якщо усі її члени прийшли до згоди з приводу цієї зміни.

Недоліками цього підходу деякі фахівці вважають нездійсненність в такому стилі досить великих і складних проектів, неможливість планувати терміни і трудомісткість проекту на досить довгу перспективу і чітко передбачити результати тривалого проекту в термінах співвідношення якості результату і витрат часу і ресурсів.

Scrum методологія

Достоїнствами XP є велика гнучкість і простота розуміння, проте в повному об'ємі XP не була використана навіть її авторами. Крім того, відомі і успішно впроваджуються окремі практики XP, як, наприклад, парне програмування, колективне володіння кодом, і рефакторинг коду. Ідея простого, ненадмірного дизайну проекту також зробила значний вплив насвіт розробників ПЗ. Більше практичним «гнучким» методом розробки є Scrum[4].

Історія

В 1986 японські фахівці Hirotaka Takeuchi і Ikujiro Nonaka опублікували повідомлення про новий підхід до розробки нових сервісів і продуктів (не обов'язково програмних). Основу підходу складала згуртована робота невеликої універсальної команди, яка розробляє проект на усіх фазах. Наводилася аналогія з регбі, де уся команда рухається до воріт супротивника як єдине ціле, передаючи (пасуючи) м'яч своїм гравцям як вперед, так і назад. На початку 90-х років цей підхід став застосовуватися в програмній індустрії і набув назви **Scum** (термін з регбі, що означає, – сутичка), в 1995 році Jeff Sutherland і Ken Schwaber представили опис цього підходу на OOPSLA '95 – одній з найавторитетніших конференцій в області програмування. Відтоді метод активно використовується в індустрії і багаторазово описаний в літературі.

Загальний опис

Метод Scrum дозволяє гнучко розробляти проекти невеликими командами (7 чоловік плюс-мінус 2) в ситуації змінюючихся вимог . При цьому процес розробки ітеративен і надає велику свободу команді.

Спочатку створюються вимоги до усього продукту. Потім з них вибираються найактуальніші і створюється план на наступну ітерацію. Впродовж ітерації плани не міняються (цим досягається відносна стабільність розробки), а сама ітерація триває 2-4 тижні. Вона закінчується створенням працездатної версії продукту (робочий продукт), яку можна пред'явити замовникові, запустити і

продемонструвати, хай і з мінімальними функціональними можливостями. Після цього результати обговорюються і вимоги до продукту коригуються.

Ролі

У Scrum є всього три види ролей.

Власник продукту (Product Owner) – це менеджер проекту, який представляє в проекті інтереси замовника. У його обов'язки входить розробка початкових вимог до продукту (Product Backlog), своєчасна їх зміна вимог, призначення пріоритетів, дат постачання і ін. Важливо, що він абсолютно не бере участь у виконанні самої ітерації.

Scrum-мастера (Scrum Master) забезпечує максимальну працездатність і продуктивну роботу команди – як виконання Scrum-процесса, так і рішення господарських і адміністративних завдань. Зокрема, його завданням є обгороджування команди від усіх дій ззовні під час ітерації.

Scrum-команда (Scrum Team) – група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Першим завданням команди є постановка для ітерації реально досяжних і пріоритетних для проекту в цілому завдань (на основі Project Backlog і при активній участі власника продукту і Scum-мастера). Другим завданням є виконання цього завдання щоб то не було, у відведені терміни і із заявленою якістю.

Практики

У Scrum визначені наступні практики.

Sprint Planning Meeting. Проводиться на початку кожного Sprint. Спочатку Product Owner, Scurm-мастер, команда, а також представники замовника і інші зацікавлені особи визначають, які вимоги з Project Backlog найбільш пріоритетні і їх слід реалізовувати у рамках цього Sprint. Формується Sprint Backlog. Далі Scurm-мастер і Scrum-команда визначають те, як саме буде досягнута певна вище мета з Sprint Backlog. Для кожного елементу Sprint Backlog визначається список завдань і оцінюється їх трудомісткість.

Daily Scrum Meeting – п'ятнадцятихвилинна щоденна нарада, метою якої є досягти розуміння того, що сталося з часу попередньої наради, скоректувати робочий план згідно реаліям сьогоднішнього дня і позначити шляхи рішення існуючих проблем. Кожен учасник Scrum-команди відповідає на три питання: що я зробив з часу попередньої зустрічі, мої проблеми, що я робитиму до наступної зустрічі?

У цій нараді (15-20 хвилин) може брати участь будь-яка зацікавлена особа, але тільки учасники Scrum-команди мають право приймати рішення. На них лежить відповідальність за їх власні слова, і, якщо хтось з боку втручається і приймає рішення за них, тим самим він знімає відповідальність за результат з учасників команди.

Sprint Review Meeting. Проводиться у кінці кожного Sprint. Спочатку Scrum-команда демонструє Product Owner зроблену впродовж Sprint роботу, а той у свою чергу веде цю частину мітингу і може запросити до участі усіх

зацікавлених представників замовника. Product Owner визначає, які вимоги з Sprint Backlog були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в Sprint Backlog для наступної ітерації. У другій частині мітингу робиться аналіз минулого спринту, який веде Scrum-мастер. Scrum-команда аналізує в останньому Sprint позитивні і негативні моменти спільної роботи, робить висновки і приймає важливі для подальшої роботи рішення. Scrum-команда також шукає шляхи для збільшення ефективності подальшої роботи. Потім цикл повторюється (рис. 1).

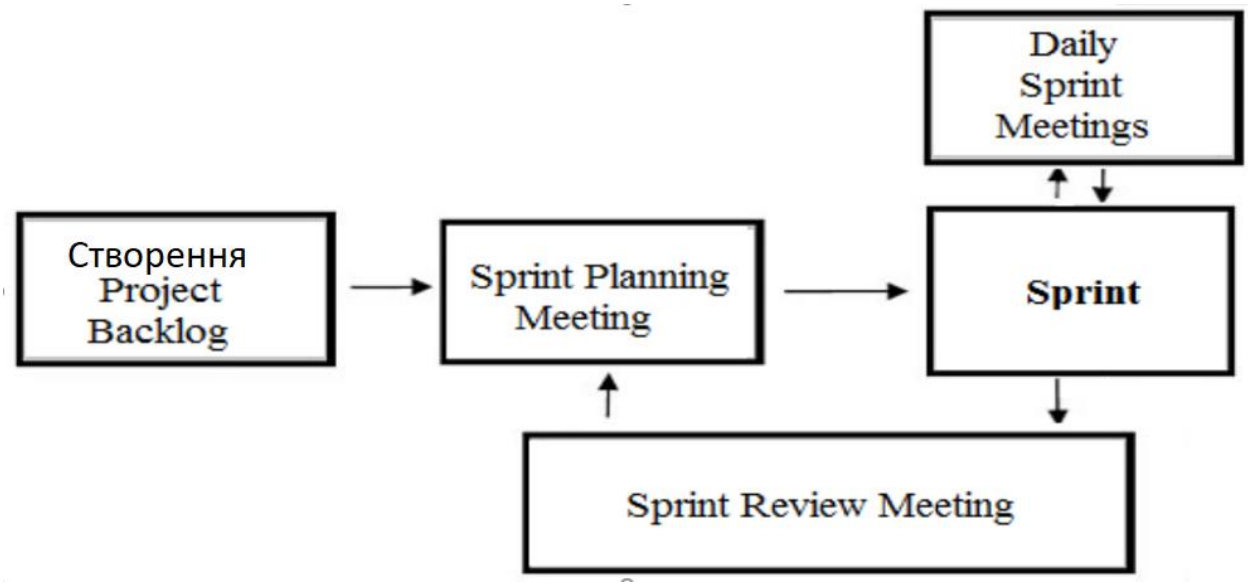


Рисунок 1 – Ітераційна схема створення ПЗ

Контрольні питання

1. Перерахуйте переваги і недоліки канонічної технології проектування ІС.
2. Перерахуйте переваги і недоліки RUP-технологии проектування ІС.
3. Перерахуйте переваги і недоліки XP-технологии.
4. Перерахуйте переваги і недоліки SCRUM-технологии.
5. Які ви знаєте стадії і етапи створення ІС із застосуванням канонічної технології?

Література

1. Якобсон А., Буч Г., Рамбо Дж.. Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с.

2. Stanislaw Wrycza, Bartosz Marcinkowski. Towards a Light Version of UML2.X: Appraisal and Model, 2007.

3. Интернет-ресурс http://scrum.org.ua/wp-content/uploads/2008/12/scrum_xp-from-thetrenches-rus-final.pdf .

4. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / Пер. с англ. – 3-е изд. – М.: ИД «Вильямс», 2013. – 736 с.

4.2 Керування проектом розробки ІС

Інформаційна система розроблюється як деякий проект.

Багато особливостей керування проектами і фази розробки проекту (фази життєвого циклу) є загальними, незалежними не лише від предметної області, але і від характеру проекту (неважливо, інженерний це чи економічний). Ваша магістерська дисертація – це теж проект.

Проект – це обмежена за часом цілеспрямована зміна окремої системи із спочатку чітко визначеним цілями, досягнення яких визначає завершення проекту, а також зі встановленими вимогами до термінів, результатів, ризику, рамок витрачання засобів і ресурсів.

Мета будь-якого програмного проекту полягає в створенні певного програмного продукту. Поняття «продукт» задає не лише текст на мові програмування і відкомпільований двійковий код, але і туди включаються документація, звіти по проміжних підсумках, результати перевірок, оцінки якості і т. д. Ці елементи зазвичай називають *артефактами*.

Розглядаючи планування проектів і керування ними, необхідно чітко усвідомлювати, що йдеться про керування деяким динамічним об’єктом. Тому система керування проектом має бути досить гнучкою, щоб допускати можливість модифікації без глобальних змін в робочій програмі.

Підбір команди і організаційні питання

При розробці великих проектів керівникам цих проектів доводилося шукати відповіді на безліч питань, які розпочинаються із слова «чому». Чому надвисокого IQ недостатньо для того, щоб ефективно керувати програмістами? Чому тільки менше 20% проектів розробки ПЗ завершуються в строк і укладаються до бюджету, а майже третина проектів анулюється до їх завершення?

Виявилось, що використання кращих мов і технологій програмування, найдосконаліших інструментів розробки і систем якості не гарантують успішність програмного проекту. «Саме людські якості забезпечують успіх тому або іншому проекту, саме вони є чинником першорядної ваги, грунтуючись на якому потрібно будувати прогнози про проект» [1].
Таким чином, головний елемент в розробці будь-якого проекту – це зовсім не новітні комп’ютери, або новітні технології, або новітня версія C++. *Головне – це люди, талановиті люди*. На них все тримається.Тобто, все вирішують

таланти. Проста мобілізація засобів і зусиль не може забезпечити прогрес. Як писав Ф. Брукс [2], «Якщо проект не
вкладається в терміни, то додавання робочої сили затримає його ще більше». Ідею багатства тепер зв'язують не з грошима, а з людьми, не з фінансовим капіталом, а з «людським».

Як відомо, суть усіх гнучких технологій: Scrum, xProgramming і інших, – полягає саме в тому, що вони декларують своєю вищою цінністю командну роботу, орієнтованість на людей і їх взаємодію, а не на процеси і засоби. Кожна розробка збирає навколо себе команду проекту. Ця команда проекту складається із зацікавлених в проекті осіб, які можуть грати в нім наступні ролі.

Кінцеві користувачі. Здійснюють введення в систему, яка розроблюється, забезпечують зворотний зв'язок в проекті інтерфейсу, проводять бета-тестування і допомагають керувати визначенням досягнення кінцевої мети. (Бета-тестування (англ. beta testing) або випробування — інтенсивне використання майже готової версії продукту (як правило, програмного або апаратного забезпечення) з метою виявлення максимальної кількості помилок в його роботі для їх подальшого усунення перед остаточним виходом (релізом) продукту на ринок, до масового споживача. На відміну від альфа-тестування, яке проводиться силами штатних розробників або тестувальників, бета-тестування передбачає залучення добровольців з числа звичайних майбутніх користувачів продукту, яким доступна версія продукту (так звана бета-версія).)

Розробники. Один з розробників є керівником команди проекту, її лідером, який регулює потік інформації між членами команди. Це найбільш авторитетна людина в команді проекту, до чиєї думки прислухаються, хто приймає більшість технічних рішень по ходу проекту. В команді може бути і менеджер проекту, який в класичному розумінні виконує тільки адміністративні обов'язки. Часто лідер і менеджер проекту – одна особа.

Керівник відділу. Ця особа, яка очолює відділ, в якому виконується розробка. На нього покладається відповідальність за достовірність даних, які видаються цим відділом інформаційній системі. Це людина, яка затверджує розроблювану документацію.

Відповідальний за гарантію якості. Ця роль в кожному проекті полягає в тому, щоб переконатися, що:

1. Проект досягне намічених цілей.
2. Проект відповідає опису системи.
3. План тестування відповідає вимогам.
4. План перетворення даних відповідає вимогам.

За якість проекту зазвичай відповідає одна людина, але стежать за якістю всі члени команди проекту.

Відповідальні за бета-тестування (дослідна експлуатація). Зазвичай в

групу команди бета-тестирования входять можливі кінцеві користувачі. Існує два типи тестування. Один використовує плани спеціального тестування, розроблені відповідальним за гарантію якості. Інший використовує тести, які розробили самі відповідальні за бета-тестування, застосовуючи критерії, відповідальні за гарантію якості.

Перш, ніж перейматися створенням основ технології, керівникові (менеджерові, начальникові) проекту необхідно по максимуму вирішити усі організаційні проблеми. А саме:

1. Постаратися, наскільки можливо, розділити територіально розробників ПЗ і інших фахівців. При сучасному підході до розробки ПЗ дуже високо цінується легкість комунікації. Поза сумнівом, прекрасно, коли розробник може, зробивши два кроки, уточнити в безпосереднього замовника спосіб вирішення проблеми, вхідні дані або що-небудь ще, але, посадивши менеджера по продажах і розробника за сусідні столи, щонайменше в 25% понизится продуктивність праці розробника. Основний принцип: всі розробники мають бути доступні один для одного, як організаційно, так і фізично.

Структура в групі – однорангова, тобто існує тільки професійний авторитет, але не адміністративне ділення.

2. Заборонити, кому б то не було, окрім менеджера проекту, безпосередньо давати вказівки розробникам. Проект з великою часткою вірогідності вийде з-під контролю, якщо розробники виконуватимуть такі, нехай навіть термінові, розпорядження.

3. Виділити або людину, або час на обговорення завдань. Практика показує, що завдання по супроводу-розробці ПЗ виникають постійно впродовж дня, обставини майже завжди вимагають негайного з'ясування. Якщо ви маєте двох розробників, нехай вони по черзі спілкуються з представниками замовника, якщо у вас є спеціально виділений для цього менеджер – нехай спілкується він.

4. Виділити для розробників окремий телефонний номер. Не залучати розробників до секретарської роботи.

5. Ніколи не залучайте розробників до некваліфікованої праці. Якщо ви, звичайно, не боїтеся втратити розробника. Пам'ятайте, розробник надзвичайно цінує свою кваліфікацію. Хороший розробник завжди зможе знайти собі нову роботу, а ви навряд чи легко знайдете нового розробника.

6. Не завантажуйте розробників одночасно декількома завданнями. Постійне відвернення уваги розробника від завдання до завдання, особливо якщо вони належать до різних предметних областей або розроблюються на різних мовах програмування, окрім зниження ефективності роботи, має ще один негативний результат – настання апатії (чи як говорять деякі програмісти– «алергії») до завдання або до роботи взагалі.

7. Забезпечте для розробників навчання і обмін досвідом. Як керівник,

ви не можете бути в курсі всіх останніх змін у світі розробки ПЗ. Ви можете вважати, що людині, яка вміє програмувати, вчитися більше не потрібно, цілком достатньо досвіду, який отримується в процесі роботи. Проте якщо розробники не будуть в курсі останніх технологій, ваше ПЗ застаріє дуже швидко і стане стримуючим чинником на шляху до нових завоювань ринку.

Керівник проекту

Однією з поширених помилок є вибір керівника проекту, який не має відповідних технічних знань для реалізації цього проекту. Ця проблема зазвичай зустрічається на великих проектах, де потрібна велика команда програмістів. Часто існує технічний лідер, який може керувати проектом также добре, як і вирішувати технічні питання.

В процесі аналізу вимог замовника важливо, щоб в переговорах брав участь один з членів команди розробників, а в кращому разі, провідний технічний фахівець або технічний керівник проекту. Якщо в процесі обговорення бере участь тільки адміністративна особа, або керівник проекту, далекий від проблем безпосереднього кодування, то виникає безліч проблем і питань, пов'язаних з можливою оптимізацією окремих операцій, створенням словника баз даних, системними вимогами до створюваного програмного забезпечення і так далі. З іншого боку, участь в обговоренні проекту технічних фахівців може привести до помітного спрощення проекту за рахунок приведення окремих вимог користувача до вже існуючих і раніше розроблених технологій задоволення цих вимог. Якщо вказані рекомендації будуть дотримані, то технічна сторона розробки буде розглянута більш повно, що дасть можливість згодом уникнути багатьох помилок, пов'язаних з нерозумінням тією або іншою стороною технічних особливостей проекту.

Є багато обов'язків хорошого керівника проектом, перерахуємо тільки деякі.

1. Сучасний керівник – це цілеспрямований організатор, у якого є натхнення, який створює силові поля, які притягують таланти, а не просто службовців, прагнучих зайняти робочі місця.
2. Ніхто не вірить в керівників, які завжди праві і прикидаються, що знають більше, ніж підлеглі. «Людьми не потрібно « керувати». Завдання – направляти людей. Мета – зробити максимально продуктивними специфічні навички і знання кожного окремого працівника».
3. Керівник відповідає за щоденне керівництво проектом. Керівник може делегувати (передавати) іншій людині виконання яких-небудь повноважень керівника, але не відповідальність.
4. Керівник не повинен дозволяти втручатися комусь іншому в керівництво проектом. Тому що за проект відповідає саме керівник, це його особиста відповідальність.
5. Керівник проекту, окрім планування завдання, здійснює ще і розподіл ресурсів, ґрунтуючись на своєму відчутті і дуже хорошому знанні завдання.

6. Керівник відповідає за переговори із замовником. Тільки він їх веде. Він представляє інтереси групи і берет на себе зобов'язання з її боку.

7. Людський чинник – один з найважливіших в керуванні проектами, але частенько його ігнорують. Головне – це люди, талановиті люди. Тому одне з головних завдань керівника – це уміння знаходити і виховувати талановитих розробників.

Інше завдання керівника – відводити різні проблеми від групи, щоб група розробників, працюючих над проектом, просто не знала про ті бурі, які бушують за стінами цього колективу, і не відволікалася на них. Неправильно поступають ті керівники, які, отримавши прочухан від замовника або власного керівництва, передають його тут же своїм виконавцям. Це абсолютно неприпустимо.

Командні ролі

Фахівці в області оцінки і розвитку команд стверджують, що існує лише дев'ять командних ролей, баланс яких є вирішальним чинником успіху або невдачі в командній роботі. Деякі з них наступні.

Генератор ідей. Оригінальний мислитель, який дає життя новим ідеям. Незалежний співробітник з розвиненою уявою, але подібно до інших людей має негативні риси вдачі – може бути надто чутливий до критики. Для успіху генератору ідей потрібні конструктивні стосунки з керівником або координатором групи.

Дослідник ресурсів. Так само, як і генератор ідей, в змозі привнести нові ідеї до групи, але ці ідеї будуть запозичені ззовні, завдяки широким контактам. Дещо безцеремонний, гнучкий і шукає сприятливі можливості. До негативних якостей характеру відносяться лінь, самовдоволення.

Координатор. Зазвичай формальний лідер групи. Керує і направляє групу у бік досягнення цілей. Може заздалегідь визначити, хто з працівників хороший для виконання необхідних завдань. Зазвичай спокійний, впевнений і розпорядливий. Проте іноді схильний до зайвого домінування, і група стає продовженням його сильного «Я».

Мотиватор. Енергійний і в змозі впроваджувати ідеї. Бачить світ як проект, який вимагає впровадження. Зазвичай впевнений, динамічний, емоційний і імпульсивний. Мотор групи, але може бути дратівливим, невгамовним, нелюб'язним.

Аналітик. Оцінює пропозиції і займає позицію спостерігача за просуванням. Не дає групі рухатися неправильним шляхом. Обачний, безпристрасний, має аналітичний склад розуму. Може здаватися байдужим, незацікавленим, іноді стає надмірно критичним.

Натхненник команди. Прагне об'єднувати і вносити гармонію в стосунки між членами групи. Займає позицію того, що розуміє чужі проблеми, прагне допомогти і згладжує конфлікти. За вдачею людина добра, прагне налагоджувати неформальні стосунки. Проте буває нерішучим в складних або

кризових ситуаціях.

Фахівець. Професіонал, самостійний, прагне стати експертом у своїй області. Має високу професійну/технічну експертизу і знання, гордиться своєю роботою. Приносить вклад тільки у вузькій сфері своєї професійної експертизи.

Комунікації

Найважливішим неформальним макропоказником стану проекту є комунікації, їх якість і кількість. Тільки завдяки ефективним комунікаціям можна досягати синергетичного ефекту, який відрізняє команду від просто групи. Вже з XP-технології відомо, що освоєння нової технології парою програмістів, які здійснюють інтенсивний обмін знаннями, відбувається мінімум в 3 рази швидше, ніж у разі, коли ту ж роботу виконує один програміст.

Недостатня кількість комунікацій свідчить, як правило, про відсутність команди, кожен поглиблений у своє завдання і не цікавиться, що роблять його колеги. В результаті буде зроблено не те, що треба, а те, що буде зроблено, навряд чи вдасться інтегрувати в єдину систему.

Для побудови ефективної комунікації необхідно обов'язково враховувати індивідуальні особливості людей. Для кожного типу особи існує свій найбільш ефективний спосіб спілкування. Якщо той, хто говорить не бере до уваги індивідуальні особливості того, хто слухає, комунікації, як правило, заходять у безвихідь. При цьому буде відсутня (повністю або частково) передача інформації.

Не може бути ефективної команди, якщо учасники не знають і не вміють робити свою справу. Важливо приводить список чинників незрілості і неефективності співробітника:

1. Не дотримується інструкції.
2. Погано контролює час.
3. Не любить, коли контролюють його роботу.
4. Не звертає увагу на якість роботи.
5. Не може сконцентруватися на роботі.
6. Має особисті проблеми.
7. Перебільшує свої здібності.
8. Не виконує свою долю роботи.
9. Не любить змін в роботі.
10. Не повідомляє про помилки.
11. Не лояльний по відношенню до своєї компанії.

У більшості своїй ці негативні якості носять тимчасовий характер і походять від відсутності досвіду і недостатньої самостійності фахівця. Дружня підтримка і допомога, як правило, дозволяють впоратися з більшістю перерахованих проблем.

В свою чергу, ефективний програміст окрім технічних знань і умінь

повинен володіти ще і особистими компетенціями, необхідними для командної роботи:
1. Займає активну позицію, прагне розширити свою відповідальність і збільшити особистий вклад в загальну справу.
2. Постійно придбає нові професійні знання і досвід, висуває нові ідеї, спрямовані на підвищення ефективності досягнення спільних цілей, домагається поширення своїх знань, досвіду і ідей серед колег.
3. Отримує задоволення від своєї роботи, гордиться її результатами і прагне, щоб ці ж почуття переживали усі колеги.
4. Чітко усвідомлює свої особисті і загальні цілі, розуміє їх взаємообумовленість, наполегливо прагне до їх досягнення.
5. Впевнений в собі і у своїх колегах, об'єктивно оцінює їх досягнення і успіхи, уважно відноситься до їх інтересів і думок, активно шукає взаємовигідне рішення в конфліктах.
6. Сприймає кожну нову проблему, як додаткову можливість підтвердити власний професіоналізм.

Буває так, що начебто, сильний фахівець приносить команді більше шкоди, ніж користі. До патологій поведінки, яка неприйнятна в команді, слід віднести наступні:
1. Непорядність, брехливість, відсутність совісті і почуття справедливості.
2. Неповага і неувага до партнерів. Схильність до негативних оцінок інших. Грубість. «Кожен сам за себе! – ніхто тобі не допоможе»!.
3. Завищена самооцінка. Відчуття власної переваги.
4. Мудрування. Людина сильно переоцінює свій особистий вклад в загальну справу і тому вважає, що він повинен працювати менше, ніж його «менш здібні» колеги.

Оптимальний варіант, коли користувач має уявлення про технічну сторону обговорюваного завдання, а команда програмістів має досвід у сфері діяльності користувача. Коли поєднуються такі якості користувача і розробника, приблизно половина питань відразу знімається з обговорення за непотрібністю.

Керування персоналом

Найціннішим ресурсом програмного проекту є люди. В першу чергу важливі технічні навички інженерів-розробників. Проте ці навички необхідно застосовувати для вирішення проблем в потрібний час і в потрібному місці. Отже, передбачається комбінація двох стилів: робота в команді і лідерство. Організація команди, яка забезпечує ефективну роботу, є дуже складним завданням для менеджера – керівника проекту. Хороша команда повинна демонструвати сплав найрізноманітніших якостей: професійні навички, досвід, згуртованість, дух товариства. Структура команди повинна стимулювати творчу роботу усіх і кожного.

Підбір членів команди

Передусім, керівник повинен організувати правильний підбір членів команди: вони можуть доповнювати один одного по навичках і досвіду і мають бути сумісні один з одним психологічно. При роботі з кандидатом менеджер зазвичай враховує наступні аспекти:

– досвід роботи у багатьох апаратно-програмних середовищах;
– знання мов програмування;
– освіта і досвід роботи за фахом;
– комунікабельність і здатність адаптуватися;
– особові якості.

Освіта є комплексним показником початкових знань і навичок кандидата, а також його здібності до навчання. Досвід же характеризує кінцеві знання і навички фахівця.

Комунікабельність характеризує можливості спілкування з колегами, керівниками і іншими зацікавленими в проекті особами. Здатність до адаптації може пояснювати «послужний список» – наявний робочий стаж.

Особові якості оцінити, мабуть, найважче. Тут і психологічний портрет, і темперамент, і ініціативність, і цілеспрямованість, і багато що інше. Саме ці якості визначають сумісність кандидата з колективом.

Важливо також правильно вибрати лідера команди. Він відповідає за технічне керівництво або за адміністративне керування (можливо і поєднання цих обов'язків). Лідери мають бути в курсі повсякденної діяльності команди, забезпечуючи її ефективну роботу і співпрацю з керівництвом проекту. Вони повинні ладнати з усіма членами колективу, згладжуючи напруженості і усувати неприємності.

При програмуванні без персоналізації усі робочі продукти (моделі, код, документація) **вважаються власністю усієї команди**, а не окремого співробітника, який займався їх створенням.

Переваги розробки без персоналізації:

– спрощення процедур перевірки, критики недоліків, підвищення їх об'єктивності;
– заохочення стилю невимушеного обговорення робочих завдань, достоїнств і недоліків окремих рішень;
– активізація дружніх стосунків, підвищення рівня щирості;
– швидкий ріст майстерності (завдяки роботі пліч-о-пліч);
– поліпшення якості, вдосконалення результатів роботи.

Взаємодії в команді

Для команди програмного проекту просто потрібна розвинена система взаємодії, іншими словами, спілкування і хороші засоби зв'язку між співробітниками. Співробітники повинні інформувати один одного про хід роботи, рішення, які приймаються, а також про зміни, які внесені в попередні рішення. Постійна взаємодія теж сприяє згуртованості і підвищенню якості

роботи, оскільки співробітники спільно обговорюють рішення, починають краще розуміти мотивацію своїх колег.

На ефективність взаємодії впливають наступні параметри.

1. Розмір/структура команди. З ростом числа учасників кількість зв'язків по взаємодії росте квадратично. Наприклад, між трьома учасниками є три зв'язки, чотири учасники мають шість зв'язків, п'ять чоловік – десять зв'язків, тобто *n* чоловік мають $(n-2) + ... + 1 = n(n-1)/2$ зв'язків (кожен з кожним). Отже, 50 чоловік повинні брати участь в 1225 взаємодіях. Але ж це неможливо!

На початку 1990-х років засновник фірми Borland, Філіп Кан постулював формулу продуктивності розробки програмного забезпечення. Він назвав цю формулу продуктивності розробки ПЗ як «Закон Філіпа». Закон свідчить, що продуктивність розробника програмного забезпечення в команді з «*N» людей зменшується, розділивши його на корінь кубічний з «N»*, що показує: чим більше в команді розробки ПЗ фахівців, тим нижче продуктивність кожного з них. Якщо вірити закону Філіпа, то створення прикладного коду об'ємом в 1 Гбайт за допомогою традиційної техніки зажадало б велетенських трудовитрат.

Для великих команд альтернативою є їх розділення на групи. Кожна група відповідає за певну частину проекту і працює над нею. Зазвичай чисельність групи не перевищує восьми чоловік. У таких групах проблеми взаємодії частково зникають. Для взаємодії з іншими групами в кожній групі виділяється один співробітник. Така структура зберігає переваги невеликих команд, але дозволяє великій кількості людей створювати великі програмні продукти.

2. Ієрархія команди. Співробітники в команді з горизонтальною організацією (один рівень, усі співробітники рівні) легше спілкуються між собою, чим в командах з багаторівневою організацією і ієрархією стосунків (начальники/підлеглі). У останніх командах взаємодія відбувається між рівнями, в ієрархічній послідовності.

Склад групи

Не існує універсального рецепту для визначення оптимального складу групи розробників. Цей склад залежить від великої кількості різноманітних чинників: стилю менеджменту, прийнятого в організації, предметної області і розміру проекту, професійних можливостей співробітників організації і т. д.

Типові ролі:

1. Аналітик – відповідає за розвиток і інтерпретацію вимог замовника; має бути експертом в предметній області, але працювати в тісному контакті з іншими співробітниками.

2. Архітектор – відповідає за проектування і розвиток архітектури продукту, є одним з найбільш кваліфікованих фахівців, які мають досвід ухвалення стратегічних рішень; окрім досвіду проектування, архітектор повинен вміти

програмувати, оскільки його рішення втілюються в програмному коді.
3. Конструктор компонентів – головний творець компонентів (будівельної цегли, з якої компонується продукт).
4. Фахівець з інтеграції – відповідає за зборку сумісних версій компонентів і перевірку правильності їх спільної роботи, підтримує випуск версій продукту.
5. Фахівець з документації – документує усі реалізовані рішення, готує документацію для користувача.
6. Системний програміст – відповідає за створення і адаптацію програмних утиліт, які полегшують розробку в проекті.
7. Системний адміністратор – керує фізичними комп'ютерними ресурсами в проекті.

Зрозуміло, не кожен проект вимагає виконання усіх цих ролей. У невеликих проектах співробітники можуть грати відразу декілька ролей.

Процес розробки

Найбурхливіша дискусія про процеси розробки розгортається навколо вибору між ітеративною і водоспадною моделями.

При організації роботи в стилі **водоспаду** проект ділиться на підставі виду робіт. Щоб створити програмне забезпечення, необхідно зробити певні дії: проаналізувати вимоги, створити проект, виконати кодування і тестування.Такий річний проект може включати двомісячну фазу аналізу, за якою йде чотиримісячна фаза дизайну (моделювання), а потім тримісячна фаза кодування і, нарешті, тримісячна фаза тестування.

Ітеративний стиль ділить проект за принципом функціональності продукту. Можна взяти рік і розділити його на тримісячні ітерації. У першій ітерації береться чверть вимог і виконується повний цикл розробки програмного забезпечення для цієї чверті: аналіз, дизайн, кодування і тестування. До кінця першої ітерації у вас є система, яка має чверть необхідної функціональності. Потім ви приступаєте до другої ітерації і через шість місяців отримуєте систему, яка робить половину того, що їй покладене.

При розробці способом водоспаду після кожного етапу зазвичай в якому-небудь виді виконується формальна здача, але має і місце повернення назад. В процесі кодування можуть з'ясуватися обставини, які змушують знову повернутися до етапів аналізу і дизайну. Звичайно, на початку кодування не слід думати, що аналіз завершений. І рішення, прийняті на стадії аналізу і дизайну, неминуче переглядатимуться пізніше. Проте ці зворотні потоки є виключеннями і мають бути по можливості зведені до мінімуму.

При ітеративному процесі розробки перед початком реальної ітерації зазвичай спостерігається деяка дослідницька активність. Як мінімум на вимоги буде кинутий поверхневий погляд, достатній, принаймні, для розділення вимог на ітерації для подальшого виконання. В процесі такого дослідження можуть бути прийняті деякі рішення по дизайну самого вищого рівня.

З іншого боку, незважаючи на те що в результаті кожної ітерації має з'явитися інтегроване програмне забезпечення, готове до постачання, часто буває, що воно ще не готове і потрібний деякий стабілізаційний період для виправлення останніх помилок.

Звичайно, можна не передавати систему на реалізацію у кінці кожної ітерації, але вона повинна знаходитися в стані виробничої готовності. Проте ітеративний процес найчастіше передбачає передачу підсистеми в реалізацію, оскільки можна на кожній ітерації оцінити працездатність системи і отримати якіснішу зворотну реакцію. У цій ситуації часто говорять про проект, що має декілька версій, кожна з яких ділиться на декілька ітерацій.

Можливий і змішаний підхід (так званий життєвий цикл **поетапної доставки**), відповідно до якого спочатку виконуються аналіз і проектування верхнього рівня в стилі водоспаду, а потім кодування і тестування, розділені на ітерації.

При використанні методу водоспаду важко стверджувати, що розробка якогось проекту дійсно йде у вірному напрямі. Занадто легко оголосити перемогу на ранньому етапі і приховати помилки планування. Зазвичай єдиний спосіб, яким ви дійсно можете показати, що слідуєте по такому шляху, полягає в тому, щоб отримати протестоване, інтегроване програмне забезпечення. У разі ітеративного процесу це повторюється багаторазово, і в результаті, якщо щось йде не так, як потрібно, ви своєчасно отримуєте відповідний сигнал.

Саме тільки з цієї причини рекомендується уникати методу водоспаду в чистому вигляді. Принаймні, необхідно застосовувати поетапну доставку, якщо неможливо використати ітеративний метод в повному об'ємі.
Особливо важливо, щоб при ітеративному процесі кожна ітерація завершувалася створенням протестованого, інтегрованого програмного продукту, який би мав якість, як можна ближчу до якості серійної продукції.

Загальним прийомом при ітеративній розробці є упаковка за часом. Таким чином, ітерація займатиме фіксований проміжок часу. Якщо виявилося, що ви не в змозі виконати усе, що планували зробити за час ітерації, то необхідно викинути деяку функціональність з цієї ітерації, але не слід змінювати дату виконання. У більшості проектів, заснованих на ітеративному процесі, протяжність ітерацій однакова, і це дозволяє вести розробку в постійному ритмі.

Одним з найбільш загальних аспектів ітеративної розробки є питання переробки (рефакторинга). Ітеративна розробка недвозначно припускає, що ви перероблятимете і видалятимете існуючий код на останній ітерації проекту. В багатьох областях людської діяльності, наприклад в промисловому виробництві, переробка вважається збитком. Але створення програмного забезпечення не схоже на промислове виробництво – часто буває вигідніше переробити існуючий код, чим латати код невдало спроектованої програми.

Ефективність керівництва програмним проектом цілком визначається

правильністю планування робіт, які потрібні для його виконання. План допомагає передбачати можливі проблеми розробки ПЗ і ввести захисні заходи для їх попередження і рішення. План створюється на початковому етапі проекту і розглядається менеджерами і інженерами-розробниками як керівний документ, виконання якого повинне привести до успішного завершення проекту.

Ці питання ми докладно розглянемо в наступній лекції.

Контрольні запитання

1. Дайте визначення поняттю *об'єкту*.
2. Перерахуйте командні ролі проекту.
3. Перерахуйте обов'язки керівника проекту.
4. Дайте визначення поняттю *адаптивні вимоги і адаптивний замовник*.
5. Що таке *ітеративний стиль* розробки ПЗ?
6. Що таке *прогнозуюче і адаптивне планування*?
7. Дайте визначення поняттю *керування конфігурацією*.

Література

1 Алистэр Коуберн, Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения, Humans and Technology, Октябрь, 1999.

2 Брукс Фредерик. Мифический человеко-месяц, или Как создаются программные комплексы. – СПб.: Символ-Плюс, 2006. – 304 с.

3 Интернет-ресурс http://scrum.org.ua/wp-content/uploads/2008/12/scrum_xp-from-thetrenches-rus-final.pdf.

4.3 Планування програмного проекту

Розглядаючи планування проектів, необхідно чітко усвідомлювати, що маємо справу з деяким динамічним об'єктом, тому система керування проектом має бути досить гнучкою, щоб допускати можливість модифікації без глобальних змін в робочій програмі.

Ефективність керівництва програмним проектом цілком визначається правильністю планування робіт, які потрібні для його виконання. План допомагає передбачати можливі проблеми розробки ПЗ і ввести захисні заходи для їх попередження і рішення. План створюється на початковому етапі проекту і розглядається менеджерами і інженерами-розробниками як керівний документ, виконання якого повинне привести до успішного завершення проекту. Цей початковий план повинен максимально детально описувати усі

етапи роботи проекту.

Планування є багатокроковим ітераційним процесом. Дуже важливо, щоб план регулярно переглядався – адже в міру роботи в проект безперервно поступають нові відомості. Важливими чинниками, які повинні враховуватися при розробці плану, є контрактні зобов'язання фірми, вимоги замовника, бюджетні і тимчасові обмеження.

Планування розпочинається з оцінки предметної області програмної системи, її розміру, трудовитрат і часу на розробку. Формується команда виконавців, розподіляються їх функції. При цьому враховуються обмеження за часом, бюджету, наявності і можливостям співробітників, матеріально-технічному забезпеченню. Потім визначаються етапи розробки, контрольні віхи проекту і перелік артефактів – результатів кожного етапу.

Якщо зафіксовані внутрішні проблеми або замовник змінив/розширив список вимог, можливий перегляд первинних оцінок проекту. Такий перегляд може привести до модифікації початкового (чи вже проміжного) графіку робіт. Якщо зміни впливають на терміни завершення або вартість проекту, із замовником узгоджуються нові обмеження проекту. Після цього продовжують проект – переходять до наступного етапу роботи.

Структура плану керування програмним проектом

План керування проектом має бути складений так, щоб кожен знав, що і коли йому потрібно робити. Існує безліч стандартів для таких планів. Але у будь-якому випадку більшість планів містять наступні розділи.

1. Вступ.

1.1. Огляд проекту. Повинен визначати проект, але не намагатися охопити усі вимоги до нього. Самі вимоги наводяться в Специфікації вимог до програмного забезпечення.

1.2. Результуючі артефакти проекту. Список усіх документів, початкових файлів і кінцевих програмних продуктів, які мають бути створені.

1.3. Розвиток плану. Напрями очікуваного розширення і зміни.

1.4. Посилальні матеріали.

1.5. Визначення і абревіатури.

2. Організація проекту.

2.1. Модель процесу. Посилаються на тип процесу розробки, який буде використаний (наприклад, водоспадний, спіральний, інкрементний).

2.2. Організаційна структура. Описується внутрішня організація команди.

2.3. Організаційні рамки і взаємозв'язки. Шляхи можливої взаємодії між організаціями. Усе це залежить від зацікавлених в проекті сторін. Наприклад, тут визначається, яким чином здійснюватиметься взаємодія між відділом розробки і маркетинговим, чи будуть це регулярні зустрічі або листування по електронній пошті і т. д.

2.4. Відповідальність за проект. Визначає межі відповідальності, тобто хто за що відповідає. Наприклад, за що несе відповідальність координатор

підвищення ефективності команди при горизонтальній організації. Чи відповідає він за загальний успіх проекту, чи надає персональні рекомендації або займається тільки керівництвом?

3. Аналіз ризиків.

3.1. Цілі і пріоритети. Проголошується робоча філософія проекту.

3.2. Допущення, залежності і обмеження.

3.3. Керування ризиками.

3.4. Механізми моніторингу і контролю. Визначають, хто керуватиме, контролюватиме і (чи) здійснюватиме перевірку проекту, а також пропонує, як і коли це повинно бути зроблено.

3.5. План розставляння кадрів.

4. Технічний процес.

4.1. Методи, інструменти і технології. Накладаються обмеження на мови і використовувані інструменти. Може містити інформацію про повторно використовувані вимоги і використання такої техніки, як зразки проектування.

4.2. Документація програмного забезпечення.

4.3. Функції супроводу проекту. Описані дії для підтримки процесу розробки, такі як керування конфігурацією і забезпечення якості. Якщо ж функція підтримки представлена в різних документах (наприклад, в плані керування конфігураціями або в плані якості), то в цьому пункті будуть посилання на ці документи. Інакше тут повністю специфікуються функції підтримки.

5. Розподіл робіт, графік і бюджет.

5.1. Розподіл робіт. Описує те, як робота повинна розподілятися і надаватися після виконання. Оскільки програмна архітектура ще не затверджена, перша версія цього пункту буде поверхневою. Деталі з’являються в подальших версіях плану.

5.2. Потреби в ресурсах. Оцінюються трудовитрати, апаратне і програмне забезпечення, необхідні для зборки і технічної підтримки продукту. Можуть бути приведені результати оцінки вартості. Цей пункт уточнюється і деталізується з кожною ітерацією.

5.3. Виділення бюджету і ресурсів. Розподіляються ресурси між різними частинами проекту впродовж усього його життєвого циклу. Наводяться оцінки вартості людино-дня, можуть вказуватися оцінки вартості апаратури і програмного забезпечення.

5.4. План-графік. Містить розклад, визначальний, як і коли мають бути виконані різні етапи процесу.

В міру виконання проекту план повинен регулярно переглядатися.

Структура графіку робіт програмного проекту

Складання графіку – одна з найвідповідальніших робіт менеджера проекту. Тут менеджер оцінює тривалість проекту, визначає ресурси, необхідні

для реалізації робочих завдань, і розгортає послідовність завдань в часі. Як правило, ця дія виконується за допомогою спеціалізованої утиліти планування проекту.

Плануючі утиліти дозволяють:

– визначити критичний шлях (ланцюжок завдань, задаючих тривалість усього проекту);

– визначити тривалість критичного шляху;

– встановити для кожного завдання найбільш вірогідну часову оцінку;

– вирахувати межі, які визначають тимчасове вікно для окремого завдання.

Першими виконуваними завданнями є збір вимог і аналіз вимог. Вони закладають фундамент для подальших паралельних завдань. Збір вимог проводиться з метою:

– з'ясування потреб замовника;

– оцінки здійснимості програмної системи;

– виконання економічного і технічного аналізу;

– визначення вартості і обмежень планування;

– створення системної специфікації.

Аналіз вимог дає можливість:

– уточнити функції і характеристики програмного продукту;

– позначити інтерфейс продукту з іншими системними елементами;

– визначити проектні обмеження програмного продукту;

– побудувати моделі: процесу, даних, режимів функціонування продукту;

– створити такі форми представлення інформації і функцій системи, які можна використати в ході проектування.

Результати аналізу зводяться в специфікацію аналізу, яка містить конкретизовані вимоги до програмного продукту.

Завдання по проектуванню і плануванню тестів можуть бути розпаралелені. Завдяки модульній природі ПЗ для кожного модуля можна передбачити паралельний шлях для детального (процедурного) проектування, кодування і тестування. Після отримання всіх модулів ПЗ вирішується завдання тестування інтеграції – об'єднання елементів в єдине ціле. Далі проводиться тестування правильності, яке забезпечує перевірку відповідності за вимогами замовника.

Прогнозуюче і адаптивне планування

Прогнозуючий підхід спрямований на виконання роботи на початковому етапі проекту, для того щоб краще зрозуміти, що треба робити надалі. Таким чином, настає момент, коли частину проекту, яка залишилася, можна оцінити з достатньою мірою точності.

В процесі *прогнозуючого планування* проект розділяється на дві стадії. На першій стадії складаються плани, і тут передбачати важко, але друга стадія

більше передбачувана, оскільки плани вже готові. Проте все ще йдуть гострі дискусії про те, чи багато проектів можуть бути передбачуваними. Суть цього питання полягає в аналізі вимог. Одна з найістотніших причин складності програмних проектів полягає в трудності
розуміння вимог до програмних систем.

Більшість програмних проектів піддаються істотному **перегляду вимог**: зміні вимог на пізній стадії виконання проекту. Наслідки перегляду можна запобігти, заморозивши вимоги на ранній стадії проекту і не дозволяючи змінам з'являтися, але це призводить до ризику поставити клієнтові систему, яка більше не задовольняє вимогам користувачів.

Прибічники іншої школи стверджують, що перегляд вимог неминучий, що у багатьох проектах важко стабілізувати вимоги в такому ступені, щоб була можливість використати прогнозуюче планування. Це може бути або слідством того, що виключно важко уявити, що може робити програмний продукт, або слідством того, що умови ринку диктують непередбачувані зміни. Ця школа підтримує адаптивне **планування** відповідно до твердження, що прогнозованість – це ілюзія.

Відмінність між прогнозуючими і адаптивними проектами проявляється різними шляхами, коли люди говорять про стан проекту. Коли стверджується, що виконання проекту йде добре, оскільки робота ведеться відповідно до плану, то мається на увазі метод прогнозування.

При адаптивній розробці не можна сказати «відповідно до плану», оскільки план увесь час міняється. Це не означає, що адаптивні проекти не плануються; зазвичай планування займає значний час, але план трактується як основна лінія проведення послідовних змін, а не як пророцтво майбутнього.

На основі прогнозуючого плану можна розробити контракт з фіксованою функціональністю за фіксованою ціною. У такому контракті точно вказується, що повинно бути створено, скільки це коштує і коли продукт буде поставлений.

У адаптивному плані така фіксація неможлива. Ви можете позначити бюджет і терміни постачання, але ви не можете точно зафіксувати функціональність поставляємого продукту. Адаптивний контракт припускає, що користувачі співпрацюватимуть з командою розробників, щоб регулярно переглядати необхідну функціональність і переривати проект, якщо прогрес занадто незначний. Як такий процес адаптивного планування може визначати проект зі змінними межами функціональності за фіксованою ціною.

Природно, адаптивний підхід менш бажаний, оскільки усі віддають перевагу великій передбачуваності програмних проектів. Проте передбачуваність залежить від точності, коректності і стабільності безлічі вимог. Звідси витікають дві важливі поради.

1. Не складайте прогнозуючий план до тих пір, поки не отримаєте точні і коректні вимоги і не будете впевнені, що вони не піддадуться істотним змінам.

2. Якщо ви не можете отримати точні, коректні і стабільні вимоги, то використайте метод адаптивного планування.

Вибір процесу розробки

За останній час виріс інтерес до гнучких процесів розробки програмного забезпечення. Гнучкий (agile) – це широкий термін, який охоплює велику кількість процесів, які мають загальну множину величин і понять, визначених Маніфестом гнучкої розробки програмного забезпечення (Manifesto of Agile Software Development, http://agileManifesto.org). Прикладами таких процесів є XP (Extreme Programming – екстремальне програмування), Scrum (сутичка, зіткнення), FDD (Feature Driven Development – розробка, керована можливостями) і DSDM (Dynamic Systems Development Method – метод розробки динамічних систем).

Гнучкі процеси виключно адаптивні за своєю природою. Вони також мають чітку орієнтацію на людину. Гнучкі підходи припускають, що найбільш важливим чинником успішного завершення проекту є кваліфікація виконавців і їх хороша спільна робота з людської точки зору. Значущість процесів або інструментів, ними використовуваних, безперечно стоїть на другому місці.

Гнучкі методи в основному спрямовані на використання коротких, обмежених за часом ітерацій, які найчастіше закінчуються через місяць або раніше. Оскільки їх вклад в документацію невеликий, то в гнучкому підході не передбачається застосування UML в режимі проектування. Частіше усього UML використовується в режимі ескізування і рідше в якості мови програмування. (З UML ви знайомі з попередніх курсів. Ми про UML будемо говорити в наступних лекціях).

В більшості своїй гнучкі процеси не занадто **формалізовані.** Сильно формалізовані або ваговиті процеси мають багато документації і постійний контроль під час виконання проекту. Гнучкий підхід припускає, що формалізм заважає проведенню змін і суперечить природі талановитих осіб. Тому гнучкі процеси часто називають **полегшеними** (lightweight).

Важливо розуміти, що недостатня формализованность є наслідком адаптивності і орієнтації фахівців, а не фундаментальною властивістю. Один з прибічників ітеративного процесу розробки (Мартін Фаулер) говорить у своїй книзі [4]: «Застосовуйте ітеративний метод розробки тільки в проектах, яким ви бажаєте успіху».

При грамотному застосуванні ітеративної розробки, вона є дуже важливим методом, здатним допомогти в ранньому виявленні можливих ризиків і в поліпшенні керованості процесом розробки. Ітеративна розробка вимагає ретельного планування. Але це надійний підхід, і тому будь-яка книга з об’єктно-орієнтованої розробки рекомендує його застосовувати – і небезпідставно.

Одна з великих переваг ітеративної розробки полягає в можливості часто удосконалювати процес. У кінці кожної ітерації слід проводити її

ретроспективний аналіз, збираючи команду на наради, щоб розглянути, як йдуть справи і що можна поліпшити. В кінці розробки проекту або його основної версії можна провести формальніший ретроспективний аналіз проекту, який може зайняти пару днів.

Налаштування UML під процес

При розгляді графічних мов моделювання зазвичай про них думають в контексті водоспадного процесу. Водоспадний процес, як правило, супроводжується документами, які виступають прес-релізами між стадіями аналізу, дизайну і кодування. Часто графічні моделі можуть займати основну частину цих документів.

Незалежно від того, застосовуєте ви метод водоспаду або ні, так або інакше ви проводите аналіз, дизайн, кодування і тестування. Використання UML не має на увазі обов'язкову розробку документів або завантаження складних CASE-систем.

Аналіз вимог. В процесі аналізу вимог необхідно зрозуміти, що клієнти і користувачі програмного забезпечення чекають від системи. У вашому розпорядженні є безліч прийомів UML:

1. Прецеденти, які описують, як люди взаємодіють з системою.
2. Діаграма класів, яка будується з точки зору концептуальної перспективи і може служити хорошим інструментом для побудови точного словника предметної області.
3. Діаграма діяльності, яка показує робочий потік організації, способи взаємодії програмного забезпечення і користувачів. Діаграма діяльності може показати контекст для використання прецедентів, а також деталі роботи складних прецедентів.
4. Діаграма станів, яка може виявитися корисною, якщо концепція має своєрідний життєвий цикл з різними станами і подіями, які змінюють ці стани.

Аналізуючи стани, пам'ятайте, що найважливіше – ця взаємодія з вашими користувачами і клієнтами. Звичайно це непрограмісти, і вони не знайомі з UML і іншими подібними технологіями. Будьте готові у будь-який момент відійти від правил UML, якщо це допоможе поліпшити взаєморозуміння. Найбільший ризик у разі застосування UML для аналізу полягає в тому, що ви будуєте діаграми, не зовсім зрозумілі фахівцям в конкретній предметній області. Така діаграма гірша, ніж даремна; вона лише здатна вселити в розробників неправдиве почуття впевненості.

При розробці моделі ви можете широко застосовувати діаграми. Можна використати більше нотацій і при цьому бути точнішим. Ось деякі корисні прийоми:

1. Діаграми класів з точки зору програмного забезпечення. Вони показують класи програми і їх взаємозв'язок.
2. Діаграми послідовності для загальних сценаріїв. Правильний підхід

полягає у витяганні найбільш важливих і цікавих сценаріїв з прецеденту, а також у використанні і діаграм послідовності з метою зрозуміти, що відбувається в програмі.
3. Діаграми пакетів, які показують високорівневу організацію програмного продукту.
4. Діаграми станів для класів із складним життєвим циклом.
5. Діаграми розгортання, які показують фізичну конфігурацію програмного забезпечення.

Створення моделей зазвичай відбувається на ранній стадії ітерації і може бути зроблене по частинах для різних розділів функціональності, призначеної для цієї ітерації. Крім того, ітерація має на увазі зміну існуючої моделі, а не побудову кожного разу нової моделі.

Застосування UML в режимі ескізування – рухливіший процес. Один з підходів полягає у виділенні пари днів на початку ітерації на створення ескізного дизайну для цієї ітерації. Можна також проводити короткі сесії проектування у будь-який момент в ході ітерації, влаштовуючи короткі півгодинні наради всякий раз, коли розробники починають сперечатися з приводу нетривіальної функції.

В режимі проектування очікується, що програмна реалізація будуватиметься відповідно до діаграм. Зміну моделі слід вважати відхиленням, яке вимагає, щоб проектувальники, які створили цю модель, її переглянули.

Ескіз зазвичай трактується як перший зріз дизайну. Якщо в ході кодування виявляється, що ескіз не зовсім точний, то розробники повинні мати можливість змінити дизайн. Розробники, впроваджувальні дизайн, повинні самі вирішувати, чи варто влаштовувати широку дискусію, щоб зрозуміти усі можливі варіанти.

Керування конфігурацією

Керування конфігурацією – це координація різних версій і частин документації і програмного коду. Керування конфігурацією ПЗ – захисна діяльність, вживана на усіх етапах життєвого циклу ПЗ. Вона забезпечує керування змінами в ПЗ, яке включає наступні дії:
1. Ідентифікація зміни.
2. Контроль зміни.
3. Гарантія правильної реалізації зміни.
4. Формування повідомлення про зміни.

Керування конфігурацією стартує з початком програмного проекту і закінчується з припиненням використання ПЗ.

За час життя програмний код продукту зазнає зміни двох категорій: додавання нових частин, підключення нових версій існуючих частин. Звичайно, слід враховувати обидві категорії змін.

Інформацію на виході процесу розробки ПЗ можна розділити на три категорії:

1. Комп’ютерні програми (в вигляді виконуваних кодів).
2. Документи, які описують програми (як для технічного персоналу, так і для користувачів).
3. Структури даних (як зовнішні, так і внутрішні).

Сукупність усіх елементів інформації, які виробляються як частину процесу розробки ПО, називають конфігурацією ПО.

З розвитком процесу розробки ПЗ кількість елементів конфігурації нестримно росте.

Мінімальна конфігурація ПЗ включає наступні базові елементи:

1. Системна специфікація.
2. План програмного проекту.
3. Специфікація вимог до ПЗ.Працюючий або паперовий макет.
4. Попередній посібник користувача.
5. Специфікація проектування.
6. Лістинги початкових текстів програм.
7. План і методика тестування. Тестові варіанти і отримані результати.
8. Керівництво по роботі і інсталяції.
9. Виконуваний код програм.
10. Опис бази даних.
11. Посібник користувача по налаштуванню.
12. Документи супроводу. Звіти про проблеми ПЗ.Запити супроводу. Звіти про зміни.
13. Стандарти і методики розробки ПО.

Керування конфігурацією полягає в застосуванні дій для керування змінами впродовж усього життєвого циклу ПО.

Зміна – неодмінний факт життєвого циклу ПЗ.Замовники хочуть змінити вимоги. Розробники хочуть модифікувати технічний підхід.

Контрольні запитання

1. Дайте визначення поняттю *об’єкту*.
2. Перерахуйте командні ролі проекту.
3. Що таке *ітеративний стиль* розробки ПЗ?
4. Що таке *прогнозуюче і адаптивне планування*?
5. Дайте визначення поняттю *керування конфігурацією*.

Література

1 Алистэр Коуберн, Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения, Humans and Technology, Октябрь, 1999.
2 Брукс Фредерик. Мифический человеко-месяц, или Как создаются программные комплексы. – СПб.: Символ-Плюс, 2006. – 304 с.
3 Интернет-ресурс http://scrum.org.ua/wp-content/uploads/2008/12/scrum_xp-from-thetrenches-rus-final.pdf.
4 Фаулер М. UML. Основы, 3-е изд. – СПб: Символ-Плюс, 2004. – 192 с.

5. Інструментальні засоби проектування обчислювальних систем.

5.1 . *Моделювання розробки програмних систем.*

ОСНОВНІ ВІДОМОСТІ ПРО МОВУ UML

Трудомісткість створення сучасних додатків на початкових етапах проекту, як правило, оцінюється значно нижче зусиль, які реально витрачаються, що служить причиною незапланованих витрат і затягування програм. В процесі розробки додатків змінюються *функціональні вимоги* замовника, що ще більше віддаляє момент закінчення роботи програмістів. Збільшення остаточних термінів готовності розмірів програм змушує залучати понадштатних програмістів, що, у свою чергу, вимагає додаткових ресурсів для організації їх погодженої роботи. В розробці і впровадженні сучасних корпоративних інформаційних систем бере участь багато фахівців різної кваліфікації, для яких однакове розуміння архітектури і функціональності є серйозною проблемою. Таким чином, усі ці особливості призводять до наполегливої необхідності моделювання структури і процесу функціонування програмних систем до початку написання відповідного коду. При цьому неодмінною умовою успішного завершення проекту стає побудова попередньої *моделі* програмної системи.

Модель – абстракція фізичної системи, яка створюється з метою досягнення чого-небудь перед тим, як воно буде створено, і представлена на деякій мові, або в графічній формі. Моделі бувають різні – матеріальні і нематеріальні, штучні і природні, декоративні і математичні. Наведемо декілька прикладів. Знайомі усім нам пластмасові іграшкові автомобільчики – це не що інше, як *матеріальна штучна декоративна* модель реального автомобіля. Звичайно, в такому «авто» немає двигуна, ми не заповнюємо його бак бензином, але як модель ця іграшка свої функції цілком виконує: вона дає

дитині уявлення про автомобіль, оскільки відображає його характерні риси – наявність чотирьох коліс, кузова, дверей, вікон, здатність їхати.

В ході медичних досліджень досліди на тваринах часто передують клінічним випробуванням медичних препаратів на людях. У такому разі тварина виступає в ролі *матеріальної природної* моделі людини.

Рівняння

$$mg - a\frac{dx}{dt} = m\frac{d^2x}{dt^2}$$

– це теж модель, але це модель математична і описує вона рух матеріальної точки під дією сили тяжіння.

Оскільки модель не містить несуттєвих деталей, працювати з нею виявляється простіше, ніж з модельованою суттю. Абстрагування – це одне з найважливіших людських умінь, яке дає нам можливість працювати із складними речами.

Основна вимога до моделі програмної системи – вона має бути зрозуміла замовникові і всім фахівцям проектної групи, включаючи бізнес-аналітиків і програмістів, оскільки моделі програмних систем можуть відображати різні аспекти системи. Наприклад, структурна модель відображає статичну організацію системи, а модель поведінки підкреслює динамічні процеси, властиві системі.

Саме для розробки такої нотації знадобилися зусилля групи фахівців провідних фірм виробників програмного і апаратного забезпечення, які привели до появи мови UML. Розробка і використання *моделей мови UML здійснюється в рамках загальної концепції об'єктно-орієнтованого аналізу і проектування,* яка, у свою чергу, є узагальненням методології *об'єктно-орієнтованого програмування* [1; 2].

Поняття UML

UML є абревіатурою назви *Unified Modeling Language* і перекладається *– уніфікована мова моделювання.*

Отже, UML – це мова і головним словом в цьому поєднанні є слово «мова». *Мова* – це знакова система для зберігання і передачі інформації.

Як Ви знаєте, розрізняються мови *формальні,* правила вживання яких строго і явно визначені, і *неформальні,* вживання яких засноване на практиці, яка склалася. Розрізняються також мови *природьні,* які з'являються як би самі собою в результаті неперсоніфікованих (тобто загальних) зусиль маси людей, і мови *штучні*, які є плодом видимих зусиль певних осіб.

Мова, на якій написана ця лекція, є неформальною і природньою. З іншого боку, переважна більшість мов програмування є формальними і

штучними. Зустрічаються і інші комбінації: наприклад, мова формул алгебри вважається формальною і природньою, а есперанто – неформальною і штучною. UML можна охарактеризувати як **формальну штучну мову**, хоча і не в такій степені, як багато поширених мов програмування. Ознакою штучності служить наявність трьох загальновизнаних авторів – Гради Буча, Айвара Якобсона і Джеймса Рамбо [1].

Для опису формальних штучних мов (зокрема, для опису мов програмування) придумано і використовується безліч різних способів. Проте на практиці склалася загальноприйнята структура таких описів. Вважається, що формальна штучна мова описана належним чином, якщо цей опис містить, щонайменше, такі частини:

– ***синтаксис*** - визначення правил складання конструкцій мови;

– ***семантика*** - визначення правил приписування сенсу конструкціям мови;

– ***прагматика*** - визначення правил використання конструкцій мови для досягнення певної мети.

Як формальна штучна мова UML має синтаксис, семантику і прагматику, хоча ці частини названі в деяких випадках інакше і описані інакше, ніж це прийнято в текстових мовах програмування так як: по-перше, ***UML - мова графічна, а не текстова,*** а по-друге, ***UML - мова моделювання, а не програмування.***

UML – це мова моделювання

Слово «моделювання», яке входить в назву UML, має багато змістових відтінків і способів вживання. Зокрема, англійські слова *modeling* і *simulation* перекладаються одним словом «моделювання», хоча означають різні речі. В першому випадку йдеться про складання моделі, яка використовується тільки для опису об’єкту або явища, яке моделюється. В другому випадку мається на увазі складання моделі, яка може бути використана для отримання істотної інформації про модельований об’єкт або явище. При цьому в другому випадку зазвичай додається уточнювальний прикметник: *чисельне моделювання, математичне моделювання та ін.* UML є мовою моделювання в першому сенсі.

Стосовно розробки програмного забезпечення так склалося, що результати фаз аналізу і проектування, оформлені засобами певної мови, прийнято називати ***моделлю***. *Діяльність зі складання моделей природньо назвати* ***моделюванням.***

Таким чином, ***модель UML* – це, передусім, опис об’єкту або явища**, а також і дещо інше, а саме усе, що авторам UML вдалося включити в мову, не порушуючи принципу уніфікації, про що ми поговоримо в далі.

UML – це уніфікована мова моделювання

Поширення *об’єктно-орієнтованого підходу* до розробки програмних систем викликало потребу у відповідних засобах. Іншими словами, появи

чогось подібного на UML вже чекали практики і три видатні фахівці в цій області, автори найпопулярніших методів, наважилися об'єднати зусилля саме з метою уніфікації своїх (і не лише своїх) розробок відповідно до соціального замовлення. Вони змогли звести воєдино (*уніфікувати*) велику частину того, що було відомо і до них. В результаті уніфікації вийшла теоретично витончена і практично корисна річ – UML. Але уніфікація UML носить не лише історичний характер. UML докладає зусиль (і в основному успішно) в уніфікації декількох різних областей:

- UML надає візуальний синтаксис для моделювання впродовж усього життєвого циклу розробки програмного забезпечення – від постановки вимог до реалізації;
- UML використовується для моделювання всіх аспектів – від апаратних вбудованих систем реального часу до систем підтримки ухвалення рішень;
- UML є незалежною від мов і платформ.

Існує багато технологій і інструментальних засобів, за допомогою яких можна реалізувати оптимальний проект ІС, починаючи з етапу аналізу і закінчуючи створенням програмного коду системи. Ці технології представлені CASE-засобами верхнього рівня, або CASE-засобами повного життєвого циклу (upper CASE tools або full life – cycle CASE tools). Вони не дозволяють оптимізувати діяльність на рівні окремих елементів проекту, і, як наслідок, багато розробників перейшли на так звані CASE-засоби нижнього рівня (lower CASE tools). Проте вони зіткнулися з новою проблемою – проблемою організації взаємодії між різними командами, які реалізовують проект. Уніфікована мова об'єктно-орієнтованого моделювання Unified Modeling Language (UML) стала засобом досягнення компромісу між цими підходами. Існує достатня кількість інструментальних засобів, які підтримують за допомогою UML життєвий цикл інформаційних систем, і, одночасно, UML є досить гнучкою для налаштування і підтримки специфіки діяльності різних команд розробників.

Потужний поштовх до розробки цього напряму інформаційних технологій дало поширення *об'єктно-орієнтованих* мов програмування у кінці 1980-х – на початку 1990-х років. Користувачам хотілося отримати єдину мову моделювання, яка об'єднала б у собі всю потужність об'єктно-орієнтованого підходу і давала б чітку модель системи, що відбиває усі її суттєві сторони. До середини дев'яностих явними лідерами в цій області стали методи Booch (Гради Буч, Grady Booch), OMT – 2 (Джим Рамбо, Jim Rumbaugh), OOSE – Object – Oriented Software Engineering (Айвар Якобсон, Ivar Jacobson). Проте ці три методи мали свої сильні і слабкі сторони: OOSE був кращим на стадії аналізу проблемної області і аналізу вимог до системи, OMT – 2 був найприйнятнішим на стадіях аналізу і розробки інформаційних систем, Booch краще всього підходив для стадій дизайну і розробки. Усе йшло до створення

єдиної мови, яка об'єднувала б сильні сторони відомих методів і забезпечувала найкращу підтримку моделювання. Такою мовою виявилася *UML*.

Створення UML почалося в жовтні 1994 р., коли Джим Рамбо і Гради Буч з Rational Software Corporation стали працювати над об'єднанням своїх методів OMT і Booch [1]. Восени 1995 р. побачила світ перша версія об'єднаної методології, яку вони назвали Unified Method 0.8. Після приєднання в кінці 1995 р. до Rational Software Corporation Айвара Якобсона і його фірми Objectory, зусилля трьох творців найпоширеніших *об'єктно-орієнтованих* методологій були об'єднані і спрямовані на створення UML.
Історію розвитку UML можна було б викласти одним реченням, але враховуючи Вашу спеціальність, розглянемо більш детально.

Зараз консорціум користувачів *UML* Partners включає представників таких грандів інформаційних технологій, як Rational Software, Microsoft, IBM, Hewlett– Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology.
Мова UML прийнята на озброєння практично усіма найбільшими компаніями – виробниками ПЗ (Microsoft, IBM, Hewlett – Packard, Oracle, Sybase та ін.).

Основна ідея UML – можливість моделювати програмне забезпечення і інші системи як набори взаємодіючих об'єктів. Це добре підходить для ОО програмних систем і мов програмування, але також добре працює і для бізнес-процесів і інших прикладних завдань. Таким чином, UML призначена для моделювання. Самі автори UML визначають її наступним чином:
Мова UML – це графічна мова моделювання загального призначення, призначена для специфікації, візуалізації, проектування і документування усіх артефактів, які створюються при розробці програмних систем.

Специфікація і візуалізація

В типових випадках у процесі розробки додатків беруть участь як мінімум дві дійові особи: ***замовник*** (конкретна людина або група осіб, або організація) і ***розробник*** (це може бути програміст-одинак, тимчасова команда проекту або ціла організація, яка спеціалізується на розробці програмного забезпечення). Через те, що дійових осіб двоє, дуже багато що залежить від міри їх взаєморозуміння.

Одним з ключових етапів розробки додатка є визначення того, яким вимогам повинен задовольняти розроблюваний додаток. В результаті цього етапу з'являється формальний або неформальний документ (артефакт), який називають по-різному, маючи на увазі приблизно одне і те ж: *постановка завдання, вимоги, технічне завдання, зовнішні специфікації*.

Аналогічні за призначенням, але, можливо, відмінні за формою і змістом артефакти з'являються і на інших етапах розробки, особливо якщо в розробку включено багато дійових осіб. Для них також використовуються різні назви: *функціональні специфікації, архітектура додатка та ін.* Усі такі артефакти

називатимемо *специфікаціями*.

Специфікація – це декларативний опис того, як щось влаштоване або працює. У нашому випадку специфікація – детальний опис системи, який повністю визначає її мету і функціональні можливості.

Необхідно брати до уваги три тлумачення специфікацій:

– те, яке має на увазі дійову особу, яка є джерелом специфікації (наприклад, замовник);

– те, яке має на увазі дійову особу, яка є споживачем специфікації (наприклад, розробник);

– те, яке об'єктивно обумовлене природою об'єкту, який специфікується.

Ці три трактування специфікацій можуть не співпадати, і, на жаль, як показує практика, часто не співпадають, причому значно.

Замовник може не усвідомлювати своїх об'єктивних потреб, або невірно їх інтерпретувати, або помилятися стосовно природи своїх утруднень.

Розробник може не розбиратися в предметній області замовника і інтерпретувати формулювання специфікацій абсолютно іншим чином. Якщо ж у формулюванні специфікацій бере участь розробник, то зловживання технічною термінологією може абсолютно дезорієнтувати замовника.

Не слід також забувати, що замовник і розробник мають, як правило, абсолютно різне розуміння сенсу цього артефакту. Адже окрім цього є ще аналітики, менеджери, бізнес-консультанти. Кожен з них називає специфікації по-своєму: *вимоги користувача, постановка завдання, технічне завдання, функціональна специфікація, архітектура системи*. Причому усі ці люди, будучи фахівцями в абсолютно різних предметних областях, говорять кожний своєю мовою і часто просто не розуміють один одного. Ось через те і виникає проблема, яку може вирішити тільки наявність єдиного, уніфікованого засобу створення специфікацій, досить простого і зрозумілого для усіх зацікавлених осіб.

Розрізняють специфікації трьох видів: словесні специфікації на природній мові, модельні специфікації і формальні специфікації.

Словесні специфікації на природній мові якраз і викликають масу проблем, оскільки створюються різними фахівцями на «їх мові».

Іншим видом специфікацій є *формальні специфікації.* Зрозуміло, що формальна специфікація є, по суті, математичною моделлю завдання і тому для обчислювальних завдань усе виглядає досить просто. Формалізація ж завдань з інших галузей знань може виявитися складнішою і трудомісткою проблемою.

Коли ми говоримо про те, що UML – це засіб ***візуалізації,*** то ми маємо на увазі *модельні специфікації*. Відомо, що вивчення чогось нового йде набагато простіше, якщо документ містить не лише текст, а ще і ілюстрації до нього. Такі картинки наочні і інтуїтивно зрозумілі, причому майже однозначно розуміються будь-якими зацікавленими особами, так що можуть використовуватися як засіб спілкування між людьми. UML дозволяє

створювати такі прості і зрозумілі картинки (моделі), які описують систему з різних сторін, які можна показати замовникові і обговорити з ним, тобто такі моделі служить засобом комунікації в команді.

Таким чином, основне призначення **UML –** надати, з одного боку, досить формальний, з іншого боку, досить зручний, і, з третього боку, **досить універсальний засіб,** що дозволяє деякою мірою понизити ризик розбіжностей в тлумаченні специфікацій.

Проектування і документування

В оригіналі це призначення UML визначене за допомогою слова construct, яке ми передаємо терміном «проектування». Йдеться про те, що UML призначена не лише для опису абстрактних моделей додатків, але і для безпосереднього маніпулювання артефактами, які входять до складу цих додатків, в тім рахунку такими, як програмний код [3]. Іншими словами, одним з призначень UML є, наприклад, створення таких моделей, для яких можлива **автоматична генерація програмного коду** (чи фрагментів коду) відповідних додатків. Більше того, природа моделей UML така, що можливий і зворотний процес: автоматична побудова моделі за кодом готового додатку. Англійською мовою автоматична побудова моделі за кодом готового додатку називається ***reverse engineering*** і зазвичай перекладається як «*зворотне проектування*». UML-моделі самі по собі вже є документами і досить зрозумілими, навіть для неспеціаліста. Причому будь-який елемент на будь-якій діаграмі може бути забезпечений текстовим коментарем. Тобто, побудова набору діаграм вже є процесом документування майбутньої системи. Більше того, більшість інструментів UML-проектирования вміють витягати текстову інформацію з моделей і генерувати відносно легкі для читання тексти.

Інструментальна підтримка

Розглянемо, як співвідноситься сьогоднішня практика використання UML. Як приклад розглянемо одну з діаграм UML – діаграму використання (опис цього типу діаграм розглянемо нижче).

Можна виділити три основні варіанти використання UML (рис. 1).

Варіант використання ***drawing*** («Малювання діаграм») має на увазі *зображення діаграм UML* з метою обдумування, обміну ідеями між людьми, документування і тому подібного. Значимим для користувача **User** результатом в цьому випадку є саме зображення діаграм. Іноді малювання діаграм від руки фломастером з подальшим фотографуванням цифровим апаратом може виявитися практичним.

Варіант використання ***modeling*** («Моделювання систем») має на увазі *створення і зміну моделі* системи в термінах тих елементів моделювання, які передбачаються метамоделлю UML. Значимим результатом в цьому випадку є ***машинно-читаний артефакт*** з описом моделі. Скорочено називатимемо

такий артефакт просто ***моделлю,*** діяльність зі складання моделі називатимемо моделюванням, а суб'єкта моделювання називатимемо архітектором Architect.

Варіант використання ***development*** («Розробка додатків») має на увазі *детальне моделювання, реалізацію і тестування* додатка в термінах UML.

Значимим для користувача ***Developer*** результатом в цьому випадку є працюючий додаток, який може бути скомпільований в мову, підтримувану

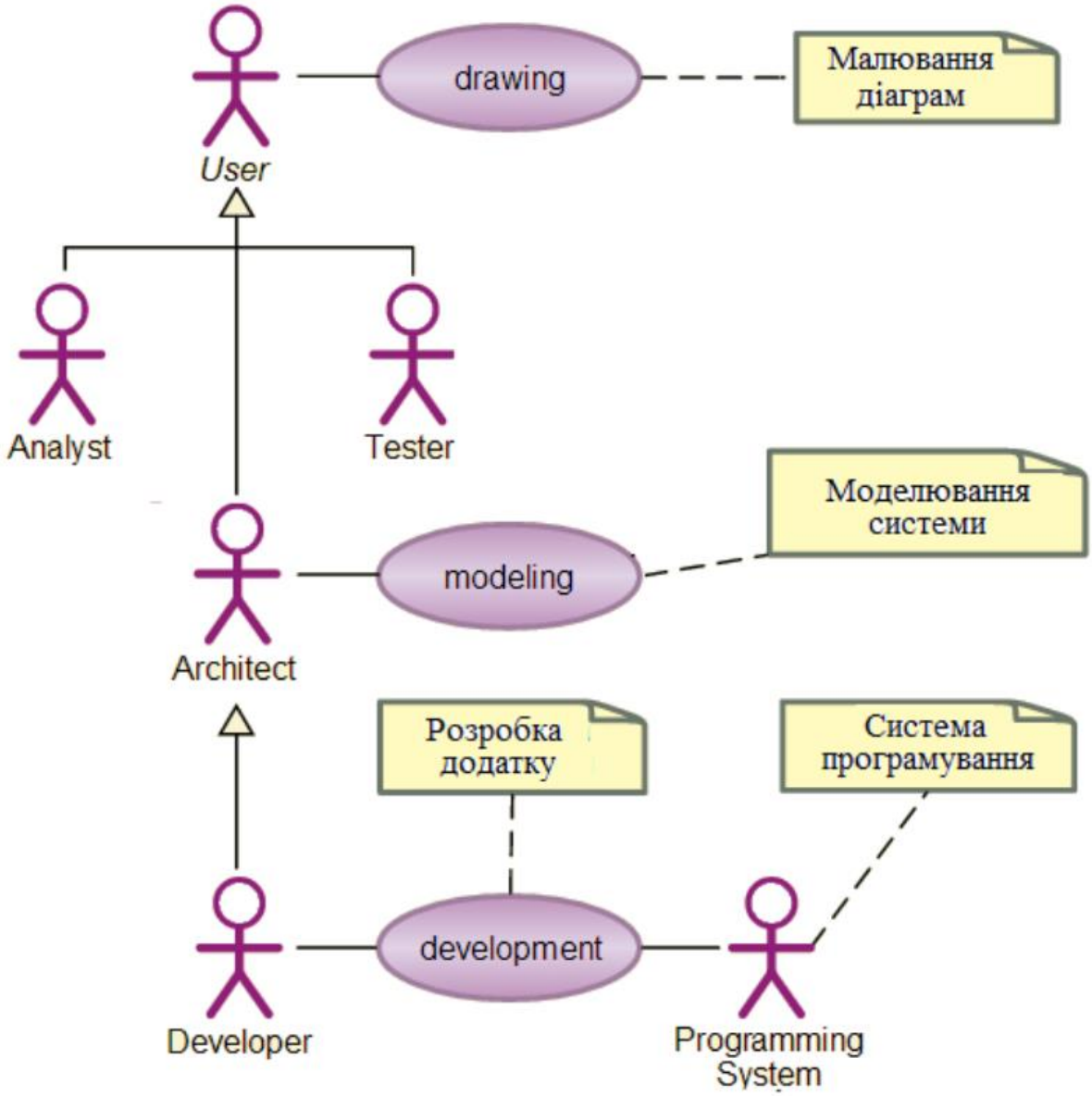


Рисунок 1 – Інструментальна підтримка

конкретною системою програмування або відразу інтерпретований середовищем виконання інструменту. Цей варіант використання найскладніший в реалізації.

Сучасні інструменти підтримують вказані варіанти використання далеко не в рівній мірі. Усі інструменти вміють (погано або добре) візуалізувати всі типи діаграм UML, деякі інструменти дозволяють побудувати модель, яка допускає якесь подальше використання, але тільки небагато інструментів

можуть генерувати виконуваний код і то (ні в якому разі) не для всіх діаграм.

Способи використання UML

UML призначена для вирішення різних завдань, відповідно вона може бути використана і практично використовується по-різному. Далі вказані різні способи використання UML.

Малювання картинок. Графічні засоби UML можна і треба використати безвідносно до всього іншого. Навіть малювання діаграм олівцем на папері дозволяє упорядкувати думки і зафіксувати для себе істотну інформацію про модельований додаток, або іншу систему.

Обмін інформацією. Співтовариство людей, які застосовують і розуміють UML, нестримно росте. Якщо ви використовуватимете UML, то вас розумітимуть інші, і ви розумітимете інших «з напівпогляду».

Специфікація систем. Це найважливіший спосіб використання UML. І хоча не в усіх випадках UML виявляється абсолютно адекватним засобом специфікації, в міру розвитку мови все менше залишатиметься таких виключень, де UML непридатна.

Повторне використання архітектурних рішень. Повторне використання раніше розроблених рішень – ключ до підвищення ефективності.

Слід зауважити, що моделі UML поки що повторно використовуються в обмежених масштабах.

Генерація коду. Генерувати код треба і можна, але можливості наявних інструментів не варто переоцінювати.

Метод визначення UML

В основу опису UML покладений метод розкручування, тобто використання мови, яка визначається, для визначення цієї мови. А саме, основні конструкції UML формально визначені за допомогою UML. Це описано в UML неформально, за допомогою текстів природньою (англійською) мовою.

Метод розкручування часто застосовується при визначенні формальних мов. Наприклад, можна дуже витончено визначити операційну семантику мови програмування, якщо написати транслятор, або інтерпретатор цієї мови цією ж мовою. Одним з перших цей прийом використав Н. Вірт, створюючи мову Паскаль.

В описі UML використовуються три мовні рівні:

1. *Мета-метамодель*, тобто опис мови, якою описана метамодель.
2. *Метамодель*, тобто опис мови, якою описуються моделі.
3. *Модель*, тобто опис самої модельованої предметної області.

Використання приставки **мета** (від грецького μετα, що означає «між», «після», «через») може бути дещо незвичним. Нехай ця мова називається *X*. Якщо опис хороший, то на самому початку вказується мова (іноді її так і називають – метамова), яка використовується для опису мови *X*. Наприклад, наводиться фраза такого типу: «синтаксис мови *X* описаний за допомогою контекстно-вільної граматики, правила якої записані в формі Бекуса-Наура

(БНФ), контекстні умови і семантика описані природною мовою».
Якщо опис не дуже хороший, то така фраза може бути і відсутньою, але вона все одно неявно мається на увазі і від читача вимагається зрозуміти використовувану метамову за контекстом. Далі за допомогою метамови більш менш формально описуються конструкції мови *X*. Усе, що не вдається описати формально, описується природною мовою.

Таким чином, основна ідея опису UML цілком традиційна і узгоджується із загальноприйнятою практикою: ***мета-метамодель*** **–** це опис використовуваного формалізму; ***метамодель*** **–** це і є власне опис мови (елементів моделювання); там, де формалізм не спрацьовує, на допомогу приходить природна мова.

Термінологія і нотація

Проблема термінології є однією з найхворобливіших при обговоренні мови UML. По-перше, автори UML намагалися зробити мову ***незалежною від конкретних мов програмування***, моделей обчислювальності і тому подібного. З цією метою вони частенько вводили нові терміни для визначуваних понять, щоб випадковий збіг зі старим терміном, вже зайнятим у якій-небудь суміжній області програмування, не вводив в оману користувача. По-друге, мова UML порівняно молода, тому сталої
термінологічної традиції доки немає.

Щоб підкреслити, що UML мова графічна, автори називають правила запису (малювання) моделей не синтаксисом, а ***нотацією***. При розробці UML були запропоновані і прийняті розумні рекомендації щодо вибору нотації. Автори виходили з того, що UML використовуватиметься
по-різному: починаючи від не дуже акуратного малювання від руки на листку паперу, друку чорно-білих зображень в книгах і закінчуючи створенням складних діаграм за допомогою комп'ютера. Тому в якості основних графічних елементів були вибрані такі, які було б легко використати в усіх випадках.

Типів елементів нотації п'ять:
1) фігура (shape);
2) лінія (line);
3) значок (icon);
4) текст (text);
5) рамка (frame).

Фігури (shape) в UML використовуються двовимірні (тобто їх можна намалювати на площині) і замкнуті (тобто внутрішня і зовнішня частини). Фігури можуть міняти свої розміри і форму, зберігаючи при цьому свої інтуїтивно відмітні ознаки. Наприклад, серед фігур UML є прямокутники і еліпси. Вони можуть бути зображені багатьма способами: різного розміру, з різним співвідношенням довжин сторін, по-різному орієнтовані відносно меж сторінки і так далі, але в усіх випадках прямокутник відмінний від еліпса і не може бути з ним сплутаний. Всередині фігур можуть розміщуватися інші

елементи нотації: тексти, лінії, значки і навіть інші фігури. Єдина вимога: має бути однозначно зрозуміло, що елемент нотації знаходиться всередині фігури, зокрема, його зображення не повинне перетинати межу фігури.

Лінії (line) в UML завжди приєднуються своїми кінцями до фігур або значків, вони не можуть бути намальовані самі по собі. Форма ліній довільна: це можуть бути прямі, ломані, плавні криві – значення це не має. Товщина ліній також довільна. А ось стиль лінії має значення. І в UML використовується тільки два стилі ліній, які важко сплутати: суцільні і пунктирні лінії. До ліній можна примальовувати різні додаткові елементи: стрілки на кінцях, тексти і так далі. Єдина вимога: повинно бути ясно, що додатковий елемент належить саме до цієї лінії. Лінії можуть перетинатися, і це нічого не означає, але рекомендується уникати таких випадків, оскільки це утрудняє сприйняття.

Значки (icon) в UML схожі на фігури тим, що вони двовимірні, а відрізняються тим, що не мають внутрішності, в яку можна щось розмістити, і, як правило, не міняють свою форму і розміри.

Тексти (text) в UML – це послідовності помітних символів деякого алфавіту. Алфавіт не фіксований – він тільки має бути зрозумілий читачеві моделі. Гарнітура, розмір і колір шрифту не мають значення, а ось зображення шрифту має: в UML розрізняються прямі, *курсивні* і підкреслені тексти.

Рамки (frame) з'явилися в UML 2. Рамка – це окремий випадок фігури, яка використовується виключно як контейнер для інших фігур, ліній, значків і текстів. Рамка має прямокутну форму і, як правило, ярличок в лівому верхньому кутку, в якому вказується тип і ім'я рамки.

Загалом, нотація UML досить вільна: малювати можна як завгодно, аби не виникало непорозумінь. Використання кольорів для заливки фігур і розфарбовування ліній, тіні у значків і фігур, різні шрифти в текстах, нарешті, анімація зображень – усе це, звичайно, корисні речі, оскільки підвищують наочність картинок. Важливо при цьому знати міру, а міра дуже проста і навіть має назву – ***канонічна нотація***. Згідно з нею будь-яка модель може бути описана монохромними рисунками з текстовими поясненнями. При цьому рисунки повинні залишатися зрозумілими після друку на чорно-білому принтері.

В паперовій монохромній книзі зазвичай використовують канонічну нотацію, але в електронній книзі в діаграмах краще використати кольори (для наочності). Тут потрібно дотримуватися певної палітри кольорів, що відображає вибране представлення моделі [3]:

 – моделювання використання (бузковий);

 – моделювання структури (жовтий);

 – моделювання поведінки (блакитний);

 – елементи, використовувані і для моделювання структури, і для моделювання поведінки (зелений);

 – стереотипи, обмеження і інші значимі елементи (сірий).

Модель і її елементи

UML складається всього з трьох «будівельних блоків»:

1. Сутності – це самі елементи моделі.
2. Відношення, які зв'язують сутності. Відношення визначають, як семантично пов'язані дві, або більше за сутності.
3. Діаграми – це представлення моделей UML. Вони показують набори сутностей, які «розповідають» про програмну систему і є способом візуалізації того, що робитиме система (аналітичні діаграми), або як вона робитиме це (проектні діаграми).

Модель UML – це сукупність кінцевої множини конструкцій мови, головні з яких **–** це ***сутності*** і ***відношення*** між ними. Розглядаючи модель UML з найзагальніших позицій, можна сказати, що це граф, в якому вершини і ребра навантажені додатковою інформацією і можуть мати складну внутрішню структуру. Вершини цього графа називаються *сутностями,* а ребра – *відношеннями*.

Сутності

Для зручності огляду сутності в UML можна розділити на чотири групи: 1) структурні; 2) поведінкові; 3) такі, що групують; 4) анотації.

1. Структурні сутності призначені для опису структури. На рис. 2 наведена стандартна нотація в мінімальному варіанті для структурних сутностей.

Об'єкт **(object) (**1) **–** сутність, яка має унікальність і інкапсулює в собі стан і поведінку.

Клас **(class)** (2)**–** опис множини об'єктів із загальними атрибутами, які визначають стан, і операціями, які визначають поведінку.

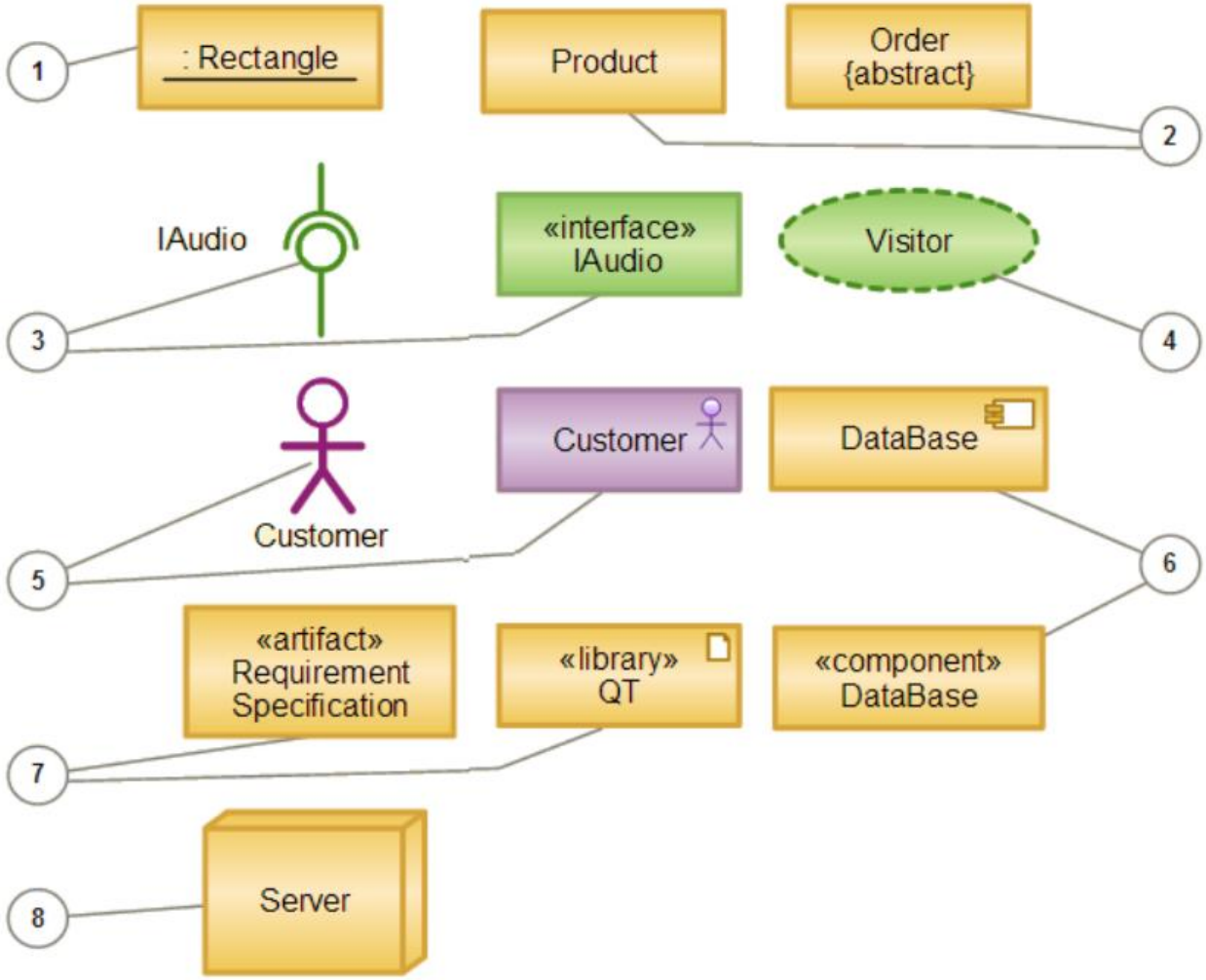


Рисунок 2 – Нотація структурних сутностей

Інтерфейс (interface) (3) – іменована множина операцій, яка визначає набір послуг, які можуть бути запитані споживачем і надані постачальником послуг.

Кооперація* (*collaboration) (4) – сукупність об'єктів, які взаємодіють для досягнення певної мети.

***Дійова особа* (*actor*) (**5) – сутність, яка знаходиться поза модельованою системою і безпосередньо взаємодіє з нею.

Компонент* (*component) (6) – модульна частина системи з чітко певним набором необхідних, надаваємих інтерфейсів.

***Артефакт* (*artifact*)** (7) – елемент інформації, який використовується або породжується в процесі розробки програмного забезпечення. Іншими словами, артефакт – це фізична одиниця реалізації, яка отримується з елементу моделі.

***Вузол* (*node*)** (8) – обчислювальний ресурс, на якому розміщуються і при необхідності виконуються артефакти.

2. Поведінкові сутності (рис. 3) призначені для опису поведінки.

Основних поведінкових сутностей всього дві***:*** *стан* і *дія* (іноді вживається ще і *діяльність*, яка можна розглядається як особливий випадок стану).

***Стан* (*state*)** (1) – період в життєвому циклі об'єкту, знаходячись в якому об'єкт задовольняє деякій умові і здійснює власну діяльність, або чекає настання деякої події.
***Діяльність* (*activity*)** (2) можна вважати частковим випадком стану, який характеризується тривалими (за часом) не атомарними обчисленнями.
***Дія* (*action*)** (3) – примітивне атомарне обчислення.
***Варіант використання* (*use case*)** (4) – множина сценаріїв, об'єднаних за деяким критерієм і виконуваних системою дій, які описують послідовності, які доставляють значущий для деякої дійової особи результат. На рис. 3 наведена стандартна нотація в мінімальному варіанті для поведінкових сутностей.

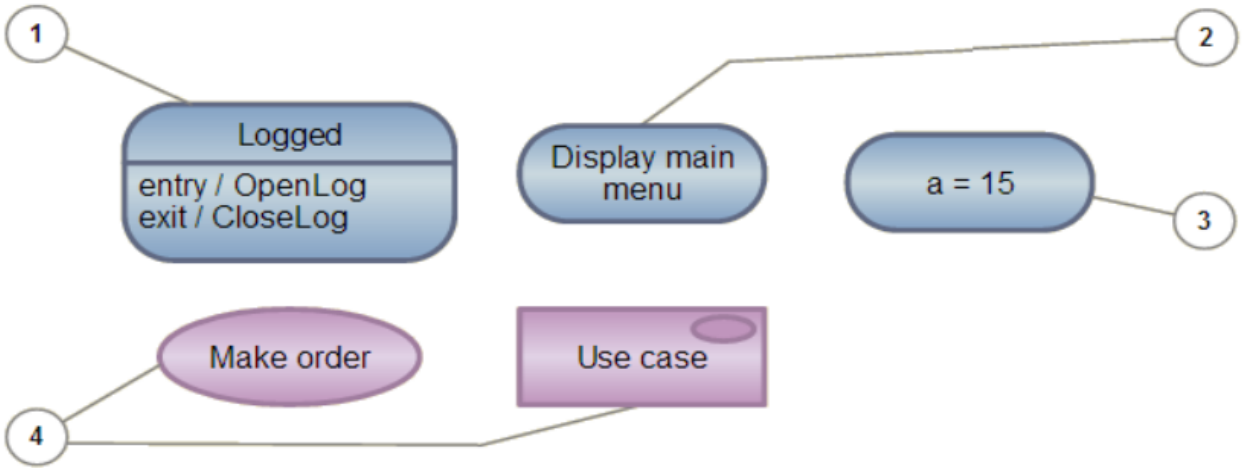


Рисунок 3 – Нотація поведінкових сутностей

3. Групуюча сутність в UML одна – пакет, але універсальна. *Пакет (package)* (1) – група елементів моделі (в тім рахунку пакетів, рис. 4).

4. Сутність анотації теж одна – коментар (рис 4). *Коментар (comment*) (2) – довільний за форматом і змістом опис одного або декількох елементів моделі.

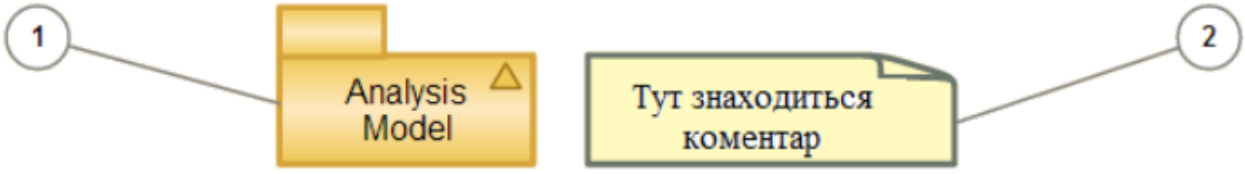


Рисунок 4 – Групуюча нотація і коментар

Відношення

В UML використовуються чотири основні типи відношень:
– залежність (dependency);
– асоціація (association);
– узагальнення (generalization);
– реалізація (realization).

Залежність – це найзагальніший тип відношення між двома сутностями.

Відношення залежності вказує на те, що зміна незалежної сутності якимсь чином впливає на залежну сутність. Графічно відношення залежності зображається у вигляді пунктирної лінії із стрілкою (1) , спрямованою від залежної сутності (2) до незалежної (3) , як показано на рис. 5. Як правило, семантика конкретної залежності уточнюється в моделі за допомогою додаткової інформації. Наприклад, залежність із стереотипом «**use**» означає, що залежна сутність використовує (викликає операцію) незалежну сутність.

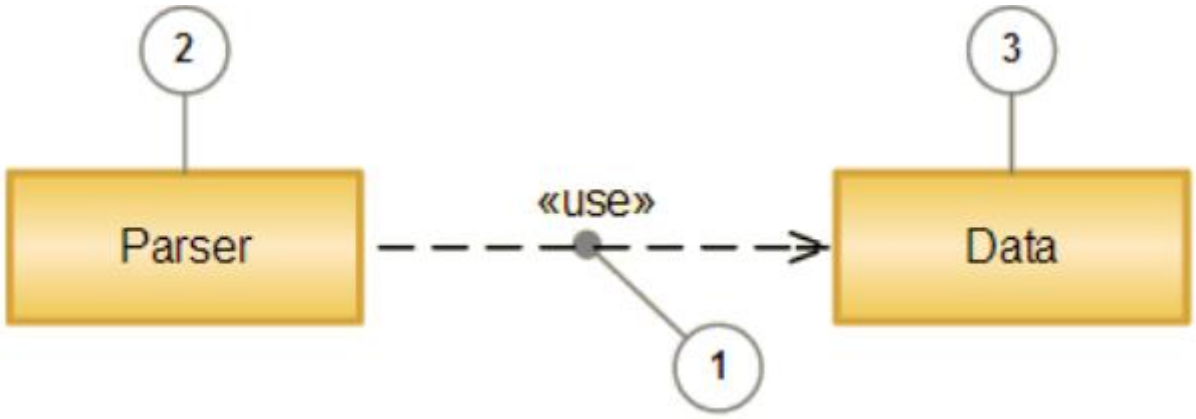


Рисунок 5 – Відношення залежності

Відношення асоціації має місце, якщо одна сутність безпосередньо пов’язана з іншою (чи з іншими – асоціація може бути не лише бінарною). Графічно асоціація зображається у вигляді суцільної лінії (1) з різними доповненнями, що сполучає пов’язані сутності, як показано на рис. 6. На програмному рівні безпосередній зв’язок може бути реалізований по-різному, головне, що асоційовані сутності знають одна про іншу. Наприклад, відношення частина-ціле є частковим випадком асоціації і називається *відношенням агрегації*

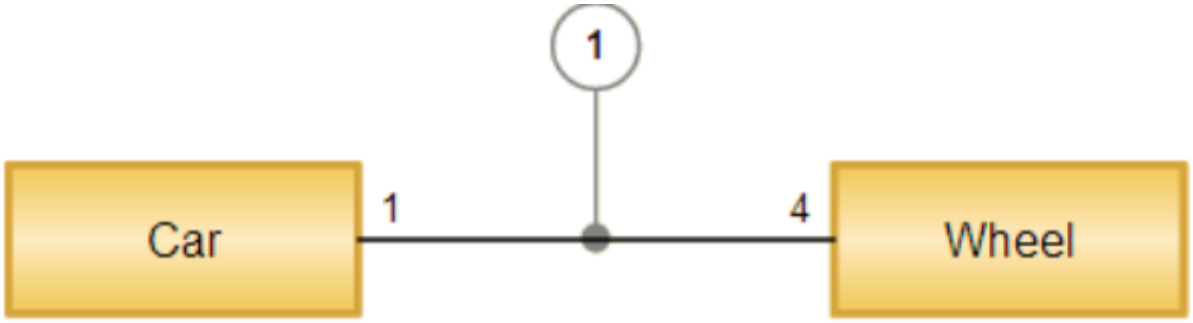


Рисунок 6 – Відношення асоціації

Узагальнення – це відношення між двома сутностями, одна з яких є частковим (спеціалізованим) випадком іншої.

Графічно узагальнення зображається у вигляді лінії з трикутною незафарбованою стрілкою на кінці (1) , спрямованою від частки (2) (підкласу)

до загального (3) (суперкласу), як показано на рис. 7.
Відношення реалізації використовується дещо рідше, ніж попередні три типи відношень, оскільки часто маються на увазі за умовчанням.

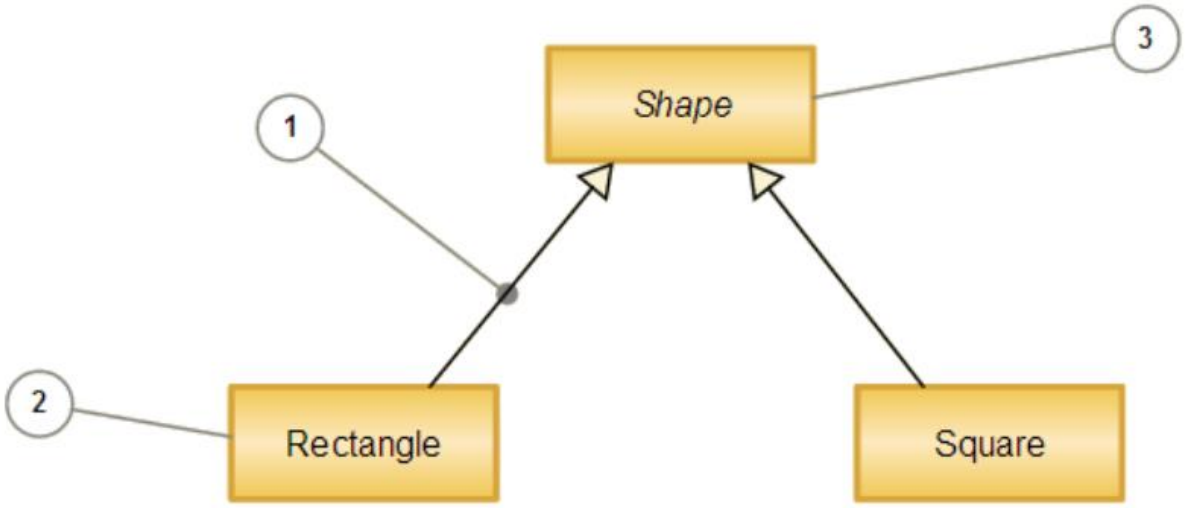


Рисунок 7 – Відношення узагальнення

Відношення реалізації вказує, що одна сутність є реалізацією іншої. Наприклад, клас є реалізацією інтерфейсу. Графічно реалізація зображається у вигляді пунктирної лінії з трикутною незафарбованою стрілкою на кінці (1), спрямованою від реалізовуючої сутності (2) до тієї, яка реалізовується (3) , як показано на рис. 8.
Перераховані типи відношень є основними, різні їх варіації і додаткові відношення (Агрегація ( ) – цільовий елемент є частиною початкового елементу; Композиція ( ) – строга, більше обмежена форма агрегації;
Включення ( ) – початковий елемент містить цільовий елемент) детально буде розглянутий далі.

Діаграми

В попередньому була змалювана множина слів UML (лексем, графічних примітивів, елементів моделювання). Далі перейдемо до синтаксису, а саме до опису того, як із слів конструюються пропозиції.

На перший погляд, усе дуже просто: беруться сутності і, якщо потрібно, вказуються відношення між ними. В результаті виходить модель, тобто граф (з різнорідними вершинами і ребрами), навантажений додатковою інформацією. Але при уважнішому розгляді виявляються проблеми. Розглянемо таку аналогію з природною мовою. Кожна трійка «сутність» – «відношення» – «сутність» в моделі цілком може розглядатися як просте

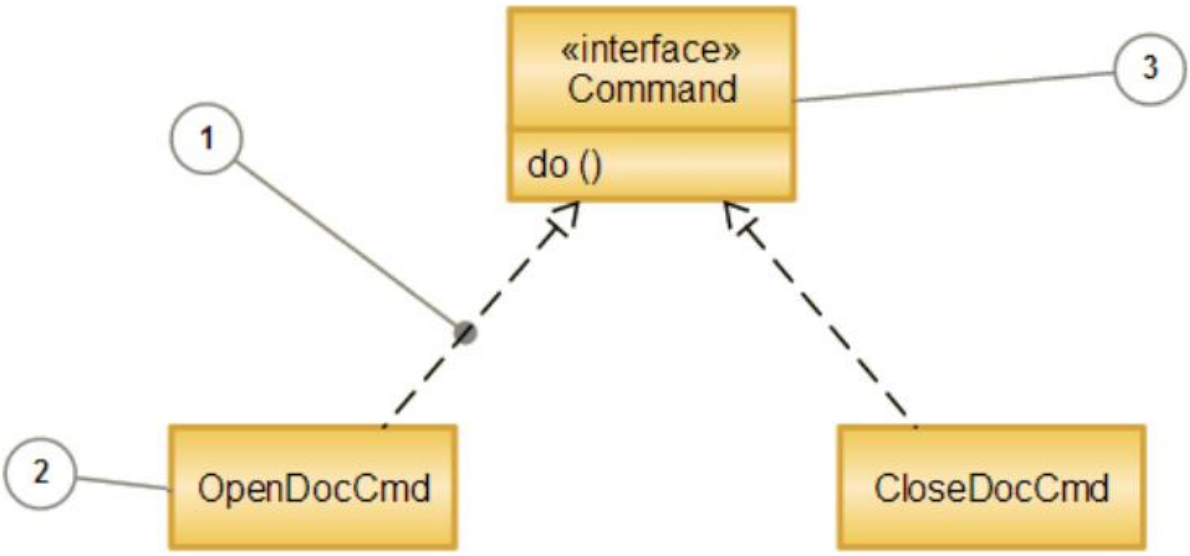


Рисунок 8 – Відношення реалізації

твердження: 2 < 5, ртуть важча за залізо, місяць є супутником Землі (усе це приклади відношень). Поки усе добре, але згадаємо, що в графі (у моделі) ніякої упорядковуючої структури немає: не можна сказати, що ця вершина перша, а ця – друга.

Продовжуючи таку аналогію, виходить, що модель – це множина незв'язаних між собою пропозицій, ніяк не впорядкована. Якщо взяти який-небудь «нормальний» текст, то можна помітити, що окрім структури пропозицій, є маса додаткових структур: пропозиції об'єднані в абзаци, абзаци зібрані в параграфи і розділи, у яких є заголовки, окрім звичайних абзаців і заголовків є примітки і виноски. І усі ці додаткові структури по суті нічого не додають до змісту книги, але серйозно впливають на її читабельність. Текст, в якому немає цих структур, зрозуміти важко. Звідси висновок: окрім сутностей і відношень, в моделі має бути якась структура, яка б допомагала її побудові і розумінню. Діаграми UML і є та основна структура, яка накладається на модель, яка полегшує створення і використання моделі.

Діаграма – це графічне представлення деякої частини графа моделі. Діаграми – це свого роду картини, або представлення моделі. ***Діаграма це не модель!*** Насправді, відмінність між діаграмою і моделлю є дуже важливою для розуміння, оскільки сутність або відношення можуть бути видалені з діаграми, або навіть з усіх діаграм, але як і раніше вони продовжують існувати в моделі. Вони залишатимуться в моделі до тих пір, поки не будуть явно видалені з неї.

Взагалі кажучи, в діаграму можна було б включити будь-які (допустимі) комбінації сутностей і відношень, але свавілля в цьому питанні утруднило б розуміння моделей. Тому автори UML визначили набір рекомендованих до використання типів діаграм, які дістали назву ***канонічних.***

Інструменти моделювання, як правило, забезпечують роботу з усіма канонічними діаграмами, але роблять це досить догматично, не дозволяючи

відійти від канону ні на крок, навіть якщо це треба по суті завдання. Було б зручно, якби набір канонічних діаграм пропонувався за умовчанням, але користувач міг би налагодити, змінити і перевизначити цей набір в разі потреби, приблизно так, як це робиться з шаблонами Microsoft Word. Деякі інструменти, але далеко не всі, підтримують такі можливості.

Слід відмітити, що окрім сутностей і відношень на діаграмі є присутніми інші елементи моделі, які ми також називатимемо ***конструкціями*** мови. Це тексти, які можуть бути написані всередині фігур сутностей або поряд з лініями відношень, рамки діаграм і їх фрагментів, значки, які приєднуються до ліній або поміщаються всередину фігур.

Класифікація діаграм

В UML 1 усього визначено 9 канонічних типів діаграм. Нижче перераховані їх назви (в інших джерелах можуть бути відмінності):

1. Діаграма використання (Use Case diagram).
2. Діаграма класів (Class diagram).
3. Діаграма об’єктів (Object diagram).
4. Діаграма станів (State chart diagram).
5. Діаграма діяльності (Activity diagram).
6. Діаграма послідовності (Sequence diagram).
7. Діаграма кооперації (Collaboration diagram).
8. Діаграма компонентів (Component diagram).
9. Діаграма розміщення (Deployment diagram).

Канонічні діаграми зовсім не утворюють повного ортогонального набору: вони перетинаються як за включеними в них засобами, так і за сферою застосування. Більше того, деякі з них є окремими випадками інших, є просто семантично еквівалентні пари, можна навести приклади допустимих діаграм, для яких складно вказати однозначно, до якого саме з канонічних типів діаграма належить.

Сказане можна проілюструвати умовною класифікацією діаграм, наведеною нижче (рис. 9). Тут на рисунку діаграма використання поміщена окремо, вона не належить ні до діаграм опису структури, ні до діаграм опису поведінки. У більшості джерел діаграми використання відносять до опису поведінки. Окрім відношень узагальнення, на діаграмі класів, додатково показані деякі залежності між окремими UML діаграмами. Ці залежності у кожному конкретному випадку носять різний характер, що відображено за допомогою використання різних стереотипів. Детальніше про стандартні стереотипи залежностей буде сказано пізніше.

В UML 2 внесені значні корективи як в список канонічних діаграм (саме їх число збільшилося до 13), так і в список доступних конструкцій мови, що значно розширило сферу її застосування. Окрім цього, дві діаграми були перейменовані: ***діаграма кооперації*** була перейменована в ***діаграму***

комунікації, а діаграма станів в діаграму автомата.

Список нових діаграм і їх назв наведений нижче.

1. Діаграма внутрішньої структури (Composite Structure diagram).
2. Діаграма пакетів (Package diagram).
3. Діаграма автомата (State machine diagram).
4. Діаграма комунікації (Communication diagram).
5. Оглядова діаграма взаємодії (Interaction Overview diagram).
6. Діаграма синхронізації (Timing diagram).

На рис. 9, 10, 11 наведені діаграми класів, які відбражають взаємозв'язок діаграм в UML 1 и UML 2.

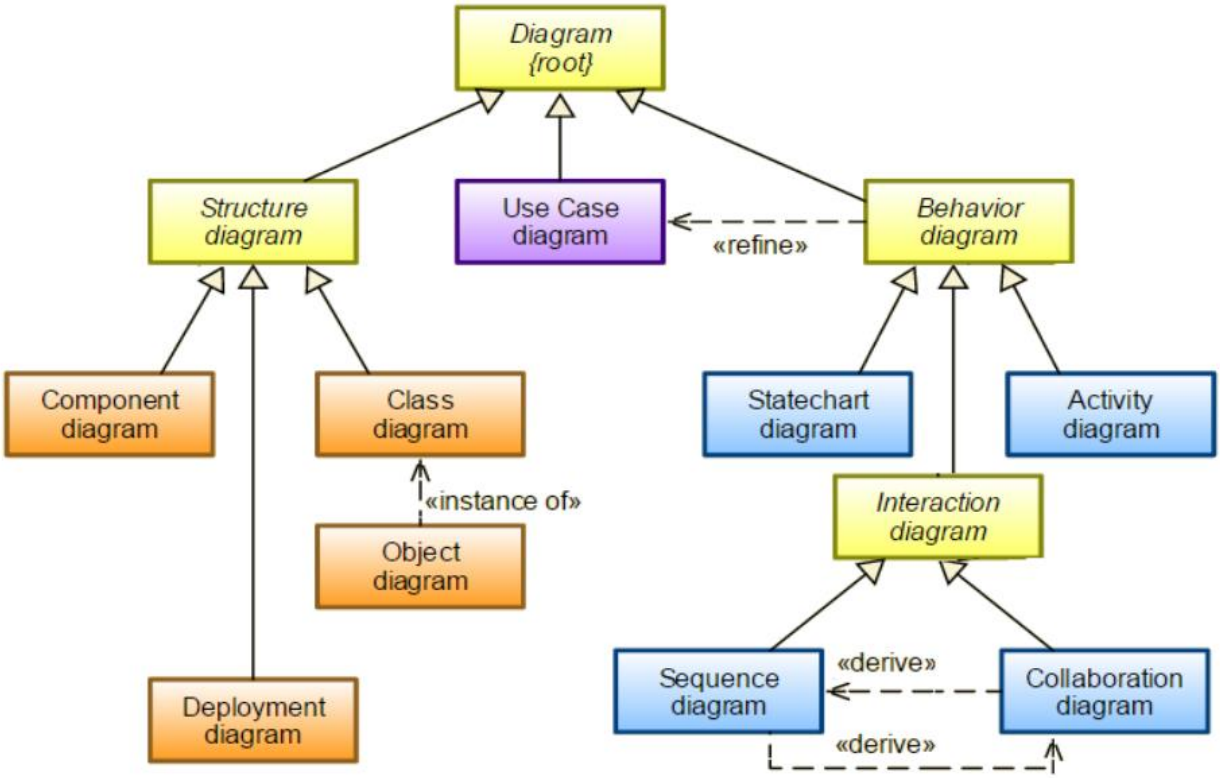


Рисунок 9 – Ієрархія типів діаграм для UML 1

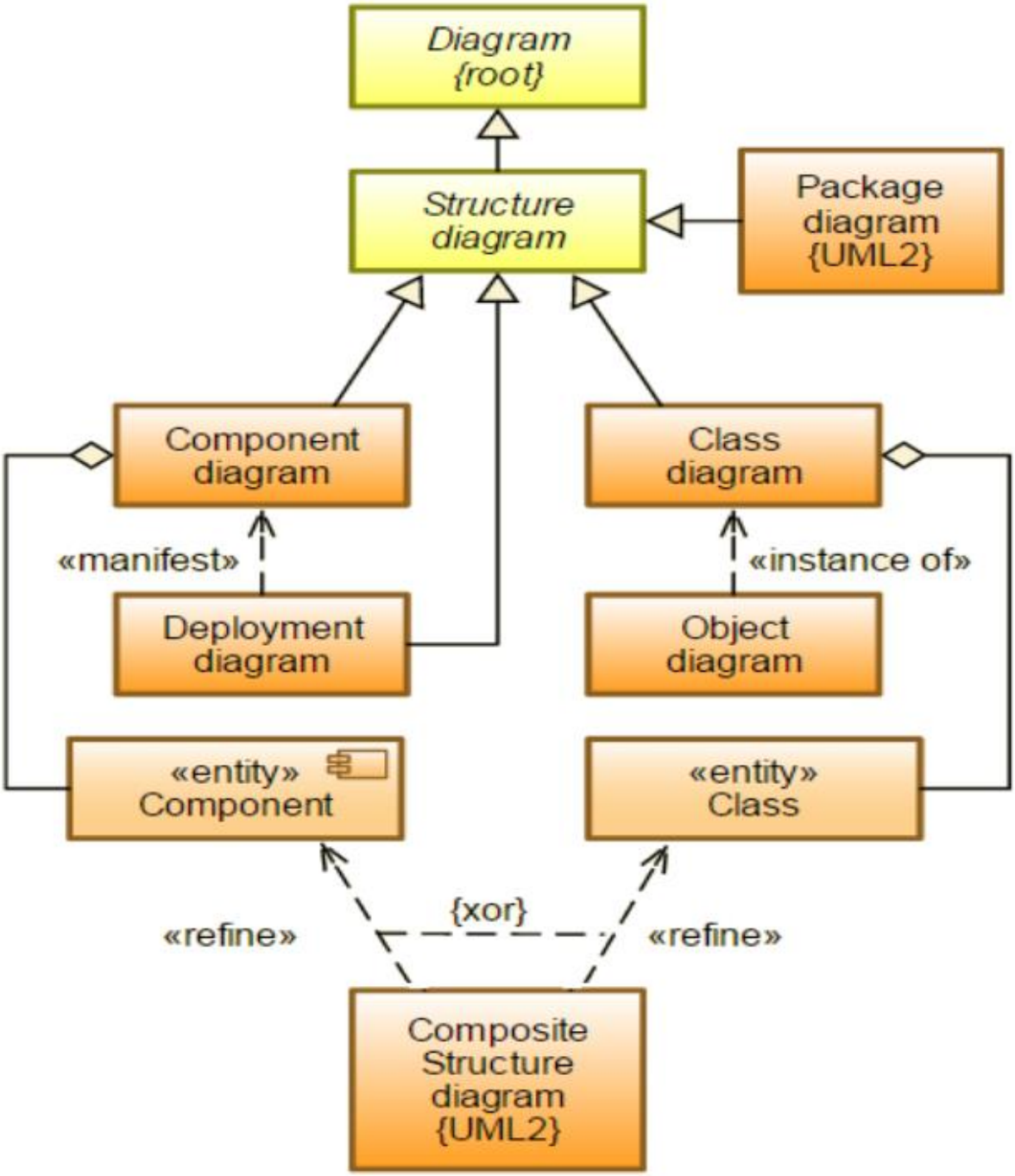


Рисунок 10 – Ієрархія типів діаграм для UML 2

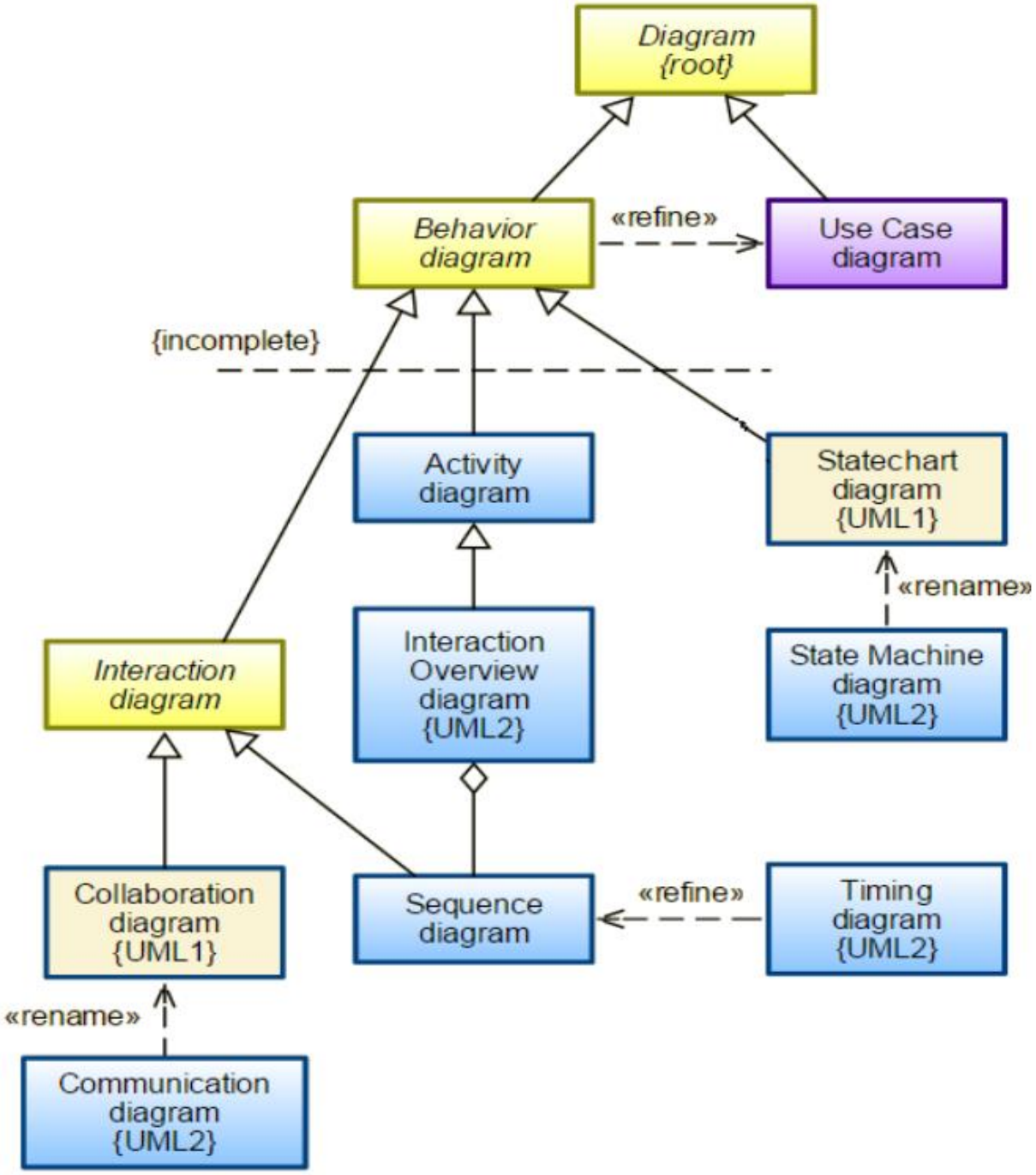


Рисунок 11 – Ієрархія типів діаграм для UML 2 (частина 2)

Основних елементів оформлення два: зовнішня рамка і ярличок з назвою діаграми. Якщо з рамкою усе просто – це прямокутник, який обмежує область в якому повинні знаходитися елементи діаграми, то назва діаграми записується в спеціальному форматі, наведеному на рис. 12.

Рисунок 12 – Нотація для діаграм

Вказана форма ярличка є відносно складною і підтримується не усіма інструментами. Втім, це не обов'язково, оскільки семантика первинна, а нотація вторинна. Далі будемо використовуватимемо як ярличок діаграми прямокутник. Можливі **теги** (типи) для діаграм приведені в таблиці 1. Теги, які пропонуються стандартом, записані в другий стовпець. Проте, як показала практика, пропоновані стандартом правила не завжди зручні і логічно обгрунтовані, тому третій стовпець таблиці містить альтернативні теги.

Таблиця 1 – Типи і теги діаграм

Назва діаграми	Тег (стандартний)	Тег (пропонований)
Діаграма використання	**use case** або **uc**	use case
Діаграма класів	class	class
Діаграма автомата	**state machine** або **stm**	state machine
Діаграма діяльності	**activity** або **act**	activity
Діаграма послідовності	**interaction** або **sd**	sd
Діаграма комунікації	**interaction** або **sd**	comm
Діаграма компонентів	**component** або **cmp**	component
Діаграма розміщення	не визначений	deployment
Діаграма об'єктів	не визначений	object
Діаграма внутрішньої структури	class	**class** або **component**
Оглядова діаграма взаємодії	**interaction** або **sd**	interaction
Діаграма синхронізації	**interaction** або **sd**	timing
Діаграма пакетів	**package** або **pkg**	package

Література

1. Гради Буч, Джеймс Рамбо, Ивар Якобсон Введение в UML от создателей языка. Пер: Н. Мухин – М.: ДМК Пресс, 2011г. – 496с.

2. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Пер. с англ. – СПб.: Символ-Плюс, 2011. – 192 с.

5.2 *Діаграми. Представлення. Процес моделювання.*

МОВА UML

Загальні діаграми

Усі діаграми UML можна умовно розбити на дві групи, перша з яких – *загальні діаграми*. Загальні діаграми практично не залежать від предмету моделювання і можуть застосовуватися у будь-якому програмному проекті.

Діаграма використання

***Діаграма використання* (діаграма прецедентів, use case diagram) –** це найзагальніше представлення функціонального призначення системи. Діаграма використання покликана відповісти на головне питання моделювання: що робить система у зовнішньому світі?

На діаграмі використання (рис. 1) застосовуються два типи основних сутностей: варіанти використання (1) і дійові особи (актори – користувачі, пристрої) (2) , між якими встановлюються такі основні типи відношень:

1. Асоціація між дійовою особою і варіантом використання (3).
2. Узагальнення між дійовими особами (4).
3. Узагальнення між варіантами використання (5).
4. Залежності (різних типів) між варіантами використання (6).

На діаграмі використання, як і на будь-якій іншій, можуть бути присутніми коментарі (7). Більше того, це настійно рекомендується робити для поліпшення читаності діаграм. Детальний опис варіацій діаграм використання розглянемо пізніше, а на рисунку 2 показані тільки основні елементи нотації.

Діаграма класів

***Діаграма класів* (class diagram) –** основний спосіб опису структури системи. Це не дивно, оскільки UML в першу чергу об’єктно-орієнтована мова, і класи є основним (якщо не єдиним) «будівельним матеріалом».

На діаграмі класів (рис.2) застосовується один основний тип сутностей: класи (1) (включаючи численні окремі випадки класів: інтерфейси, примітивні типи, класи-асоціації і багато інших), між якими встановлюються наступні основні типи відношень:

– асоціація між класами (2) (з множиною додаткових подробиць);
– узагальнення між класами (3);
– залежності (різних типів) між класами (4) і між класами та інтерфейсами.

Аналогічно опис варіацій діаграм класів розглянемо пізніше, а на рисунку 2 показані тільки основні елементи нотації.

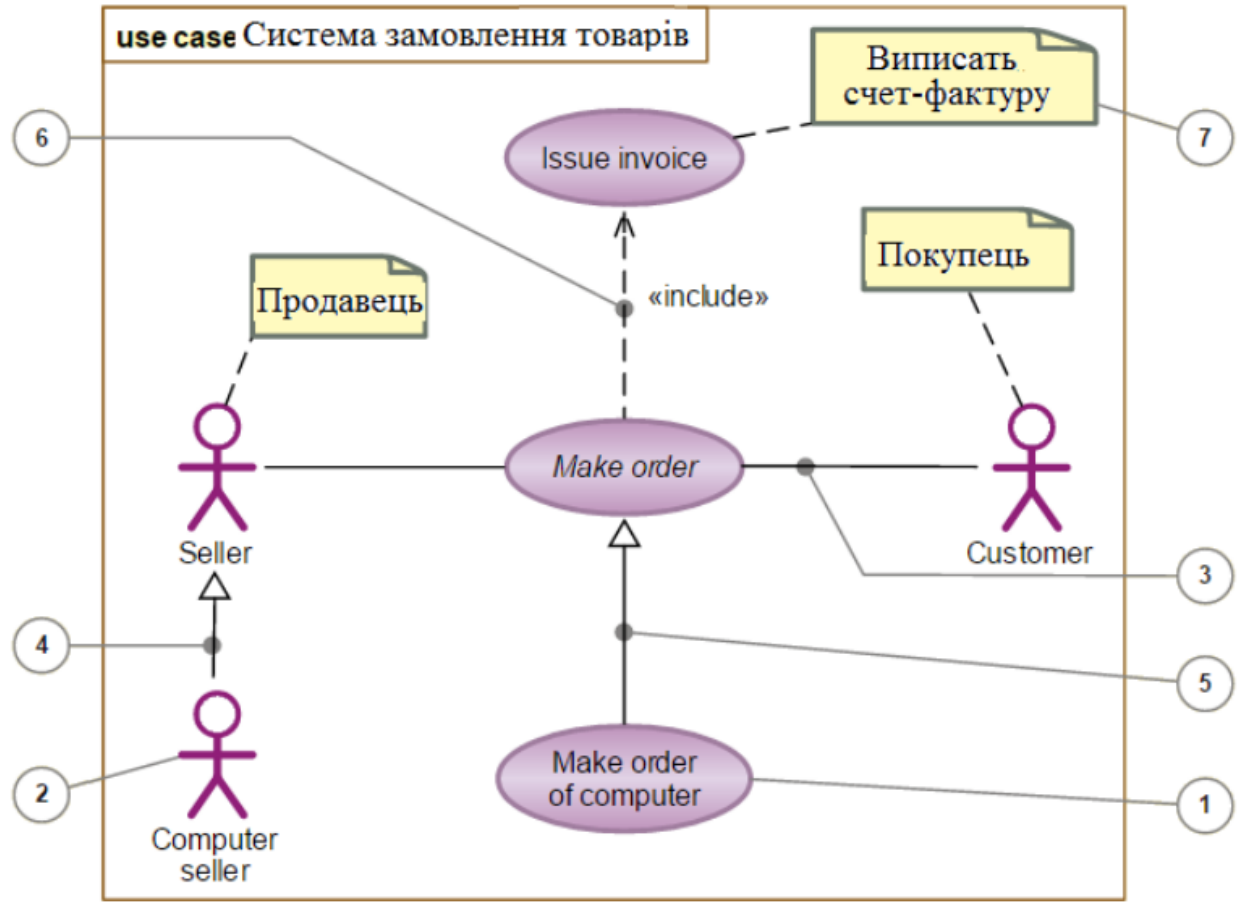


Рисунок 1 – Нотація діаграми використання (прецедентів)

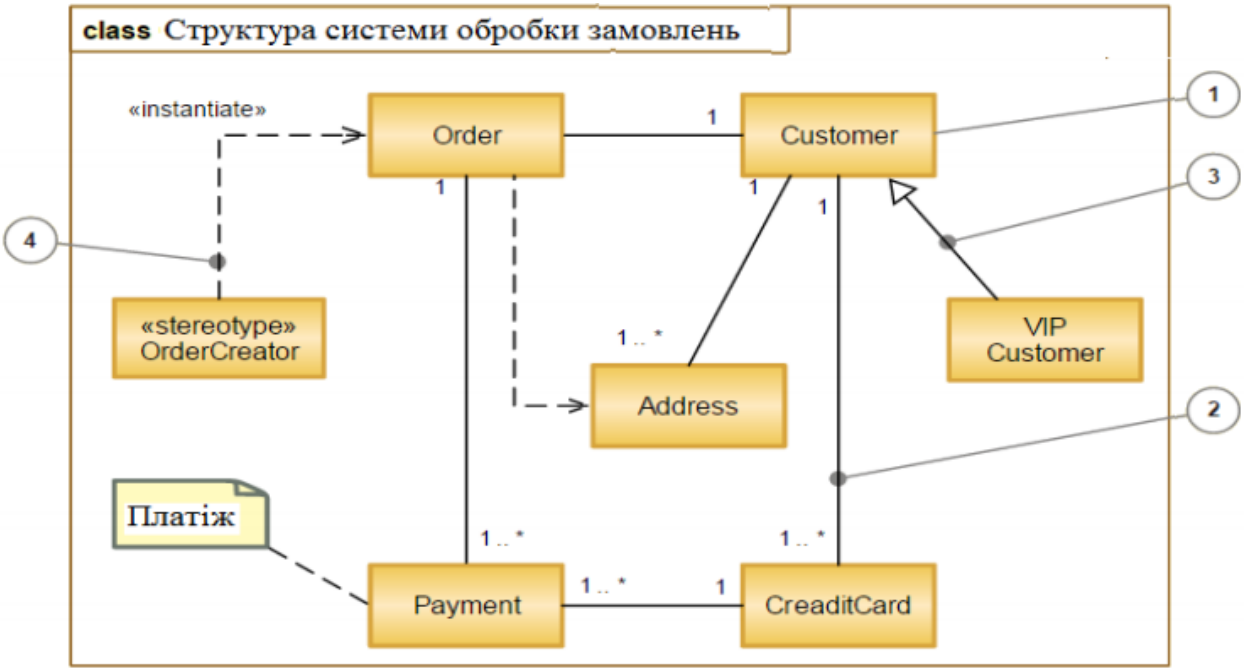


Рисунок 2 – Нотація діаграми класів

Діаграма автомата

Діаграма автомата (state machine diagram, *діаграма станів*) – це один із способів детального опису поведінки в UML на основі явного виділення станів і опису переходів між станами. По суті, діаграмою автомата, як це витікає з назви, є граф переходів станів, навантажений множиною додаткових деталей і подробиць. На діаграмі автомата (рис. 3) застосовують один основний тип сутностей – стан (1), і один тип відношень – переходи (2), але і для тих і для інших визначено багато різновидів, спеціальних випадків і додаткових позначень. Детальний опис усіх варіацій діаграм автомата розглянемо пізніше, а на рисунку (3) показані тільки основні елементи нотації.

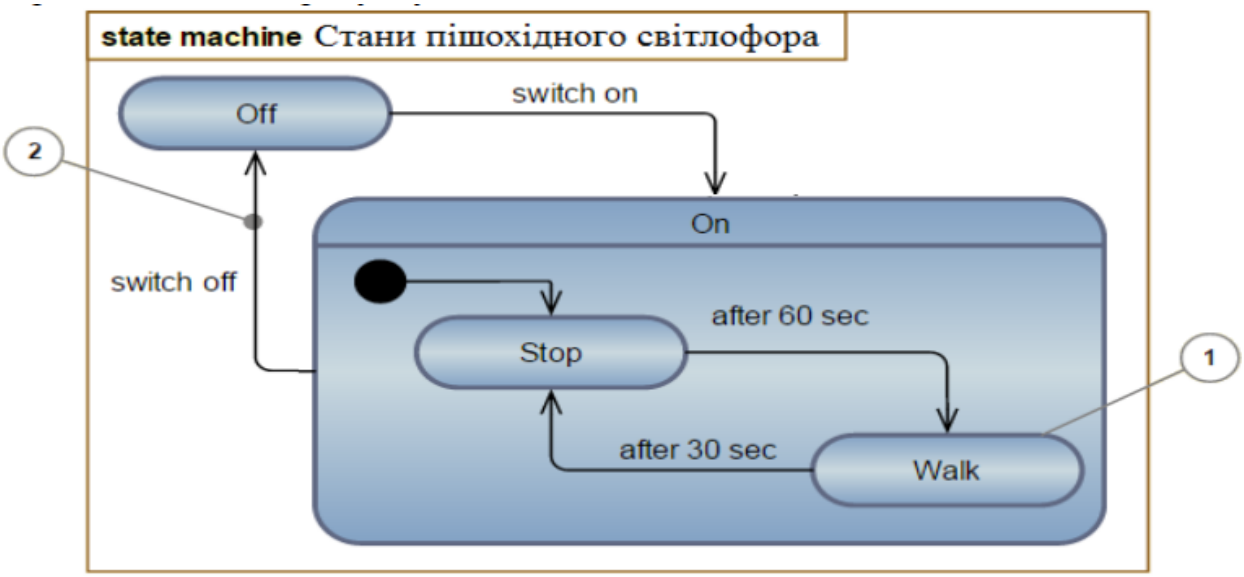


Рисунок 3 – Нотація діаграми автомата

Діаграма діяльності

Діаграма діяльності (activity diagram) – спосіб опису поведінки на основі вказівки потоків управління і потоків даних.

Діаграма діяльності – ще один спосіб опису поведінки, який візуально нагадує стару добру блок-схему алгоритму. Проте за рахунок модернізованих позначень, погоджених з об’єктно-орієнтованим підходом, діаграма діяльності UML є потужним засобом для опису поведінки системи.

На діаграмі діяльності (рис. 4) застосовують один основний тип сутностей – дія (1) , і один тип відношень – переходи (2) (передачі керування і даних). Також використовуються такі конструкції як розвилки, злиття, з’єднання, галуження (3) , які схожі на сутності, але такими насправді не є, а є графічним способом зображення деяких окремих випадків багатомісних відношень. Семантика елементів діаграм діяльності буде розібрана далі. Основні елементи нотації діаграмі діяльності показані на рис. 4.

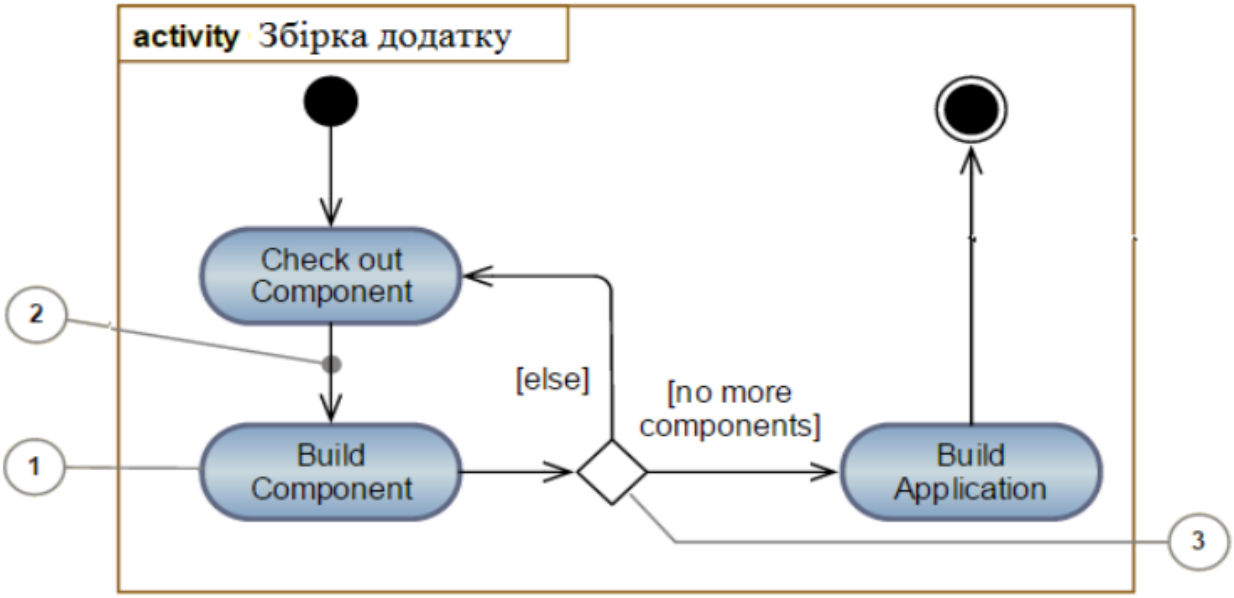


Рисунок 4 – Нотація діаграми діяльності

Діаграма послідовності

Діаграма послідовності (sequence diagram) – це спосіб опису поведінки системи на основі вказівки послідовності передаваних повідомлень. Фактично, діаграма послідовності – це запис протоколу конкретного сеансу роботи системи (чи фрагмента такого протоколу). У об'єктно-орієнтованому програмуванні найістотнішою під час виконання є пересилка повідомлень між взаємодіючими об'єктами. Саме послідовність посилок повідомлень відображається на цій діаграмі, звідси і назва.

На діаграмі послідовності (рис. 5) застосовують один основний тип сутностей – екземпляри взаємодіючих класифікаторів (1) (в основному класів, компонентів і дійових осіб), і один тип відношень – зв'язки (2) , по яких відбувається обмін повідомленнями (3) . Передбачені декілька способів посилки повідомлень, які в графічній нотації розрізняються видом стрілки, яка відповідає відношенню.

Важливим аспектом діаграми послідовності є явне відображення плину часу. На відміну від інших типів діаграм, окрім хіба що діаграм синхронізації, на діаграмі послідовності має значення не лише наявність графічних зв'язків між елементами, але і взаємне розташування елементів на діаграмі. А саме, вважається, що є (невидима) вісь часу, за умовчанням спрямована зверху вниз, і те повідомлення, яке відправлене пізніше, намальоване нижче. Вісь часу може бути спрямована горизонтально, в цьому випадку вважається, що час тече зліва направо.

На рис. 5 показані основні елементи нотації, вживані на діаграмі послідовності. Для позначення самих взаємодіючих об'єктів застосовується стандартна нотація – прямокутник з ім'ям екземпляру класифікатора.

Пунктирна лінія, яка виходить з нього, називається лінією життя (lifeline) (4). Це не позначення відношення в моделі, а графічний коментар, покликаний направити погляд читача діаграми в правильному напрямі. Фігури у вигляді вузьких смужок, накладених на лінію життя, також не є зображеннями модельованих сутностей. Це графічний коментар, який показує відрізки часу, протягом яких об'єкт володіє потоком керування (execution occurrence) (5) або іншими словами має місце активація (activation) об'єкту. Складені кроки взаємодії (combined fragment) (6) дозволяють на діаграмі послідовності відбивати і алгоритмічні аспекти протоколу взаємодії. Інші деталі нотації діаграми послідовностей будуть розглянуті далі.

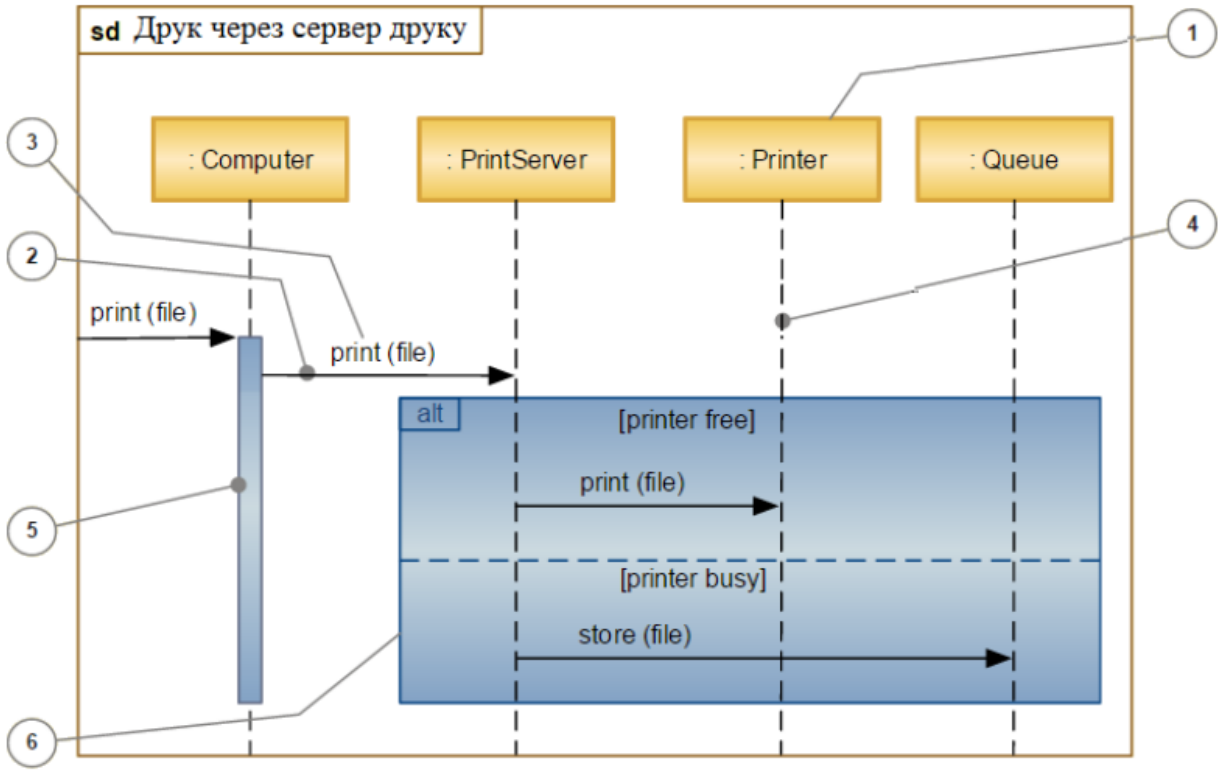


Рисунок 5 – Нотація діаграми послідовності

Діаграма комунікації

Діаграма комунікації (communication diagram) – спосіб опису поведінки, семантично еквівалентний діаграмі послідовності. Фактично, цей такий же опис послідовності обміну повідомленнями взаємодіючих екземплярів класифікаторів, тільки виражений іншими графічними засобами. Таким чином, на діаграмі комунікації (рис. 6) також як і на діаграмі послідовності застосовують один основний тип сутностей – екземпляри взаємодіючих класифікаторів (1) і один тип відношень – зв'язку (2). Проте тут акцент робиться не на часі, а на структурі зв'язків між конкретними екземплярами.

Для позначення самих взаємодіючих об'єктів застосовується стандартна нотація – прямокутник з ім'ям екземпляру класифікатора. Взаємне положення елементів на діаграмі комунікації (кооперації) не має значення – важливі тільки

зв’язки (найчастіше екземпляри асоціацій), уздовж яких передаються повідомлення (3) . Для відображення впорядкованості повідомлень в часі застосовується ієрархічна десяткова нумерація.
Порівняйте цей рисунок з рис. 5 *Нотація діаграми послідовності* (на них зображена одна і та ж поведінка), і вам усе стане зрозуміло. Інші деталі нотації діаграми комунікації розглянемо далі.

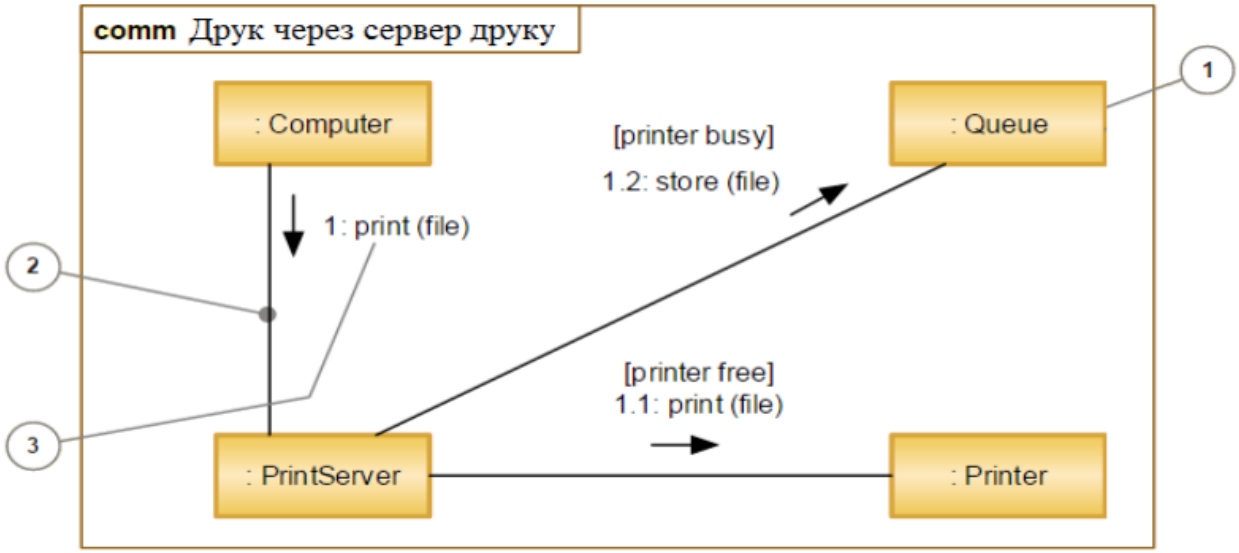


Рисунок 6 – Нотація діаграми комунікації

Діаграма компонентів

Діаграма компонентів (component diagram) – показує взаємозв’язки між модулями (логічними або фізичними), з яких складається модельована система.

Основний тип сутностей на діаграмі компонентів (рис. 7) – це самі компоненти (1) , а також інтерфейси (2) , за допомогою яких вказується взаємозв’язок між компонентами. На діаграмі компонентів застосовуються такі відношення:

– реалізації між компонентами і інтерфейсами (компонент реалізує інтерфейс);

– залежності між компонентами і інтерфейсами (компонент використовує інтерфейс) (3) .

Детальний опис усіх варіацій діаграм компонентів наведений далі, а на рисунку 7 показані тільки основні елементи нотації.

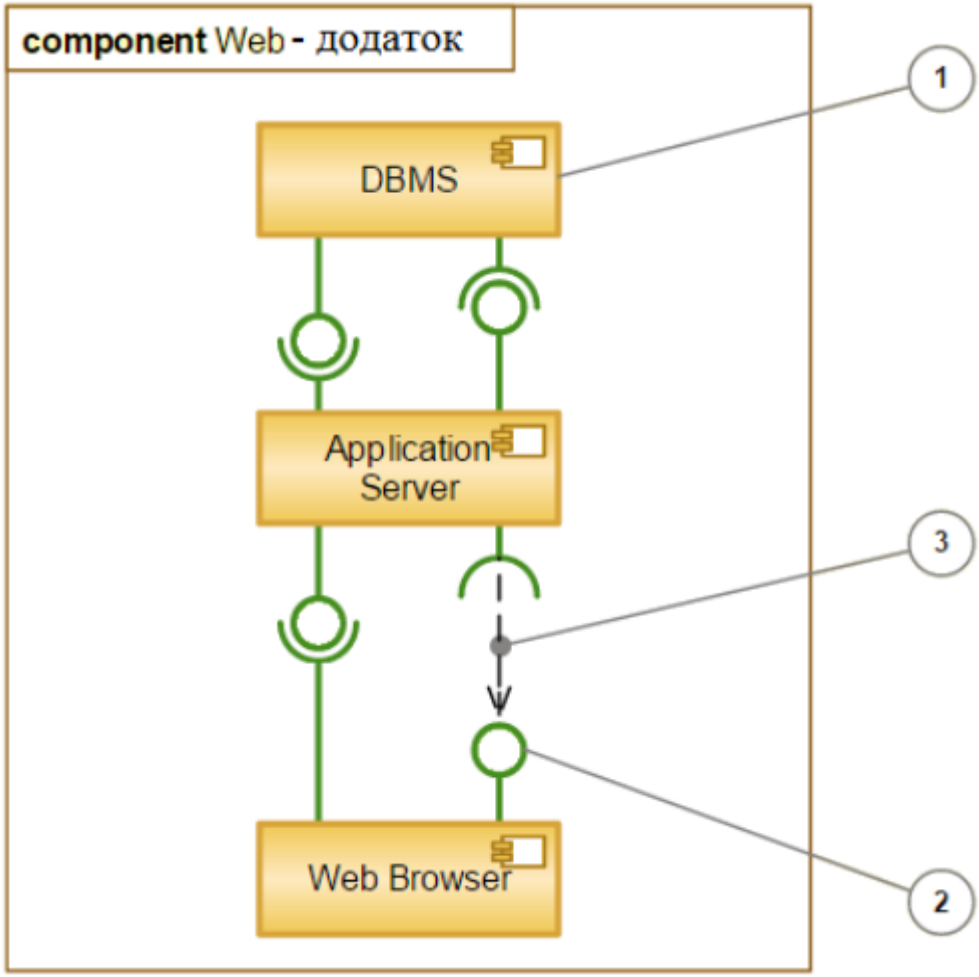


Рисунок 7 – Нотація діаграми компонентів

Діаграма розміщення

Діаграма розміщення (deployment diagram) разом з відображенням складу і зв'язків елементів системи показує, як вони фізично розміщені на обчислювальних ресурсах під час виконання.

Таким чином, на діаграмі розміщення (рис. 8), в порівнянні з діаграмою компонентів, додається два типи сутностей: артефакт (1) , який є реалізацією компонента (2) і вузол (3) (може бути як класифікатор, який описує тип вузла, так і конкретний екземпляр), а також відношення асоціації між вузлами (4) , що показує, що вузли фізично пов'язані під час виконання.
На рис. 8 показані основні елементи нотації, вживані на діаграмі розміщення. Для того щоб показати, що одна сутність є частиною іншої, застосовується або відношення залежності «deploy» (5) , або фігура однієї сутності розміщується всередину фігури іншої сутності (6). Детальний опис діаграми приведений в наступних викладенях.

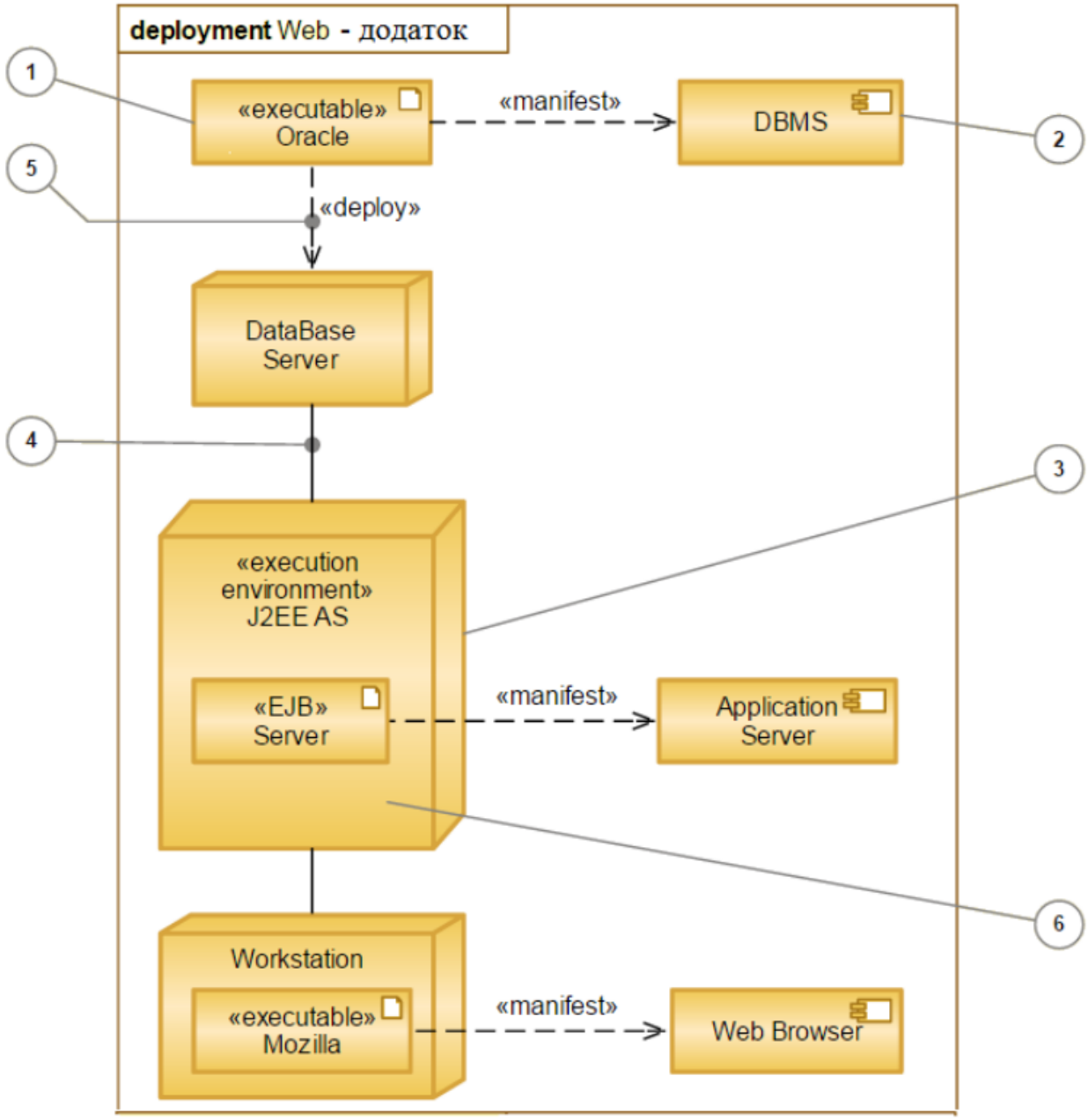


Рисунок 8 – Нотація діаграми розміщення

Спеціальні діаграми

Спеціальні діаграми характеризуються тим, що найчастіше служать для доповнення якої-небудь загальної діаграми, наприклад, є її частковим випадком або ж просто грають допоміжну роль, уточнюючи деякі деталі.

Діаграма об'єктів

Діаграма об'єктів (object diagram) – є екземпляром діаграми класів. На діаграмі об'єктів (рис. 9) застосовують один основний тип сутностей: об'єкти (1) (екземпляри класів), між якими вказуються конкретні зв'язки (2) (найчастіше екземпляри асоціацій).

Діаграми об'єктів мають допоміжний характер – по суті це приклади (можна сказати, дампи пам'яті), які показують, які є об'єкти і зв'язки між ними в деякий конкретний момент функціонування системи.

Основні елементи нотації, вживані на діаграмі об’єктів, показані нижче, а детальний опис приведений в наступних викладенях.

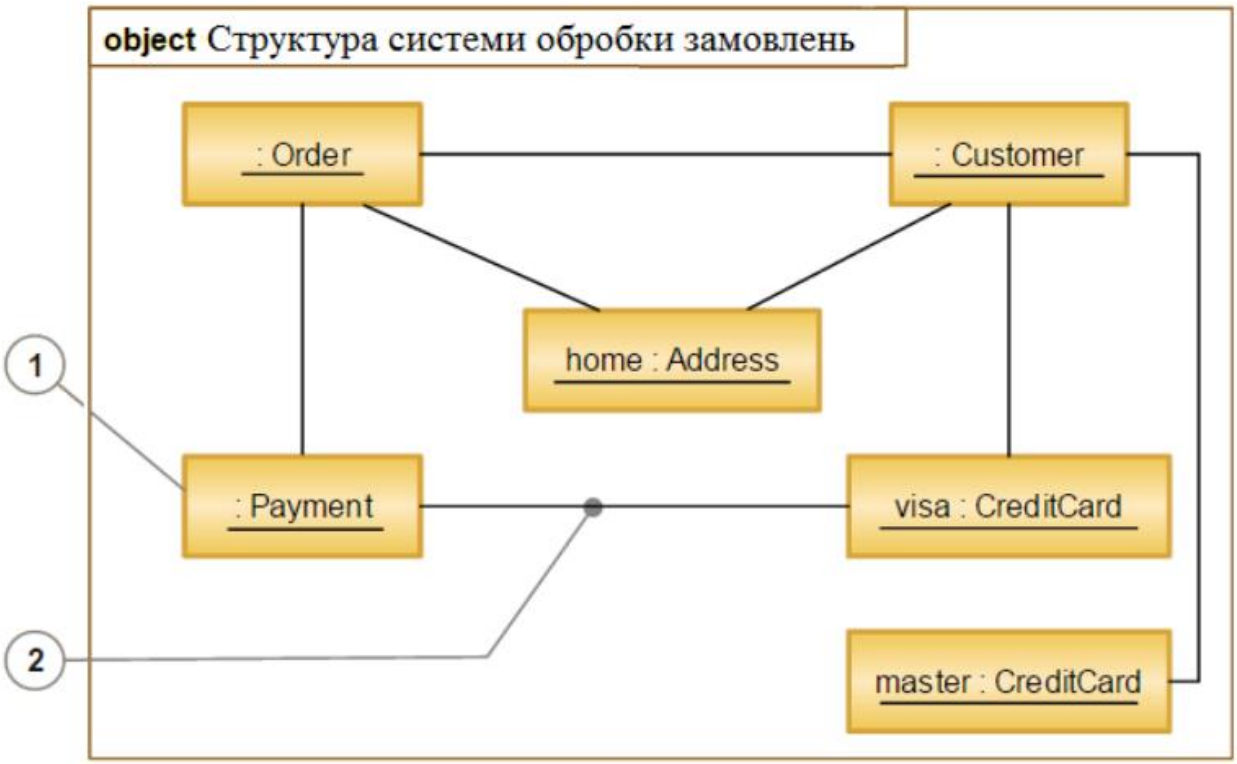


Рисунок 9 – Нотація діаграми об’єктів

Діаграма внутрішньої структури

Діаграма внутрішньої структури (composite structure diagram) використовується для детальнішого представлення структурних класифікаторів, передусім класів і компонентів.

Структурний класифікатор (рис. 10) зображається у вигляді прямокутника (1), в верхній частині якого міститься ім’я класифікатора (2). Усередині знаходяться *частини (parts)* (3) . Частин може бути декілька. Кожна частина є екземпляром деякого іншого класифікатору. Частини можуть взаємодіяти одна з одною. Це позначається за допомогою *з’єднувачів (connectors)* (4) різних видів. Місце на зовнішній межі частини, до якої приєднується з’єднувач, називається *портом (port)* (5) . Порти розташовуються також на зовнішній межі структурного класифікатору (6) , забезпечуючи йому зв’язок із зовнішнім світом.

Детальний опис усіх варіацій діаграм внутрішньої структури наведений в наступних викладенях, а на рисунку 10 показані тільки основні елементи нотації.

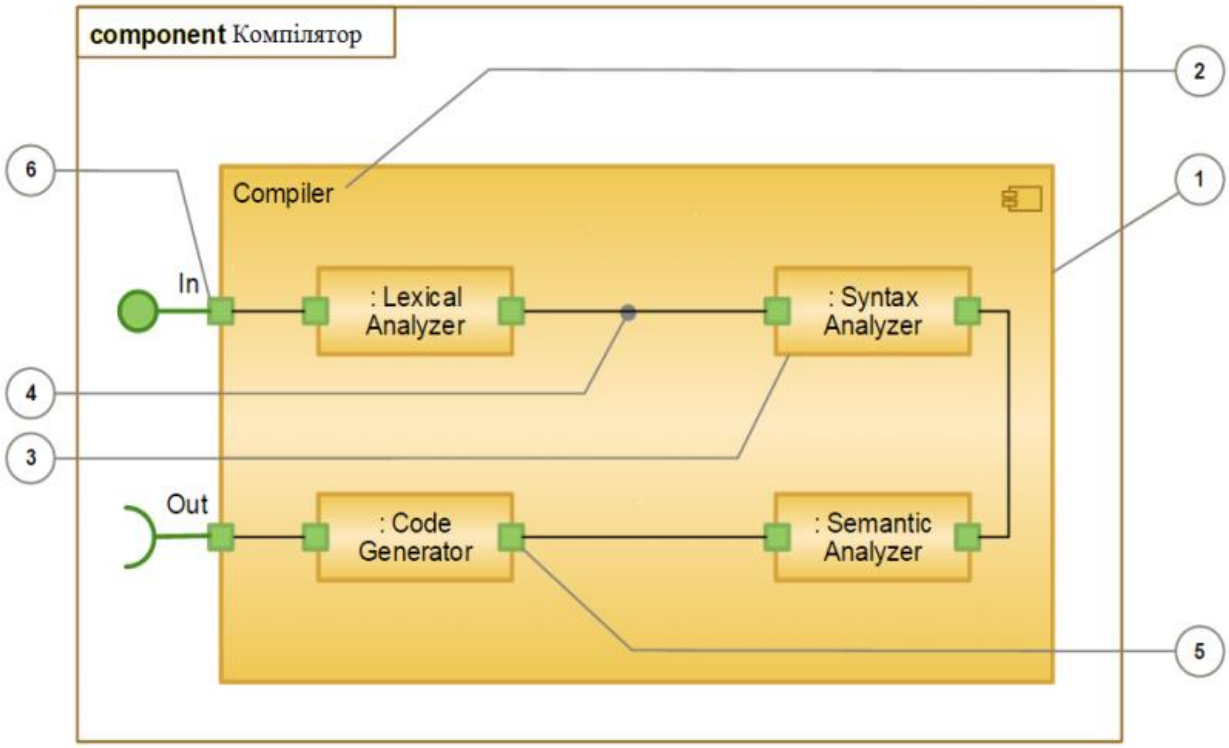


Рисунок 10 – Нотація діаграми внутрішньої структури

Оглядова діаграма взаємодії

Оглядова діаграма взаємодії (interaction overview diagram) є різновидом діаграми діяльності з розширеним синтаксисом. Елементами оглядової діаграми взаємодії можуть виступати *посилання на взаємодії* (interaction use) (1) , які визначаються діаграмами послідовності (рис. 11)**.** Основні елементи нотації показані на рис. 11 , а детальний опис наведений в викладені далі.

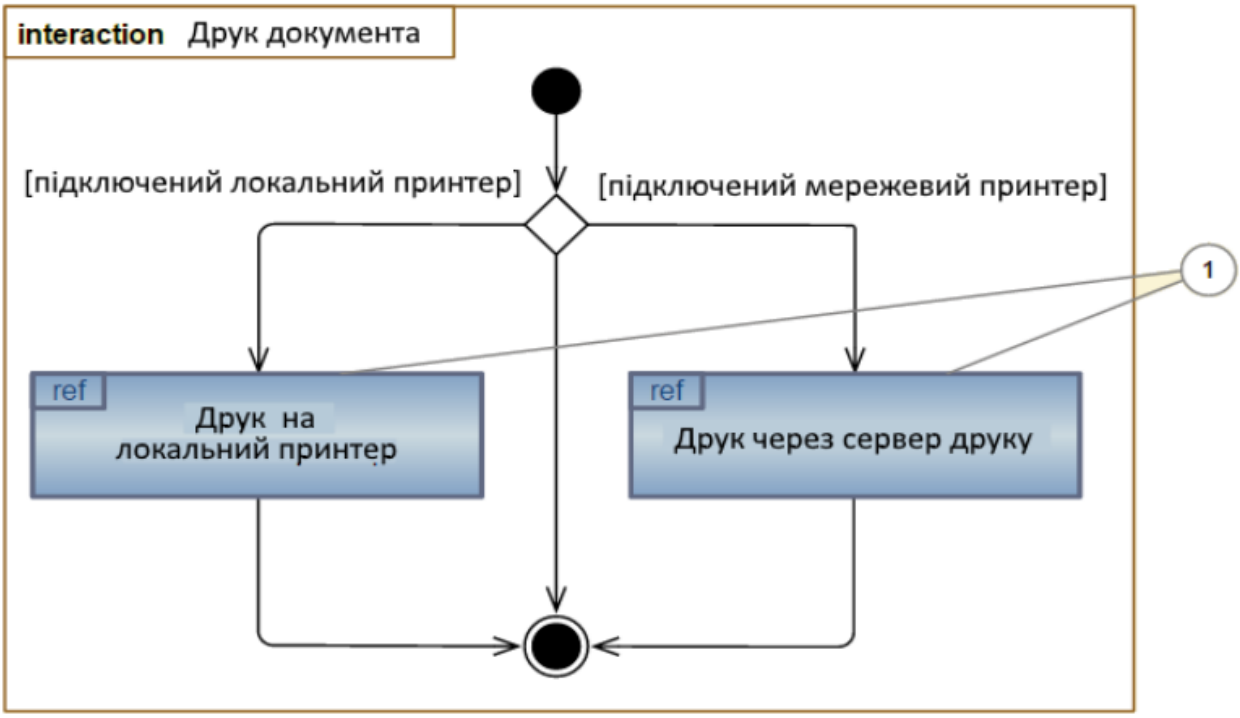


Рисунок 11 – Нотація оглядової діаграми взаємодії

Діаграма синхронізації

Діаграма синхронізації (timing diagram) є особливою формою діаграми послідовності, на якій особлива увага приділяється зміні станів (1) різних екземплярів класифікаторів і їх тимчасової синхронізації (2) (рис. 12)**.** Основні елементи нотації показані на рис. 12, а детальний опис наведений далі.

Рисунок 12 – Нотація діаграми синхронізації

Діаграма пакетів

Діаграма пакетів (package diagram) – засіб групування елементів моделі. Діаграма пакетів (рис. 13) – єдиний засіб, який дозволяє керувати складністю самої моделі. Основні елементи нотації – пакети (1) і залежності з різними стереотипами (2) , що вживаються на діаграмі, показані на рис. 13.

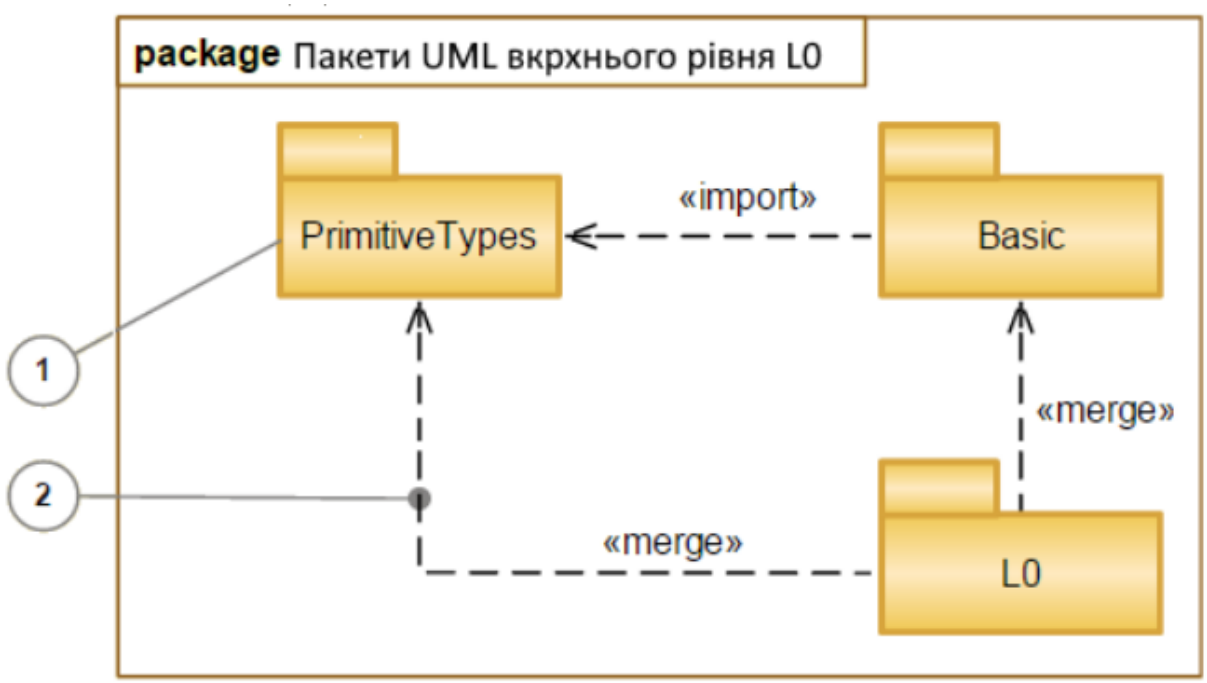


Рисунок 13 – Нотація діаграми пакетів

Моделі і їх представлення

Як показує практичний досвід, починаючи з деякого порогу складності, не вдається описати з єдиної точки зору усі без виключення аспекти модельованої системи. Дійсно, в моделі треба відобразити багато речей: інтерфейси для взаємодії із зовнішнім світом, внутрішню логічну структуру програми, структуру зберігаємих даних, алгоритми функціонування і багато що інше. Звідси слідує висновок: *моделювати складну систему слід з декількох різних точок зору*, кожного разу зважаючи на один аспект модельованої системи і абстрагуючись від інших. Ця теза є одним із засадничих принципів UML, може бути найважливішим принципом, який зумовив практичний успіх UML.

Ідея полягає в тому, що абстрактний граф моделі, яка складається з багатьох різнотипних сутностей і відношень, не підлягає конструюванню або вивченню в цілому. Кожного разу для візуалізації, зміни або інших маніпуляцій з цього загального графу вичленюють тільки сутності і відношення, релевантні для певного аспекту модельованої системи, а усі інші ігноруються. Такий вид з певної точки зору проекції моделі називають ***представленням*** *(view)* – засобом логічної структуризації моделі**.**

Призначення і рівні моделей

Моделі UML можуть створюватися і використовуватися з різними цілями. Іноді буває навіть так, що автор створював модель з однією певною метою, а використовується вона іншими людьми абсолютно несподіваним для автора чином. Таким чином, призначення моделі не є чимось постійним і важко змінюваним. Трансформації призначення моделей цілком можливі, але

практика моделювання підказує, що моделі дають більший ефект, якщо при моделюванні брати до уваги призначення моделі.

Призначення моделей може бути різним, усі випадки описати неможливо. Зазвичай пропонують таку класифікацію призначень моделі, засновану на відповіді на просте питання «а що з цією моделлю трапиться потім?», яке автор моделі повинен задавати собі під час моделювання. Розглядаємо три типові варіанти відповіді.

1. Концептуальна модель (conceptual model). На діаграми такої моделі дивитимуться, їх обмірковуватимуть, але з самою моделлю нічого робити не будуть. Це не означає, що модель не потрібна – це означає, що модель використовується тільки для керування розумовим процесом, для розуміння. Тому такі моделі називають концептуальними (також застосовуються терміни ***модель аналізу*** (analysis model) або ***аналітична модель***). Такий тип використання моделей один з найважливіших, оскільки так використовуються моделі, які виходять в результаті аналізу предметної області.

Концептуальні моделі досить стабільні: якщо не міняється предметна область, то немає потреби міняти і модель. Головна вимога до моделі предметної області: ***концептуальна цілісність***.

2. Модель проектування. Модель проектування є високорівневим (на рівні підсистем) і низькорівневим (на рівні класів, якщо йдеться про використання ООП) описом програмної системи. Модель проектування призначена для того щоб, керуючись нею, розробити програмний код додатку.

Як правило, архітектура (високорівневий опис) і код розроблюються ітеративно, і розробникам в процесі розробки необхідно модифікувати модель проектування, щоб вона відповідала рішенням, які приймалися. Головна вимога до моделі проектування – ***зрозумілість*** (usability). Дійсно, щоб вести розробку, розробники повинні повністю розуміти модель.

3. Модель реалізації (implementation model). Модель реалізації призначена для автоматичного перетворення в істотно інший вид, наприклад, в здійснимий код. Таке призначення вимагає вказівки необхідних деталей реалізації в моделі, оскільки «від себе» комп’ютер їх додати не зможе. Головна вимога до моделей реалізації: ***повнота*** (completeness).

Ще раз підкреслимо, що між моделями різного призначення немає непереборного бар’єру, але відмінності все-таки є. На рис. 14 наведено приклад, де розглянуті відношення між авторами «Author» і книгою «Book».

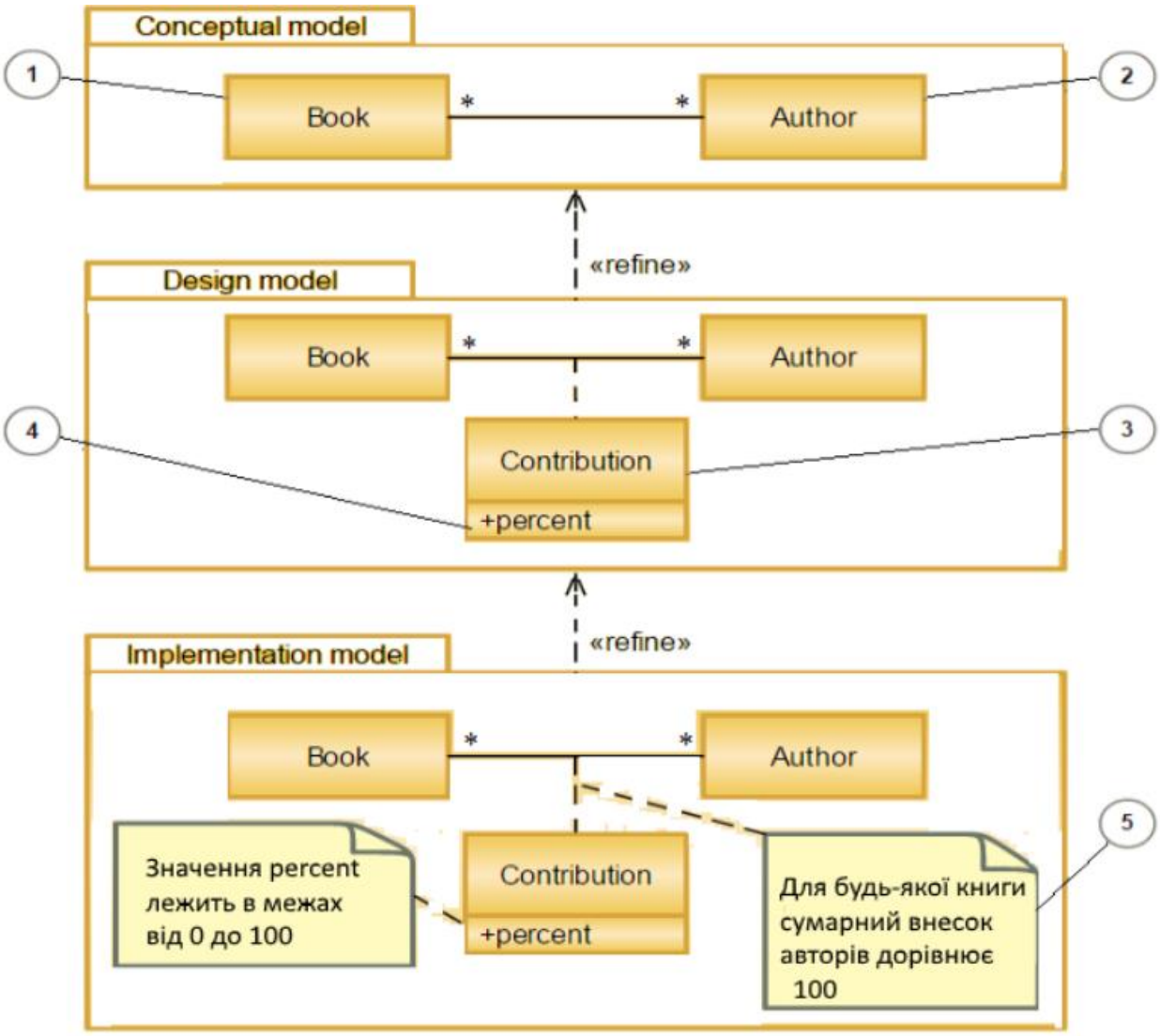


Рисунок 14 – Концептуальна модель, модель проектування і модель реалізації

З концептуальної точки зору книга (1) – це тип видання, який має авторів (2) , причому співавторів у книги може бути декілька і кожен може бути автором декількох книг. Важливо також врахувати, що в створення книги кожен співавтор вніс певний вклад (3). В моделі проектування необхідно уточнити, що абстрактний творчий вклад вимірюється в конкретних відсотках (4) , а в моделі реалізації додати, що необхідно перевіряти інваріантне (яке залишається незмінним при тих або інших перетвореннях) співвідношення (5) , яке полягає в тому, що для кожної книги сума вкладів усіх її авторів має бути рівна 100%. В якості доповнення до моделі реалізації можна навести пояснюючу діаграму об’єктів (рис. 15), де вказати конкретну книгу (1) , конкретних авторів (2) і конкретний вклад кожного з авторів (3).

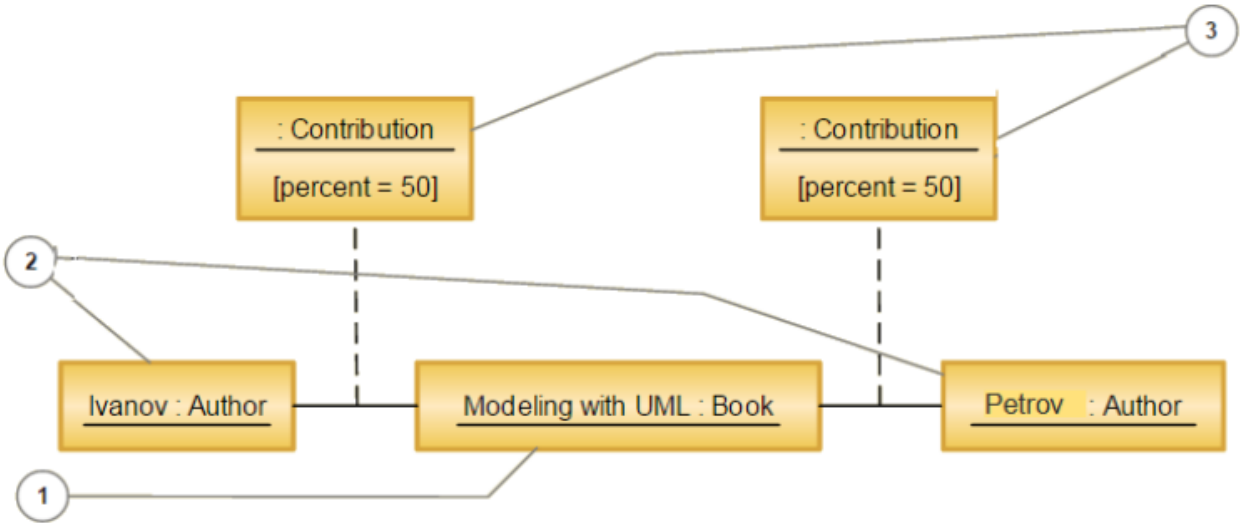


Рисунок 15 – Пояснююча діаграма об’єктів

Класичні представлення з UML 1 і 2

Набір використовуваних представлень (видів) моделі є ще менш формальним, чим набір канонічних діаграм і класифікація призначення і рівня моделей.

Одним з найпопулярніших є набір представлень, описаних авторами мови в UML 1 і показаних на рис. 16.

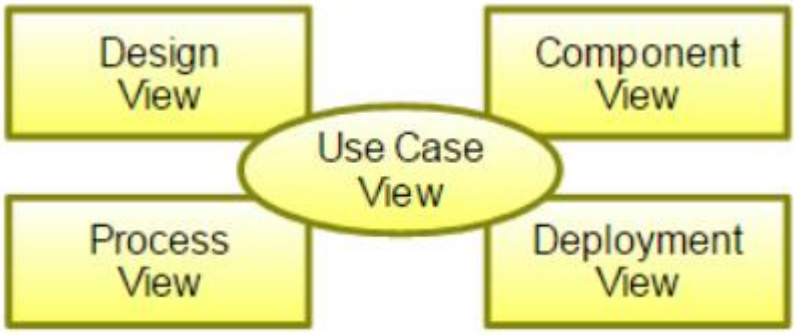


Рисунок 16 – Представлення з UML 1

1. Представлення використання (Use Case View) – цей опис поведінки системи в термінах варіантів використання з точки зору зовнішніх стосовно системи дійових осіб. Це представлення описує не те, як організована система, а ті функціональні вимоги, яким вона повинна задовольняти. При цьому структурні аспекти передаються *діаграмами використання, а поведінкові аспекти – діаграмами взаємодії, станів і діяльності*.

2. Представлення проектування (Design View) призначене для опису словника предметної області, тобто, в парадигмі об’єктно-орієнтованого програмування, класів, а також таких допоміжних сутностей як, наприклад, інтерфейси або кооперації. Структурні аспекти передаються *діаграмами класів і об’єктів,* а поведінкові аспекти – *діаграмами взаємодії, станів і діяльності*.

3. Представлення процесів (Process view) – цей опис взаємодії елементів керування (процесів, потоків) під час роботи системи. Воно відбиває такі

нефункціональні вимоги, як, наприклад, забезпечення паралелізму. Структурні аспекти передаються за допомогою концепції *активних класів,* які представляють процеси і потоки, а поведінкові аспекти – *діаграмами взаємодії, станів і діяльності.*

4. Представлення компонентів (Component view) – цей опис системи на рівні артефактів (компонентів, файлів тощо), використовуваних для складання, випуску, конфігурації програмного продукту. Структурні аспекти передаються *діаграмами компонентів,* а поведінкові аспекти – *діаграмами взаємодії, станів і діяльності*.

5. Представлення розміщення (Deployment view) відбиває топологію зв'язків апаратних засобів і розміщення на них компонентів. Структурні аспекти передаються *діаграмами розміщення,* а поведінкові аспекти – *діаграмами взаємодії, станів і діяльності*.

Первинні п'ять представлень, які асоціюються з UML 1, при переході до UML 2 були доповнені і в результаті утворили набір вже з восьми представлень.

1. Статичне представлення (Static view) відображає статичну структуру системи і не описує динаміку у будь-якому її прояві. Найчастіше статичне представлення відбиває логічні концепції додатку, основою якого служать класи і їх відношення.

2. Представлення проектування (Design view) є деталізованішим варіантом статичного представлення, виділяючи класифікатори, які забезпечують необхідну функціональність системи.

3. Представлення використання (Use Case view) описує функціонування системи в термінах варіантів використання і розглядає їх з точки зору зацікавлених дійових осіб.

4. Представлення автоматів (State machine view) специфікує поведінку окремих елементів, для яких можна ввести поняття життєвого циклу, що описується набором станів і переходів між ними.

5. Представлення діяльності (Activity view) описує функціонування системи з точки зору різних елементів діяльності, сполучених потоками керування і потоками даних.

6. Представлення взаємодії (Interaction view) відбиває послідовність обміну повідомленнями між елементами системи під час виконання додатку.

7. Представлення розгортання (Deployment view) описує розміщення артефактів на обчислювальних вузлах під час виконання додатку, а також компоненти і інші елементи, маніфестацією яких є ці артефакти.

8. Представлення керування моделлю (Model Management view) відбиває внутрішню організацію моделі, описуючи її розбиття на пакети і вказуючи відношення між ними.

Враховуючи неформальний характер самого поняття представлення, і спираючись на досвід використання UML, більшість авторів пропонують свій

варіант набору представлень. Деякі пропонуюь тільки три представлення:

1. Представлення використання. По суті це те ж саме представлення, яке було вказано вище. Представлення використання покликане відповідати на питання, ***що робить система корисного*.** Визначальною ознакою для віднесення елементів моделі до представлення використання являється явне зосередження уваги на факті наявності у системи зовнішніх меж, тобто виділення зовнішніх дійових осіб, які взаємодіють з системою, і внутрішніх варіантів використання, які описують різні сценарії такої взаємодії. Таким чином, єдиним виразним засобом представлення використання виявляються *діаграми використання*.

2. Представлення структури. Представлення структури покликане відповідати (з різною мірою деталізації) на питання: ***з чого складається система.*** Визначальною ознакою для віднесення елементів моделі до представлення структури являється явне виділення структурних елементів – складових частин системи – і опису взаємозв'язків між ними. Принциповим є чисто статичний характер опису, тобто відсутність поняття часу у будь-якій формі, зокрема, у формі послідовності подій і/або дій. Представлення структури описується, передусім, і головним чином *діаграмами класів,* а також, якщо потрібно, *діаграмами компонентів, розміщення, внутрішньої структури* і, в окремих випадках, *діаграмами об'єктів*.

3. Представлення поведінки. Представлення поведінки покликане відповідати на питання: ***як працює система*.** Визначальною ознакою для віднесення елементів моделі до представлення поведінки являється явне використання поняття часу, зокрема, у формі опису послідовності подій/дій, тобто у формі алгоритму. Представлення поведінки описується *діаграмами автомату* і *діяльності*, а також оглядовою *діаграмою взаємодії, діаграмами комунікації* і *послідовності*.

Такий набір представлень є погодженим з класифікацією діаграм. Більше того, процес моделювання (незалежно від призначення моделі) є не лінійним послідовним, а ітеративним і паралельним, приблизно таким, як показано на рис. 17.

Іншими словами, ***процес моделювання циклічний***, на кожному кроці може бути присутнім для уточнення представлення використання, за яким йде паралельне моделювання структури і поведінки.

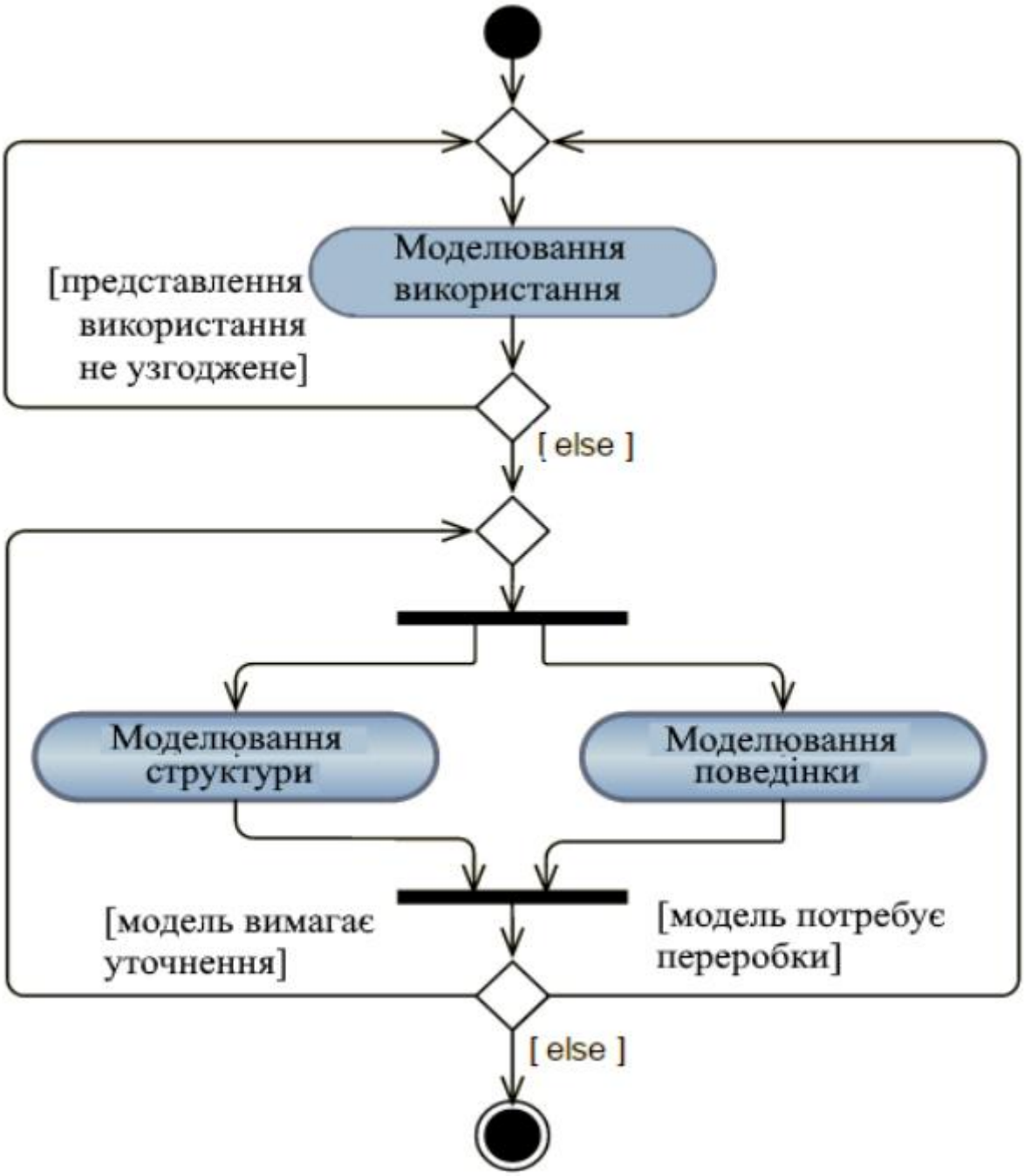


Рисунок 17 – Процес моделювання

Механізми розширення

Механізми розширення **–** це вбудований у мову спосіб її змінити. Автори UML при уніфікації різних методів постаралися включити в мову якомога більше різних засобів (утримуючи об'єм мови у рамках розумного), так щоб мова виявилася застосовною в різних контекстах і предметних областях. Але не можна обійняти неосяжного! Цілком можуть виникати і виникають випадки, коли стандартних елементів моделювання бракує, або вони не цілком адекватні.

Механізми розширення дозволяють визначати нові елементи моделі на основі існуючих керованим і уніфікованим способом. Таких механізмів три: *помічені значення*; *обмеження*; *стереотипи*.

Ці механізми не є незалежними – вони тісно пов'язані між собою.

Помічене значення **–** це пара: ім’я властивості і значення властивості, яка може бути додана до будь-якого стандартного елементу моделі. Іншими словами, помічене значення – це властивість, яка додається до представлення моделі. До будь-якого елементу моделі можна додати будь-яке помічене значення, яке зберігатиметься так само, як і усі стандартні властивості цього елементу. Спосіб обробки поміченого значення, визначеного користувачем, стандартом не описується і віддається на відкуп інструменту.

Помічені значення записуються в моделі у вигляді рядка тексту, який має наступний синтаксис: в фігурних дужках вказується пара: ім’я і значення, розділені знаком рівності. На рис. 18 наведений приклад використання помічених значень.

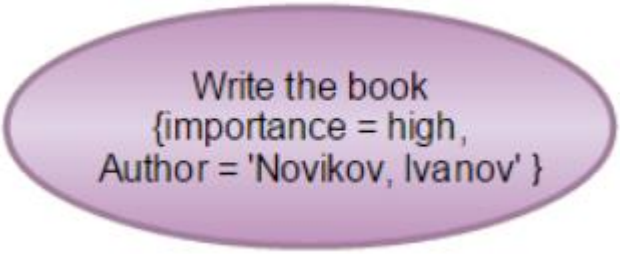


Рисунок 18 – Приклад використання помічених значень

Обмеження – це логічне твердження стосовно значень властивостей елементів моделі.

Логічне твердження може мати два значення: істина або хиба, тобто умова, що задається ним, або виконується, або не виконується. Вказуючи обмеження для елементу моделі, ми змінюємо його семантику, вимагаючи, щоб обмеження виконувалося. Обмеження може належати до окремого елементу або до сукупності елементів моделі.

Обмеження записуються у вигляді рядку тексту, поміщеного у фігурні дужки. Це може бути неформальний текст природною мовоюі, логічний вираз мови програмування, що підтримується інструментом, або вираження на деякій іншій формальній мові.

Аналогічно поміченим значенням, обмеження можна написати після імені елементу або в окремому коментарі, приєднаному до елементу. На рис.19 наведений приклад використання обмеження.

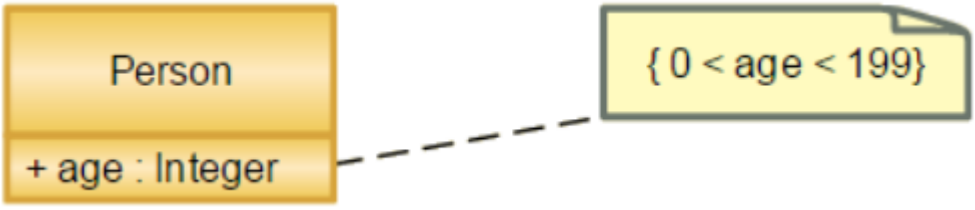


Рисунок 19– Приклад використання обмеження

Для помічених значень і обмежень одна і та ж синтаксична конструкція – текст у фігурних дужках – використовується не випадково. Насправді помічене

значення можна вважати обмеженням. А саме, якщо в моделі вказано помічене значення {A = B}, те це можна розглядати як запис умови: «властивість А обов'язково має значення В».

Стереотип **–** це визначення нового елементу моделювання на основі існуючого.

Визначення стереотипу робиться таким чином. Узявши за основу деякий існуючий елемент моделі, до нього додають нові помічені значення (розширюючи тим самим внутрішнє представлення), нові обмеження (змінюючи семантику) і доповнення, тобто нові графічні елементи.
Після того як стереотип визначений, його можна використати як елемент моделі нового типу. Якщо при створенні стереотипу не використовувалися доповнення і графічна нотація узята від базового елементу моделі, на основі якого визначений стереотип, то стереотип елементу позначається за допомогою імені стереотипу, поміщеного в подвійні кутові дужки (друкарські лапки «»), яке поміщується перед ім'ям елементу моделі. Якщо ж для стереотипу визначена своя нотація, наприклад, новий графічний символ, то вказується цей символ. Втім, можна вказати як ім'я стереотипу, так і графічний символ (значок). На рис. 20 наведений приклад визначення і використання стереотипу «PowerUser», який ми визначили на базі стандартного стереотипу дійової особи.

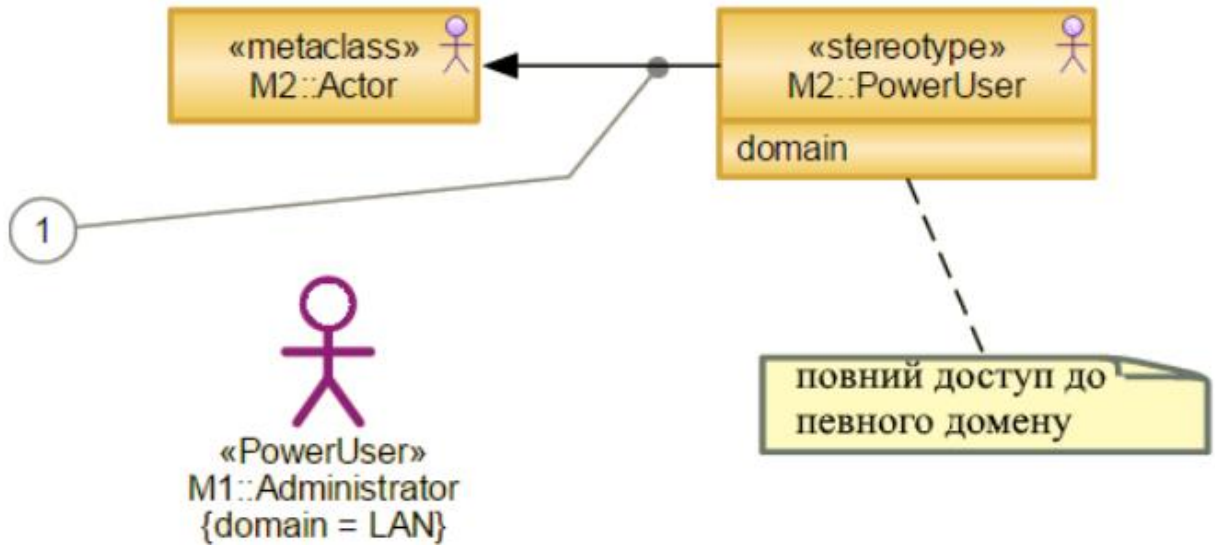


Рисунок 20 – Приклад використання стереотипу

Відношення між елементом моделі, який взятий за основу M2::Actor і новим елементом моделі M2::PowerUser називається відношенням ***розширення*** *(extension)* 1. В UML є велика кількість зумовлених стереотипів. Стереотипи використовуються дуже часто. Стереотипи є потужним механізмом розширення мови, проте ними слід користуватися помірно.

Контрольні запитання

1. Дайте визначення поняттю *моделі* програмної системи.
2. Які різновиди моделей ви знаєте?
3. Призначення UML
4. Дайте визначення поняттю *специфікація*.
5. Перерахуйте різні способи використання UML.
6. Які основні графічні елементи UML ви знаєте?
7. Перерахуйте способи використання UML.
8. Що таке *модель UML?*
9. З яких трьох «будівельних блоків» складається UML?
10. Які чотири сутності в UML ви знаєте?
11. Які чотири основні типи відношень в UML ви знаєте?
12. Перерахуйте 9 канонічних типів діаграм в UML 1.

Література

1. Гради Буч, Джеймс Рамбо, Ивар Якобсон Введение в UML от создателей языка. Пер: Н. Мухин – М.: ДМК Пресс, 2011г. – 496с.

2. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Пер. с англ. – СПб.: Символ-Плюс, 2011. – 192 с.

5.3. Проектування інтерфейсу користувача. Тестування і розробка тестів.

1 Основні правила створення інтерфейсу

Фахівці зазвичай формулюють деякий набір принципів і правил, які дозволяють як оцінювати зручність інтерфейсу, так і пропонувати рішення, які підвищують його зручність. Наведемо їх [1].

Правило доступності. Система має бути настільки зрозумілою, щоб користувач, який ніколи раніше не бачив її, але добре розбирається в предметній області, міг без жодного навчання почати її використати. Це правило служить деяким ідеалом, до якого потрібно прагнути, оскільки на практиці досягти такої міри доступності майже ніколи не вдається. Проте, потрібно робити в цьому напрямку все можливе.

Правило ефективності. Система не повинна перешкоджати ефективній роботі досвідчених користувачів, які працюють з нею довгий час. Очевидним прикладом порушення цього правила є націленість системи тільки на новачків, використання засобів, які добре підходять для недосвідченого користувача,

обмежуючи його в можливості зробити щось не так, але неефективні для експерта, який і так знає, що і як йому потрібно зробити.

Правило безперервного розвитку. Система повинна сприяти безперервному росту знань, умінь і навичок користувача і пристосовуватися до його досвіду, який мінюється. Погані результати приносить надання тільки базових можливостей або залишення початкуючого користувача наодинці із складним інтерфейсом, яким упевнено користуються експерти. Порушення безперервності при переході від одного набору можливостей до іншого також приносить незручності, оскільки користувач вимушений розбиратися з доданими можливостями в новому контексті.

Більшість користувачів можна розділити на три групи: новачки, досвідчені і середні, які вже знають більше, ніж новачки, і не роблять стільки помилок, але ще не придбали автоматизму при виконанні більшості операцій. Новачкам потрібна допомога в освоєнні нової для них системи і контроль за їх діями досвідченим користувачем. Про середніх же користувачів часто забувають, хоча переважна більшість користувачів програмного забезпечення належать саме до цієї категорії. Їм потрібні досить висока ефективність і гнучкість разом з можливістю швидко отримувати адекватну допомогу з різноманітних питань, які виникають час від часу.

Правило дотримання контексту. Система має бути погоджена з контекстом, в якому їй належить працювати. Це правило вимагає від системи бути працездатною не «взагалі», а саме в тому оточенні, в якому нею користуватимуться. У контекст можуть входити специфіка і об'єми вхідних і вихідних даних, тип і цілі організацій, в яких система повинна працювати, рівень користувачів, зашумленість приміщень тощо.

2 Принципи розробки інтерфейсу користувача

Наведені вище правила визначають загальні вимоги, яким повинен задовольняти зручний інтерфейс. Наступні принципи дозволяють знаходити рішення, які підвищують зручність призначеного для користувача інтерфейсу.

Принцип структуризації. Призначений для користувача інтерфейс має бути доцільно структурований. Близькі за змістом, споріднені його частини мають бути пов'язані видимим чином, а незалежні – розділені; схожі елементи повинні виглядати схоже, а несхожі – розрізнятися.

Принцип простоти. Найпоширеніші операції повинні виконуватися максимально просто. При цьому мають бути видимі посилання на складніші процедури.

Принцип видимості. Усі функції і дані, які необхідні для розв'язання певного завдання, мають бути видні, коли користувач намагається його вирішити.

Принцип зворотного зв'язку. Користувач повинен отримувати повідомлення про дії системи і про важливі події всередині неї. Повідомлення

мають бути інформативними, короткими, однозначними і написаними мовою, зрозумілою користувачеві.

Принцип толерантності. Інтерфейс має бути гнучким і терпимим до помилок користувача. Збиток від помилок повинен знижуватися за рахунок можливості відміни і повтору дій і за рахунок розумної інтерпретації будь-яких розумних дій користувача і введених ним даних. По можливості слід уникати взаємодії (модальних діалогів) заснованої на обмеженні свободи користувача.

Принцип повторного використання. Слід намагатися багаторазово використати внутрішні і зовнішні компоненти, забезпечуючи тим самим уніфікованість інтерфейсу і схожість між його схожими елементами.

Далі розглянемо деякі питання розробки ергономічного інтерфейсу користувача.

3 Взаємодія між користувачем і комп'ютером

Людиномашинний інтерфейс забезпечує зв'язок між користувачем і комп'ютером, він дозволяє досягати поставлених цілей, успішно знаходити розв'язання поставленої задачі. Взаємодія – обмін діями і реакціями на ці дії між комп'ютером і користувачем. Існує ряд стилів взаємодій, які підрозділяються на два види:

1. Використання інтерфейсу мови команд – введення команд текстовими засобами.
2. Безпосереднє маніпулювання.

Таким чином, є ряд способів, якими користувач міг би зв'язуватися з комп'ютером:

– мови команд – користувач керує системою, вводячи відповідні команди в текстовому режимі;

– питання і відповідь – діалог, в якому комп'ютер ставить питання, а користувач відповідає йому (чи навпаки);

– форми – користувач заповнює форми або поля діалогу, вводячи дані у відповідні поля;

– меню – користувач забезпечений рядом опцій і керує системою, вибираючи необхідні пункти;

– пряме маніпулювання – користувач керує об'єктами на екрані за допомогою пристрою маніпулювання типу миші.

В розроблюваній програмній системі також може бути застосований комплексний підхід до створення інтерфейсу. Тут використовується пряме маніпулювання, меню, форми і діалоги.

Мета створення ергономічного інтерфейсу полягає в тому, щоб відобразити інформацію настільки ефективно, наскільки це можливо для людського сприйняття і структурувати відображення на екрані так, щоб притягнути увагу до найбільш важливих одиниць інформації. Основна ж мета в тому, щоб мінімізувати загальну інформацію на екрані і представити тільки те, що є необхідним для користувача.

4. Розміщення інформації на екрані

Кількість інформації, яка відображається на екрані, називається *екранною щільністю*. Дослідження показали, що чим менше екранна щільність, тим інформація, яка відображається, доступніша і зрозуміліша для користувача і навпаки, якщо екранна щільність велика, то це може викликати труднощі в засвоєнні інформації і її ясному розумінні. Проте досвідчені користувачі можуть віддавати перевагу інтерфейсам з великою екранною щільністю.

Інформація на екрані може бути згрупована і впорядкована в значимі частини. Це може бути досягнуто використанням кадрів (фреймів), методів типу колірного кодування, рамок, негативного зображення або інших методів для привертання уваги.

Виділення елементів інтерфейсу яскравістю. Для привертання уваги до яких-небудь елементів інтерфейсу можна скористатися виділенням цих елементів більшою яскравістю на тлі інших, темніших. Проте важливо не перестаратися з цим методом, оскільки велика кількість яскравих елементів здатна викликати дискомфорт у користувача і можна отримати зворотний ефект – перевантаження інтерфейсу. Застосовувати цей метод слід тільки при необхідності. Існує декілька способів виділення яскравістю:

– рух (миготіння або зміна позиції) – ефективний метод;

– яскравість – не дуже ефективний метод, оскільки люди здатні розрізнити лише декілька рівнів яскравості;

– колір – використання може бути надзвичайно ефективним;

– форма (символ, шрифт, форма символу) використовується для того, щоб диференціювати різні категорії даних;

– розмір (тексту, символів) – зазвичай застосовують збільшення виділеного об’єкту в 1,5 рази;

– відтінення (різна текстура об’єктів) – ефективний метод для привертання уваги до якої-небудь частини екрану;

– оточення (підкреслення, рамки, інвертоване зображення) – дуже ефективний спосіб.

Використання кольору при проектуванні ергономічного інтерфейсу.

Колір може поліпшити інтерфейс, але для багатьох систем використання кольору практично не впливає на ефективність роботи користувача. Основне призначення кольору – створення інтерфейсів, цікавіших для користувачів, проте є випадки, коли колір може допомогти проектувальникові інтерфейсу. Особливо це ефективно при групуванні інформації, виділенні відмінностей між інформацією або простими повідомленнями (помилки, стани тощо).

Колір – потужний візуальний інструмент, і застосовувати його потрібно дуже обережно, щоб не викликати у користувача дискомфорт від помилкових колірних комбінацій. Перерахуємо деякі принципи використання кольору, якими бажано керуватися при проектуванні ергономічного інтерфейсу:

– необхідно обмежити число кольорів до 4 на екрані і до 7 – для

послідовності екранів;

– для неактивних елементів краще використати бліді кольори;

– при відображенні стану, як правило, червоний означає небезпеку (стоп), зелений – продовження роботи, жовтий – колір попередження;

– для привертання уваги найефективніші білий, жовтий і червоний кольори;

– при необхідності впорядкування даних (чи розділення даних) можна використати спектр 7 кольорів (веселка);

– для згрупування даних, об'єднання і подібності треба використати сусідні кольори спектру: помаранчево-жовті, синьо- фіолетові.

Важливо відмітити, що близько 9% людей не розрізняють кольорів (зазвичай, червоно-зелені поєднання); проте ці люди можуть відрізняти чорно-білі відтінки, тому проектувальники автоматизованих систем повинні перевіряти, чи не порушує використання різних кольорів в інтерфейсах сприйняття користувачів цієї категорії.

Несуперечність і стандартизація. Дані на екрані слід розташовувати так, щоб користувач знав, де знайти і де чекати виведення необхідної інформації:

– інформація, на яку слід негайно звернути увагу, повинна завжди відображатися на видному місці, щоб захопити увагу користувача (наприклад, застережливі повідомлення і повідомлення про помилки);

– не дуже часто потрібна інформація (наприклад, довідка) не повинна відображатися, але має бути доступна за потреби. Наприклад, іконка «довідки» або відповідна опція меню має бути доступна на кожному екрані.

Тексти і діалоги. Наведемо деякі принципи, якими необхідно керуватися при створенні текстових діалогів і відображень:

– текст в нижньому регістрі читається приблизно на 13% швидше, ніж текст, надрукований повністю у верхньому регістрі;

– символи верхнього регістру найефективніші для передачі інформації, яка повинна привернути увагу. Не використовуйте верхній регістр, якщо ви не хочете виділяти яку-небудь інформацію;

– вирівняний по правому краю текст важче читати, ніж рівномірно розподілений текст з невирівняним правим полем;

– оптимальний інтервал між рядками рівний або трохи більше, ніж висота символів.

Меню. Необхідний елемент автоматизованої системи – меню, що дозволяє користувачеві виконувати завдання всередині додатку і керувати процесом рішення. Меню – набір опцій, які відображаються на екрані, де користувачі можуть вибирати і виконувати дії, тим самим роблячи зміни в стані інтерфейсу. Перевага меню в тому, що користувачі не повинні пам'ятати назву елементу або дії, яку вони хочуть виконати, вони повинні тільки розпізнати

його серед пунктів меню.

Таким чином, меню може використовувати навіть недосвідчений користувач. Проте проект меню має бути ретельно продуманий – щоб меню було ефективним, назви його пунктів мають бути очевидними. Меню може займати багато екранного місця, але вирішенням цієї проблеми може бути використання спливаючого або спадаючого меню, яке викликається клацанням по піктограмі, рядку меню або іншому об'єкту.

Основні принципи створення меню. В процесі проектування системи меню додатка необхідно прийняти найкращий спосіб відображення меню, щоб воно було зрозумілим і легким у використанні. Зазвичай команди меню впорядковані деяким ієрархічним способом. Основна проблема полягає в тому, щоб правильно розподілити різні пункти меню по різних рівнях і правильно їх згрупувати.

Принципи проектування меню:

– структура меню повинна відповідати структурі завдання, яке розв'язується системою; організація меню повинна відобразити найефективнішу послідовність кроків, які ведуть до розв'язання поставленої задачі;

– пункти меню мають бути короткими, граматично правильними і відповідати своєму заголовку. Порядок пунктів меню вибирається відповідно до угоди, частоти і порядку використання, а також залежно від потреб завдання або користувача;

– вибір пунктів меню має бути забезпечений декількома способами – за допомогою клавіатури, за допомогою миші і через інші об'єкти призначеного для користувача інтерфейсу. Важливо зафіксувати поєднання клавіш, що легко запам'ятовуються, для швидшого доступу до пунктів меню, оскільки це дуже економить час.

Форми. Форми – основний елемент інтерфейсу. Призначення форм – зручне введення і перегляд даних, стану, повідомлень автоматизованої системи.

Основні принципи проектування форм:

– форма проектується для зручнішого, зрозумілого і швидкого рішення поставленої задачі. Якщо форма переноситься з паперової форми, то пересування по суміжних полях не повинне викликати утруднень у користувача;

– розміщення інформаційних одиниць на просторі форми повинне відповідати логіці її майбутнього використання: це залежить від необхідної послідовності доступу до інформаційних одиниць, частоти їх використання, а також від відносної важливості елементів;

– логічні групи елементів необхідно відділяти пропусками, рядками, колірними або іншими візуальними засобами;

– взаємозалежні або пов'язані елементи повинні відображатися в одній формі.

При розробці форм необхідно продумати і вказати, які кнопки в смузі системного меню мають бути доступні в тому або іншому вікні, чи повинне вікно допускати зміни користувачем його розміру, яким має бути заголовок вікна. Без особливої необхідності не треба робити вікна зі змінюваними розмірами. При зміні розмірів, якщо не застосовані спеціальні прийоми, порушується компонування вікна, що негативно позначається на роботі користувача. Має сенс створити вікно зі змінюваними розмірами, якщо це дозволяє користувачеві змінювати корисну площу розташованих в нім компонентів відображення і редагувати інформацію: текст, зображення, списки тощо. Якщо проектується змінюване вікно, то необхідно вжити заходи, щоб компоненти у вікні при цьому теж змінювали свої розміри або місце розташування, рівномірно розподіляючись по площі вікна і не залишали порожніх місць.

При проектуванні форм необхідно прагнути до використання обмеженого набору кольорів і приділяти увагу їх правильному поєднанню. Для фону форми обираються нейтральні кольори (світло-сірі). Колір не повинен використовуватися як основний засіб передачі інформації, потрібно вибирати системні кольори, які користувач може перебудовувати на свій розсуд.

Керуючі елементи і функціонально пов’язані з ними компоненти екрану слід зорово об’єднувати в групи, заголовки яких коротко і чітко пояснюють їх призначення. Кожне вікно повинне мати деяку центральну тему, яка підпорядковується його композиції. Користувач повинен розуміти, для чого призначено це вікно і що в нім найважливіше. Неприпустимо перевантажувати вікно великим числом елементів керування, введення і відображення інформації. Також неприпустимо, щоб схожі за функціями органи керування в різних вікнах називалися по-різному або розміщувалися в різних місцях вікон.

Важливо потурбуватися про те, як додаток впишеться в загальну організацію робочого простору системи і як це взаємодіятиме з іншими застосуваннями.

Дизайн заголовків і полів. Для окремих полів заголовок має бути вирівняний по лівому краю; для полів списків заголовок має бути вищий і лівіший по відношенню до основного поля; числові поля вирівнюються по правому полю.

Довгі колонкові поля або довгі стовпці інформаційних одиниць з поодинокими полями необхідно об’єднувати в групи по п’ять елементів, які розділяються порожнім рядком, – це допомагає користувачеві подумки обробляти інформацію по виділених групах.

У формах з великою кількістю інформації важливо використати назви розділів, які однозначно свідчать про характер інформації, яка належить їм. Необхідно чітко розділити відображення заголовків і безпосередньо полів введення. Заголовки мають бути короткими, знайомими і змістовними. Поля, які необов’язкові для заповнення або не мають особливої важливості, повинні відрізнятися візуально (кольором або іншими ефектами)

від полів важливих і обов'язкових для заповнення.

Формати введення. Слід забезпечити введення значень за замовчуванням в усі поля, які це допускають і де така функція не дратуватиме користувача. Можна призначити клавіші або коди для введення значень, які часто повторюються. Вхідні дані мають бути значимими і загальноприйнятими.

Не слід об'єднувати поля введення чисел і символів, оскільки числові і алфавітні клавіші знаходяться незручно один відносно одного на клавіатурі. Необхідно виключити часте перемикання між верхнім і нижнім регістрами для прискорення введення даних. Не можна вимагати від користувача введення незначимих цифр (наприклад, замість 00000010 користувач повинен ввести тільки 10). Аналогічно, не можна вимагати від користувача ввести інформацію, яка була заздалегідь введена або яка може бути автоматично отримана з системи. Бажано використати значення за замовчуванням, щоб мінімізувати процес введення інформації.

Організація системи навігації і системи відображення станів.

Навігація забезпечує користувачеві можливість переміщення між різними екранами, інформаційними одиницями і підпрограмами в автоматизованій системі. У повноцінній системі користувач завжди може отримати інформацію про стан системи, про процес виконання або активну підпрограму.

Загальні принципи проектування. Існує ряд навігаційних засобів і прийомів, які допомагають користувачеві орієнтуватися в системі. Вони включають використання заголовків сторінок для кожного екрану, номерів сторінок, рядків і стовпців, відображення поточного імені файлу вверху екрану.

Тип системи навігації залежить від прийнятого стилю інтерфейсу. Для інтерфейсів мови команд існує дуже мало способів забезпечення повноцінної навігації. У інтерфейсах з меню можна використати ієрархічно структуроване меню. Діалогові інтерфейси самі по собі захищають користувача від помилкових дій. Інформація стану зазвичай відображається внизу екрану і містить в собі дані про кількість записів, число оброблених одиниць, процес друку, черги друку і так далі

Проектування повідомлень. Повідомлення потрібні для спрямування дій користувача в потрібну сторону підказок і попереджень при виконанні необхідних дій на шляху розв'язання задачі. Вони також включають підтвердження дій з боку користувача і підтвердження з боку системи, що завдання виконані успішно або з якихось причин не виконані. Повідомлення можуть бути виведені у формі діалогу, екранних заставок і тому подібне.

Повідомлення можуть запропонувати користувачеві:

– вибрати із запропонованих альтернатив опцію або набір опцій;

– ввести інформацію;

– вибрати опцію з набору опцій, які можуть змінюватися залежно від поточного контексту;

– підтвердити фрагмент введеної інформації перед продовженням введення.

Повідомлення можуть бути поміщені в модальні діалогові вікна, які змушують користувача відповісти на питання перш, ніж почне виконуватися будь-яка інша дія. Це може бути корисно, коли система змушена змусити користувача обдумати рішення перед продовженням роботи. Немодальні діалогові вікна дозволяють працювати з іншими елементами інтерфейсу, тоді як саме вікно може ігноруватися.

5. Запобігання, виявлення і виправлення помилок

Звичайна людина у нормальному стані здійснює багато помилок різного роду. Можна сказати, що людина, на відміну від комп'ютера, є адаптивною аналоговою системою і успішність її «функціонування» в набагато більшому ступені визначається не точністю виконання дій і формулювання думок, а здатністю швидко видати хороше наближення до потрібного результату і досить швидко виправитись, якщо це необхідно.

Помилки користувача можуть бути засновані на неправильному розумінні дії або порядку дій, або бути випадковими, неумисними, наприклад, друкарська помилка при введенні тексту.

Помилки другого виду можуть бути розділені ще на шість підвидів:

– неточність у виборі опції (наприклад, користувач випадково натиснув кнопку «Вихід» і програма закрилася);

– втрата активності, коли користувач забуває необхідну послідовність дій для продовження роботи;

– помилка режиму або стану, коли користувач думає, що він знаходиться в одному стані, а фактично – в іншому; наприклад, режим вставки замість режиму друку поверх тексту в текстовому процесорі.

Користувач завжди робитиме помилки навіть у відмінній програмній системі, тому в розроблюваный системі завжди має бути передбачений захист від помилок. Техніка такого захисту включає такі аспекти:

– примусові дії в системі, які запобігають або утрудняють появу помилок;

– забезпечення хороших і інформативних повідомлень про помилки;

– забезпечення нормальної діагностики системи, в процесі якої користувачеві пояснюється, в чому суть помилки, і вказуються шляхи її виправлення.

Розглянемо основні принципи обробки помилок у формах введення:

– забезпечення можливості посимвольного редагування введених записів для виправлення помилок введення (друкарських помилок);

– якщо помилка виявлена системою, бажано повернути курсор в поле з помилковими даними і яким-небудь чином виділити це поле;

– виводити значимі повідомлення про помилки, які використовують стиль мови користувача і відповідну термінологію;

– виводити повідомлення про помилки, які пояснюють і пропонують шляхи їх усунення.

Ефективність запобігання і подолання помилок користувачів тим вище,

чим рідше користувачі помиляються при роботі з цим інтерфейсом і чим менше часу і зусиль вимагається для подолання наслідків вже зроблених помилок.

Рівні тестування

Тестуванням називається виконання програми з метою виявлення помилок. Локалізація і виправлення помилок відбувається при налаштуванні .

Модульне тестування є процесом перевірки окремих програмних процедур (модулів) і підпрограм, які входять до складу програм або підпрограмних систем. Модульне тестування робиться безпосереднім розробником і дозволяє перевіряти всі внутрішні структури і потоки даних в кожному модулі.

Інтеграційне тестування проводиться для перевірки спільної роботи окремих модулів і передує тестуванню всієї системи як єдиного цілого. В ході інтеграційного тестування перевіряються зв'язки між модулями, їх сумісність і функціональність. Воно здійснюється незалежним тестувальником і входить до складу етапу тестування.

Елементи інтеграційного тестування:

– перевірка функціональності – перевірка відповідності окремих функцій, які виконуються сукупностями модулів, функціям, заданим в специфікаціях вимог;

– перевірка проміжних результатів – перевірка всіх проміжних результатів і файлів на наявність і коректність;

– перевірка інтеграції – перевірка коректності взаємної передачі модулями інформації.

Системне тестування призначене для перевірки програмної системи в цілому, її організації і функціонування на відповідність специфікаціям вимог замовника. Його проводить незалежний тестувальник після успішного завершення інтеграційного тестування.

Елементи системного тестування:

– граничне тестування – тестування в граничних умовах;

– прогоночне тестування – тестування всіх функціональних характеристик реальної роботи системи;

– цільове тестування – тестування на цільовій платформі;

– перевірка документації – перевірка призначеної для користувача документації на коректність;

– інші тести, які визначаються тестувальником.

Вихідне тестування – завершальний етап тестування, на якому перевіряється готовність програмного продукту для постачання замовникові. Цей вид тестування проводить незалежний тестувальник.

Приймальне тестування проводиться організацією, яка відповідає за інсталяцію, супровід програмної системи і навчання кінцевого користувача.

Технології тестування

Технологія тестування, яка застосовується на етапі розробки програмного забезпечення, називається тестуванням «скляного ящика» («білого ящика») в протилежність класичному поняттю «чорного ящика».

При тестуванні «чорного ящика» програма розглядається як об'єкт, внутрішня структура якого невідома. Тестувальник вводить дані і аналізує результат, але він не знає, як саме працює програма.

При тестуванні «скляного ящика» ситуація абсолютно інша. Тестувальник (в даному випадку сам програміст) розробляє тести, грунтуючись на знанні початкового коду, до якого він має повний доступ. В результаті він отримує певні переваги.

1. Спрямованість тестування. Програміст може тестувати програму частинами, розробляти спеціальні тестові підпрограми, які викликають тестований модуль і передають йому дані, які цікавлять програміста. Окремий модуль набагато легше протестувати саме як «скляний ящик».
2. Повне охоплення коду. Програміст завжди може визначити, які саме фрагменти коду працюють в кожному тесті. Він бачить, які ще гілки коду залишилися непротестованими, і може підібрати умови, в яких вони будуть протестовані. Нижче описано, як відстежувати міру охоплення програмного коду проведеними тестами.
3. Можливість керування потоком команд. Програміст завжди знає, яка функція повинна виконуватися в програмі наступною і яким має бути її поточний стан. Щоб з'ясувати, чи працює програма так, як він думає, програміст може включити в неї налагоджувальні команди, які відображають інформацію про хід її виконання, або скористатися для цього спеціальним програмним засобом, який називається відладчиком.
4. Можливість відстежування цілісність даних. Програмістові відомо, яка частина програми повинна змінювати кожен елемент даних. Відстежуючи стан даних (за допомогою того ж відладчика), він може виявити такі помилки, як зміна даних не тими модулями, їх невірна інтерпретація, або невдала організація. Програміст може самостійно автоматизувати тестування.

Тестування «скляного ящика» – частина процесу програмування. Програмісти виконують цю роботу постійно, вони тестують кожен модуль після його написання, а потім ще раз після інтеграції його в систему. При виконанні модульного тестування можна використати технологію або структурного, або функціонального тестування, або одночасно і ту і іншу.

Програмні помилки

Всі програмні помилки можна віднести до відповідних категорій.

Розглянемо ті, що зустрічаються найчастіше.

1. Функціональні недоліки властиві програмі, якщо вона не робить того, що повинна, виконує одну зі своїх функцій погано, або не повністю. Функції програми мають бути детально описані в її специфікації, і саме на основі

затвердженої специфікації тестувальник будує свою роботу.
2. Недоліки призначеного для користувача інтерфейсу. Оцінити зручність і правильність роботи призначеного для користувача інтерфейсу можна тільки в процесі роботи з ним. Бажано, щоб в цій роботі брав участь сам користувач.
3. Недостатня продуктивність. При розробці програмного продукту важливою його характеристикою може виявитися швидкість роботи, іноді цей критерій задається у вимогах замовника. Погано, якщо у користувача створюється враження, що програма працює повільно.
4. Некоректна обробка помилок. Процедури обробки помилок – важлива частина програми. Правильно визначивши помилку, програма повинна видати про неї повідомлення.
5**.** Некоректна обробка граничних умов. Існує багато різних граничних ситуацій. Будь-який аспект роботи програми, до якого застосовані поняття «більше» або «менше», «раніше або пізніше», «коротше» або «довше», обов'язково має бути перевірений на межах діапазону.
6. Помилки обчислень. Сюди належать помилки, викликані неправильним вибором алгоритму обчислень, неправильними формулами. Найпоширенішими є помилки округлення.
7. Помилки керування потоком. За логікою роботи програми слідом за першою дією повинна бути виконана друга. Якщо замість цього виконується третя або четверта дія, значить, в керуванні потоком припустилися помилки.
8**.** Перевантаження. Збої в роботі програми можуть відбуватися із-за нестачі пам'яті, або відсутності інших необхідних системних ресурсів.

Контрольні запитання

1. Перерахуйте загальні принципи і правила створення зручного інтерфейсу.
2. Як використовується колір при розробці ергономічного інтерфейсу?
3. Перерахуйте основні вимоги до компонування форм.
4. Як можна запобігти появі помилок при введенні?
5. Як правильно створити меню?
6. Що відображають схеми роботи системи?
7. Дайте визначення технологіям тестування «білого ящика» і «чорного ящика».
8. Перерахуйте основні види тестування.
9. Що означає ситуація перегонів? Яким чином її можна протестувати?
10. Що є програмною помилкою? Які категорії програмних помилок ви знаєте?

Література

1 Рудаков А.В., Федорова Г.Н. Технология разработки программных продуктов. Практикум. 4-е изд. – М.: ИЦ «Академия», 2014. – 192 с.

6. CASE-технології

6.1 *Вступ в CASE-технології*

Давно минули часи, коли одна людина цілком могла справлятися з реалізацією програмного проекту, який забезпечує функціональність великих підприємств. Постійне зростання складності та комплексності не лише цілей проекту, а й інструментарію їх реалізації призводить до того, що вже важко обійтися силами окремих фахівців, а потрібна злагоджена робота цілої команди. Для того, щоб успішно виконати проект, об'єкт проектування має бути перш за все правильно і адекватно описаний, тобто необхідно побудувати повноцінні та функціональні інформаційні моделі об'єкта проектування.

Донедавна проектування інформаційних систем виконувалося головним чином на інтуїтивному рівні із застосуванням не формалізованих методів, які ґрунтувалися на практичному досвіді, експертних оцінках та дорогих експериментальних перевірках якості функціонування подібних систем. Але, природно, під час розробки та функціонування інформаційних систем потреби користувачів можуть змінюватись або уточнюватися, що ще більше ускладнює розробку та супровід.

У 1970-80-х роках при розробці інформаційних систем широко застосовувалася структурна методологія, що надає розпорядникам у розпорядження суворі формалізовані методи опису ІС і прийнятих технічних рішень. Ця методологія ґрунтувалася на наочній графічній техніці, інакше кажучи, для опису проекту використовувалися різноманітні схеми та діаграми. Наочність та суворість засобів структурного аналізу дозволяла розробникам та майбутнім користувачам системи від самого початку неформально брати участь у її створенні, обговорювати та закріплювати розуміння основних технічних рішень. Однак широке застосування цієї методології та дотримання її рекомендацій при розробці конкретних проектів зустрічалося досить рідко, оскільки її практично неможливо реалізувати належним чином ручним, неавтоматизованим способом. Вручну дуже важко розробити та графічно уявити суворі формальні специфікації системи, перевірити їх на повноту та несуперечність, і тим більше змінити. Якщо все ж таки вдається створити сувору систему проектних документів, то її переробка при появі серйозних змін практично неможлива. Якщо учасники проекту намагалися вдатися до ручної розробки, перед ними виникали такі проблеми:

• неадекватна специфікація вимог;

• нездатність виявляти помилки у проектних рішеннях;

• низька якість документації, що знижує експлуатаційні якості;

• затяжний цикл та незадовільні результати тестування.

Ці аргументи не завжди сягали розробників. Адже відомо, що проектувальники інформаційних систем в останню чергу використовують комп'ютерні технології для підвищення якості та продуктивності своєї роботи. Але рано чи пізно мали з'явитися спеціалізовані програмно-технологічні засоби розробки проектів, зокрема, заснованих на інформатизації. Ними стали засоби, які реалізують CASE-технологію створення та супроводу інформаційних систем.

Термін CASE (Computer-Aided Software Engineering) сьогодні розуміється досить широко. Початкове значення терміна, обмежене питаннями автоматизації розробки програмного забезпечення (ПЗ), нині набуло нового сенсу, і тепер це поняття охоплює процес розробки складних інформаційних систем загалом. Тепер під терміном CASE-засоби розуміються програмні засоби, які підтримують процеси створення та супроводу подібних систем, включаючи аналіз та формулювання вимог, проектування прикладного ПЗ (додатків) та баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне управління та управління проектом та т. д. CASE-засоби разом із системним ПЗ та технічними засобами утворюють повне середовище розробки.

Активні дослідження в галузі методології програмування призвели до того, що програмування набуло рис системного підходу з розробкою та впровадженням мов високого рівня, методів структурного та модульного програмування, мов проектування та засобів їх підтримки, формальних та неформальних мов описів системних вимог та специфікацій тощо. Крім того, появі CASE-технології сприяли і такі фактори, як:

• підготовка аналітиків та програмістів, сприйнятливих до концепцій модульного та структурного програмування;

• широке впровадження та постійне зростання продуктивності комп'ютерів, що дозволило використовувати ефективні графічні засоби та автоматизувати більшість етапів проектування;

• впровадження мережевої технології, що надала можливість об'єднання зусиль окремих виконавців у єдиний процес проектування шляхом використання

розділяємої бази даних,яка містить необхідну інформацію про проект. Таким чином, CASE-технологія є методологією проектування ІС, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки та супроводу ІС та розробляти додатки відповідно до потреб користувачів. Більшість CASE-засобів використовує методологію структурного (в основному) або орієнтованого аналізу та проектування, що використовують специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів. Згідно із західними дослідженнями CASE-технологія потрапила до розряду найстабільніших інформаційних технологій. Втім, CASE-засоби, як і будь-який інструмент, потрібно вміти застосовувати. Існує безліч прикладів їх невдалого впровадження. У зв'язку з цим слід зазначити таке:

• CASE-засоби не обов'язково дають негайний ефект; він може бути отриманий тільки через якийсь час;

• реальні витрати на використання CASE-засобів зазвичай набагато перевищують витрати на їх придбання;

• CASE-засоби забезпечують можливості отримання значної вигоди лише після успішного завершення процесу їх впровадження.

Сучасні CASE-засоби охоплюють широку сферу підтримки численних технологій проектування інформаційних систем - від простих засобів аналізу та документування до повномасштабних засобів автоматизації, що покривають весь життєвий цикл програмного забезпечення. Найбільш трудомісткими етапами розробки інформаційних систем є аналіз та проектування, у процесі яких CASE-засоби забезпечують якість прийнятих технічних рішень та підготовку проектної документації. У цьому велику роль грають методи візуального представлення інформації. Це передбачає побудову структурних чи інших діаграм у реальному масштабі часу, використання різноманітної палітри кольорів, наскрізну перевірку синтаксичних правил. Графічні засоби моделювання дозволяють розробникам у наочному вигляді вивчати існуючу інформаційну систему, перебудовувати її відповідно до поставлених цілей та наявних обмежень. До розряду CASE-засобів потрапляють як відносно дешеві системи для персональних комп'ютерів з обмеженими можливостями, так і дорогі системи для неоднорідних обчислювальних платформ та операційних середовищ. Так, сучасний ринок програмних засобів налічує більше 300 різних CASE-засобів, найпотужніші з яких використовуються практично всіма провідними західними компаніями. Зазвичай до CASE-засобів відносять будь-

який програмний засіб, що автоматизує ту чи іншу сукупність процесів життєвого циклу ПЗ і має такі особливості:

• потужні графічні засоби для опису та документування ІС, які забезпечують зручний інтерфейс з розробником та розвивають його творчі можливості;

• інтеграція окремих компонентів CASE-засобів, що забезпечує керованість процесом розробки інформаційної системи;

• використання спеціальним чином організованого сховища проектних метаданих (репозиторія). Інтегрований CASE-засіб (або комплекс засобів, що підтримують повний життєвий цикл програмного забезпечення) містить такі компоненти:

• репозиторій, що є основою CASE-засобу. Він повинен забезпечувати зберігання версій проекту та його окремих компонентів, синхронізацію надходження інформації від різних розробників при груповій розробці, контроль метаданих на повноту та несуперечність;

• графічні засоби аналізу та проектування, що забезпечують створення та редагування ієрархічно пов'язаних діаграм (DFD, ERD та ін.), що утворюють моделі інформаційної системи;

• засоби розробки додатків, включаючи мови 4GL та генератори кодів;

• засоби конфігураційного управління;

• засоби документування; • засоби тестування;

• засоби управління проектом;

• засоби реінжинірингу.

Всі сучасні CASE-засоби можна класифікувати за типами та категоріями. Класифікація за типами відбиває функціональну орієнтацію CASE-засобів ті чи інші процеси життєвого циклу. Класифікація за категоріями визначає ступінь інтегрованості за функціями, що виконуються, і включає окремі локальні засоби, що вирішують невеликі автономні завдання (tools), набір частково інтегрованих засобів, що охоплюють більшість етапів життєвого циклу інформаційних систем (toolkit) і повністю інтегровані засоби, що підтримують весь життєвий цикл інформаційних систем і пов'язані загальним репозиторієм. Крім цього CASE-засоби можна класифікувати за методологіями і моделями застосовуваних систем і БД; ступеня інтегрованості із СУБД; доступним платформам. Класифікація за типами переважно збігається з компонентним складом CASE-засобів і включає:

• засоби аналізу (Upper CASE), призначені для побудови та аналізу моделей предметної галузі (Design/IDEF (Meta Software), BPwin (Logic Works));

• засоби аналізу та проектування (Middle CASE), що підтримують найбільш поширені методології проектування та використовуються для створення проектних специфікацій (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE . Аналітик (МакроПроджект)). Виходом таких засобів є специфікації компонентів та інтерфейсів системи, архітектури системи, алгоритмів та структур даних;

• засоби проектування баз даних, що забезпечують моделювання даних та генерацію схем баз даних (як правило, мовою SQL) для найбільш поширених СУБД. До них відносяться ERwin (Logic Works), S-Designor (SDP) та DataBase Designer (ORACLE). Засоби проектування баз даних є також у складі CASE-засобів Vantage Team Builder, Designer/2000, Silverrun та PRO-IV;

• засоби розробки програм. До них відносяться кошти 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) та ін.) та генератори кодів, що входять до складу Vantage Team Builder, PRO-IV та частково - у Silverrun;

• засоби реінжинірингу, що забезпечують аналіз програмних кодів та схем баз даних та формування на їх основі різних моделей та проектних специфікацій. Засоби аналізу схем БД та формування ERD входять до складу Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin та S-Designor. В області аналізу програмних кодів найбільшого поширення набувають об'єктно-орієнтовані CASE-засоби, що забезпечують реінжиніринг програм мовою C++ (Rational Rose (Rational Software), Object Team (Cayenne)).

Допоміжні типи включають:

• засоби планування та управління проектом (SE Companion, Microsoft Project та ін.);

• засоби конфігураційного управління (PVCS (Intersolv));

• засоби тестування (Quality Works (Segue Software));

• засоби документування (SoDA (Rational Software)).

Зважаючи на різноманітну природу CASE-засобів, було б помилково робити будь-які твердження щодо реального задоволення тих чи інших

очікувань від їх впровадження. Можна перерахувати такі фактори, що ускладнюють визначення можливого ефекту від використання CASE-засобів:

• широке розмаїття якості та можливостей CASE-засобів;

• відносно невеликий час використання CASE-засобів у різних організаціях та нестача досвіду їх застосування;

• широка різноманітність у практиці впровадження різних організацій;

• відсутність детальних метрик та даних для вже виконаних та поточних проектів;

• широкий діапазон предметних галузей проектів;

• різний рівень інтеграції CASE-засобів у різних проектах.

Одні вважають, що реальна вигода від використання деяких типів CASE-засобів може бути отримана тільки після одно- або дворічного досвіду. Інші вважають, що вплив може реально виявитися у фазі експлуатації життєвого циклу інформаційних систем, коли технологічні покращення можуть призвести до зниження експлуатаційних витрат. Для успішного впровадження CASE-засобів організація повинна мати такі якості:

• Технологія. Розуміння обмеженості існуючих можливостей та здатність прийняти нову технологію;

• Культура. Готовність до впровадження нових процесів та взаємовідносин між розробниками та користувачами;

• Управління. Чітке керівництво та організованість по відношенню до найважливіших етапів та процесів впровадження.

Якщо організація не має хоча б однією з перелічених якостей, то використання CASE-засобів може закінчитися невдачею, незалежно від рівня ретельності проходження різним рекомендаціям по впровадженню. Для того, щоб прийняти виважене рішення щодо інвестицій у CASE-технологію, користувачі змушені проводити оцінку окремих CASE-засобів, спираючись на неповні та суперечливі дані. Ця проблема часто посилюється недостатнім знанням всіх можливих "підводних каменів" використання CASE-засобів. Серед найважливіших проблем виділяють такі:

• достовірна оцінка віддачі від інвестицій у CASE-засоби скрутна через відсутність прийнятних метрик та даних щодо проектів та процесів розробки ПЗ;

• використання CASE-засобів може бути тривалий процес і не принести негайної віддачі. Можливе навіть короткострокове зниження продуктивності внаслідок зусиль, що витрачаються на впровадження. Внаслідок цього керівництво організації-користувача може втратити інтерес до CASE-коштів та припинити підтримку їх впровадження;

• відсутність повної відповідності між тими процесами та методами, що підтримуються CASE-засобами, та тими, що використовуються в даній організації, може призвести до додаткових труднощів;

• CASE-засоби найчастіше важко використовувати у комплексі з іншими подібними засобами. Це як різними парадигмами, підтримуваними різноманітними засобами, і проблемами передачі і управління від одного засобу до іншого;

• деякі CASE-засоби вимагають надто багато зусиль для того, щоб виправдати їх використання у невеликому проекті, проте можна отримати вигоду з тієї дисципліни, до якої зобов'язує їх застосування;

• Негативне ставлення персоналу до впровадження нової CASE-технології може бути головною причиною провалу проекту.

Користувачі CASE-засобів повинні бути готові до необхідності довгострокових витрат на експлуатацію, частої появи нових версій та можливого швидкого морального старіння коштів, а також постійних витрат на навчання та підвищення кваліфікації персоналу. Грамотне, продумане і обґрунтоване використання CASE-технології здатне принести такі вигоди:

• високий рівень технологічної підтримки процесів розробки та супроводу ПЗ;

• позитивний вплив на деякі або всі з цих факторів: продуктивність, якість продукції, дотримання стандартів, документування;

• прийнятний рівень віддачі від інвестицій у CASE-засоби.

Отже, ви зважилися на використання CASE-засобів. Процес впровадження складається з наступних етапів:

• визначення потреб у CASE-засобах;

• оцінка та вибір CASE-засобів;

• виконання пілотного проекту;

• практичне використання CASE-засобів.

Визначення потреб у CASE-засобах можна проілюструвати наступною діаграмою (рис. 1).

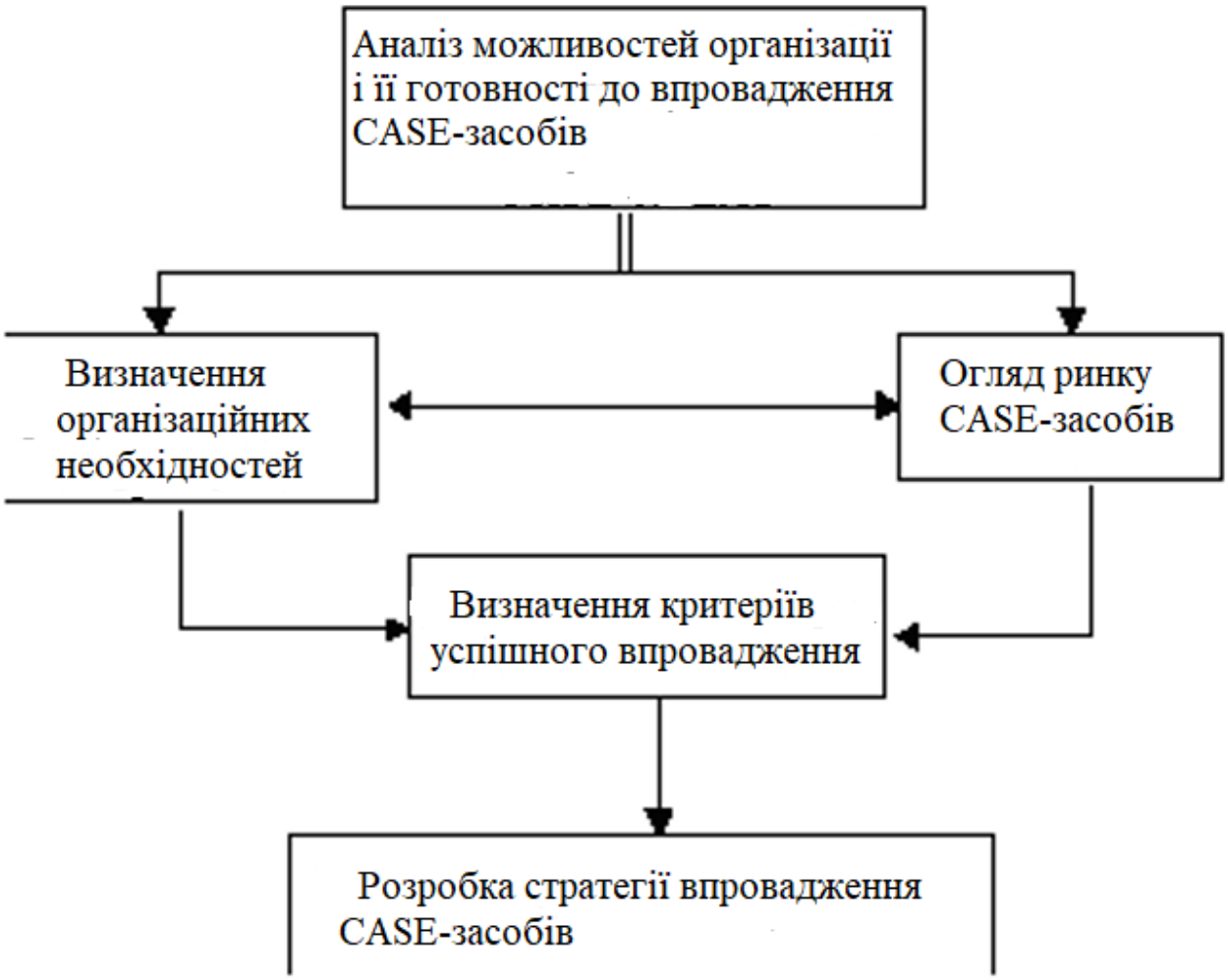


Рис. 1

Цей етап включає досягнення розуміння потреб організації та технології подальшого процесу впровадження CASE-засобів. Він повинен призвести до виділення тих сфер діяльності організації, в яких застосування CASE-засобів може принести реальну користь. Результатом цього етапу є документ, визначальний стратегію застосування. Процес оцінки та вибору CASE-засобів можна розглянути у вигляді моделі. Цей процес може переслідувати кілька цілей і включати:

• оцінку кількох CASE-засобів та вибір одного або більше з них;

• оцінку одного або більше CASE-засобів та збереження результатів для подальшого використання;

• вибір одного чи більше CASE-засобів із використанням результатів попередніх оцінок.

Нижче наведено діаграму, що описує найбільш загальну ситуацію оцінки та вибору, а також показує залежність між ними (рис. 2).

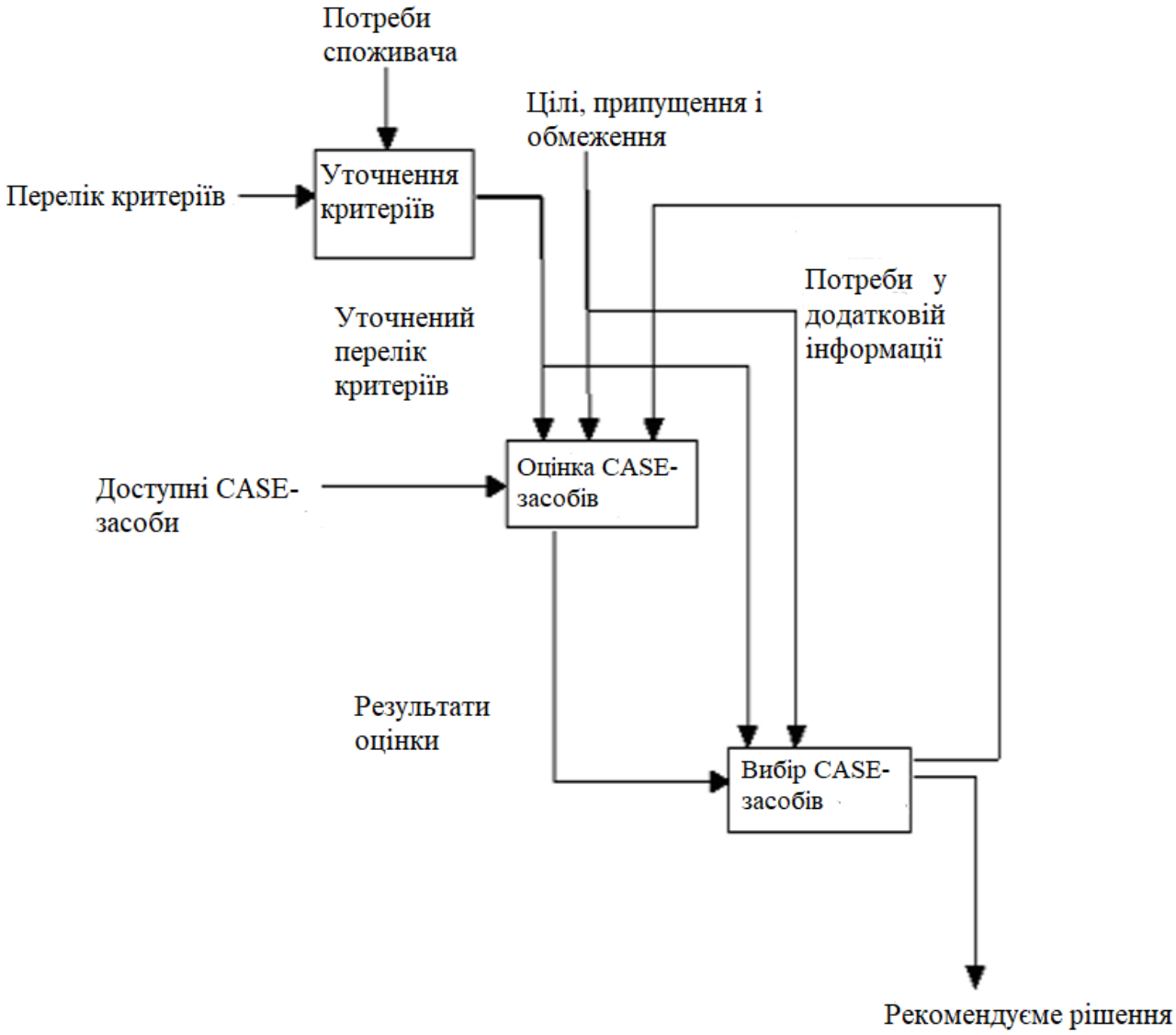


Рис. 2

Як очевидно з малюнку, вхідною інформацією для процесу оцінки є:

• визначення потреб користувача;

• цілі та обмеження проекту;

• дані про доступні CASE-засоби;

• список критеріїв, що використовуються у процесі оцінки.

Результати оцінки можуть містити результати попередніх оцінок. При цьому не слід забувати, що набір критеріїв, що використовуються під час попередньої оцінки, повинен бути сумісним з поточним набором. Конкретний варіант реалізації процесу (оцінка та вибір, оцінка для майбутнього вибору або

вибір, що ґрунтується на попередніх оцінках) визначається переліченими вище цілями.

Елементи процесу включають:

• цілі, припущення та обмеження, які можуть уточнюватися під час процесу;

• потреби користувачів, що відображають кількісні та якісні вимоги користувачів до CASE-засобів; • критерії, що визначають набір параметрів, відповідно до яких проводиться оцінка та ухвалення рішення про вибір;

• формалізовані результати оцінок одного чи більше засобів;

• рекомендоване рішення (зазвичай або про вибір, або подальша оцінка).

Процес оцінки та/або вибору слід розпочинати лише тоді, коли особа, група чи організація повністю визначила для себе конкретні потреби та формалізувала їх у вигляді кількісних та якісних вимог у заданій предметній галузі. Далі термін "потреби користувача" означає саме такі формалізовані вимоги. Користувач повинен визначити конкретний порядок дій та прийняття рішень із будь-якими необхідними ітераціями. Наприклад, процес можна подати у вигляді дерева рішень з його послідовним обходом та вибором підмножин кандидатів для більш детальної оцінки. Опис послідовності дій має визначати потік даних з-поміж них.

Визначення списку критеріїв засноване на користувацьких вимогах і включає:

• вибір критеріїв для використання наведеного далі переліку;

• визначення додаткових критеріїв;

• визначення сфери використання кожного критерію (цінка, вибір або обидва процеси);

• визначення однієї чи більше метрик кожного критерію оцінки;

• призначення ваги кожному критерію під час вибору.

Перед повномасштабним використанням обраного CASE-кошти в організації виконується пілотний проект. Його мета — експериментальна перевірка правильності рішень, ухвалених на попередніх етапах, та підготовка до впровадження. Пілотний проект є початковим реальним використанням CASE-засобу і зазвичай передбачає більш широкий масштаб використання CASE-засобу по відношенню до того, якого було досягнуто під час оцінки.

Пілотний проект повинен мати багато характеристик реальних проектів, для яких призначений даний засіб. Він має такі цілі:

1. підтвердити достовірність результатів оцінки та вибору;

2. визначити, чи дійсно CASE-засіб підходить для використання в даній організації, і якщо так, то визначити найбільш відповідну сферу його застосування;

3. зібрати інформацію, необхідну розробки плану практичного застосування;

4. Набути власний досвід використання CASE-кошти.

Пілотний проект дозволяє отримати важливу інформацію, необхідну для оцінки якості функціонування CASE-засобу та його підтримки з боку постачальника після встановлення. Його реалізацію можна проілюструвати наступною схемою (рис. 3).

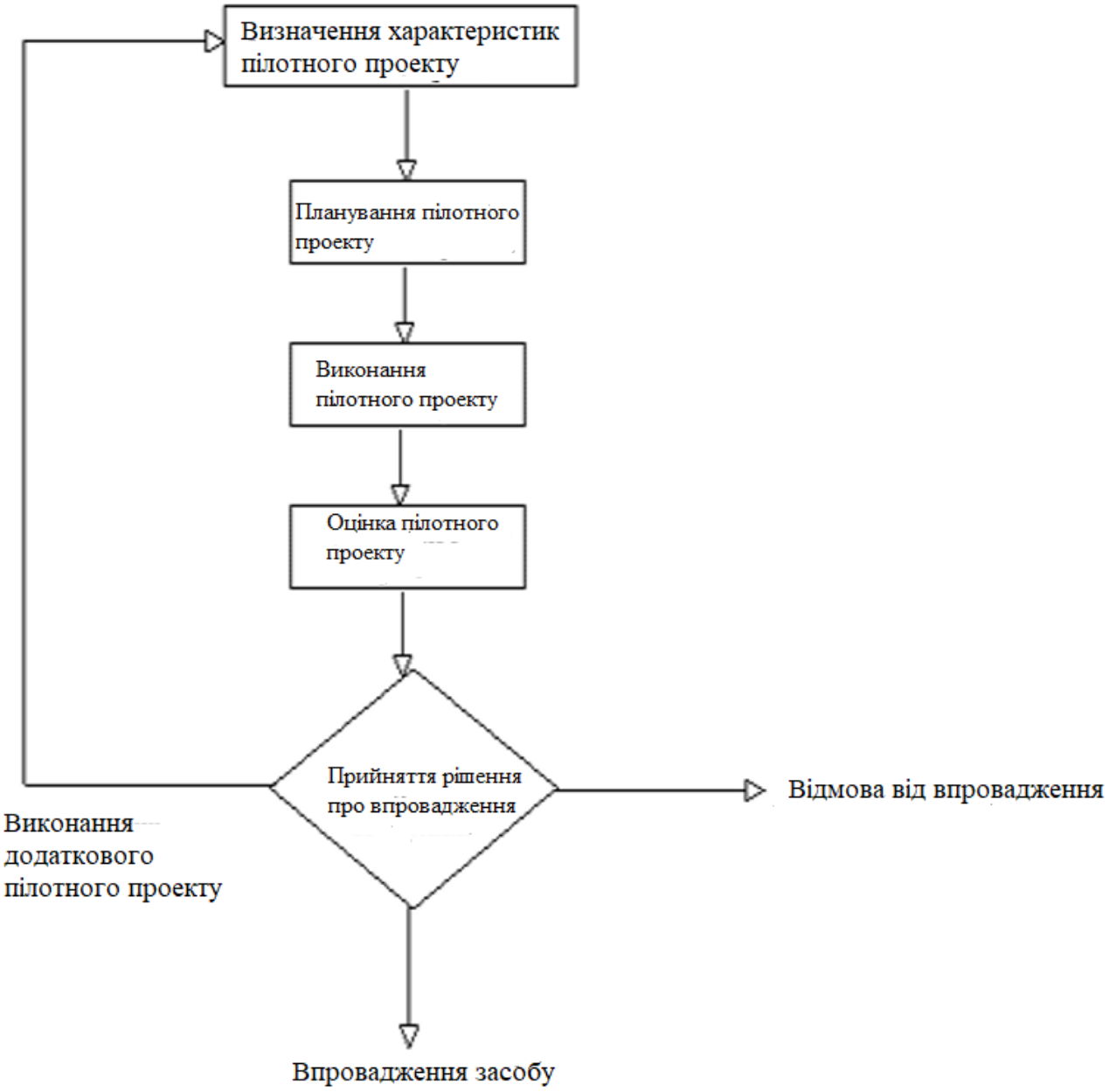


Рис. 3

Важливою функцією пілотного проекту є ухвалення рішення щодо придбання чи відмови від використання CASE-засобу. Провал пілотного проекту дозволяє уникнути більш значних і дорогих невдач надалі, оскільки він зазвичай пов'язаний із придбанням щодо невеликої кількості ліцензій та навчанням вузького кола фахівців. Ну і нарешті настає перехід до практичного використання CASE-засобів. Він починається з розробки та подальшої реалізації плану переходу. План переходу має включати таке:

• Інформація щодо цілей, критеріїв оцінки, графіка та можливих ризиків, пов'язаних з реалізацією плану.

• Інформація щодо придбання, встановлення та налаштування CASE-засобів.

• Інформацію щодо інтеграції кожного засобу з існуючими засобами, включаючи як інтеграцію CASE-засобів один одним, так і їх інтеграцію в процеси розробки та експлуатації програмного забезпечення, що існують в організації.

• Очікувані потреби в навчанні та ресурси, які використовуються протягом та після завершення процесу переходу.

• Визначення стандартних процедур використання коштів.

Реалізація плану переходу вимагає постійного моніторингу використання CASE-засобів, забезпечення поточної підтримки, супроводу та оновлення коштів у міру потреби. Досягнуті результати повинні періодично піддаватися експертизі відповідно до графіка, а план переходу — коригуватися за необхідності. Необхідно постійно приділяти увагу задоволенню потреб організації та критеріям успішного впровадження CASE-засобів. Значною та невід'ємною частиною реалізації плану є також навчання та перенавчання. Як правило, всі розуміють: навчання є центральною ланкою, що забезпечує нормальне використання CASE-засобів в організації. Проте існує досить поширена помилка, що початкове навчання необхідне групи непідготовлених користувачів, тому все обмежується мінімальним поточним навчанням. Учасники пілотного проекту, які мають початкове навчання, можуть бути висококваліфікованими ентузіастами нової технології, які прагнуть використовувати її будь-що-будь. З іншого боку, розробникам, які надалі візьмуть участь у проекті, може знадобитися більш інтенсивне та глибоке навчання, а також поточна підтримка використання коштів. На додаток до цього слід зазначити, що кожна категорія співробітників (наприклад, адміністратори коштів, служба підтримки робочих місць, інтегратори коштів, служба супроводу та розробники додатків) потребує різного навчання. Навчання не повинно замикатися тільки на користувачах CASE-засобів, навчатися повинні і ті співробітники, на діяльність яких так чи інакше впливає використання CASE- засобів. При подальшому застосуванні CASE-засобів організація повинна орієнтуватися навчання як співробітників, знову прийнятих працювати, і фахівців, виконують проекти з допомогою цих коштів. Саме тому навчання має стати невід'ємною частиною нормативних матеріалів щодо діяльності організації, які пропонуються новим співробітникам. Щоб визначити, наскільки ефективно новий CASE-засіб підвищує продуктивність та/або якість, організація повинна спиратися на деякі базові дані. На жаль, лише небагато організацій нині накопичують дані для реалізації програми поточної кількісної оцінки та вдосконалення процесів.

Для доказу ефективності CASE-засобів та їх можливостей покращувати продуктивність необхідні такі базові метричні дані:

• використаний час;

• час, виділений персонально для спеціалістів;

• розмір, складність та якість ПЗ;

• зручність супроводу.

Ще до початку впровадження CASE-засобів метрична оцінка має починатися з реальної оцінки поточного стану середовища та підтримувати процедури постійного накопичення даних. Період, протягом якого виконується кількісна оцінка впливу, що надається впровадженням CASE-засобів, є вельми значущою величиною з точки зору визначення ступеня успішності переходу. Деякі організації, які успішно впровадили зрештою CASE-засоби, зіткнулися з короткочасними негативними ефектами на початку процесу. Інші, успішно розпочавши, недооцінили довгострокові витрати на супровід та навчання. Таким чином, щоб подолати будь-які негативні ефекти на початковому етапі, а також змоделювати майбутні довготривалі витрати, найбільш прийнятний часовий інтервал для оцінки ступеня успішності впровадження має бути досить великим. З іншого боку, цей інтервал має відповідати цілям організації та очікуваним результатам. Зрештою, досвід, отриманий при впровадженні CASE-засобів, може частково змінити цілі організації та очікування, що покладаються на CASE- засоби. Наприклад, організація може зробити висновок, що кошти доцільно використовувати для більшого чи меншого кола користувачів та процесів у циклі створення та супроводу ПЗ. Такі зміни в очікуваннях часто дають позитивні результати, але можуть і внести відповідні корективи визначення ступеня успішного впровадження CASE-засобів. Підсумком цього етапу є впровадження CASE-засобів у повсякденну практику організації, причому більше не потрібно якогось спеціального планування. Крім того, підтримка CASE-засобів включається до плану поточної підтримки ПЗ у цій організації.

6.2 Огляд CASE-засобів

Накопичений досвід проектування ІС показує, що це логічно складна, трудомістка і тривала за часом робота, яка вимагає високої кваліфікації фахівців, які беруть участь в ній [2]. Усе це сприяло появі різних технологій програмування і програмно-технологічних засобів спеціального класу – CASE-засобів (Computer Aided Software Engineering), реалізовуючих CASE-технологию створення і супроводу ІС.

Термін CASE має широке тлумачення. Первинне значення терміну CASE обмежувалося питаннями автоматизації розробки тільки програмного забезпечення, а нині воно набуло нового сенсу і охоплює процес розробки складних ІС в цілому. Тепер під терміном CASE- засобів розуміються програмні засоби, які підтримують процеси створення і супроводу ІС відповідно до інформаційних потреб користувачів, включаючи: аналіз і формулювання вимог, в наочній формі моделювання предметної області, аналіз цієї моделі на усіх етапах розробки і супроводу ІС.

Розпочнемо з пакету , який є фактичним стандартом в області UML-проектування.

IBM Rational Rose

Rational Rose – сучасний і потужний засіб аналізу, моделювання і розробки програмних систем. Rational Rose згодиться при рішенні практично будь-яких завдань проектування інформаційних систем: від аналізу бізнес-процесів до *кодогенерації* на певній мові програмування. Такий арсенал дозволить не лише спроектувати нову систему, але і допрацювати стару, зробивши процес *зворотного проектування*.

Випускається декілька версій продукту.

1. Rational Rose Modeler. Ця версія дозволить аналітикам і проектувальникам проводити аналіз бізнес-процесів і проектувати систему. Ця редакція не підтримує *кодогенерацію*.

2. Rational Rose Professional. Як видно з назви, це професійна редакція продукту. Залежно від вибраної мови програмування дозволяє виконувати пряме і *зворотне проектування*. Rose Professional замовляється тільки в певній конфігурації (наприклад, Rose Professional C++ або Rose Professional C++ DataModeler). Rational Rose Professional, звичайно, не створює 100 % виконуваного коду. На виході розробник отримує каркасний код інформаційної системи на певній (замовленій) мові програмування, яку згодом треба ще програмувати і програмувати. Продукт націлений і на аналітиків, і на розробників.

3. Rational Rose RealTime. Версія продукту, створена спеціально для отримання 100% виконуваного коду в реальному масштабі часу. RealTime дозволяє проводити пряме і *зворотне проектування* на мовах C або C++. За завіреннями розробників, на виході модель автоматично компілюється і збирається у виконуваний файл. Продукт призначений саме для розробників.

4. Rational Rose Enterprise. Абсолютно повна версія. Підтримуються всі функції інших редакцій, за винятком можливості 100% *кодогенерацій.* Таким чином, ця версія продукту покриває весь спектр завдань з проектування, аналізу і кодогенерації. Це програмний пакет для всіх учасників проекту.

5. Rational Rose DataModeler. Це не конкретний варіант продукту, а функціональність з проектування баз даних. Функції DataModeler входять до

складу Rose Enterprise або Professional.

На жаль, немає безкоштовної версії продукту, але для освітніх установ все програмне забезпечення IBM доступно безкоштовно (для використання з навчальною метою) в рамках програми IBM Academic Initiative.

Залежно від постачання, в Rational Rose може бути розширений або звужений набір візуальних компонент (можливих діаграм). Втім, Rational Rose і так досить функціональний. Основні можливості продукту:

1. Пряме і *зворотне проектування* на мовах: ADA, Java, C, C++, Basic.
2. Підтримка технологій COM, DDL, XML.
3. Можливість генерації схем БД Oracle і SQL.

Інформацію про продукти Rational можна знайти на сайті: http://www-306.ibm.com/software/rational/ Це офіційний сайт Rational, де ви зможете знайти інформацію про Rational Rose та інші продукти Rational.

Borland Together

Borland Together ControlCenter – це інтегрована платформа розробки, що дозволяє спростити і прискорити *аналіз, дизайн, розробку* і *розгортання* комплексних корпоративних застосувань. Ці можливості поєднуються в одному інтегрованому рішенні з підтримкою UML, що допомагає командно розробляти високоякісні системи швидше і ефективніше. Деякі особливості Borland Together:

1. Підтримка XP («екстремальне програмування»). Together підтримує «гнучкі» процеси моделювання, а також надає інтерактивні можливості моделювання і підтримує всі види діаграм UML.

2. Прискорення процесів розробки шляхом застосування патернів.

Ще одна модна тенденція в програмній інженерії: використання патернів, або шаблонів проектування, – деяких стандартних рішень, зразків в області проектування. Використовуючи ці зразки, експерт або розробник може швидко створити модель і привести її у відповідність з корпоративними стандартами і кращими практиками кодування.

3. Розгортання додатків на декілька серверів виконується швидко, без перекодування. Додаток можна розгорнути на декілька серверів додатків, просто написавши декілька рядків. З Together ControlCenter додаток може бути побудований для одного сервера додатків і легко перемкнутий на інший, розгорнутий на складній інфраструктурі.

4. Функція контролю якості полегшує життя розробників. Вбудоване *функціональне тестування* допомагає виявити проблеми ще в процесі розробки, що дійсно важливо, оскільки вартість виправлення помилок тим вище, чим пізніше вони виявлені.

З вищесказаного ясно, що Borland Together – це щось значно більше, ніж просто пакет для малювання «картинок в стилі UML».

Інформацію про продукти Borland Together можна знайти на сайті: http://www.borland.com/together/

Microsoft Visio

Visio – рішення для побудови діаграм від Microsoft. За словами розробників, Visio допомагає перетворити технічні і бізнес-концепції в візуальну форму.

Можливості Visio дійсно широкі. Використовуючи зумовлені фігури Visio Professional, drag – and – drop і майстри, можна швидко і просто створювати зрозумілі і інформативні діаграми. Можливості Visio можна легко розширювати, використовуючи нові шаблони бізнес-діаграм. Можна включати зовнішні джерела даних, сховища або колекції шаблонів, які зберігаються.

Visio Professional пов'язане з Microsoft Office Project, що дозволяє, наприклад, імпортувати звідти інтегруєме завдання для членів команди. За допомогою шаблонів UML можна створювати UML-диаграммы статичної структури ПЗ або проводити *зворотне проектування* за допомогою Visio 2003 Reverse Engineer Wizard.

Інформацію про продукти Microsoft Visio можна знайти на сайті: http://office.microsoft.com/en-us/FX 010857981033.aspx

StarUML, Dia, Draw.io

Чудові і **абсолютно безкоштовні**) засоби UML-моделювання.

StarUML – це пакет з відкритим програмним кодом, написаний на Delphi і працюючий під керуванням ОС сімейства Windows. StarUML підтримує UML 2.0 (плюс його профайли) і MDA (Model Driven Architecture). Функціонал пакету можна розширити за рахунок використання плагінів, так що кожен охочий може створити свій власний модуль для StarUML на будь-якій COM-сумісній мові (C++, Delphi, C#, ...).

Dia – вільний кросплатформенный редактор діаграм, частина GNOME Office, але може бути встановлений незалежно. Підтримує українську і російську мови. Він може бути використаний для створення різних видів діаграм: блок-схем алгоритмів програм, статичних структур UML, баз даних, діаграм суть-зв'язок, радіоелектронних елементів, потокових діаграм, мережевих діаграм і інших.

На http://younglinux.info/book/export/html/169 є підручник Dia російською мовою.

Draw.io - інструмент для створення діаграм UML і блок-схем **онлайн.** Нагадує MS Visio і можливо зроблений під нього, але додаток від Microsoft - платна, а онлайн сервіс Draw.io - безкоштовний. Сервіс підтримує декілька мов. Є версія з російським інтерфейсом. Сервіс дуже простий і зручний, інтерфейс приємний і зрозумілий.

З допомогою онлайн сервісу Draw.io можна створювати: Діаграми; Моделювання на UML; Графіки; Блок-схеми; Форми; Вставляти в діаграму зображення. Усі створені діаграми ви можете зберегти на комп'ютер, в Google Drive або Dropbox.

Основні відомості про CASE-засоби Rational Rose

Введення в Rational Rose

Rational Rose – сімейство об'єктно-орієнтованих CASE-засобів фірми Rational Software Corporation (у 2003 році її поглинула корпорація IBM) – призначене для автоматизації процесів аналізу і проектування програмного хабезпечення, а також для генерації кодів на різних мовах і випуску проектної документації. Rational Rose використовує метод об'єктно-орієнтованого аналізу і проектування, заснований на мові UML.

Поточна версія Rational Rose реалізує генерацію кодів програм для C++, Visual C++, Visual Basic, Java, PowerBuilder, CORBA Interface Definition Language (IDL), генерацію описів баз даних для ANSI SQL, Oracle, MS SQL Server, IBM DB2, Sybase, а також дозволяє розробляти проектну документацію вигляді діаграм і специфікацій. Крім того, Rational Rose містить засоби реверсного інжинірингу програм і баз даних.

Структура і функції. У основі роботи Rational Rose лежить побудова діаграм і специфікацій UML, що визначають архітектуру системи, її статичні і динамічні аспекти. В складі Rational Rose можна виділити шість основних структурних компонентів: репозиторій, графічний інтерфейс користувача, засоби перегляду проекту (браузер), засоби контролю проекту, засоби збору статистики і генератор документів. До них додаються генератор кодів (індивідуальний для кожної мови) і аналізатор для C++, що забезпечує реверсный інжиніринг.

Засоби автоматичної генерації кодів програм на мові C++, використовуючи інформацію, яка міститься в діаграмах класів і компонентів, формують файли заголовків і файли описів класів і об'єктів. Створюваний таким чином скелет програми може бути уточнений шляхом прямого програмування на мові C++.

Аналізатор кодів C++ реалізований в вигляді окремого програмного модуля. Його призначення – створювати модулі проектів Rational Rose на основі інформації, яка міститься в визначуваних користувачем початкових текстах на C++. В процесі роботи аналізатор здійснює контроль правильності початкових текстів і діагностику помилок.

Аналізатор має широкі можливості налаштування по входу і виходу. Наприклад, можна визначити типи початкових файлів, базовий компілятор, задати, яка інформація має бути включена у формовану модель, і які елементи вихідної моделі слід виводити на екран. Таким чином, Rational Rose/C++ забезпечує можливість повторного використання програмних компонентів.

В результаті розробки проекту за допомогою CASE-засобу Rational Rose формуються такі документи:

– діаграми UML, які в сукупності є моделлю розроблюваної програмної системи;

– специфікації класів, об'єктів, атрибутів і операцій;

– заготовки текстів програм.

Тексти програм є заготовками для подальшої роботи програмістів. Склад інформації, яка включається в програмні файли, визначається або за замовчуванням, або на розсуд користувача.

Взаємодія з іншими засобами і організація групової роботи. Для підтримки командної роботи над проектом на кожній стадії життєвого циклу ПЗ є інтегрований набір продуктів Rational Suite. Rational Suite існує в наступних варіантах:

– Rational Suite AnalystStudio – призначений для визначення і керування повним набором вимог до розроблюємої системи;

– Rational Suite DevelopmentStudio – призначений для проектування і реалізації ПЗ;

– Rational Suite TestStudio – є набір продуктів, призначених для автоматичного тестування додатків;

– Rational Suite Enterprise – забезпечує підтримку повного життєвого циклу ПЗ і призначений як для менеджерів проекту, так і для окремих розробників, які виконують декілька функціональних ролей.

Робота в середовищі Rational Rose

Елементи екрану.

П’ять основних елементів інтерфейсу Rose – це браузер, вікно документації, панелі інструментів, вікно діаграми і журнал (log).

Їх призначення полягає в наступному:

– браузер (browser) – використовується для швидкої навігації по моделі;

– вікно документації (documentation window) – застосовується для роботи з текстовим описом елементів моделі;

– панелі інструментів (toolbars) – застосовуються для швидкого доступу до найпоширеніших команд;

– вікно діаграми (diagram window) – використовується для перегляду і редагування однієї або декількох діаграм UML;

– журнал (log) – застосовується для перегляду помилок і звітів про результати виконання різних команд.

На рис.1 показані різні частини інтерфейсу Rose.

Браузер. Браузер – це ієрархічна структура, яка дозволяє здійснювати навігацію по моделі. Все, що додається в модель – дійові особи, варіанти використання, класи, компоненти – буде показано в вікні браузеру. За допомогою браузеру можна:

– додавати в модель елементи (дійові особи, варіанти використання, класи, компоненти, діаграми і так далі);

– переглядати існуючі елементи і зв’язки між елементами моделі;

– переміщувати й перейменовувати елементи моделі;

– додавати елементи моделі до діаграми;

- групувати елементи в пакети.

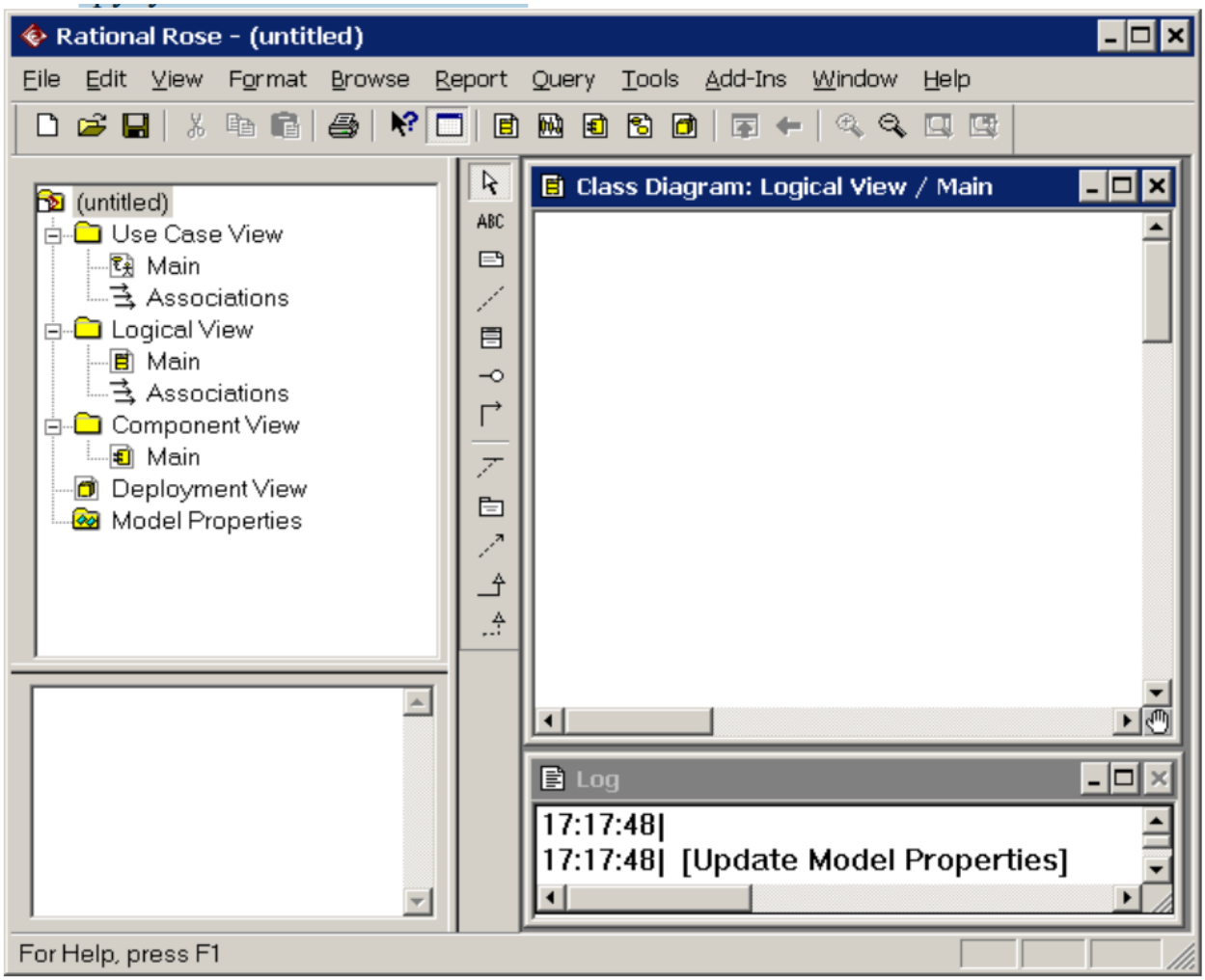


Рисунок 1 – Інтерфейс Rational Rose

Браузер підтримує чотири представлення (view): представлення варіантів використання, компонентів, розміщення і логічне представлення. Всі вони і елементи моделі, які містяться в них, описані нижче.

Браузер організований в деревовидному стилі. Кожен елемент моделі може містити інші елементи, які знаходяться нижче його в ієрархії. Знак «-» біля елементу означає, що його гілка повністю розкрита. Знак «+» – що його гілка згорнута.

Вікно документації. За його допомогою можна документувати елементи моделі Rose. Наприклад, можна зробити короткий опис кожної дійової особи. При документуванні класу все, що буде написано у вікні документації, з’явиться потім в вигляді коментаря в згенерованому коді, що позбавляє від необхідності згодом вносити ці коментарі вручну. Документація виводитиметься також у звітах, які створюються в середовищі Rose.

Панелі інструментів. Панелі інструментів Rose забезпечують швидкий доступ до найпоширеніших команд. В цьому середовищі існує два типи панелей інструментів: стандартна панель і панель діаграми. Стандартну панель видно завжди, її кнопки відповідають командам, які можуть використовуватися для роботи з будь-якою діаграмою. Панель діаграми своя для кожного типу діаграм UML.

Всі панелі інструментів можуть бути змінені і налагоджені користувачем. Для цього виберіть пункт меню Tools > Options, потім виберіть вкладку Toolbars.

Щоб показати або приховати стандартну панель інструментів (чи панель інструментів діаграми):

1. Виберіть пункт Tools > Options.
2. Виберіть вкладку Toolbars.
3. Щоб зробити видимою або невидимою стандартну панель інструментів, помітьте (чи зніміть позначку) контрольний перемикач Show Standard ToolBar (чи Show Diagram ToolBar).

Щоб збільшити розмір кнопок на панелі інструментів:

1. Клацніть правою кнопкою миші на необхідній панелі.
2. Виберіть в спливаючому меню пункт Use Large Buttons (Використати великі кнопки).

Щоб налаштувати панель інструментів:

1. Клацніть правою кнопкою миші на необхідній панелі.
2. Виберіть пункт Customize (налаштувати)
3. Щоб додати або видалити кнопки, виберіть відповідну кнопку і потім клацніть мишею на кнопці Add (додати) або Remove (видалити), як показано на рис. 2.

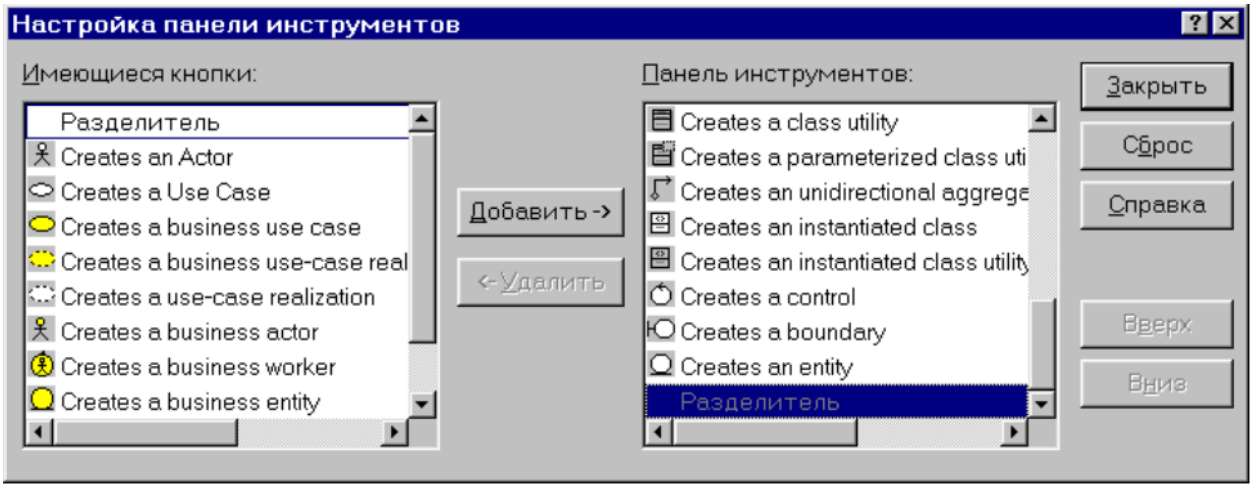


Рисунок 2 – Налаштування стандартної панелі інструментів

Вікно діаграми. В вікні діаграми видно, як виглядає одна або декілька діаграм UML моделі. При внесенні в елементи діаграми змін Rose автоматично відновить браузер. Аналогічно, при внесенні змін до елементу за допомогою браузеру Rose автоматично відновить відповідні діаграми. Це допомагає підтримувати модель в несуперечливому стані.

Журнал. В міру роботи над вашою моделлю певна інформація спрямовуватиметься в вікно журналу. Наприклад, туди поміщуються повідомлення про помилки, які виникають при генерації коду.

Не існує способу закрити журнал зовсім, але його вікно може бути мінімізоване.

Чотири представлення моделі Rose.

В моделі Rose підтримується чотири представлення (views) – представлення варіантів використання, логічне представлення, представлення компонентів і представлення розміщення. Кожне з них призначене для своїх цілей і для відповідної аудиторії. В подальшому ми коротко розглянемо кожне з вказаних представлень.

Представлення варіантів використання. Це представлення містить всіх дійових осіб, усі варіанти використання і їх діаграми для конкретної системи. Воно може також містити деякі діаграми послідовності і кооперативні діаграми.

На рис. 3 показано, як виглядає представлення варіантів використання в браузері Rose.

Представлення варіантів використання містить:

1. Дійових осіб.
2. Варіанти використання.
3. Документацію по варіантах використання, що деталізує процеси (потоки подій), які відбуваються в них, включаючи оброблення помилок.

Піктограмами зображаються зовнішні файли, які прикріплені до моделі Rose. Вид піктограми, залежить від додатку, використовуваного для документування потоку подій. В даному випадку (рис. 3) застосовувався Microsoft Word.

4. Діаграми варіантів використання. Зазвичай у системи буває декілька таких діаграм, кожна з яких показує підмножину дійових осіб і/або варіантів використання.
5. Пакети, які є групами варіантів використання і/або діючих осіб.

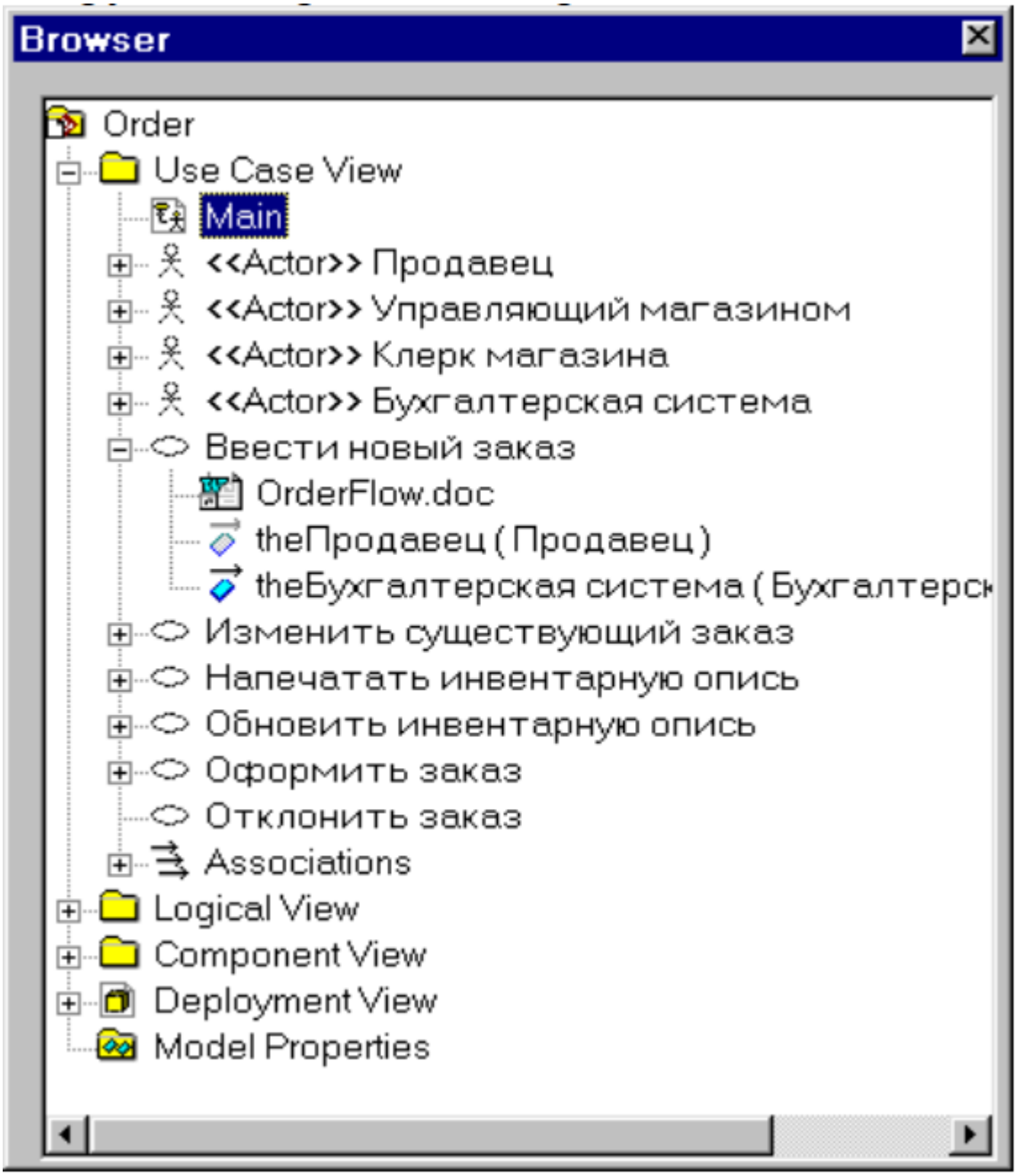


Рисунок 3 – Представлення варіантів використання

Логічне представлення. Логічне представлення, яке показане на рис. 4, концентрується на тому, як система реалізовуватиме поведінку, описану в варіантах використання. Воно дає детальну картину складових частин системи і описує взаємодію цих частин. Логічне представлення включає, окрім іншого, конкретні необхідні класи, діаграми класів і діаграми станів. За їх допомогою конструюється детальний проект створюваної системи.

Логічне представлення містить:

1. Класи.
2. Діаграми класів. Як правило, для опису системи використовується декілька діаграмів класів, кожна з яких відображає деяку підмножину всіх класів системи.

3. Діаграми взаємодії, вживані для відображення об’єктів, які беруть участь в одному потоці подій варіанту використання.

4. Діаграми станів.

5. Пакети, що є групами взаємозв’язаних класів.

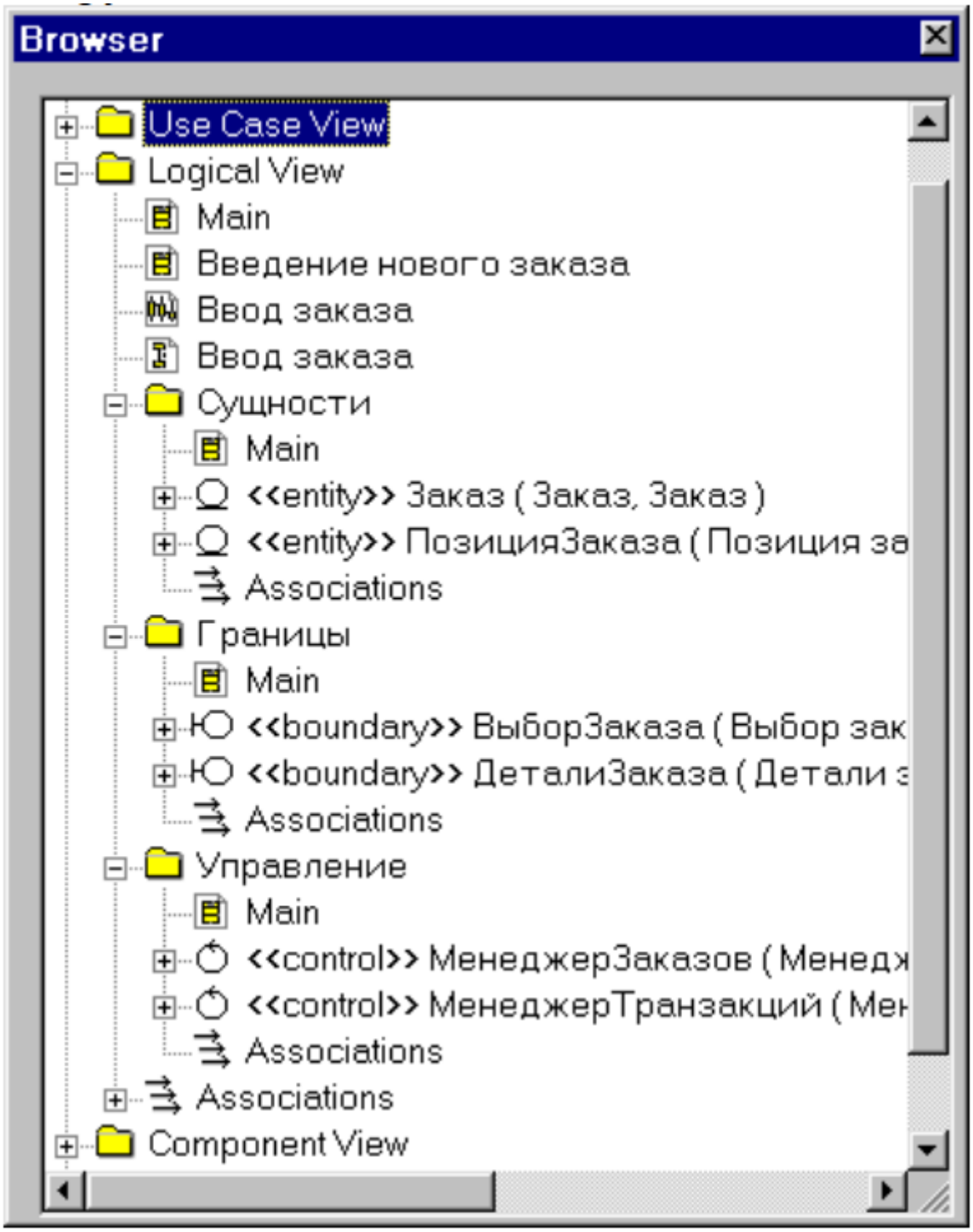


Рисунок 4 – Логічне представлення системи

Представлення компонентів. Представлення компонентів містить:

1. Компоненти, які є фізичними модулями коду.
2. Діаграми компонентів.
3. Пакети, що є групами пов’язаних компонентів.

Представлення розміщення. Останнє представлення Rose – це представлення розміщення. Воно відповідає фізичному розміщенню системи, яке може відрізнятися від її логічної архітектури.

В представлення розміщення входять:

1. Процеси, що є потоками (threads), які виконуються у відведеній для них області пам'яті.
2. Процесори, які включають будь-які комп'ютери, здатні обробляти дані. Будь-який процес виконується на одному або декількох процесорах.
3. Пристрої, тобто будь-яка апаратура, не здатна оброблювати дані. До таких пристроїв належать, наприклад, термінали введення-виведення і принтери.
4. Діаграма розміщення.

Параметри налаштування відображення.

Зображення атрибутів і операцій на діаграмах класів. У Rose є можливість налаштувати діаграми класів так, щоб:

1. Показувати всі атрибути і операції.
2. Приховати операції.
3. Приховати атрибути.
4. Показувати тільки деякі атрибути або операції.
5. Показувати операції разом з їх повними сигнатурами або тільки їх імена.
6. Показувати або не показувати видимість атрибутів і операцій.
7. Показувати або не показувати стереотипи атрибутів і операцій.

Значення кожного параметра за умовчанням можна задати за допомогою вікна, що відкривається при виборі пункту меню Tools > Options.

В цього класу на діаграмі можна:

– показати всі атрибути;
– приховати всі атрибути;
– показати тільки вибрані атрибути;
– подавити виведення атрибутів.

Подавлення виведення атрибутів призведе не лише до зникнення атрибутів з діаграми, але і до видалення лінії, яка показує місце розташування атрибутів в класі.

Існує два способи зміни параметрів представлення атрибутів на діаграмі. Можна встановити потрібні значення в кожного класу індивідуально. Можна також змінити значення потрібних параметрів за замовчуванням до початку створення діаграми класів. Внесені таким чином зміни вплинуть тільки на новостворювані діаграми.

Щоб показати всі атрибути класу:

1. Виділіть на діаграмі потрібний клас.
2. Клацніть на нім правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. В ньому оберіть Options > Show All Attributes.

Щоб показати в класу тільки обрані атрибути:
1.Виділіть на діаграмі потрібний клас.
2.Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3.У ньому оберіть Options > Select Compartment Items.
4.Вкажіть потрібні атрибути в вікні Edit Compartment.

Щоб подавити виведення всіх атрибутів класу діаграми:
1. Виділіть на діаграмі потрібний клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. У ньому виберіть Options > Suppress Attributes.

Щоб змінити вигляд набраного за замовчуванням атрибуту:
1. В меню моделі виберіть пункт Tools > Options.
2. Перейдіть на вкладку Diagram.
3. Для установки значень параметрів відображення атрибутів за замовчуванням скористайтеся контрольними перемикачами Suppress Attributes і Show All Attributes. Зміна цих значень за замовчуванням вплине тільки на нові діаграми. Вид існуючих діаграм класів не зміниться.

Як і у разі атрибутів, є декілька варіантів представлення операцій на діаграмах.
1. Показати всі операції.
2. Показати тільки деякі операції.
3. Приховати всі операції.
4. Подавити виведення операцій.

Крім того, можна:
1. Показати тільки ім’я операції. Це означає, що на діаграмі буде представлено тільки ім’я операції, але не аргументи або тип значення, що повертається.
2. Показати повну сигнатуру операції. На діаграмі буде представлено не лише ім’я операції, але і усі її параметри, типи даних параметрів і тип значення операції, що повертається.

Щоб показати всі операції класу:
1. Виділіть на діаграмі потрібний клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. У ньому виберіть Options > Show All Operations.

Щоб показати тільки обрані операції класу:
1. Виділіть на діаграмі потрібний вам клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. У ньому виберіть Options > Select Compartment Items.
4. Вкажіть потрібні операції в вікні Edit Compartment.

Щоб подавити виведення всіх операцій класу діаграми:
1. Виділіть на діаграмі потрібний вам клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. В ньому виберіть Options > Suppress Operations.

Щоб показати на діаграмі класів сигнатуру операції:
1. Виділіть на діаграмі потрібний клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. У ньому виберіть Options > Show Operation Signature.

Щоб змінити набраного за умовчанням вигляду операції:
1. В меню моделі виберіть пункт Tools > Options.
2. Перейдіть на вкладку Diagram.
3. Для установки значень параметрів відображення операцій за замовчуванням скористайтеся контрольними перемикачами Suppress Operations, Show All Operations і Show Operation Signatures.

Щоб показати видимість атрибуту або операції класу:
1. Виділіть на діаграмі потрібний клас.
2. Клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
3. У ньому виберіть Options > Show Visibility.

Щоб змінити набутого за умовчанням значення параметра показу видимості:
1. В меню моделі виберіть пункт Tools > Options.
2. Перейдіть на вкладку Diagram.
3. Для установки параметрів відображення видимості за замовчуванням скористайтеся контрольним перемикачем Show Visibility.

Для перемикання між нотаціями видимості Rose і UML:
1. В меню моделі виберіть пункт Tools > Options.
2. Перейдіть на вкладку Notation.
3. Для перемикання між нотаціями скористайтеся перемикачем Visibility as Icons. Якщо цей перемикач помічений, використовуватиметься нотація Rose. Якщо ні – то нотація UML. Зміна цього параметру вплине тільки на нові діаграми. Існуючі діаграми класів залишаться попередніми.

Контрольні запитання

1. Призначення CASE-засобу *IBM Rational Rose*.
2. Які версії продукту *Rational Rose* ви знаєте?
3. Структура і функції *Rational Rose.*
4. Які п’ять основних елементів інтерфейсу *Rose* ви знаєте?
5. Призначення панелі інструментів *Rose.*
6. Назвіть основне призначення *Браузера* в *Rose.*

7. Грід та хмарні середовища для проектування ІС.

7.1 Введення в інформаційні технології Грід.

Аналізуючи історію розвитку інформаційних технологій та сучасні тенденції можна зробити висновок, що еволюційний виток ІТ, який почався разом з епохою великих обчислювальних машин (ЕОМ) понад п'ятдесят років тому, замкнувся – разом з хмарами ми повернулися до централізації ресурсів, але на цей раз не на рівні мейнфреймів, а на новому технологічному рівні.

Виступаючи на конференції, присвяченій проблемам сучасних процесорів, професор Массачусетського технологічного інституту Ананд Агарвал сказав: "Процесор – це транзистор сучасності. Новий рівень відрізняється тим, що тут також збираються мейнфрейми, але віртуальні, і не з окремих транзисторів, як півстоліття тому, а з цілих процесорів, або цілком з комп'ютерів. На зорі ІТ численні компанії та організації створювали власні комп'ютери з дискретних компонентів, монтуючи їх на саморобних друкованих платах - кожна організація робила свою машину, і ні про яку стандартизацію чи уніфікацію мови не могло бути. Зараз ситуація повторюється – точно так саме: з серверів, комп'ютерів, різноманітного мережевого обладнання збираються зовнішні та приватні хмари. Одночасно спостерігається та ж сама технологічна роз'єднаність і відсутність уніфікації: Microsoft, Google, IBM, Aptana, Heroku, Rackspace, Ning, Salesforce будують глобальні мейнфрейми, а хтось під власні потреби створює приватні хмари. Залишається припустити, що попереду винахід інтегральної схеми і мікропроцесора”.

Трохи історії.

Перша інтегральна мікросхема була сворена в вересні 1958р.

Через 6 років в 1965р один із засновників компанії Інтел Гордон Мур підмітив тенденцію, згідно якої кількість транзисторів на кристалі подвоювалась приблизно через рік, на основі чого був сформульований так званий закон Мура. В 1975р Мур підкоректував свій закон, по якому кількість транзисторів на чіпі інтегральної мікросхеми подвоюється через 2 роки – потім Інтел скорегував цей термін на 18 місяців.

Закон імперичний. Зараз він перестав виконуватись. Чому?

Перші мікросхеми створювались на технологічних нормах в декілька мікрометрів, микропроцесори Пентіум створювалися на технологічних нормах 0,6 мкм, а зараз технологічні норми досягли 1,2 нанометрів. Технологічні норми – це мінімальна ширина токопровідної доріжки в планарній інтегральній мікросхемі, яка технологічно освоєна в серійному виробництві. Досягнуті норми такі значні, що вже назріла криза в розвитку комп’ютерів із-за

атомарної природи твердого тіла. Подолання цієї кризи бачать в квантових обчисленнях, на що вже зараз татять величезні сили і ресурси.

Інформатизація сьогодні вступила в четвертий етап свого розвитку. Перший був пов'язаний з появою великих комп'ютерів, другий - зі створенням персональних комп'ютерів, третій - з появою Інтернету. Четвертий етап інформатизації включає ряд нових технологій.

Інтернет, WWW - World Wide Web, Grid і хмарні обчислення - пов'язані між собою, але різні технології:

• Інтернет - це глобальна мережа, що об'єднує безліч комп'ютерів і локальних (порівняно невеликих) мереж і дозволяє їм взаємодіяти один з одним.

• Веб (паутина) - це спосіб доступу до інформації, яка знаходиться на віддаленому, але включеному в Інтернет комп'ютері.

• Web-служби (Web Services) - це віддалені сервісні об'єкти, які реалізують за запитом користувача деяку функціональність.

Історично першими з'явилися великі комп’ютери, які в західній термінології називають “мейнфрейми”, у них відсутні обмеження на обсяг обладнання, тому їх називають ще великими ЕОМ. Це відбулося в 1964 році, коли компанія IBM створила перший комп’ютер сімейства IBM-360, які були багато років присутні на ринку ЕОМ. Слід зауважити, що в той час в Україні була створена Інститутом Кібернетики ЕОМ “Дніпро-2” - повний функціональний аналог IBM-360: 64-розрядна ЕОМ з 128-розрядною “плаваючою” арифметикою, оперативною пам’яттю до 65536 байт і повним набором периферійних пристроїв (пристрій паралельного друку АЦПУ-128, пристрій послідовного друку, пристрої вводу і виводу інформації на перфострічку, пристрої вводу та виводу на перфокарту, зовнішня пам’ять на магнітній стрічці в складі 8-ми накопичувачів, екранний пульт: алфавітно-цифровий та графічний та ін.). Ця ЕОМ (“Дніпро-2”) серійно вироблялась в Києві до 1971р. Незважаючи на це була прийнята постанова керівництва країни про припинення виробництва вітчизняної розробки і створення радянського повного клону IBM-360. Головний конструктор «Дніпра-2» ак.Глушков В.М. не зміг переконати керівництво тодішнього СРСР, що це змусить нас поступитися предовими позиціями і перейти в розряд відстачих. Так вітчизняна ІТ-індустрія перетворилася з передової в «вічнодоганяючу». За короткий час після постанови про створення вітчизняного клону IBM-360 досить швидко було розроблене сімейство-аналог: ЕС ЕОМ та ЕОМ АСВТ (ЕОМ агрегатної системи засобів обчислювальної техніки). В 1969р в Києві була створена, а в 1970 році в Україні почалося серійне виробництво повного аналога IBM-360 ЕОМ АСВТ М-3000.

СуперЕОМ - це продовження великих ЕОМ в бік збільшення числа АЛУ, процесорів, цілих машин, об'єднаних в одну систему для виконання паралельних обчислень в рамках однієї задачі. Такі ЕОМ включають сотні і тисячі процесорів, розташованих локально, так що ефективність визначається простим законом Амдала.

Можливості централізованого обчислення обмежені, оскільки такі системи за своєю природою обмежені в обладнанні, тому для великих задач більший інтерес представляють розподілені системи, сумарна потужність яких за визначенням перевершує потужність будь-якої централізованої системи.

Коротко розглянемо засоби для організації таких разподілених систем.

Інтернет

Інтернет є єдиним інформаційним простором, в якому можна побудувати різні конструкції - сайти, хости, сервери і т.д. Історія Інтернету почалася з 1958 року, коли в відповідь на запуск першого супутника, США створили організацію під назвою ARPA (Advanced Research Projects Agency). До 68-го року в ARPA і в інших організаціях велася робота по з'єднанню комп'ютерів у мережі. В 67-му році пройшов симпозіум трьох незалежних розробників комп'ютерних мереж: ARPA, NPL, RAND. В 1968-м році була побудована перша мережа, заснована на сучасних принципах Інтернету.

Вона складалася з 4-х комп’ютерів. Слід зазначити, що тоді ж (в 1968р) на заводі обчислювальних машин в м.Києві (вул. Велика Окружна 4) була створена автоматизована система управління АСУ підприємства в складі 4-х ЕОМ «Дніпро 2», які були розміщені в різних приміщеннях заводу і об’єднані швидкісною локальною мережею.

Впродовж наступних десяти років в мережі ARPANET зв'язували багато організацій та університетів. До 1978-го року були розроблені всі базові протоколи (тобто мови спілкування між комп'ютерами), які і зараз використовуються в Інтернеті. Для того, щоб комп'ютери різних моделей могли спілкуватися в Інтернеті, потрібно було розробляти єдині протоколи. Крім того, потрібно було встановити спосіб розмежування одного комп'ютера в мережі від іншого, а також - спосіб визначення їх місця розташування.

Цій цілі служать IP адреси (адреси Інтернет-протоколу). IP адреси - це чотири числа від 0 до 255. Зазвичай їх розділяють крапками. Наприклад: 123.23.110.1. Однак такий спосіб добре підходить тільки для спілкування самих комп'ютерів, але дуже незручний для людини. Для зручності ж людини служать домени. Домен - це більш простий і зрозумілий спосіб ідентифікувати комп'ютери в Інтернеті. Наприклад, комп'ютер з IP-адресою 209.155.82.19 має доменне ім'я www.cdrom.com.

Домени мають кілька рівнів. Наприклад, в доменном імені www.cdrom.com, "com" - домен першого рівня, "cdrom" - другого рівня, "www" - третього рівня. Тот, хто володіє доменом необмеженого рівня, може створювати скільки-небудь бажаних доменів нижчих рівнів. Домени першого рівня не підлягають продажі. Вони визначаються організацією по розвитку Інтернету.

Загальноприйнято, що за протоколом http спілкуються по порту 80. І ваш браузер, і програма, яка обслуговує порт 80, вміють говорити на протоколі http. HTTP означає Hyper Text Transfer Protocol, або Протокол передачі гіпертексту. Гіпертекст - це текст, в якому є посилання на інші гіпертексти або місця в цьому тексті. При відправці пошти, Ваша поштова програма використовує протокол 'SMTP', який використовує порт 25, а при отриманні - протокол POP3 на порту 110. Є ще величезна кількість різних протоколів і портів. Їх для простоти можна називати сервісами, а комп'ютери, які очікують, що до них прийдуть запити на використовуваних протоколах, - серверами.

WWW

WWW - спроба організувати всю інформацію в Internet, а також будь-яку локальну інформацію за вашим вибором як набір гіпертекстових документів. Ви переміщується по мережі, переходячи від одного документа до іншого по посиланнях. Всі ці документи написані на спеціально розробленій для цього мові, що називається HyperText Markup Language. World Wide Web, Всесвітня павутина, WWW, Web, Інтернет, - це все назви одного й того ж сервісу, який був придуманий в 1991 році і використовує протокол HTTP для передачі гіпертекстових документів та інших файлів від Веб- сервера до клієнтів. Головна відмінність WWW від інших інструментів для роботи з Internet полягає в тому, що WWW дозволяє працювати практично з усіма доступними зараз на комп'ютері видами документів: це можуть бути текстові файли, ілюстрації, звукові і відеоролики і т.д. Принцип роботи WWW наступний. Користувач запускає у себе програму, яка розуміє протокол HTTP і спеціальну мову, на якій створюється вміст WWW. Ця програма називається «програмою перегляду HTML-сторінки», або по-англійськи - browser (браузер). При цьому HTML (Hyper Text Markup Language) - це «мова розмітки гіпертексту», або мова, за допомогою якої створюється гіпертекст. Далі користувач набирає адресу www серверу. Браузер звертається до сервера з проханням віддати документ, розташований за цією адресою. Сервер віддає документ. Браузер отримує документ, обробляє його і, якщо в ньому є картинки, також просить сервер віддати йому їх, як і інші матеріали документу. Цей документ прийнято називати сторінкою, або WEB (Веб) -сторінкою, або HTML- сторінкою. Після цього браузер оброблює всі прийняті дані і показує готову сторінку на Вашому екрані. Деякі елементи сторінки (тест, картинки, кнопки) можуть бути

посиланнями. Якщо Ви натиснете на них, то Ваш браузер пошле запит серверу, щоб попросити у нього документ.

Таким чином, Ви можете пересуватися від документа до документа, від сервера до сервера, що перетворює весь Інтернет в одну гігантську Мережу, яка пов'язує документи і сервери один з одним нитками гіперпосилань.

WWW - система в цілому складається з наступних компонент:
• Мова гіпертекстової розмітки HTML
• Протокол передачі гіпертексту HTTP
• Специфікацій на типи даних в Internet (Internet Media Types)
• Системи WWW-адресації (URL, URN, URI etc.).

Мова HTML, як уже згадувалося раніше, дуже проста. З чисто з практичної точки зору HTML є розмітка, зроблена звичайними англійськими словами усередині документу. HTML була розроблений для того, щоб виділити в документах логічну структуру. Абревіатура URL розшифровується як Uniform Resource Locator, що можна перекласти, як "єдиний покажчик на ресурс". Практично, це адреса документу.

Web service

Спочатку визначимо, що таке Web-служба. Web-сервіс - це серверний об'єкт, який реалізує певний елемент функціональності, з яким можуть взаємодіяти віддалені програми по протоколу HTTP за допомогою повідомлень на мові XML.

Технологія Web Services призначена для створення розподілених додатків, що функціонують в гетерогенному (неоднорідною) середовищі Інтернет, компоненти яких взаємодіють на базі стандартних Web - протоколів. Архітектура мережі Web Services і взаємодія між клієнтами і службами представлені на рис. 1. Архітектура Web-служб передбачає слабку зв'язність між компонентами мережі (loose coupling). Слабка зв'язність означає, зокрема, що компонентам системи зовсім не обов'язково знати, як влаштовані взаємодіючі з ними підсистеми, а для взаємодії немає необхідності у визначенні нових форматів даних та створенні спеціального програмного забезпечення. Принцип слабкої зв'язності далеко не новий. Вважається, що саме слабка зв'язність дозволила web-технологіям в рекордно короткі терміни стати надзвичайно популярними. Як вже було сказано, Web Services базуються на застосуванні відкритих стандартах і протоколах, які затверджуються консорціумом IT-спільноти, ключовими з яких є наступні:

1. SOAP (Simple Object Access Protocol) - протокол доступу до простих об'єктів, тобто механізм для передачі інформації між об'єктами на базі протоколу HTTP і деяких інших Інтернет-протоколів;

2. WSDL (Web Services Description Language) - мова опису Web-сервісів;
3. UDDI (Universal Description, Discovery and Integration) - універсальне опис, - спрощено кажучи, протокол пошуку ресурсів в Інтернеті.

Основою для реалізації всіх цих протоколів є мова XML (EXtensible Markup Language).

З урахуванням згаданих стандартів алгоритм опису деякого завдання в системі, представленої на рис. 1, можна описати таким чином. Клієнт спочатку звертається на UDDI до реєстру ресурсів (лінія 1), щоб за заданим ним признаком необхідного ресурсу отримати відомості про наявність потрібного ресурсу і місце його розташування (лінія 2). Такий запит ми зазвичай робимо в будь-якій пошуковій системі, наприклад, Google. Отримана з реєстру відповідь містить один або кілька адрес ресурсів. Клієнт по одному з отриманих адрес звертається до ресурсу (лінія 3), щоб отримати WSDL на інтерфейс ресурсу (лінія 4). Інтерфейсом називається видима користувачу частина ресурсу (взагалі, інтерфейс - це набір правил і домовленостей, за якими функціонують обчислювальні засоби). В інтерфейсі вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт SOAP передає ресурсу завдання (лінія 5) і через деякий час отримує результат. У загальному випадку завдання може виконуватися не одним, а безліччю ресурсів.

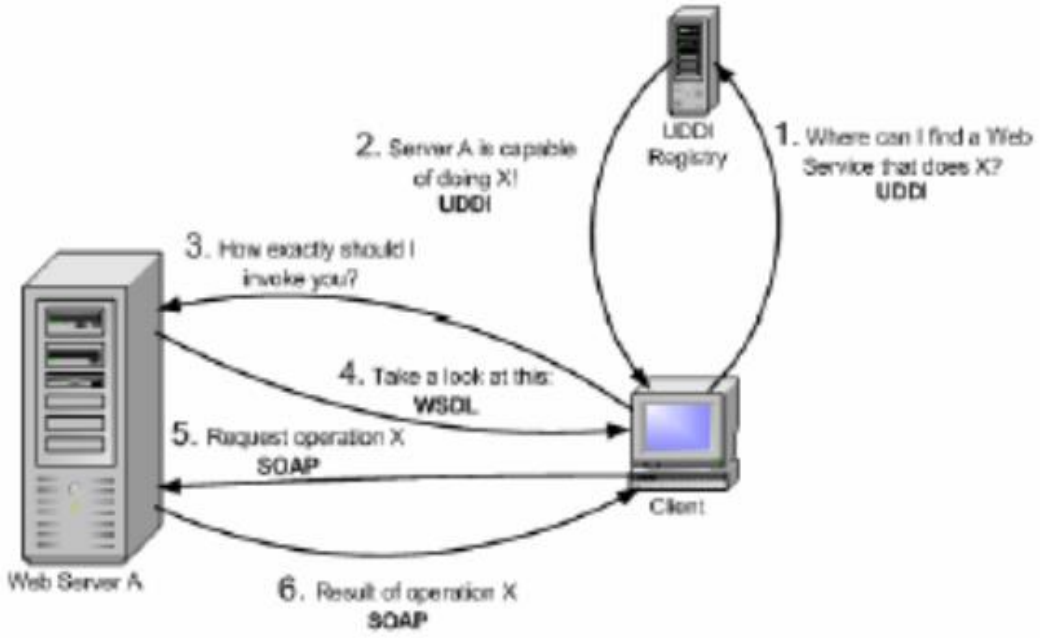


Рис. 1 Архітектура Web Services

Мова XML .

XML (EXtensible Markup Language) - це в перекладі "розширювана мова розмітки". HTML і XML створювалися з різними цілями:

• HTML створювався для демонстрації даних і фокусується на тому, як дані виглядають.
• XML створювався для опису даних і фокусується на тому, чим є дані. Ще потрібно написати якесь програмне забезпечення, щоб відправити цю записку, отримати її або показати її на екрані. На відміну від HTML, XML-теги ідентифікують дані (вказує тип даних), а не спосіб їх відображення. Якщо HTML-тег вказує, наприклад, "відобразити ці дані жирним шрифтом" (<b> ... </ b>), XML-тег діє як ім'я поля у вашій програмі. Він ставить мітку на частину даних, які ідентифікує (наприклад: <message> ... </ message>). Ніякої інформації про структуру, тільки теги візуального відображення, мінімум логічної розмітки. Теги заголовків попередньо описуються, можна описати і форматування абзаців. Але що ще краще, різним записам можна задати унікальні стильові ідентифікатори, якими в подальшому можна маніпулювати. Наприклад, зміна атрибутів виведення конкретного стилю призведе до відповідних змін у всіх документах сайту.

Грід

Під англійським терміном GRID (решітка) розуміється сукупність просторово розподілених обчислювальних вузлів, пов'язаних деякою мережею для обміну даними. Надалі замість GRID будемо використовувати слово Грід. Web Service дозволяє клієнту виконати на обладнанні власника ресурсу деяку функцію зі списку, складеного власником цього обладнання. Грід принципово відрізняється від Web Services.

Грід – це метод використання глобальних процесорних потужностей і систем зберігання інформації (дискові системи великої місткості) без їх фізичного переміщення в просторі. Природно, Грід включає як ресурси і все напрацьоване в WebServices. . Більш того, протоколи WebServices (WSDL, SOAP, UDDI), засоби адресації в розширеному варіанті є основними протоколами Грід.

Грід характеризується наступним чином:

• Географія ресурсів глобальна

• Ресурси неоднорідні, відрізняються по апаратурі, операційним системам, протоколам функціонування і зв'язку

• Склад ресурсів змінний, в загальному випадку непередбачуваний

• Елементи середовища можуть бути ворожими, тобто можуть пошкодити або захопити закриті дані.

Існують наступні два основні різновиди Грід:

• Комунальний Грід. Такий Грід дозволяє надавати на запит клієнта відповідний ресурс.

• Метакомпьютер. В цьому випадку об'єднуються ресурси ряду корпорацій, можливо - весь Інтернет для вирішення загальнолюдських завдань. Спрощена структура Грід показана на рис. 2. Тут клієнт спочатку звертається до реєстру ресурсів MDS, щоб отримати по заданих ним ознакам відомості про наявність потрібного ресурсу і місце його розташування. Клієнт по одній з отриманих адрес звертається до ресурсу, щоб отримати інтерфейс ресурсу. В інтерфейсі вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт передає ресурсу завдання і через деякий час отримує результат.

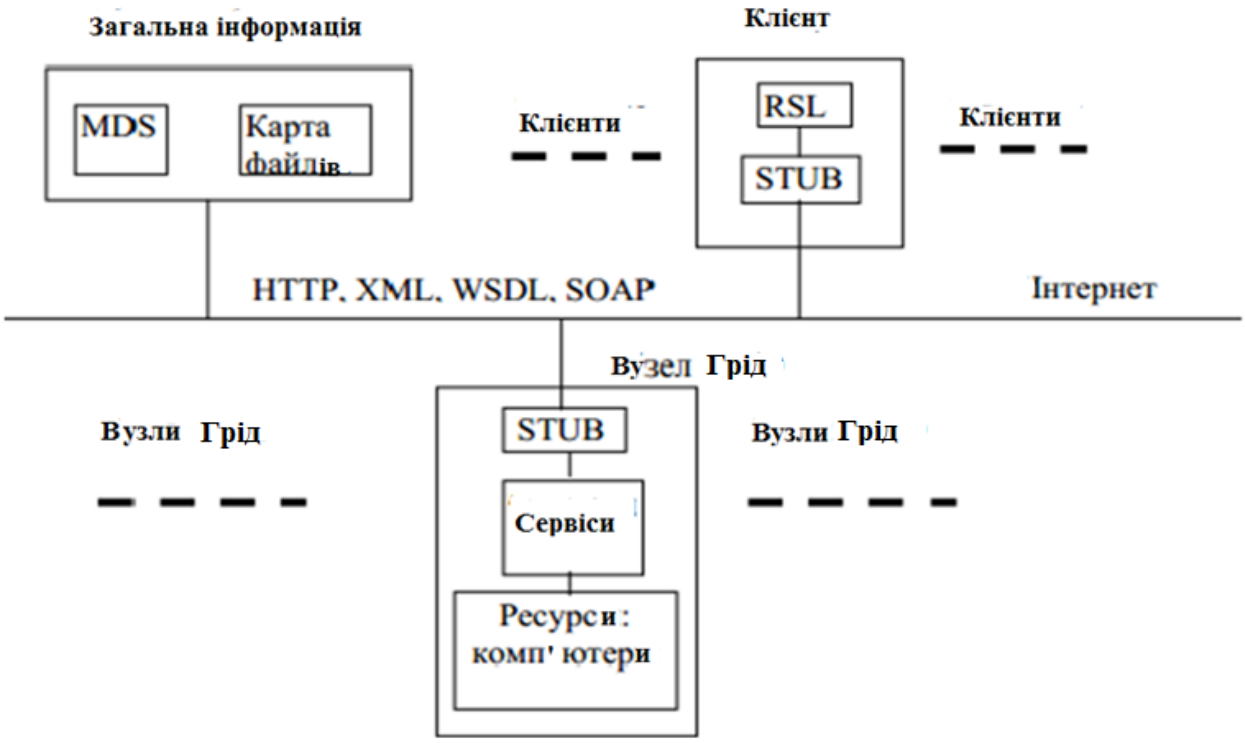


Рис. 2 Структура і функціонування Грід

Тут RSL (Resource Specification Language) - мова для опису завдання, яке потрібно виконати: програма, дані, де все це розміщено, куди послати результат, вимоги до ресурсів та ін.).

Отже, Ґрід — це географічно розподілена інфраструктура, яка об'єднує множину різних типів обчислювальних ресурсів , доступ до яких користувач може отримати з будь-якої точки, незалежно від місця їх розміщення. Ґрід є формою розподілених обчислень, в якому багато комп'ютерів об'єднані в один

потужний віртуальний комп'ютер, і які працюють разом для виконання трудомістких завдань.

Ґрід-обчислення з'єднує комп'ютери з багатьох адміністративних доменів для досягнення певних цілей, Одніє з основних стратегій ґрід-обчислень є використання проміжного програмного забезпечення (ППЗ), яке може адаптуватися під завдання, що розв'язується в одному віртуальному домені, для того щоб розподілити програму серед декількох комп'ютерів, іноді навіть серед тисяч. Основним проміжним ПЗ є Globus Toolkit, gLite, UNICORE, EMI (Europen middleware initiative).

Координація додатків на Грід- системах може бути складним завданням, особливо коли координують потоки інформації через розподілені обчислювальні ресурси.

Грід-обчислення покладається на цілі комп'ютери (з повною комплектацією), під'єднані до комп'ютерної мережі (приватної або публічної) звичайним мережевим інтерфейсом, в той час як звичайний суперкомп'ютер містить безліч процесорів, підключених до локальної високошвидкісної шини. Також є певні відмінності у програмуванні та устаткуванні. Писати програми, які працюють у середовищі суперкомп'ютера, що може мати унікальну операційну систему, може бути дорого і складно. Якщо проблема може бути адекватно розпаралелена, тонкий шар ґрід-інфраструктури може дозволити звичайній програмі запуститись на декількох машинах.

Термін «грід-обчислення» з'явився на початку 1990-х років, як метафора, що демонструє можливість простого доступу до обчислювальних ресурсів як і до електричної мережі (англ. Power grid), у збірнику під редакцією Яна Фостера і Карла Кессельмана "The Grid: Blueprint for a new computing infrastructure ".

Використання вільного часу процесорів і добровільного комп'ютингу стало популярним в кінці 1990-х років після запуску проектів добровільних обчислень GIMPS в 1996 році, distributed.net в 1997 році і SETI @ home в 1999 році. Ці перші проекти добровільного комп'ютингу використовували потужності приєднаних до мережі комп'ютерів звичайних користувачів для вирішення дослідницьких завдань, що вимагають великих обчислювальних потужностей.

Ідеї Грід-системи (включаючи ідеї з областей розподілених обчислень, об'єктно-орієнтованого програмування, використання комп'ютерних кластерів, веб-сервісів та ін.) були зібрані і об'єднані Іеном Фостером, Карлом Кессельманом і Стівом Тікі, яких часто називають батьками Грід-технології. Вони почали створення набору інструментів для Грід-комп'ютингу Globus Toolkit, який включає не тільки інструменти менеджменту обчислень, але й інструменти управління ресурсами зберігання даних, забезпечення безпеки

доступу до даних і до самого Грід, моніторингу використання і пересування даних, а також інструментарій для розробки додаткових Грід-сервісів. В даний час цей набір інструментарію є де-факто стандартом для побудови інфраструктури на базі технології Грід, хоча на ринку існує безліч інших інструментаріїв для Грід-систем.

Програмування та особливості планування в Грід-системах

1. Суперкомп'ютери, грід-системи та хмарні середовища

Наразі суперкомп'ютери, грід-системи та хмарні середовища є найпотужнішими обчислювальними засобами. Але користувачі часто не усвідомлюють, якого типу застосунки слушно виконувати на одному з цих засобів. Тому далі розглянемо особливості та відміни цих архітектур з точки зору найбільшої придатності для реалізації застосувань.

1.1. Порівняння Грід-систем та комп'ютерних кластерів

Під Грід-системою розуміють обчислювальну систему з декількох обчислювальних елементів (СЕ), зв'язаних мережею, які виконують паралельно складне завдання. Така обчислювальна система має три характерні ознаки: її СЕ слабо позв'язані, можуть бути розташовані географічно на великій відстані та можуть бути різного типу (гетерогенними). На відміну від Грід-системи, система з тісно зв'язаних СЕ, така як комп'ютерний кластер (КК), як правило, має пам'ять, що розділяється між СЕ та інші загальні ресурси, які зв'язані між собою через швидкісну систему коротких міжз'єднань.

Як правило, слабко зв'язана система будується з автономних СЕ за принципом майстер – виконавці. Причому СЕ-майстер розділяє велике завдання на незалежні підзавдання, розподіляє їх серед СЕ-виконавців, запускає їх на виконання, збирає від них результати та обробляє їх для формування кінцевого результату.

Грід-система – це система з слабко позв'язаних СЕ. Ці СЕ можуть виконувати окремі незалежні завдання, допоки операційна система (ОС) не об'єднає їх для виконання функції Грід-системи. Причому такий СЕ може продовжувати виконання свого автономного завдання у багатопотоковому режимі, виконуючи функції частини Грід-системи. Цей слабкий зв'язок між СЕ дає змогу географічно віддаленного розміщення СЕ. Оскільки СЕ зв'язані між собою через стандартну мережу (найчастіше – Інтернет), дійсно, не має значення відстань між цими СЕ. Така географічно розподілена мережа має ряд переваг. Загальне завдання у такій мережі вирішується менш витратно, оскільки вона дає змогу краще завантажити недовантажжені СЕ. Ця мережа не має проблем з електричним живленням, оскільки різні СЕ живляться від різних електричних мереж, які не здатні відмовити одночасно. СЕ мережі можуть мати різні ОС та різні продуктивності, хотя багато з Грід-систем побудовані на однотипних СЕ.

Високопродуктивні суперкомп'ютери аналогічні Грід-системам в тому, що серед них багато зв'язаних систем, як наприклад, КК. Але ця зв'язаність є щільною. Якщо між СЕ у Грід-системі передача даних відбувається як асинхронна пересилка повідомлень з підтвердженням їх приймання, то у

суперкомп'ютерах ця пересилка відбувається синхронно без очікування такого підтвердження. Через це СЕ суперкомп'ютерів мають бути зконцентровані в одному місці і з'єднані між собою швидкодіючими каналами зв'язку, а не через звичайну локальну мережу чи Інтернет. Причому ці СЕ мають бути однорідними.

1.2 Програмування Грід-систем та комп'ютерних кластерів

Відміни між архітектурами КК та Грід-систем роблять суттєво різними технології створення програмного забезпечення для них. Тому досі створення методів розробки матзабезпечення для цих архітектур залишається в галузі досліджень та розробки.

Однією з ключових концепцій в написанні застосувань для цих архітектур є ідея батьківських і дочірніх процесів, яка полягає у наступному. Велике завдання аналізується на можливість розділення його на дрібніші завдання. Диспетчер розподіленого обчислювального середовища розділяє завдання на частини, які розподіляються серед вільних СЕ. Перш ніж продовжити обробку, головна задача ("батько"), чекає дані, які повинні бути повернуті від кожного з дрібних підзадач («дітей»). Найбільш важливою проблемою при програмуванні Грід-систем чи КК є кодування у програмі способу, який розділяє велике завдання на дрібні під завдання так , щоб дочірні процеси завершувались якомога швидше. Таке програмування є досить складним. Найбільшою проблемою є нова архітектура, для якої програміст повинен думати в термінах батьківських та дочірніх процесів, які розподіляються по СЕ при певних умовах. Тут програміст має турбуватись не про зацикленість програми, а про баланс кількості та складності дочірніх процесів та кількості вільних ресурсів. Може здаватися, що Грід-система та КК мають однакові архітектури з точки зору програмування, але Грід-система є більш складною для програмування через асинхронність пересилок даних, які мають різні затримки. Крім того, слід мати на увазі, що як канали зв'язку, так і деякі СЕ Грід-системи можуть вимикатись на тривалий час. Тому користувацька програма повинна мати властивість динамічно змінювати хід свого виконання.

1.3 Планування завдань в Грід-системах та комп'ютерних кластерах

Як Грід-система, так і КК мають у своїх ОС планувальники завдань. Планувальник повинен мати не тільки інформацію про програми, які запускаються, але й про алгоритми, які керують породженням дочірніх процесів, про наявність вільних ресурсів в кожній ділянці обчислювального середовища, кількість яких може рахуватися сотнями та тисячами. Крім того він займається перевіркою аутентичності завдань, розподілом їх по ресурсах і фактичним ходом їх виконання.

В області високопродуктивних обчислень часто використовується планувальник, створений у Job Submission Description Language Working Group (JSDL-WG) та Maui Scheduler. Причому перший набув поширення серед Грід-систем, а другий – серед КК.

JSDL-WG – це мова керування завданнями, яка орієнтована на Грід-системи та є незалежною від будь-якої мови програмування, яка може бути використана

користувачем. У описі JSDL можна використовувати код XML, щоб визначити різні аспекти запиту ресурсів, що робить його відкритим стандартом. Це поняття відкритого доступу є критичним при розгляданні переходу від програмування Грід-систем до програмування хмарних обчислень.

Планувальник Maui Scheduler був розроблений в 1990-х роках як планувальник з відкритим вихідним кодом для ОС високопродуктивних КК, які використовують ОС Unix. Maui Scheduler дозволяє адміністраторам встановлювати розклад і пріоритети, резервувати ресурси для конкретних програм, або їх частин, а також виконувати інші завдання, які роблять системи більш функціональними.

Після поширення Maui Scheduler була розроблена в Центрі високопродуктивних обчислень Університету Юта його комерційна версія Moab. Вона додатково підтримує різні ОС, у тім рахунку Linux, Mac OS X та Windows. Також вона має такі функції, як менеджер робочих навантажень, менеджер кластеру, порталу доступу і керування ресурсами, які не доступні в Maui Scheduler. Ця тенденція, коли за основу береться програмне забезпечення з відкритим кодом і дороблюються програми, які додають можливостей, можна побачити у розвитку хмарних обчислень. У планувальнику Moab Scheduler завдання розглядаються з урахуванням політики справедливості пріоритетів, яка враховує тривалість знаходження в черзі, можливостей розширення завдання, історію використання СЕ користувачем і яка конфігурується адміністратором КК. Планувальник Moab Scheduler дає дозвіл, щоб певні СЕ були зарезервовані для певних користувачів, груп, рахунків, або проектів, що мінімізує кількість витрачених ресурсів комп'ютера, з урахуванням очікуваного простою.

Зараз Moab Scheduler використовується як інструмент для дослідження обчислювальних процесів в КК. Цей планувальник збирає велику кількість статистичних даних і може проаналізувати цю інформацію, щоб визначити ефективність того чи іншого підходу до планування, використання наявних ресурсів[1].

1.4 Особливості планування у Грід-системах

Перед тим, як розглядати обчислювальні процеси у хмарному середовищі, слід ознайомитись з процесом планування завдань в обчислювальній мережі Грід-систем. Слід зазначити, що велику роботу по стандартизації планування Грід-систем зроблено на форумі Open Grid Services Architecture (OGSA)[2]. Також значний внесок зробила фірма IBM під час розробки КК *Deep Blue*, яка випустила документацію в серії Red Book.

Процес планування в Грід-системі можна розділити на три етапи, які мають 11 кроків і описуються нижче.

Етап I: Визначення ресурсів

Крок 1: Фільтрація авторизуючих запитів.

Перед тим, як розмістити своє завдання у Грід-системі, користувач повинен пройти аутентифікацію. Крім того, інформація, необхідна для аутентифікації, включає в себе профіль користувача, що описує ресурси, які можуть бути використані, інформацію про оплату використання ресурсів. Цей профіль може також включати в себе деякі облікові дані для одного, або декількох ресурсів,

які підключаються до мережі. Код одноразового доступу, як правило, перевіряється на початковому етапі фільтрації авторизуючих запитів, дозволяючи обліковим даним, які зберігаються у одному CE в якості аутентифікаційних документів, бути застосованими для інших CE в рамках кластеру.

Крок 2: Визначення застосування.

На цьому етапі користувач вказує мінімальний набір ресурсів, які розглядає планувальник. Ця інформація включає в себе об'єм необхідної пам'яті, робочого простору (місця на дисках), перелік спеціалізованих бібліотек тощо і описується як скрипт для планувальника. Вона потрібна для того, щоб планувальник знав, які ресурси необхідні для завдання, щоб воно було виконано успішно.

Поширення стандартного опису XML, який використовується для обміну даними, зробило опис застосувань більш стандартизованим та відкритим для більшості кластерів і мереж. Такий опис не потрібно виконувати вручну, бо він, як правило, формується автоматично за допомогою відповідних додатків до Грід-системи.

Крок 3: Виділення мінімальних вимог до середовища.

В залежності від ресурсів, заявлених авторизованим користувачем, планувальник намагається знайти перший вільний слот (CE та інтервал часу), у якому буде виконуватися завдання. Незалежно від завдання і розміру КК, цей крок пов'язаний з вирішенням задачі пошуку відповідності між потребами в ресурсах і наявними ресурсами, щоб задовольнити потреби користувачів у встановлені терміни.

Етап II: Вибір системи CE

Крок 4: Збір інформації.

Хоча оптимальне рішення задачі призначення завдання на певний набір апаратних засобів може бути знайдене за одним з відомих алгоритмів, але місцеві правила можуть змусити користувача використати інший набір ресурсів. Наприклад, для університетської Грід факультети університету, здавалось, мають рівноправний доступ до Грід-ресурсів. Але часто окремі факультети, які , наприклад, виконують роботи по удосконаленню системи, можуть мати більше прав доступу і їм планувальник буде виділяти більшу квоту.

Слід зазначити, що існує можливість для зміни ресурсів до того моменту, коли завдання буде представлене для вирішення. Значна кількість досліджень робиться у галузі прогнозування розподілу ресурсів, щоб створити адаптоване виділення ресурсів, що дозволить збільшити точність прогнозу цих ресурсів.

Крок 5: Вибір системи CE.

Алгоритми, які вирішують, яку множину CE використовувати для конкретного завдання, варіюються по складності. Але у багатьох проектах розробки планувальників використовується одна і та сама тенденція. Це використання методів, основаних на алгоритмах пошуку паропоєднань Condor, обчислювальної економіки та багатьох інших.

Алгоритм пошуку паропоєднань Condor заснований на класичній схемі оголошень. Обчислювальні ресурси, – це за анологією – "продавці", які рекламують свої послуги та можливості через клас ClassAds, де вони можуть також вказати запитувану ціну для ресурсу. Користувачі та завдання – то "покупці" , які можуть створювати власні класи ClassAds, де зазначаються їх вимоги до ресурсів та заявки оплати, які вони готові платити, щоб їхні вимоги були виконані. Алгоритм Condor перебирає класи ClassAds від продавців і покупців і робить кращі паропоєднання, на основі вказаних специфікацій. Пошук паропоєднання є одноразовим статичним процесом, хоча у будь-який момент за алгоритмом Condor може відбутися повторне сканування заявок та ресурсів, щоб визначити, чи можна знайти кращі паропоєднання.

Алгоритми обчислювальної економіки включають в себе різні методи, які використовуються, щоб вирішити, які процеси призначити на певні ресурси. Ці методи запозичені з теорії класичної економіки, щоб усунути нестабільність вирішення завдання призначення, яка виникає при певних обставинах. Після застосування алгоритму обчислювальної економіки, ресурси і завдання можуть бути динамічно переоцінені в залежності від попиту на ресурси і попиту на результати обчислень, а центральний планувальник постійно приймає рішення щодо запуску завдань, які ґрунтуються на останніх оцінках. Система планування на основі алгоритмів економіки повинна мати на увазі що в будь-який момент ресурси мають використовуватися найбільш ефективно, а саме завдання повинне виконуватись з найбільшою ефективністю.

Етап III: Виконання завдання

Крок 6: Попереднє резервування ресурсів

Попереднє резервування ресурсів у Грід-системі може бути таке саме за складністю, як організація об’єднання комп’ютерів через локальну мережу, або як вирішення задачі масового планування. Але більшість користувачів замовляє ресурси у Грід-системі з надлишком, тобто «про всяк випадок» і це також слід враховувати.

Крок 7: Представлення завдання

Часто завдання користувача представлене з розбіжностями відносно стандартів, на які налаштована дана Грід-система. Тому представлення такого завдання потребує узгодження. Через це Грід-системи, основані на відкритому матзабезпеченні, завжди будуть позаду комерційних Грід-систем через недостатню увагу до стандартизації оформлення завдань.

Крок 8: Підготовка завдання

Як тільки починається виконання завдання, ресурси, такі як набори даних будуть переміщені в тимчасові файли і батьківський процес починає заповнювати інформацію для кожного дочірнього процесу. При цьому для розсилання даних використовуються FTP (File Transfer Protocol), SCP (Secure Copy Protocol), і SFTP (Secure File Transfer Protocol), а також протокол Torrent.

Крок 9: Моніторинг виконання завдання

Може так статись, що якийсь дочірній процес “зависне”, наприклад, через виключний стан, відсутність вхідних даних з відповідним простоюванням ресурсів. У гіршому випадку така поведінка може тягнути за собою “зависання”

усього завдання. Тому планувальник повинен відслідковувати подібні ситуації і переривати виконання завдання.

Крок 10: Завершення роботи

Бідь-яке завдання має метою розрахувати певного роду результати і припинити своє виконання, що має бути зафіксовано планувальником.

Крок 11: Очищення середовища

Планувальник турбується про стирання тимчасових файлів у разі, коли завдання користувача не зробила це самостійно.

Є багато причин, чому вибирають для обчислення задачі саме Грід-систему, а не КК. Це може бути економія витрат, або більша доступність для використання. Є чимало задач, які за своїми розмірами не поміщаються у КК, але можуть бути виконані у Грід-системі. З іншого боку, володіння КК дає простір для реалізації великої множини паралельних алгоритмів, які обчислюються у Грід-системі неефективно. На певному кроці розвитку як Грід-систем, так і КК, стало ясно, що доцільно сумістити ці архітектури з концепцією віртуальних процесорів.

Програмні рішення Грід

В Грід використовуються основні елементи технології WebServices. Це позволяє оформити функції Грід-технлогії (службові і ресурсні) в вигляді веб-сервісів, використовуючи для роботи з ними всі стандартні технології WebServices. На рис. 3 схематично показаний простий сервісно-орієнтований грід, в якому сервіси використовуються і для віртуалізації і для забезпечення інших функціональних можливостей Грід.

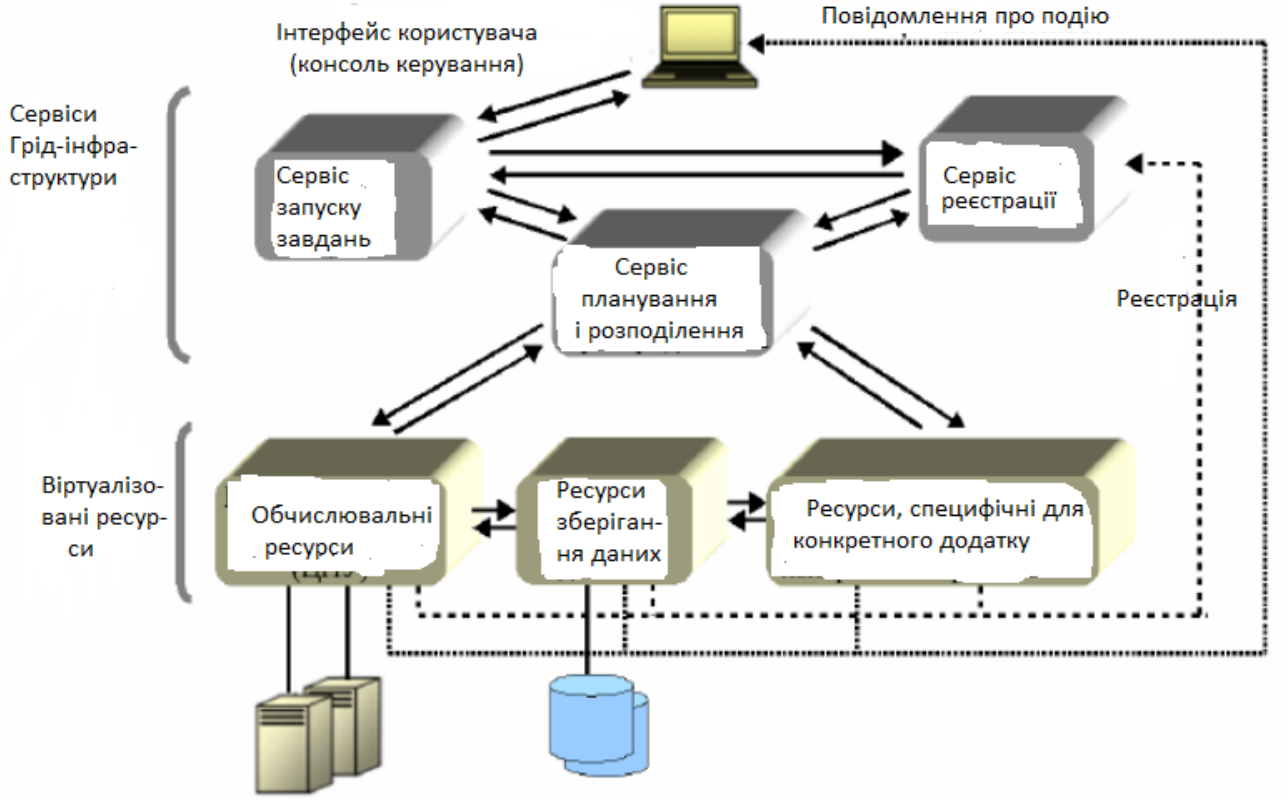


Рис. 3. Схема сервісно-орієнтованого грід

На схемі показана єдина консоль і для запуску завдань в Грід і для керування Грід-ресурсами. Програмне забезпечення інтерфейсу користувача (консолі) звертається до сервісу реєстрації для отримання інформації про існуючі Грід-ресурси. Після цього користувач за допомогою консолі входить в контакт з сервісами, які представляють (віртуалізують) кожний ресурс, щоб запросити періодичне отримання даних про роботу ресурсів і отримати повідомлення про суттєвих змінах в їх стані (наприклад, якщо ресурс стає недоступним, чи сильно завантаженим).

Користувач направляє запит на запуск в службу запуску, яка передає запит службі розподілення завдань (яку ще називають планувальником). Ця служба контактує зі службою, яка представляє додаток і запрошує інформацію про вимоги до ресурсів виконання завдання. Далі служба розподілення запитує (направляє запит) в служби реєстрації інформації про всі необхідні ресурси в Грід і напряму контактує з ними, щоб пересвідчитись в їх доступності.

Якщо необхідні ресурси доступні, планувальник вибирає найкращу доступну сукупність ресурсів і передає інформацію про них сервісу додатку з запитом на початок виконання. В іншому випадку планувальник ставить завдання користувача в чергу і виконує його, коли необхідні ресурси стають доступними. Коли виконання завдання закінчується, сервіс додатку повідомляє про результат планувальнику, який повідомляє про це сервіс запуску завдань. Сервіс запуску завдань, в свою чергу, увідомлює користувача. Слід зауважити, що наведений приклад дуже спрощений: функціонування реального Грід набагато складніше.

В середовищі Грід явним чином присутні наступні елементи:

- Програми користувача
- Ресурси (“залізо”, ОС, кластерне ПЗ та ін.)
- Проміжне програмне забезпечення (Middleware), яке виступає в ролі посередника між користувацькими програмами і ресурсами.

Middleware включає великий обсяг програмного забезпечення, яке створюється великими організаціями і строго стандартизується, щоб забезпечити його використання різними користувачами. В число найбільш відомих пакетів middleware входять:

•Globus Toolkit. Він вважається американським напрямком;

•gLITE. Являється європейським проектом і підтримується Європейським центром ядерних досліджень (CERN).

•UNICORE. Теж являється європейським проектом.

Вони розрізняються між собою, але мають багато спільного, оскільки використовують, як правило, систему Globus Toolkit.

Різні middleware–системи можуть розрізнятися по рівню. Наприклад, gLITE ставиться поверхи Globus Toolkit.

В глобальних Грід-системах в якості middleware використовують інструментарій Globus Toolkit, розроблений американськими вченими, який став de facto світовим стандартом. Він включає в себе спеціальний протокол на основі HTTP для використання обчислювальних ресурсів GRAM (Grid Resource Allocation Management); розширену версію протоколу для передачі файлів GridFTP; службу безпеки GSI (Grid Security Infrastructure); розподілений

доступ до інформації на основі протоколу LDAP; віддалений доступ до даних через інтерфейс GASS (Globus Access to Secondary Storage). Ці служби дозволяють побудувати повнофункціональну Грід- систему.

Далі коротко розглянемо розроблене і використовуємe middleware Грід.

Програмне вирішення GLOBUS

Набір програм Globus Toolkit (GT) – програмний продукт з відкритим початковим кодом і набором бібліотек, розроблений в США в національній лабораторії. Він містить набір стандартних блоків і інструментів, які можуть бути використані розробниками і системними інтеграторами. За декілька років вийшли кілька версій програми Globus: оригінальна – в кінці дев'яностих, GT2 – в 2000, GT3 – в 2003, і GT4 – в 2005, GT5, GT6. Версія GT2 послужила базисом для безлічі Грід-розробників по всьому світу. GT3 – стала першою повноцінною реалізацією інфраструктури Грід, побудованої на технології Web - сервісів, з використанням проміжної ланки GGF's OGSI. GT4 – перша версія, повністю сумісна з основними Web - сервісами так само, як GRID - сервіси засновані на WSDL і WSRF. Інструментарій підтримує операційні системи Linux (RHEL 5-7, CentOS 5-7, Debian 6-7 та ін.), MacOS (10.6 та новіші) та Windows (Windows 7 та новіші).

В стабільному релізі GT 4 Final зафіксовані всі зовнішні інтерфейси.

На сайті Globus Alliance приводиться опис складу служб, відомості про них, плани і стан реалізації, але підтримка завершилася в січні 2018 року, натомість користувачам пропонується використовувати платформу Globus Connect [4]).

На рис. 4 приведені компоненти GT 4.

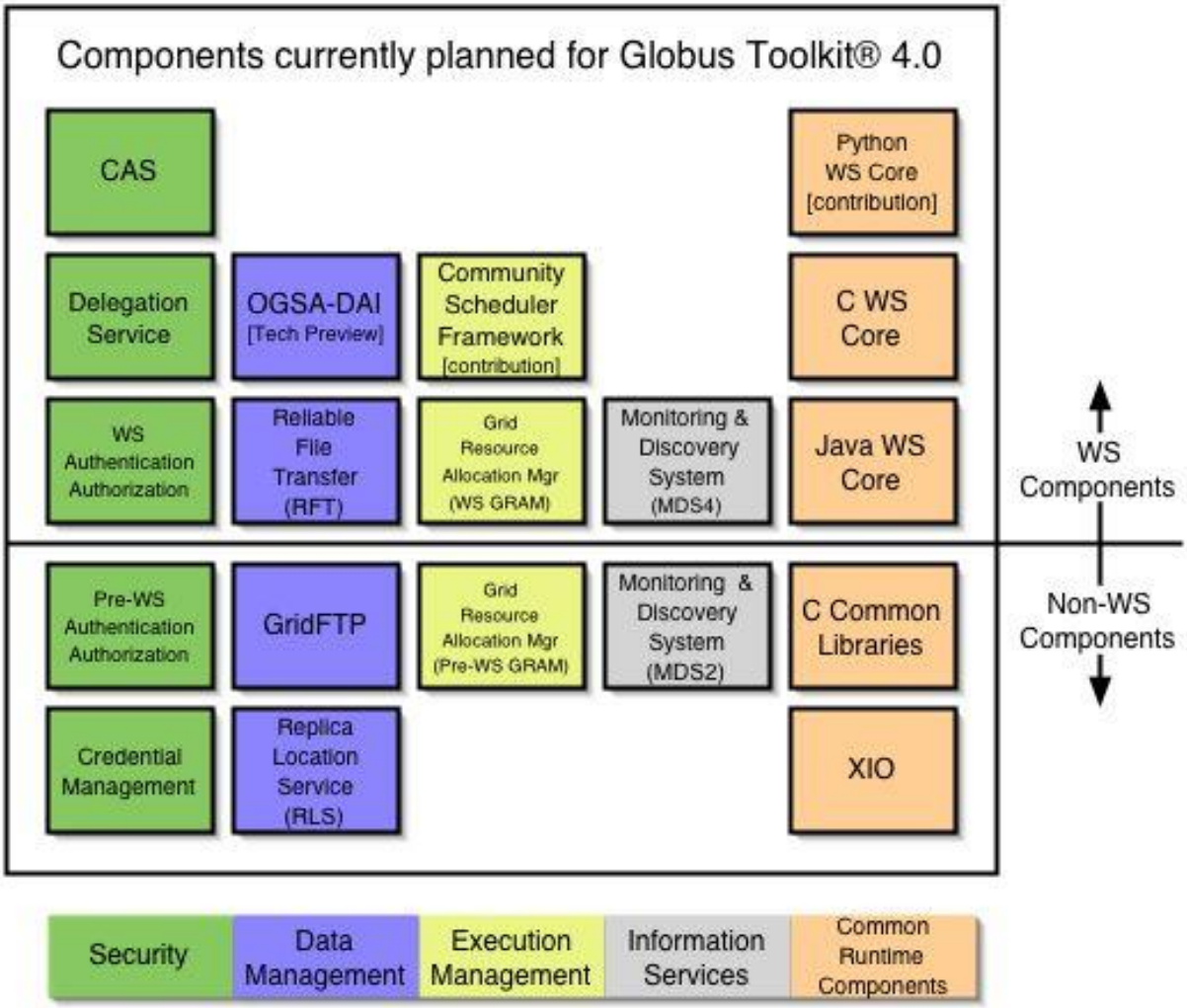


Рис. 4. Компоненти Globus Toolkit GT4

Для управління виконанням пакет надає можливість виявлення і управління ресурсами Грід, управління робочим простором і засобом планувальника співтовариства користувачів.

Для забезпечення безпеки пакет надає сервіси аутентифікації і авторизації, надання віддаленого доступу і авторизації співтовариств користувачів.

Для управління даними в програмі закладені функції надійної передачі файлів, інтеграції і доступу до даних і їх тиражування. Для забезпечення інформаційних служб, в програму закладені функції моніторингу і виявлення різних сервісів системи. Для підтримки спільної роботи система містить різні ядра Web – сервісів, бібліотеки і розширені функції підтримки введення/виводу.

GT4 містить набір стандартних служб:

1. Управління завданнями: Пакет програм виявлення і управління ресурсами (GRAM);.
2. Надійна файлопередача (RFT);.
3. Делегування функцій.
4. Система моніторингу і виявлення вільних ресурсів – індекс (MDS - index);
5. Система моніторингу і виявлення – MDS - trigge.

6. Система моніторингу і виявлення – збір даних (MDS - aggregate);
7. Авторизація співтовариства (CAS);
8. Інтеграція і доступ до даним (OGSA - DAI);
9. Робота в реальному масштабі часу.

Слід зазначити, що Globus Toolkit не містить брокера ресурсів, залишаючи завдання його реалізації розробникам, що створюють системи Грід на його основі.

Склад Globus Toolkit 5.0 (GT5) представлений на рис. 5.

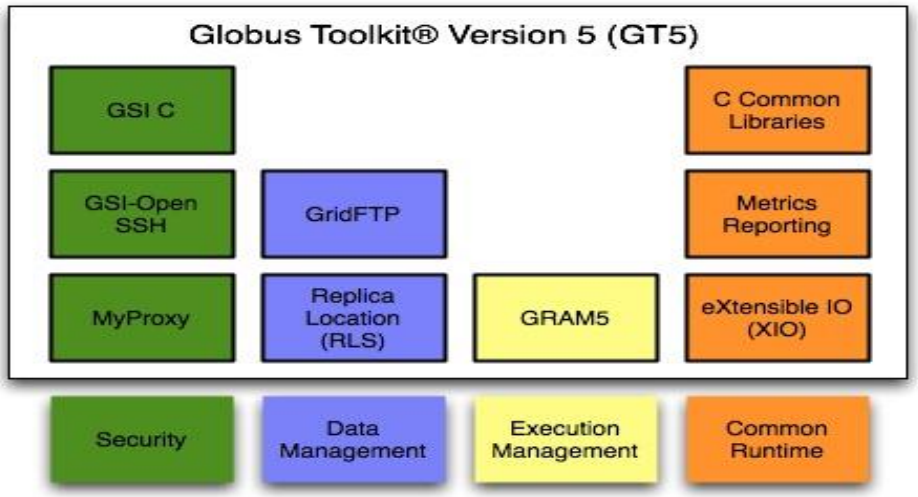


Рис. 5. Склад програмного забезпечення GlobalToolkit GT5

У порівнянні з версією 4, версія 5.0 зазнала наступних основних змін [3]:
- вдосконалення підтримки управління кореневими сертифікатами у компоненті MyProxy v5.0;
- клієнт GSI-OpenSSH типово намагаються виконувати лише автентифікацію GSI (при цьому підтримка інших методів зберігається);
- в RLS досконалена підтримка 64-бітних операційних систем, сумісність із специфікаціями ODBC при збереженні повної сумісності з GT 4 RLS;
- GRAM5, у порівнянні з GRAM2 та GRAM4, є одночасно надійним та масштабованим, при цьому збережена сумісність з GRAM2;
- покращена система журналювання в GRAM5, знижено споживання ресурсів щонайменше в 10 разів.

Версія GT6 зберігає сумісність з версіями 5.0 та 5.2 на рівні протоколів взаємодії, джерельного коду (API) та двійкових програмних інтерфейсів (ABI). Основні компоненти:
- Grid Security Infrastructure in C (GSI C) – надає інтерфейси прикладного програмування та інструменти забезпечення автентифікації, авторизації та управління сертифікатами;
- MyProxy – програмне забезпечення управління інфраструктурою публічних ключів X.509 (PKI) облікових записів (сертифікати та приватні ключі);
- GridFTP – високоефективний, безпечний та надійний протокол передачі даних, що є розширеним варіантом протоколу FTP;
- Grid Resource Allocation and Management (GRAM5) – компонент, що розподілення, моніторинг та управління завданнями в ресурсах мережі Grid;

- GSI-OpenSSH – модифікована версія OpenSSH, яка надає підтримку автентифікації та делегування з використанням X.509 проксі-сертифікатів;
- C Common Libraries – загальні бібліотеки для мови C;
- XIO – розширювана бібліотека введення/виведення написана мовою C, яка надає єдиний інтерфейс прикладного програмування (open/close/read/write) з підтримкою різноманітних протоколів взаємодії додатків. Підтримка протоколів інкапсульована у вигляді "драйверів", до стандартних входить підтримка: TCP, UDP, файли, HTTP, GSI, GSSAPI_FTP, TELNET та черги;
- SimpleCA – надає спрощену реалізацію центра сертифікації, який може видавати сертифікати X.509 для користувачів та служб Globus Toolkit.

Література

1. Foster I., Kesselman C. The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.
2. I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. - International J. Supercomputer Applications, 15(3), 2001, http://www.globus.org/research/papers/anatomy.pdf
3. Foster, C. Kesselman, S. Tuecke, J.M. Nick. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. - Morgan Kaufmann Publishers, 2002.
4 . Foster, H. Kishimoto, A. Savva, D. Berry et al. The Open Grid Services Architecture. - Global Grid Forum, 2005
5 M. Z. Zgurovsky. Development of Educational and Research Segment of Information Society in Ukraine. - //Proc. WSIS.-Tunis.- 2004.-P.103-107.
6 Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
7 GRID Computing in Research and Education, April 2005. – IBM Redbook, − 1 45 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
8 GRID Services Programming and Application Enablement, May 2004. – IBM Redbook, − 273 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
9. Anthony Velte, Toby Velte, Robert Elsenpeter: Cloud Computing: a practical approach, 2010
10. "The evolution of Cloud Computing". Retrieved 22 April 2015.

7.2 *Проектування в хмарних середовищах. Платформа Windows Azure.*

Вперше пішла мова про ідею хмарних обчислень на конференції LinuxWorld 2003. Тоді звернули увагу на систему Scyld, яка була призначена для реалізації сайтів та бізнес-застосувань. Її особливістю була та, що будь-який потік запитів до сайту завжди обслуговувався за рахунок того, що призупинялись малопріоритетні процеси, а важливі процеси динамічно переміщувались на ресурси КК, що вивільняються. Відміною від ОС паралельних систем була та, що система Scyld могла виконувати балансування

навантаження великої кількості різних СЕ, які мають власні ресурси та ОС різного типу.

Як і багато нових технологій, хмарні обчислення розвивались поетапно. На першому етапі хмарні обчислення розвивались як нова послуга – Software as a Service (SaaS), тобто матзабезпечення як послуга в рамках Інтернет - стандарту Web 2.0. Почало з'являтись багато хостинг-сайтів, які були призначені для зберігання сайтів, веб-застосунків осіб та компаній. З'явились системи керування контентом (content management systems, CMS) і відповідно, традиційні сайти почали масово замінюватись на сайти на базі CMS. Тобто, тут CMS є типовим матзабезпеченням як послуга на віддаленому сервері. Крім того, швидке зростання вартості електроенергії змушує багато фірм та організацій мінімізувати витрати, в тому числі використовуючи віддалений доступ до сервера.

Хмарні обчислення - це бізнес-модель, в якій обчислювальні ресурси, такі як обчислення та накопичення, упаковані як сервіси, які схожі на фізичні комунальні послуги.

Хмарні обчислення представляють собою динамічно масштабуємий спосіб доступу до зовнішних обчислювальних ресурсів в вигляді сервісу, який надається посередництвом Інтернету, при цьому користовачу не потрібно ніяких особливих знань про інфраструктуру «хмари» чи навиків управління цією «хмарною» технологією, або Cloud computing – це програмно-апаратне забезпечення, доступне через Інтернет в вигляді сервісу, позволяючого використовувати інтерфейс для віддаленого доступу до виділенх ресурсів (обчислювальних ресурсів , програм і даних). Комп'ютер користовача виступає при цьому рядовим терміналом, підключеним до мережі.

Хмарні середовища стали результатом досвіду експлуатації Грід-систем та уроків віртуалізації хостингу. Цей досвід показав необхідність розробки галузевих стандартів для організації хмарних обчислень. Хмарне середовище схоже як на Грід-систему, так і на КК (комп'ютерний кластер) у тому, що хмарні обчислення можуть бути організовані як у Грід-системі, так і у КК, якщо віртуальному Web-серверу користувача надати відповідні ресурси.

Появу хмарних обчислень обумовили дві ключові тенденції – консолідація і віртуалізація IT-інфраструктури.

Консолідація – це об'єднання обчислювальних ресурсів або структур керування в єдиному центрі.

Саме вона здатна суттєво зменшити витрати на IT.

Зазвичай кажуть про консолідацію:

- **серверів** – переміщення децентралізованих додатків, розподілених на різних серверах компанії, в один кластер централізованих гомогенних серверів;
- **систем зберігання** – спільне використання централізованої системи зберігання даних декількома гетерогенними вузлами;
- **додатків** – розміщення декількох додатків на одному хості.

При цьому можна виділити два базових типи консолідації - фізичну і логічну. Фізична консолідація має на увазі географічне переміщення серверів на єдину площадку (в центр даних), а логічна - централізацію управління.

Віртуалізація – це маскування від користувача складності апаратної та програмної реалізації системи та її складових, географічних відстаней між вузлами, належності вузлів різним організаціям, створення ілюзії роботи з реальним суперкомп'ютером. По статистиці средній рівень завантаження процессорных потужностей у серверів під управлінням Windows не превершує 10%, у Unix-систем цей показник кращий, але і там в средньому не превершує 20%.

В основі віртуалізації лежить можливість одного комп'ютера виконувати роботу декількох комп'ютерів завдяки розподілу його ресурсів на декілька середовищ. За допомогою віртуальних серверів і віртуальних настільних комп'ютерів можна розмістити кілька ОС і кілька додатків в єдиному розташування. Таким чином, фізичні та географічні обмеження перестають мати якесь значення.

У комп'ютерних технологіях під терміном «віртуалізація» зазвичай розуміється абстракція обчислювальних ресурсів і надання користувачеві системи, яка «інкапсулює» (приховує в собі) власну реалізацію. Простіше кажучи, користувач працює зі зручним для себе представленням об'єкту і для нього не має значення, як об'єкт влаштований в дійсності. Можливість запуску декількох віртуальних машин на одній фізичній викликає інтерес серед комп'ютерних фахівців не тільки тому, що це підвищує гнучкість IT – інфраструктури, але й тому, що це дає значну економічну вигоду.

Хмарні обчислення

Хмарні обчислення (англ. cloud computing) - технологія розподіленої обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Надання користувачеві послуг як Інтернет-сервіс є ключовим. Проте під Інтернет-сервісом не варто розуміти доступ до сервісу тільки через Інтернет, він може здійснюватися також і через звичайну локальну мережу з використанням веб-технологій.

Основою для створення і швидкого розвитку хмарних обчислювальних систем послужили великі інтернет-сервіси, такі як Google, Amazon і ін., а так само технічний прогрес, що по суті говорить про те, що поява хмарних обчислень була усього лише справою часу.

Види послуг, які надаються хмарними системами

Концепція хмарних обчислень припускає надання наступних типів послуг своїм користувачам:

- все як послуга (Everything as a Service);

При такому виді сервісу користувачеві буде надано все від програмно-апаратної частини і до управлінням бізнес-процесами, включаючи взаємодію між користувачами, від користувача потрібно тільки наявність доступу в мережу Інтернет.

- інфраструктура як сервіс (Infrastructure as a service);

Користувачеві надається комп'ютерна інфраструктура, зазвичай віртуальні платформи (комп'ютери), пов'язані в мережу.

- платформа як сервіс (Platform as a service);

Користувачеві надається комп'ютерна платформа зі встановленою операційною системою.

- програмне забезпечення як сервіс (Software as a service);

Цей вид послуги зазвичай позиціонується як "програмне забезпечення на вимогу", це програмне забезпечення розгорнуте на віддалених серверах і користувач може діставати до нього доступ за допомогою Інтернету, причому всі питання оновлення і ліцензій на це програмне забезпечення регулюється постачальником цієї послуги. Оплата в даному випадку робитися за фактичне використання програмного забезпечення.

- апаратне забезпечення як сервіс (Hardware as a Service);

В даному випадку користувачеві послуги надається устаткування, на правах оренди, яке він може використати для власних цілей. Цей варіант дозволяє економити на обслуговуванні цього устаткування, хоча за своєю суттю мало чим відрізняється від виду послуги "Інфраструктура як сервіс" за винятком того, що Ви маєте голе устаткування, на основі якого розгортаєте свою власну інфраструктуру з використанням найбільш відповідного програмного забезпечення.

- робоче місце як сервіс (Workplace as a Service);

В даному випадку компанія використовує хмарні обчислення для організації робочих місць своїх співробітників, налаштувавши і встановивши усе необхідне програмне забезпечення, необхідне для роботи персоналу.

- дані як сервіс (Data as a Service);

Основна ідея цього виду послуги полягає в тому, що користувачеві надається дисковий простір, який він може використати для зберігання великих об'ємів інформації.

- безпека як сервіс (Security as a Service).

Цей вид послуги надає можливість користувачам швидко розгортати, продукти дозволяють забезпечити безпечне використання веб-технологій, безпеку електронного листування, а також безпеку локальної системи, що дозволяє користувачам цього сервісу економити на розгортанні і підтримці своєї власної системи безпеки.

Зараз виділяють три категорії "хмар" :

1. Публічні;
2. Приватні;
3. Гібридні.

Публічна хмара – це IT-інфраструктура, яка використовується одночасно безліччю компаній і сервісів. Користувачі цих хмар не мають можливості керувати і обслуговувати цю хмару, уся відповідальність з цих питань покладена на власника цієї хмари. Абонентом пропонованих сервісів може стати будь-яка компанія і індивідуальний користувач. Приклади: онлайн сервіси Amazon EC2 і Simple Storage Service (S3), Google Apps/Docs, Salesforce.com, Microsoft Office Web.

Приватна хмара – це безпечна IT-інфраструктура, контрольована і експлуатована в інтересах однієї-єдиної організації. Організація може керувати приватною хмарою самостійно, або доручити це завдання зовнішньому підрядникові. Інфраструктура може розміщуватися або в приміщеннях замовника, або у зовнішнього оператора, або частково у замовника і частково у оператора. **Гібридна хмара** - це IT-інфраструктура використовує кращі якості публічної і приватної хмари при вирішені поставленої задачі.

Види хмарних обчислень

З поняттям хмарних обчислень пов'язують сервіси, які мають (Everything as a service) технології, такі як:

• «**Інфраструктура як сервіс»** (" Infrastructure as a Service " або " IaaS ");
• **«Платформа як сервіс»** (" Plaatform as a Service ", " PaaS ");
• **«Програмне забезпечення як сервіс»** (" Software as a Service " або "SaaS ").

Інфраструктура як сервіс (IaaS)

IaaS – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

IaaS складається з трьох основних компонентів:

• Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання);
• Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами);
• Сполучне ПЗ (наприклад, для управління системами).

IaaS заснована на технології віртуалізації, що дозволяє користувачу обладнання ділити його на частини, які відповідають поточним потребам бізнесу, тим самим збільшуючи ефективність використання наявних обчислювальних потужностей. Користувач (компанія або розробник ПЗ) повинен буде оплачувати всього лише реально необхідні йому для роботи серверний час, дисковий простір, мережеву пропускну спроможність та інші ресурси. Крім того, IaaS надає в розпорядження клієнта весь набір функцій керування в одній інтегрованої платформі.

Платформа як сервіс (PaaS)

PaaS – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб–додатків як послуги.

Для розгортання веб -додатків розробнику не потрібно купувати обладнання та програмне забезпечення, немає необхідності організовувати їх підтримку. Доступ для клієнта може бути організований на умовах оренди.

Такий підхід має такі переваги:

• масштабованість ;
• відмовостійкість ;

• віртуалізація ;
• безпека.

Масштабованість PaaS передбачає автоматичне виділення і звільнення необхідних ресурсів залежно від кількості обслуговуваних додатком користувачів. PaaS як інтегрована платформа для розробки, тестування, розгортання і підтримки веб-додатків дозволить весь перелік операцій з розробки, тестування, та розгортанню веб-додатків виконувати в одному інтегрованому середовищі, виключаючи тим самим витрати на підтримку окремих середовищ для окремих етапів. Здатність створювати вихідний код і надавати його в загальний доступ всередині команди розробки значно підвищує продуктивність по створенню додатків на основі PaaS.

Найвідомішим прикладом такої платформи є AppEngine від Google, яка пропонує хостинг для веб-додатків з можливістю купувати додаткові обчислювальні ресурси (наприклад, для тестування високих навантажень). Для запуску додатків Google AppEngine на віртуальних кластерних системах була розроблена платформа AppScale, яка не має ніякого відношення до Google.

В центрі всієї хмарної інфраструктури Microsoft – операційна система Windows Azure. Windows Azure створює єдине середовище, яке включає хмарні аналоги серверних продуктів Microsoft (реляційна база даних SQL Azure, що є аналогом SQL Server, а також Exchange Online, SharePoint Online і Microsoft Dynamics CRM Online) і інструменти розробки (.NET Framework і Visual Studio.

Програмне забезпечення як сервіс (SaaS).

SaaS – модель розгортання програми, яка надає додаток кінцевому користувачеві як послугу на вимогу (on demand). Доступ до такого додатку здійснюється за допомогою мережі, а найчастіше за допомогою Інтернет-браузера. У даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання та програмного забезпечення, що працює на ньому. Цільова аудиторія – кінцеві споживачі.

У моделі SaaS:

•додаток пристосований для віддаленого використання;
•одним додатком можуть користуватися декілька клієнтів;
•оплата за послугу стягується або як щомісячна абонентська плата, або на основі сумарного обсягу транзакцій;
•підтримка програми входить вже до складу оплати;
•модернізація програми може проводитися обслуговуючим персоналом плавно і прозоро для клієнтів.

З точки зору розробників програмного забезпечення, модель SaaS дозволить ефективно боротися з неліцензійним використанням програмного забезпечення, завдяки тому, що клієнт не може зберігати, копіювати і встановлювати програмне забезпечення.

Розвитком логіки SaaS є концепція **WaaS (Workplace as a Service – робоче місце як послуга).** Тобто клієнт отримує в своє розпорядження повністю оснащене всім необхідним для роботи віртуальне робоче місце.

Платформа *Windows Azure*

Платформа корпорації Microsoft Windows Azure (спочатку відома під назвою Azure Services Platform) - це група «хмарних» технологій, кожна з яких надає певний набір служб для розробників додатків. На рис. 1 показано, що платформа Windows Azure може бути використана як додатками, які виконуються в «хмарі», так і додатками, які працюють на локальних комп'ютерах

Рис.1 Платформа Windows Azure підтримує додатки, дані та інфраструктуру, які знаходяться в «хмарі».

Платформа Windows Azure складається з наступних компонентів:

• Windows Azure. Забезпечує сердовище на базі Windows для виконання додатків і зберігання даних на серверах в центрах обробки даних корпорації Microsoft.

• SQL Azure. Забезпечує служби даних в «хмарі» на базі SQL Server.

• .NET Services. Забезпечують розподілену інфраструктуру для «хмарних» додатків і локальних додатків.

Кожен компонент платформи Windows Azure грає власну роль. На високому рівні зрозуміти Windows Azure досить легко. Це платформа для виконання додатків Windows і зберігання їх даних в Інтернеті («хмарі»). На рис. 2 показані її основні компоненти.

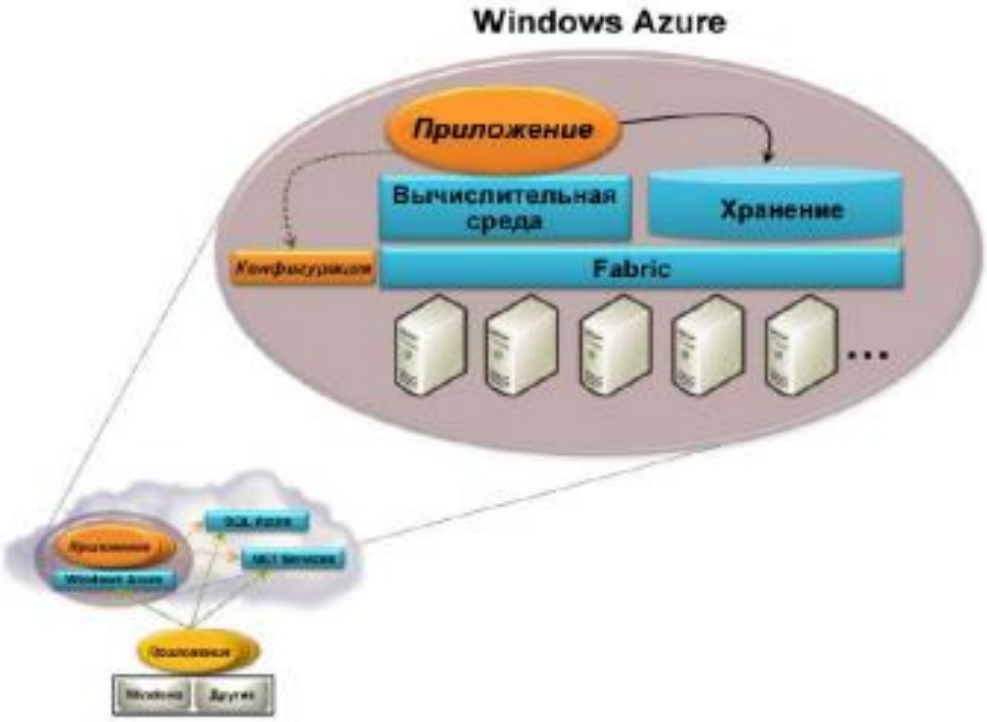


Рис. 2.

Як показано на рис. 2, Windows Azure виконується на великій кількості комп'ютерів, розташованих в центрах обробки даних корпорації Microsoft, і доступна через Інтернет. Загальна структура підключення Fabric Windows Azure з'єднує безліч обчислювальних потужностей в єдине ціле. Служби Windows Azure для обчислення і зберігання побудовані на основі цієї структури. Спочатку корпорація Microsoft дозволила виконувати на Windows Azure тільки додатки, розроблені на платформі .NET Framework. Сьогодні Windows Azure також підтримує некерований код, дозволяючи розробникам виконувати додатки, які розроблені і не на базі .NET Framework. У будь-якому випадку такі додатки написані на звичайних мовах Windows - C #, Visual Basic, C ++ і інших - за допомогою Visual Studio 2008 або інших засобів розробки. Як додатки Windows Azure, так і локальні додатки можуть отримувати доступ до служби сховища Windows Azure за допомогою підходу RESTful. Однак Microsoft SQL Server не є базовим сховищем даних. Фактично сховище Windows Azure не відноситься до реляційних систем і мова його запитів не SQL. Оскільки воно призначено для підтримки додатків на базі Windows Azure, то забезпечує більш прості і масштабовані способи зберігання. Отже, воно дозволяє зберігати великі двійкові об'єкти (binary large object - blob), забезпечує створення черг для взаємодії між компонентами додатків і навіть щось на зразок таблиць з простою мовою запитів. (Для тих додатків Windows Azure, яким потрібно звичайне реляционное сховище, платформа Windows Azure надає базу даних SQL Azure. У Windows Azure кожен додаток має конфігураційний файл. Змінюючи інформацію в цьому файлі вручну або за допомогою програми, власник додатку може контролювати різні аспекти його поведінки , такі як налаштування кількості примірників, які повинні виконуватися на платформі Windows Azure. Структура Fabric платформи

Windows Azure спостерігає за тим, щоб додаток підтримувалося в необхідному стані.

Щоб дозволити своїм замовникам створювати, налаштовувати додатки та спостерігати за ними, Windows Azure надає портал, доступний за допомогою браузера. Замовник надає Windows Live ID, а потім вирішує, створювати йому обліковий запис розміщення для виконання додатків, обліковий запис зберігання для зберігання даних або і ту і іншу. Windows Azure - це загальна платформа, яку можна використовувати в різних сценаріях.

Виконання програм в «хмарі» - один з найважливіших аспектів «хмарних» обчислень. За допомогою Windows Azure корпорація Microsoft забезпечує як платформу для виконання додатків, так і спосіб зберігання даних. У міру того, як зростає інтерес до «хмарних» обчислень, очікується створення ще більшої кількості програм Windows для цієї нової області.

Один з найбільш привабливих способів використання серверів, доступних через Інтернет - це обробка даних. Мета SQL Azure - вирішити цю проблему, пропонуючи набір веб-служб для зберігання найрізноманітнішої інформації і роботи з нею.

База даних SQL Azure Database (раніше відома під назвою SQL DataServices) забезпечує систему управління базами даних (СКБД) в Інтернеті. Ця технологія дозволяє локальним і веб-додаткам зберігати реляційні та інші типи даних на серверах Microsoft в центрах обробки даних Microsoft. База даних SQL Azure розроблена на основі Microsoft SQL Server. Другий компонент SQL Azure був заявлений під назвою Huron Data Sync. Ця технологія, розроблена на основі Microsoft Sync Framework і SQL Azure Database, дозволяє синхронізувати реляційні дані в різних локальних СУБД. Власники даних можуть визначати, що саме має синхронізуватися, як повинні вирішуватися конфлікти і багато іншого.

Виконання програм і зберігання даних в Інтернеті відносяться до важливих аспектів обчислювального мережевого середовища. Однак вони далеко не вичерпують її можливості. Інша можливість полягає в забезпеченні інфраструктури служб на базі «хмари», які можуть використовуватися локальними додатками або веб-додатками. Прикладом є служби .NET Services. Спочатку відомі як BizTalk Services, служби .NET Services пропонують функції для вирішення спільних проблем інфраструктури при створенні розподілених додатків. На рис.3 показані їх основні компоненти.

Рис.3. Служби .NET Services забезпечують інфраструктуру в «хмарі», яка може бути використана для веб-додатків і локальних додатків.

Так само як у випадку Windows Azure надається портал, доступний з допомогою браузеру, щоб дати замовникам можливість використовувати служби .NET Services за допомогою Windows Live ID. Мета корпорації Microsoft, що досягається за допомогою .NET Services, абсолютно очевидна: забезпечити корисну «хмарну» інфраструктуру для розподілених додатків.

Архітектура Windows Azure Platform

Платформа Windows Azure – це модель Платформа як сервіс, яка передбачає запуск додатків на серверах і пов'язаної мережевої інфраструктури, розміщеної в центрах обробки даних Microsoft і має доступ в Інтернет.

Платформа складається з матшабованої «хмарної" операційної системи, фабрики зберігання даних і пов'язаних сервісів доставки через фізичні або логічні (віртуалізація) екземпляри Windows Server 2008. Комплект засобів розробки Windows Azure (SDK) забезпечує розробку версії «хмарних сервісів», также добре, як інструменти і інтерфейси прикладного програмування (API), необхідні для розробки, розгортання та керування масштабованих сервісів в Windows Azure, включаючи шаблони додатків Azure для Visual Studio 2008 і 2010.

На рис. 4 зображені компоненти «хмарної» платформи і компоненти розробника. Згідно Microsoft, при використанні Azure Ви отримуєте:

• Адаптацію існуючих додатків для роботи з веб-сервісами;

• Побудова, зміна і розподіл додатків в мережі з мінімальними локальними ресурсами;

• Виконання послуги, таких як зберігання великих обсягів даних, пакетна обробка даних, обчислення великих обсягів даних і так далі;
• Створення, тестування, налагодження і розподіл веб- сервісів швидко і недорого;
• Зниження вартості і ризиків побудови і поширення місцевих ресурсів;
• Зниження витрат і зусиль на IT управління;

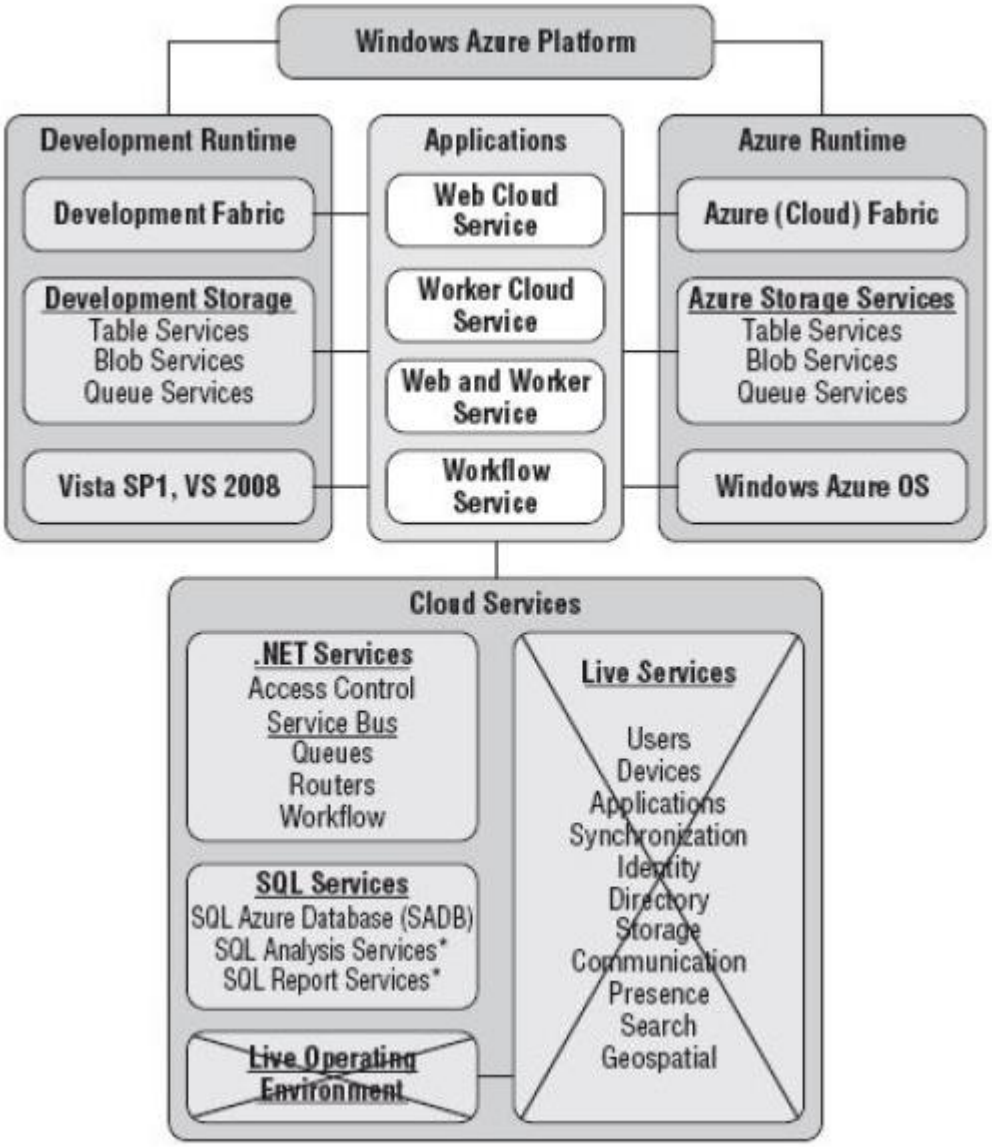


Рис. 4. Компоненти платформи Windows Azure і комплекту засобів розробки.

Вихідною точкою входу для розробників Azure для розміщення ASP.NET додатків в хмару є портал Windows Azure за адресою https://windows.azure.com/Cloud/Provisioning/Default.aspx. Портал вимагає входу з використанням Windows Live ID.

Можна вибрати центри обробки даних для розташування Hosted Services і Storage Accounts. Наприклад, USA-Northwest (Quincy, WA) і USA-Southeast (San Antonio, TX.). Ви можете додати набори Hosted Services і Storage Accounts в групу, щоб гарантувати, що сервіси та сховище розташовуються в одному і тому ж центрі обробки даних, для того, щоб збільшити продуктивність.

Якщо ви хочете розробляти веб-сайти з підтримкою Live Framework, або Mesh (програмний комплекс для синхронізації даних в кросплатформенних середовищах, розроблений компанією Microsoft) веб-додатків, необхідно запросити токен Live Framework по email meshctpe@microsoft.com. Після того, як ви отримаєте Live Framework токен, Ви можете завантажити і встановити поточні версії Live Framework SDK і Live Framework Tools для Visual Studio по посиланнях зазначених на сторінці http://dev.live.com/liveframework/sdk/. Ви повинні оплатити Live Framework токен для того, щоб завантажити Live SDK і додаткові інструменти. Немає необхідності використовувати обліковий запис Windows Azure для тестування Azure Hosted Services and Storage Services, тому що платформа розробки Azure емулює «хмарні» сервіси Azure на вашому комп'ютері.

Windows Azure Storage

Сховище Windows Azure Storage забезпечує розробникам можливість зберігання даних в хмарі. Додаток може виконувати доступ до своїх даних в будь-який момент часу з будь-якої точки планети, зберігати будь-який обсяг даних і як завгодно довго. При цьому дані гарантовано не будуть пошкоджені і втрачені. Windows Azure Storage пропонує багатий набір абстракцій даних:

• Windows Azure Table - забезпечує структуроване сховище станів сервісу.
• Windows Azure Blob - забезпечує сховище великих елементів даних.
• Windows Azure Queue - забезпечує диспетчеризацію асинхронних завдань для реалізації обміну даними між сервісами.

Azure Table Services

Windows Azure Table - структуроване сховище, котре підтримує високомасштабуємі таблиці в хмарі, які можуть містити мільярди сутностей і терабайти даних. У міру збільшення трафіку, система буде ефективно масштабуватися, автоматично підключаючи тисячі серверів. Структуроване сховище реалізовано у вигляді таблиць (Tables), в яких розташовуються сутності (Entities), що містять ряд іменованих властивостей (Properties).
Ось деякі з основних характеристик Windows Azure Table:

• Підтримка LINQ, ADO .NET Data Services і REST.
• Контроль типів під час компіляції при використанні клієнтської бібліотеки ADO .NET Data Services.
• Багатий набір типів даних для значень властивостей.
• Підтримка необмеженої кількості таблиць і сутностей без обмеження розмірів таблиць.
• Підтримка цілісності для кожної сутності.
• Нежорстке блокування при оновленнях і видаленнях.
• Для запитів, виконання яких вимагає тривалого періоду часу, або запитів, перерваних після завершення часу очікування, повертаються часткові результати і маркер продовження

Модель даних таблиці Windows Azure Table:

Облікова запись сховища (Storage Account) - для доступу до Windows Azure Storage додаток має використовувати дійсну облікову запись. Нову облікову запись можна створити через веб-інтерфейс порталу Windows Azure. Як тільки облікову запись створено, користувач отримує 256-розрядний секретний ключ, який згодом використовується для аутентифікації запитів цього користувача до системи зберігання. Зокрема, за допомогою цього секретного ключа створюється підпис HMAC SHA256 для запиту. Цей підпис передається з кожним запитом даного користувача для забезпечення аутентифікації. Ім'я облікового запису входить до складу імені хоста в URL.

Для доступу до таблиць використовується наступний формат імені хоста: <імяОбліковийЗапис> .table.core.windows.net.

• Таблиця (Table) - містить набір сутностей. Область дії імен таблиць обмежена обліковим записом. Додаток може створювати безліч таблиць в рамках облікового запису сховища.

• Сутність (рядок) (Entity (Row)) - Сутності (сутність є аналогом «Рядка») - це основні елементи даних, що зберігаються в таблиці. Сутність включає набір властивостей. У кожній таблиці є дві властивості, які утворюють унікальний ключ для сутності.

• Властивість (стовпець) (Property (Column)) - являє окреме значення сутності. Імена властивостей чутливі до регістру. Для значень властивостей підтримується багатий набір типів.

• Ключ секції (PartitionKey) - Перша властивість ключа кожної таблиці. Ця система використовує даний ключ для автоматичного розподілу сутностей таблиці по безлічі вузлів зберігання.

• Ключ рядки (RowKey) - Друга властивість ключа таблиці. Це унікальний ID сутності в рамках секції. PartitionKey в поєднанні з RowKey унікально ідентифікує сутність в таблиці.

• Тимчасова мітка (Timestamp) - Кожна сутність має версію, яка зберігається системою.

• Секція (Partition) - Набір сутностей в таблиці з однаковим значенням ключа секції.

• Порядок сортування (Sort Order) - Для CTP-версії надається всього один індекс, в якому всі сутності сортовані за PartitionKey і потім по RowKey.

Це означає, що запити із зазначенням цих ключів будуть більш ефективними, і всі повертаємі результати будуть сортовані за PartitionKey

Таблиця має гнучку схему. Windows Azure Table відстежує ім'я і типізоване значення кожної властивості кожної сутності. Додаток може моделювати фіксовану схему на стороні клієнта, забезпечуючи однаковий набір властивостей для всіх створюваних сутностей.

Деякі додаткові відомості про сутності:

• Сутність може мати до 255 властивостей, включаючи обов'язкові системні властивості: PartitionKey, RowKey і Timestamp. Імена всіх інших властивостей сутностей визначаються додатком.

• Властивості PartitionKey і RowKey строкового типу.

• Властивість Timestamp є доступною лише для читання обслуговуємою системою властивістю, яку слід розглядати як непрозору властивість.

• Відсутність фіксованої схеми - Windows Azure Table не зберігає ніякої схеми, тому всі властивості зберігаються як пари <ім'я, типізоване значення>. Це означає, що властивості сутностей однієї таблиці можуть сильно відрізнятися. В таблиці навіть може бути дві сутності, властивості яких мають однакові імена, але різні типи значень.

• Сумарний обсяг всіх даних сутності не може перевищувати 1 МБ. Сюди входить розмір імен властивостей, а також розмір значень властивостей або їх типів, включаючи і дві обов'язкові властивості ключів (PartitionKey і RowKey).

• Підтримуються типи Binary, Bool, DateTime, Double, GUID, Int, Int64, String.

Windows Azure Table забезпечує можливість масштабування таблиць до тисяч вузлів зберігання через розподіл сутностей в таблиці. При розподілі сутностей бажано забезпечити, щоб сутності, що входять в одну множину, розташовувалися в одному вузлі зберігання. Додаток формує ці множини відповідно значенням властивості PartitionKey сутностей. Додаткам повинне бути відоме робоче навантаження кожної окремо взятої секції. Для забезпечення бажаних результатів тестування має моделювати максимальне робоче навантаження.

На рис. 5 представлена таблиця, яка містить множину версій документів. Кожна сутність даної таблиці відповідає певній версії певного документу. У цьому прикладі ключем секції таблиці є ім'я документу, ключем рядку - номер версії. Ім'я документа і версія унікально ідентифікують кожну сутність таблиці. В даному прикладі секцію утворюють всі версії одного документу.

Partition Key Document Name	Row Key Version	Property 3 Modification Time	.....	Property N Description	
Examples Doc	V1.0	8/2/2007	.....	Committed version	Partition 1
Examples Doc	V2.0.1	9/28/2007		Alice's working version	
FAQ Doc	V1.0	5/2/2007		Committed version	
FAQ Doc	V1.0.1	7/6/2007		Alice's working version	Partition 2
FAQ Doc	V1.0.2	8/1/2007		Sally's working version	

рис. 5 Приклади секцій

Система відстежує характер використання секцій і автоматично рівномірно розподіляє ці секції по всіх вузлах зберігання. Це дозволяє системі і додатку масштабуватися відповідно до кількості запитів до таблиці. Тобто, якщо деякі секції запитуються більше інших, система автоматично рознесе їх на кілька вузлів зберігання, таким чином, розподіляючи трафік між множиною серверів. Однак секція, тобто всі сутності, що мають однаковий ключ секції, будуть обслуговуватися як один вузол. Але навіть незважаючи на це, обсяг даних в рамках секції не обмежений ємністю сховища одного вузла зберігання.

Сутності однієї секції зберігаються разом. Це забезпечує найбільш ефективну обробку запитів до секції. Більш того, в цьому випадку додаток може використовувати всі переваги ефективного кешування і інших оптимізацій продуктивності, які забезпечуються розташуванням даних в секції.
У наведених нижче прикладах описуються операції з таблицею «Blogs». У цій таблиці зберігаються блоги для додатку MicroBlogging. У додатку MicroBlogging є дві таблиці: Channels (Канали) і Blogs (Блоги). Є список каналів, блоги публікуються в певному каналі. Користувачі підписуються на канали і щодня отримують нові блоги цих каналів.
В даному прикладі розглянемо тільки таблицю Blogs і наведемо приклади наступних операцій з нею:

1. Опис схеми таблиці
2. Створення таблиці
3. Вставка блогу в таблицю
4. Отримання списку блогів з таблиці
5. Оновлення блогу в таблиці
6. Видалення блогу з таблиці

Схема таблиці описується як C # -клас. Таку модель використовує ADO.NET Data Services. Схема відома тільки клієнтському додатку і спрощує доступ до даних. Сервер схему не застосовує.

Розглянемо опис сутностей Blog, які зберігаються в таблиці Blogs. Кожна сутність блогу містить наступні дані:

1. Назва каналу (ChannelName) - канал, в якому розміщується блог.
2. Дата розміщення.
3. Текст (Text) - вміст тіла блогу.
4. Рейтинг (Rating) - популярність цього блогу.

Зверніть увагу, що для таблиці визначено PartitionKey, яка представляє ім'я каналу, частиною якого є блог, і в якості RowKey використовується дата розміщення блогу. PartitionKey і RowKey - ключі таблиці Blogs, вони оголошуються за допомогою атрибута класу DataServiceKey (Ключ сервісу даних). Тобто таблиця Blogs секціонована по іменах каналів (ChannelName). Це дозволяє додатку ефективно витягати самі недавні блоги каналу, на які підписаний користувач. Крім ключів, в якості властивостей оголошені характерні для користувача атрибути. Всі властивості мають відкриті (public) методи зчитування і присвоєння значення і зберігаються в таблиці Windows Azure Table. Отже, в прикладі нижче:

• Text і Rating зберігаються для екземпляра сутності в таблиці Azure.

• RatingAsString немає, тому що для нього не визначено метод присвоєння значення.

• Id не зберігається, тому що методи доступу не public.

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Blog
```

```
{
// ChannelName
public string PartitionKey { get; set; }
// PostedDate
public string RowKey { get; set; }
// визначаємі користувачем властивості
public string Text { get; set; }
public int Rating { get; set; }
public string RatingAsString { get; }
protected string Id { get; set; }
}
```

Далі розглянемо, як створити таблицю Blogs для облікового запису сховища.

Створення таблиці аналогічно створенню сутності в основній таблиці «Tables». Ця основна таблиця визначена для кожного облікового запису сховища, і ім'я кожної таблиці, використовуваної обліковим записом зберігання, повинно бути зареєстровано в основній таблиці. Опис класу основної таблиці наведено нижче, де властивість TableName (Ім'я таблиці) представляє ім'я створюваної таблиці.

```
[DataServiceKey("TableName")]
public class TableStorageTable
{
public string TableName { get; set; }
}
```

Фактично створення таблиці відбувається наступним чином:

```
// Uri сервиса: “http://<Account>.table.core.windows.net/”
DataServiceContext context = new DataServiceContext(serviceUri);
TableStorageTable table = new TableStorageTable("Blogs");
// Створюємо нову  таблицю, добавляючи нову  сущність
// в основну  таблицю "Tables"
context.AddObject("Tables", table);
// результатом визову SaveChanges являеться відгук сервера
DataServiceResponse response = context.SaveChanges();
```

serviceUri – це uri сервісу таблиці, http://<тут вказується ім’я облікового запису>.table.core.windows.net/. DataServiceContext (Контекст сервісу даних) – один із основних класів сервісу даних ADO.NET, представляє контекст часу виконання для сервісу. Він забезпечує API для вставки, оновлення, видалення та запиту сутностей за допомогою яких LINQ, або RESTful URI і зберігає стан на стороні клієнта.

Розглянемо вставку елемента Blog. Щоб вставити сутність, додаток має виконати наступне.

1. Створити новий C #-об'єкт і задати всі властивості.

2. Створити екземпляр DataServiceContext, який представляє підключення до серверу в сервісі даних ADO .NET для вашого профілю сховища.

3. Додати C #-об'єкт в контекст.

4. Викликати метод SaveChanges (Зберегти зміни) об'єкту DataServiceContext для відправки запиту серверу. Це забезпечує відправку на сервер HTTP- запиту з сутністю в XML-форматі ATOM.

Далі представлені приклади коду для перерахованих вище операцій:

```
Blog blog = new Blog {
PartitionKey = "Channel9", // ChannelName
RowKey = DateTime.UtcNow.ToString(), // PostedDate
Text = "Hello",
Rating = 3
};
serviceUri = new Uri("http://<account>.table.core.windows.net");
var context = new DataServiceContext(serviceUri);
context.AddObject("Blogs", blog);
DataServiceContext response = context.SaveChanges();
```

Запит сутностей виконується за допомогою вбудованої в C # мови запитів LINQ (Language Integrated Query). В даному прикладі отримаємо всі блоги, рейтинг яких дорівнює 3.

При обробці запиту (наприклад, за допомогою виразу foreach), він передається на сервер. Сервер відправляє результати в XML-форматі ATOM. Клієнтська бібліотека ADO .NET Data Services десеріалізує результати в C # -об'єкти, після чого вони можуть використовуватися додатком.

```
var serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext context = new DataServiceContext(serviceUri);
// LINQ-запит з використанням DataServiceContext для вибору
// із таблиці Blogs всіх сутностей блогів, для которих rating = 3
var blogs =
from blog in context.CreateQuery<Blog>("Blogs")
where blogs.Rating == 3
select blog;
// запит відправляється на сервер і виконується
foreach (Blog blog in blogs) { }
```

Оновлення суті виконується наступним чином.

1. Створюється DataContext (Контекст даних), властивості MergeOption (Варіант об'єднання) якого задається значення OverwriteChanges (Перезапис

змін) або PreserveChanges (Збереження змін) . Це забезпечує правильну обробку ETag для кожного витягнутого об'єкту.

2. За допомогою LINQ DataContext отримує сутність, яка буде оновлюватися. Витяг її з серверу гарантує оновлення ETag в сутностях, що відслідковуються контекстом, і те, що при подальших оновлення та видаленях в заголовку if-match буде використовуватися оновлений ETag. Міняємо C#-об'єкт, який представляє сутність.

3. Повертаємо C #-об'єкт в той же DataContext для поновлення. Використання того ж DataContext гарантує автоматичне повторне використання ETag, отриманого раніше для цього об'єкта.

4. Викликаємо метод SaveChanges для відправки запиту на сервер.

```
Blog blog =
(from blog in context.CreateQuery<Blog>("Blogs")
where blog.PartitionKey == "Channel9"
&& blog.RowKey == "Oct-29"
select blog).FirstOrDefault();
blog.Text = "Hi there";
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Видалення Blog

Видалення сутності аналогічно її оновленню. Для цього витягаємо сутність за допомогою DataServiceContext і викликаємо для вмісту замість методу UpdateObject метод DeleteObject (Видалити об'єкт).

```
// Отримуємо об'єкт Blog для ("Channel9", "Oct-29")
context.DeleteObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Розглянемо рекомендації по роботі з DataServiceContext:

• Об'єкт DataServiceContext не забезпечує безпеку потоків, тому він не може використовуватися спільно різними потоками, а також має нетривалий час існування.

• DataServiceContext не є об'єктом з тривалим часом життя. Замість того, щоб використовувати один DataServiceContext протягом всього життя потоку, рекомендується створювати об'єкт DataServiceContext кожен раз, коли виникає необхідність виконати ряд транзакцій з WindowsAzureTable, і потім видаляти цей об'єкт.

• Якщо для всіх вставок / оновлень / вилучень використовується один екземпляр DataServiceContext і виникає збій при виконанні SaveChanges, відомості про операції, що дала збій, зберігаються в DataServiceContext. При наступному виклику SaveChanges спроба виконати цю операцію повторюється.

• DataServiceContext має властивість MergeOption, яка використовується для управління тим, як DataServiceContext обробляє відслідковувані сутності. Можливі значення:

o AppendOnly (Тільки додавання): Це значення за замовчуванням, при використанні якого DataServiceContext не завантажує екземпляр сутності з сервера, якщо він вже є в його кеші.

o OverwriteChanges: DataServiceContext завжди завантажує екземпляр сутності з сервера і перезаписує попередній варіант сутності, тобто забезпечує відповідність екземпляра сутності її поточному стану.

o PreserveChanges: Якщо екземпляр сутності існує в DataServiceContext, він не завантажується з постійного сховища. Всі зміни, що вносяться до властивості об'єктів в DataServiceContext, зберігаються, але ETag оновлюється, тому дану опцію слід використовувати при необхідності відновлення після помилок спільного доступу з нежорстким блокуванням.

o NoTracking (Без відстеження): DataServiceContext не відслідковує екземпляри сутностей. Оновлення сутності в контексті без відстеження реалізується за допомогою Etag, який оновлюється за допомогою AttachTo. Цей варіант не рекомендується до застосування.

```
context.AttachTo("Blogs", blog, "etag to use");
context.UpdateObject(blog);
context.SaveChanges();
```

Коли MergeOption контексту задано значення AppendOnly і об'єкт DataServiceContext вже відстежує сутність в результаті попередньої операції витягнення або додавання, повторне витягнення сутності з сервера не приведе до відновлення відслідковуємої сутності в контексті. Таким чином, якщо сутність на сервері була змінена, подальші оновлення / видалення призведуть до збою необхідних умов (PreCondition). Результатом усіх розглянутих вище операцій є передача HTTP- повідомлень на і з сервера. Додаток може відмовитися від використання клієнтської бібліотеки .NET і працювати на рівні HTTP / REST.

Розглянемо паралельні поновлення. Для поновлення сутності необхідно виконати наступні операції.

1. Отримати сутність з сервера.
2. Відновити об'єкт локально і повернути його на сервер. Припустимо, два процеси, що виконуються паралельно, намагаються оновити одну і ту ж сутність. Оскільки кроки 1 і 2 не є неподільними, на будь-якому з них може виникнути ситуація внесення змін до вже застарілої версії сутності. Для вирішення цієї проблеми Windows Azure Table використовує нежорстке блокування.

1. Для кожної сутності система зберігає версію, яка змінюється сервером при кожному оновленні.
2. Під час витягу сутності, сервер відправляє цю версію клієнта у вигляді ETag HTTP.
3. Коли клієнт передає запит UPDATE (оновити) на сервер, він відправляє на нього цей ETag у вигляді заголовка If-Match.

4. Якщо версія сутності, яка зберігається на сервері, аналогічна ETag в заголовку If-Match, зміна приймається, і зберігаєма на сервері сутність отримує нову версію. Ця нова версія повертається клієнту як заголовок ETag.
5. Якщо версія сутності на сервері відрізняється від ETag в заголовку If- Match, зміна відхиляється, і клієнтові повертається HTTP-помилка «Precondition failed» (необхідна умова не виконана).

При отриманні помилки «precondition failed» типовою поведінкою клієнтського додатку буде повторення всієї операції, як показано в фрагменті коду нижче.
1. Додаток має отримати цей об'єкт знову, тобто отримати його останню версію.
2. Відновити об'єкт локально і повернути його на сервер.

При використанні клієнтської бібліотеки .NET додаток отримує HTTP-код помилки як виняток DataServiceRequestException.

У прикладі нижче два різних клієнта виконують один і той же код для зміни тексту. Ці два клієнти намагаються задати Text різні значення.
1. Вони витягають сутність. При цьому для кожної сутності витягується ETag, наприклад, «v1». Обидва клієнти вважають, що попередня версія суті - «v1».
2. Кожен клієнт локально оновлює властивість Text.
3. Кожен клієнт викликає методи UpdateObject і SaveChanges.
4. Кожен клієнт відправляє на сервер HTTP-запит з заголовком «If- Match: v1 ».
5. Запит одного з клієнтів потрапляє на сервер першим.
   a. Сервер порівнює заголовок If-Match з версією сутності. Вони збігаються.
   b. Сервер застосовує зміну.
   c. Версія сутності на сервері оновлюється і стає «v2».
   d. Як відповідь клієнту відправляється новий заголовок «ETag: v2» .__
6. Далі на сервер надходить запит іншого клієнта. На цей момент зміни першого клієнта вже застосовані.
   a. Сервер порівнює заголовок If-Match з версією сутності. Вони не збігаються, оскільки версія сутності вже змінена на «v2», тоді як в запиті вказується версія «v1».
   b. Сервер відхиляє запит.

```
// Задаємо такий варіант об'єднання, який забезпечує
// збереження оновлень, але дозволяє оновлення etag.
// За замовчуванням застосовується значення AppendOnly, при якому
// вже відстежувана сутність не буде перезаписана значеннями,
// отриманими з сервера, в результаті чого в разі зміни
// суті на сервері використовується недійсний etag.
context.MergeOption = MergeOption.PreserveChanges;
Blog blog =
(from blog in context.CreateQuery<Blog>("Blogs")
where blog.PartitionKey == "Channel9"
&& blog.RowKey == "Oct-29"

select blog).FirstOrDefault();
blog.Text = "Hi there again";
```

```
try
{
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
OperationResponse response = e.Response.First();
if (response.StatusCode == (int)HttpStatusCode.PreconditionFailed)
{
//виконуємо запит об'єкта повторно, щоб отримати
// останній etag, і проводимо оновлення}
}
}
```

Для безумовного поновлення сутності додаток виконує наступне:

1. Створює новий об'єкт DataServiceContext або, в разі використання існуючого контексту, від'єднує об'єкт, як демонструє приклад нижче.
2. Приєднуємо сутність до контексту і використовуємо «*» як нове значення ETag.
3. Оновлюємо сутність.
4. Викликаємо SaveChanges.

```
// задаємо опцію об'єднання, що дозволяє перезапис,
// щоб забезпечити можливість поновлення відслідковуємої сутності
context.Detach (blog);
// Приєднуємо сутність до контексту, використовуючи ім'я таблиці, сутність,
// яка повинна бути оновлена, і "*" як значення etag.
context.AttachTo("Blogs", blog, "*");
blog.Text = "Hi there again";
try
{
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
// Обробка помилки, але в даному випадку формування помилки PreCondition
неможливо}
```

Для запитів, які можуть повертати велику кількість результатів, система забезпечує два механізми:

1. Можливість отримувати перші N сутностей, використовуючи LINQ-функцію Take (N).
2. Маркер продовження, який позначає місце початку наступної множини результатів.

Система підтримує функцію повернення перших N відповідних запиту

сутностей. Наприклад, якщо програма розробляється на .NET, для витягу перших N сутностей (в даному прикладі це перші 100 сутностей) можна використовувати LINQ-функцію Take (N).

```
serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext svc = new DataServiceContext(serviceUri);
var allBlogs = context.CreateQuery<Blog>("Blogs");
foreach (Blog blog in allBlogs.Take(100))
{
// виконуємо деякі операції з кожним блогом}
```

Аналогічна функціональність підтримується в інтерфейсі REST через опцію рядку запиту $ top =N. Наприклад, запит «GET http: // <UriСервіса> / Blogs? $ Top = 100» забезпечував би повернення перших 100 сутностей, що відповідають запиту. Фільтрація виконується на сервері, тому у відповіді клієнту може бути передано максимум 100 сутностей.

1. У запиті вказується максимальна кількість сутностей, яка повинна бути повернута.
2. Кількість сутностей перевищує максимально дозволене сервером число сутностей у відповіді (в даний час це 1000 сутностей).
3. Загальний розмір сутностей у відповіді перевищує максимально допустимий розмір відповіді (в даний час це 4МБ, включаючи імена властивостей, але виключаючи xml-теги, які використовуються для REST).
4. На виконання запиту потрібно більше часу, ніж заданий період очікування сервера (в даний час це 60 секунд)

У будь-якому з цих випадків відповідь буде включати маркер продовження у вигляді спеціального заголовка. Для запиту до ваших сутностей використовуються такі заголовки:

- x-ms-continuation-NextPartitionKey
- x-ms-continuation-NextRowKey

Якщо клієнт отримав ці значення, він повинен передати їх з наступним запитом у вигляді опцій HTTP-запиту; в усьому іншому запит залишається незмінним. Це забезпечить повернення наступного набору сутностей, що починається з місця, позначеного маркером продовження.

Наступний запит виглядає наступним чином:
http://<UriСервиса>/Blogs?<исходный запит>&NextPartitonKey=<деяке значення >&NextRowKey=<другеЗначення>

Це повторюється до тих пір, поки клієнтом не буде отриманий відповідь без маркера продовження, що свідчить про витягнення всіх відповідних запиту результатів. Маркер продовження повинен розглядатися як непрозоре значення. Воно вказує на точку початку наступного запиту і може не відповідати фактичній суті в таблиці. Якщо в таблицю додається нова сутність, так що Key (нова сутність)> Key (остання сутність, витягнута запитом), але Key (нова сутність) < «Маркера продовження», тоді ця нова сутність не буде повернута повторним запитом, використовуємим маркером продовження. Але нові

сутності, додані так, що Key (нова сутність)> «Маркера продовження», увійдуть до результатів, які повертаються наступними використовуємимі маркер продовження запитами.

Тепер розглянемо модель узгодженості, забезпечуємую Windows Azure Table. В рамках однієї таблиці система забезпечує гарантії транзакції ACID для всіх операцій вставки / оновлення / видалення для однієї сутності.

Для запитів в рамках окремої секції виконується ізоляція моментального знімку. Запит забезпечується узгодженим поданням секції з моменту його початку і протягом всієї транзакції. Миттєвий знімок забезпечує наступне:

1. Відсутність «брудного зчитування». Транзакція не бачитиме незафіксовані зміни, внесені іншими транзакціями, які виконуються паралельно. Будуть представлені тільки ті зміни, які були завершені до початку виконання запиту на сервері .

2. Механізм ізоляції миттєвого знімку дозволяє виконувати читання паралельно з оновленням секції без блокування цього оновлення.

Ізоляція миттєвого знімку підтримується тільки в рамках секції і в рамках одного запиту. Система не підтримує ізоляцію миттєвого знімку для декількох секцій таблиці або інших фаз запиту. Додатки відповідають за збереження узгодженості між множиною таблиць.

У прикладі MicroBlogging використовувалося дві таблиці: Channels і Blogs. Додаток виконує узгодження таблиць Channels і Blogs. Наприклад, коли канал видаляється з таблиці Channels, додаток повинен видалити відповідні блоги з таблиці Blogs.

Збої можуть виникати в моменти синхронізації стану множини таблиць. Додаток має вміти обробляти такі збої і мати можливість відновлювати роботу з моменту, на якому вона була перервана.

У попередньому прикладі, коли канал видаляється з таблиці каналів, програма має також видалити всі блоги цього каналу з таблиці Blogs. В ході цього процесу можуть виникати збої програми. Для обробки таких збоїв додаток може зберігати транзакцію в Windows Azure Queues, що дозволяє користувачеві відновити операцію видалення каналу і всіх його блогів навіть в разі збою.

Повернемося до прикладу з таблицями Channels і Blogs. Channels має такі властивості: Name (Ім'я) як PartitionKey, порожній рядок як RowKey, Owner (Власник), CreatedOn (Дата створення). І Blogs має властивості Channel Name (Ім'я каналу) як PartitionKey, CreatedOn як RowKey, Title (Назва), Blog, UserId. Тепер, коли канал видалений, необхідно забезпечити, щоб всі асоційовані з ним блоги також були видалені. Для цього виконуємо наступні кроки:

1. Створюємо чергу для забезпечення узгодженості таблиць, назвемо її «DeleteChannelAndBlogs» (Видалення каналів і блогів).

2. Під час отримання запиту на видалення каналу від ролі веб-інтерфейсу, ставимо в створену вище чергу елемент, який визначає ім'я каналу.

3. Створюємо робочі ролі, які чекатимуть подію додавання елемента в чергу «DeleteChannelAndBlogs».

4. Робоча роль вилучає елемент з черги DeleteChannelAndBlogs, задаючи для витягнутого елемента черги час невидимості протягом N секунд. При цьому

елемент, який визначає ім'я каналу, який повинен бути видалений, вилучається. Якщо роль працівника видаляє елемент черги протягом цих N секунд, даний елемент буде видалено з черги. Якщо ні, елемент стане знову видимим і доступним для використання робочою роллю. Під час вилучення елементу робоча роль робить наступне:

a. У таблиці Channels позначає канал як недійсний, щоб з цього моменту ніхто не міг виконувати читання з нього.

b. Видаляє з таблиці Blogs всі записи, для яких PartitionKey = "Імені каналу", вказаного в елементі черги.

c. Видаляє канал з таблиці Channels.

d. Видаляє елемент з черги.

e. Повертається.

Якщо в ході виконання, наприклад, кроку 4, виникає якийсь збій, проводиться аварійне завершення робочого процесу, при цьому елемент черги не видаляється з неї. Таким чином, як тільки відповідний елемент черги стане знову видимим (тобто коли закінчиться час, заданий як час очікування видимості), це повідомлення буде знову вилучено з черги робочим процесом, і процес видалення відновиться з кроку 4. Більш докладно обробка черг розглядається в документації Windows Azure Queue.

Таким чином, ми ознайомилися з однією з абстракцій даних Windows Azure Storage - Windows Azure Table. Це технологія, яка забезпечує структуроване сховище станів сервісу.

Azure Blob Services

Для роботи з ***Windows Azure Storage*** користувач має створити обліковий запис сховища. Виконується це через веб-інтерфейс порталу Windows Azure Portal. При цьому користувач отримує 256-розрядний секретний ключ, який потім використовується для аутентифікації запитів користувача до системи сховища. Зокрема, цим ключем створюється підпис HMAC SHA256 для запиту. Цей підпис передається з кожним запитом даного користувача для забезпечення аутентифікації через перевірку достовірності підпису HMAC Завдячуючи Windows Azure Blob додаток отримує можливість зберігання в хмарі великих об’єктів, до 50 ГБ кожний. Він підтримує високомасштабуєму систему великих об’єктів (*blob*), в котрій найбільш часто використовуємі blob разподіляються серед великої кількості серверів для обслуговування необхідних об’ємів трафіка. Ця система характеризується високою надійністтю і тривалісттю зберігання. Дані доступні в любий момент часу з любої точки планети і продубльовані, по крайній мірі, тричі для підвищення надійності. Крім того, забезпечується строга відповідність, що гарантирує негайну доступність об’єкту при його добавлені чи обновлені: всі зміни, внесені в попередній операції записі, миттєво видні при наступному читанні.

Azure Blob

Далі розглянемо модель даних Azure Blob.

На рис.6 представлено простір імен Windows Azure Blob.

• Обліковий запис сховища - будь-який доступ до Windows Azure Storage здійснюється через обліковий запис сховища.

o Це найвищий рівень простору імен для доступу до об'єктів blob.

o Обліковий запис може мати безліч контейнерів Blob.

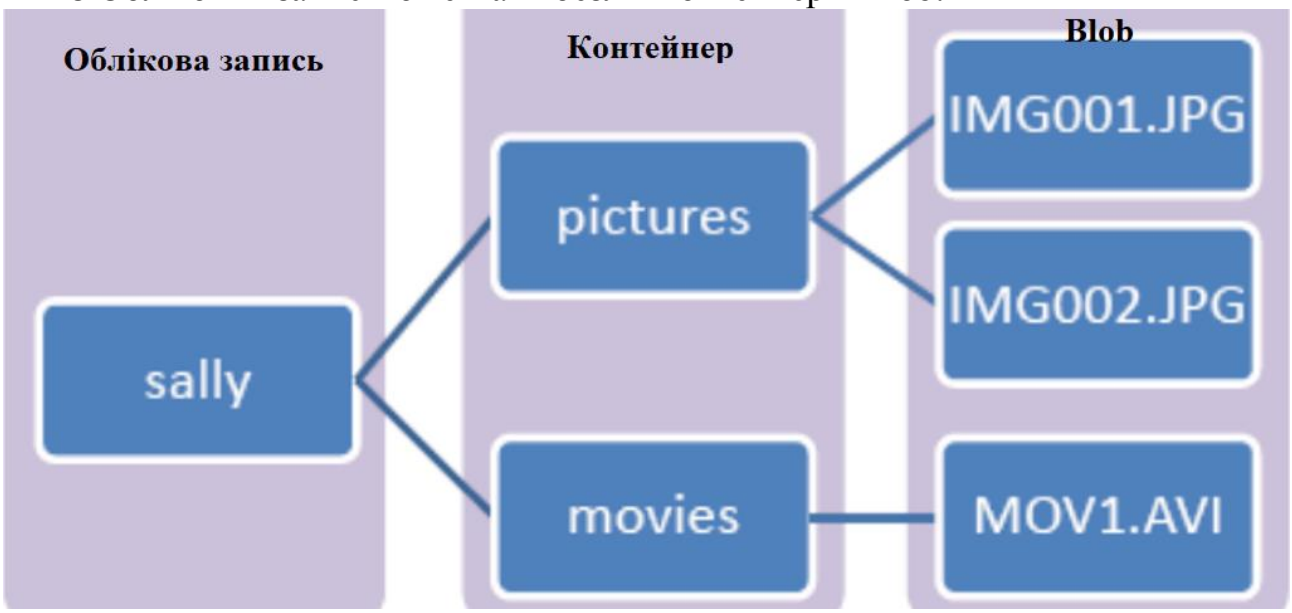


Рис. 6 Загальне уявлення сховища Blob

• Контейнер Blob - контейнер забезпечує угрупування набору об'єктів blob. Область дії імені контейнера обмежена обліковим записом.

- Політики спільного використання задаються на рівні контейнера. В даний час підтримуються «Public READ» (Відкрите читання) і «Private» (Закритий). Якщо для контейнера визначена політика «Public READ», весь його вміст відкрито, доступно для читання без необхідності аутентифікації. Політика «Private» означає доступ з аутентифікацією, тобто тільки власник відповідного облікового запису має доступ до об'єктів blob цього контейнера.
- З контейнером можуть бути асоційовані метадані, які задаються у вигляді пар <ім'я, значення>. Максимальний розмір метаданих контейнера - 8Кб.
- Існує можливість отримання списку всіх об'єктів blob контейнера.

• Blob - Об'єкти blob зберігаються в контейнерах Blob Container та їх область дії обмежена цими контейнерами. Кожен blob може бути розміром до 50ГБ і має унікальне в рамках контейнера строкове ім'я. З blob можуть бути асоційовані метадані, які задаються у вигляді пар <ім'я, значення> і можуть досягати розміру 8Кб для blob. Метадані blob можуть бути отримані і задані окремо від даних blob.

Для доступу до Windows Azure Blob використовується наведені вище підходи. URI конкретного blob структурований таким чином: *http: // <обліковийзапис>.blob.core.windows.net/<контейнер><імяblob>*

Перша частина імені хоста утворена ім'ям облікового запису сховища, за яким слідує ключове слово «*blob*». *Це забезпечує направлення запиту в частину* ***Windows Azure Storage,*** *яка опрацьовує запити* ***blob.*** За іменем хосту іде ім’я

контейнера, «/» і потім ім'я blob. Існують обмеження іменування облікових записів і контейнерів . Наприклад, ім'я контейнера не може включати символ «/». Ще кілька зауважень з приводу контейнерів:

• Як говорилося вище, область дії контейнерів обмежена обліковим записом, якій вони належать. Контейнери зберігаються розподілено, що усуває ймовірність виникнення «вузьких місць» для трафіку при роботі з ними.

• Можлива затримка при відтворенні контейнера, який був недавно вилучений, особливо якщо в цьому контейнері знаходилася велика кількість об'єктів blob. Система повинна очистити об'єкти blob цього контейнера, перш ніж контейнер з таким же ім'ям зможе бути створений знову. Поки сервер видаляє всі об'єкти blob, спроби створення контейнера з таким же ім'ям будуть закінчуватися невдачею з формуванням помилки, що вказує на те, що контейнер знаходиться в процесі видалення.

• Команди на видалення, або створення абсолютно нового контейнера швидко передаються на сервер, і з додатком повертається підтвердження про їх виконання, навіть якщо операція видалення займає деякий час.

Розглянемо інтерфейс REST об'єктів Blob. Будь-який доступ до Windows Azure Blob виконується через стандартні HTTP-команди PUT / GET / DELETE інтерфейсу REST.

До команд HTTP / REST, підтримуваним для реалізації операцій з blob, відносяться:

• PUT Blob - Вставити новий або перезаписати існуючий blob з заданим ім'ям.

• GET Blob - Отримати весь blob або діапазон байтів blob, використовуючи стандартну HTTP-операцію для повернення діапазону GET.

• DELETE Blob - Видалити існуючий blob.

Всі ці операції з blob - Put, Get і Delete, - можуть бути виконані з використанням наступного URL:

http: // <обліковийзапис> .blob.core.windows.net / <контейнер> / <імяblob>

Один запит PUT забезпечує можливість завантаження в хмару blob розміром до 64 МБ. Для завантаження blob, розмір якого перевищує 64 МБ, використовується технологія завантаження блоками, описаними в наступному розділі.

Один з цільових сценаріїв Windows Azure Blob - ефективне завантаження об'єктів blob, розміром десятки гігабайт. Windows Azure Blob забезпечує це наступним чином:

Завантаження Blob (наприклад, Movie.avi) розбивається на послідовні блоки. Наприклад, ролік розміром 10ГБ може бути розбитий на 2500 блоків по 4 Мб, при цьому перший блок буде представляти діапазон байтів даних від 1 до 4194304, другий блок - від 4194305 до 8388608 і т.д.

• Кожному блоку присвоюється унікальний ID / ім'я. Область дії цього унікального ID обмежена ім'ям завантажуємого blob. Наприклад, перший блок може бути названий «Block 0001», другий - «Block 0002» і т.д.

• Кожен блок розміщується в хмарі. Для цього використовується операція PUT із зазначенням представленого вище URL із запитом, що визначає, що це

операція розміщення блоку, і ID блоку. Продовжуючи приклад, при розміщенні першого блоку буде вказано ім'я blob «Movie.avi» і ID блоку «Block 0001».

• Коли всі блоки розміщені в Windows Azure Storage, передається список завантажених блоків, щоб представити blob, з яким вони асоційовані. Виконується це за допомогою операції PUT із зазначенням представленого вище URL із запитом, що визначає, що це команда blocklist. Після цього HTTP-заголовок містить список блоків, які повинні використовуватися в цьому blob. У разі успішного завершення цієї операції отримуємо список блоків, що представляє придатну для читання версію blob. Тепер blob може бути зчитаний за допомогою описаної вище команди GET Blob.

На рис.3.4.7 представлено місце блоків в концепції даних Windows Azure Blob.

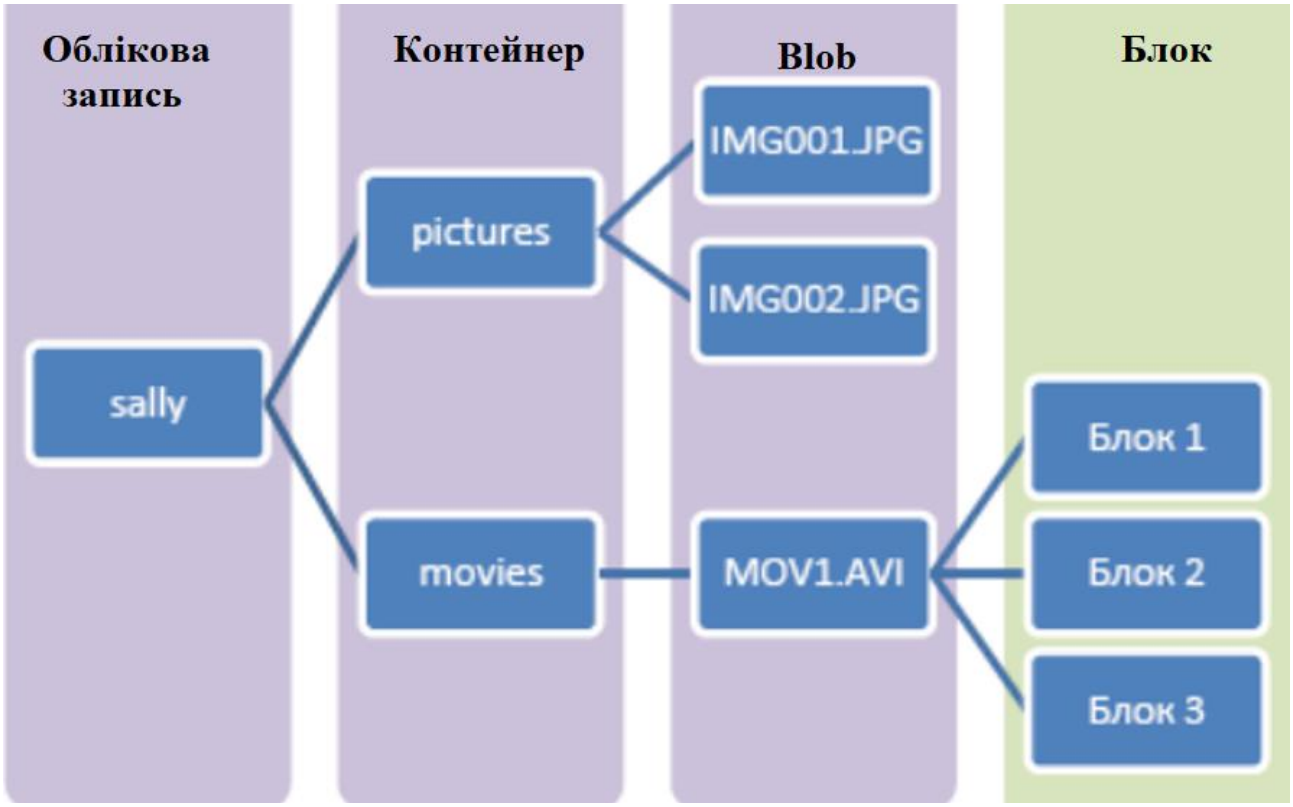


Рис. 7 Загальне уявлення сховища Blob, додавання блоків.

Доступ до об'єктів blob може здійснюватися за допомогою операцій PUT і GET з використанням наступного URL:

http: // <обліковийзапис> .blob.core.windows.net / <контейнер> / <імяblob>

У прикладі, представленому на рис. 7, зображення з наступними URL можуть бути розміщені однією операцією PUT:

http://sally.blob.core.windows.net/pictures/IMG001.JPG
http://sally.blob.core.windows.net/pictures/IMG002.JPG

Ті ж URL можуть використовуватися для повернення об'єктів blob. Одна операція PUT може забезпечити розміщення в сховищі об'єктів blob розміром до 64МБ. Для збереження об'єктів blob розміром більше 64МБ і аж до 50 ГБ необхідно спочатку розмістити всі блоки за допомогою відповідної кількості

операцій PUT і потім, за допомогою все тієї ж операції PUT, передати список блоків, щоб забезпечити придатну для читання версію blob. У прикладі, який ілюструє рис.7, тільки після розміщення всіх блоків і підтвердження їх приналежності blob за допомогою списку блоків blob може бути зчитаний з використанням наступного URL:

http://sally.blob.core.windows.net/pictures/MOV1.AVI

Операції GET завжди виконуються на рівні blob і не припускають використання блоків. Розглянемо абстракції даних блоків. Кожен блок ідентифікує ID блоку розміром до 64 байт. Область дії ID блоку обмежена ім'ям blob, тому різні об'єкти blob можуть мати блоки з однаковими ID. Блоки незмінні. Кожен блок може бути розміром до 4 Мб, і один blob може включати блоки різного розміру. Windows Azure Blob забезпечує наступні операції рівня блоку:

• PUT block - завантажити блок blob. Зверніть увагу, що успішно завантажений за допомогою операції PUT block блок не є частиною blob до тих пір, поки це не буде підтверджено списком блоків, що завантажуються операцією PUT blocklist.

• PUT blocklist - підтвердити blob через надання списку ID блоків, його складових. Зазначені в цій операції блоки повинні бути вже успішно завантажені через виклики PUT. Порядок блоків в операції PUT blocklist забезпечить придатну для читання версію blob.

• GET blocklist - витягти список блоків, переданий раніше для blob операцією PUT blocklist. У повернутому списку блоків вказуються ID і розмір кожного блоку.

У всіх розглянутих далі прикладах використовується blob «MOV1.avi», що розташовується в контейнері «movies» (ролики) під обліковим записом «sally».

Нижче представлений приклад REST-запиту для розміщення блоку розміром 4 Мб за допомогою операції PUT block. Зверніть увагу, що використовується HTTP-команда PUT. «? Comp = block» вказує на те, що це операція PUT block. Потім задається BlockID. Параметр Content-MD5 може бути заданий для захисту від помилок передачі по мережі і забезпечення цілісності. В даному випадку, Content-MD5 - це контрольна сума MD5 даних блоку в запиті. Контрольна сума перевіряється на сервері, в разі розбіжності повертається помилка. Параметр Content-Length (Довжина вмісту) визначає розмір вмісту блоку. Також в заголовку HTTP-запиту є заголовок авторизації, як показано нижче.

PUT http://sally.blob.core.windows.net/movies/MOV1.avi
? Comp = block & blockid = BlockId1 & timeout = 60
HTTP / 1.1 Content-Length: 4194304
Content-MD5: HUXZLQLMuI / KZ5KDcJPcOA ==
Authorization: SharedKey sally: F5a + dUDvef +
PfMb4T8Rc2jHcwfK58KecSZY + l2naIao =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
..... Block Data Contents .....

Нижче представлений приклад REST-запиту для операції PUT blocklist. Зверніть увагу, що використовується HTTP-команда PUT. «? Comp = blocklist» вказує на те, що це операція PUT blocklist. Список блоків задається в тілі HTTP-запиту в форматі XML, як показано в прикладі нижче. Зверніть увагу, що значення поля Content-Length в заголовку запиту відповідає розміру тіла запиту, а не розміру створюваного blob. Також в заголовку HTTP-запиту є заголовок авторизації, як показано нижче.

PUT http://sally.blob.core.windows.net/movies/MOV1.avi
? Comp = blocklist & timeout = 120
HTTP / 1.1 Content-Length: 161213
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
<? Xml version = "1.0" encoding = "utf-8"?>
<BlockList>
<Block> BlockId1 </ Block>
<Block> BlockId2 </ Block>
.....
</ BlockList>

Нижче представлений приклад REST-запиту для операції GET blob. В даному випадку використовується HTTP-команда GET. Цей запит забезпечить витяг всього вмісту заданого blob. Якщо для контейнера, якому належить blob (в даному прикладі «Movies»), задана політика спільного використання «Private», для отримання blob необхідно пройти аутентифікацію. Якщо задана політика спільного використання «Public-Read», аутентифікація не вимагається і заголовок аутентифікації в заголовку запиту не потрібен.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP / 1.1
Authorization: SharedKey sally: RGllHMtzKMi4y / nedSk5Vn74IU6 / fRMwiPsL + uYSDjY =
X-ms-date: Mon, 27 Oct 2016 17:00:25 GMT

Як показано в прикладі нижче, також підтримується операція GET для витягу діапазону байта заданого blob.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP/1.1
Range: bytes=1024000-2048000

Завантаження blob у вигляді списку блоків має наступні переваги:

• Можливість продовження - для кожного блоку окремо можна перевірити успішність завантаження, в разі збою повторити спробу завантаження і продовжити виконання з цього моменту.

• Паралельне завантаження - завантаження блоків може виконуватися паралельно, що забезпечує скорочення часу завантаження дуже великих об'єктів blob.

• Завантаження не по порядку - Блоки можуть завантажуватися в довільному порядку. Значення має лише порядок блоків в списку операції PUT blocklist.

Список блоків в операції PUT blocklist визначає придатну для читання версію blob.

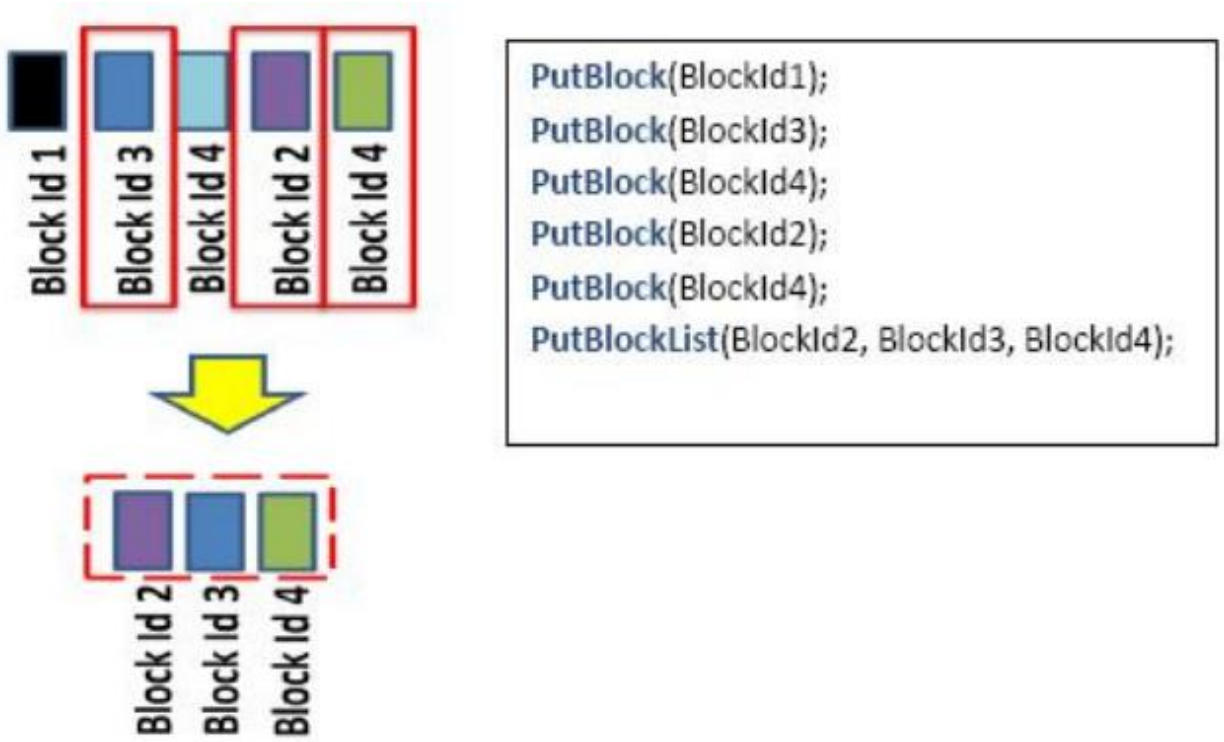


Рис. 8 Сценарій завантаження блоків

Використовуючи представлений на рис. 8 приклад, опишемо різні сценарії, можливі при використанні блоків для завантаження об'єктів blob:

• Завантаження блоків з однаковими ID блоку - коли для одного blob завантажуються блоки з однаковими ID блоку, при формуванні остаточно версії blob в операції PUT blocklist з усіх блоків з однаковим ID буде використовуватися тільки завантажений найостаннішим. В наведеному вище прикладі завантажуються два блоки з BlockId4 і тільки другий з них буде використовуватися в остаточному списку блоків blob.

• Завантаження блоків в довільному порядку - блоки можуть завантажуватися в порядку, відмінному від зазначеного в остаточному списку блоків blob. В наведеному вище прикладі в остаточному списку блоки розташовуються в порядку BlockId2, BlockId3 і BlockId4, але завантажувалися вони в іншій послідовності. Упорядкування даних blob (через операцію GET) в придатну для читання версію виконується відповідно до списку, зазначеного в операції PUT blocklist.

• Невикористані блоки - деякі блоки можуть ніколи не увійти в остаточний список блоків blob. Ці блоки будуть видалені системою в процесі збору «сміття». У розглянутому прикладі такими блоками є BlockId1 і перший блок з ID BlockId4. Точніше кажучи, як тільки blob створений за допомогою операції PUT blocklist, всі завантажені, але які не увійшли в список блоків операції PUT blocklist блоки будуть вилучені шляхом видалення «сміття».

Завантаження великого blob може займати досить тривалий час. При цьому завантажені, але не використані блоки займають місце в сховищі. Багато завантажених блоків можуть ніколи не увійти в список PUT blocklist. У разі

відсутності активності для даного blob протягом тривалого періоду часу (в даний час цей період становить тиждень), ці невикористані блоки будуть видалені системою як сміття.

Цікавим є сценарій паралельного завантаження блоків для одного blob. В цьому випадку заслуговують на увагу два питання:

• ID блоків - якщо для завантаження блоків в один blob додаток використовує множину клієнтських модулів записі, щоб уникнути конфліктів ID блоків повинні бути унікальними серед всіх цих модулів записі, або вони повинні представляти вміст записуємого блоку (таким чином, якщо один і той же блок записується декількома клієнтами, ID блоку у всіх клієнтів буде однаковим, оскільки він представляє одні й ті ж дані). Щоб уникнути помилок при потенційній можливості запису одного Blob декількома модулями записі в якості ID блоку рекомендується використовувати хеш (наприклад, MD5-хеш) вмісту блоку. Таким чином, ID блоку представлятиме його вміст.

• Пріоритет має перша фіксація - в ситуації, коли безліч клієнтів виконують завантаження блоків для одного blob паралельно, пріоритет має перша фіксація blob за операції PUT blocklist (або модуль запису, який викликав PUT blob першим). Всі інші невикористані блоки, завантажені іншими модулями записі для blob з цим ім'ям, будуть видалені в процесі видалення "сміття". Отже, для ефективного паралельного поновлення blob необхідна координація всіх клієнтів, які ведуть запис паралельно.

Windows Azure Blob підтримує умовні PUT і GET, які забезпечують реалізацію ефективної обробки паралелізму і клієнтського кешування.

Умовний PUT може використовуватися в ситуаціях, коли один blob оновлюється кількома користувачами. Наприклад, завантаження blob може виконуватися підчас останньої зміни; це гарантує, що версія змінюваного blob аналогічна версії, яку змінює клієнт. Так може бути реалізований спільний доступ з нежорстким блокуванням. Скажімо, два клієнта, A і B, оновлюють один і той же blob. Вони паралельно виконують читання версії blob, вносять в неї якісь зміни і знову завантажують в сховище. У цьому сценарії кожен з клієнтів записує час останньої зміни витягнутого зі сховища blob (нехай час останньої зміни буде X). Коли вони готові завантажити оновлену версію blob назад в сховище, вони роблять це за допомогою умовного PUT на підставі збереженого при добуванні blob часу останньої зміни. В операції повинно бути визначено, що умовою виконання PUT є «якщо не змінювався з моменту X». Таким чином, якщо blob був змінений іншим клієнтом в проміжок часу з моменту X, операція оновлення дасть збій, і клієнт отримає повідомлення про це.

Умовний GET може використовуватися для ефективної обробки питань відповідності вмісту кешів. Наприклад, клієнт має локальний кеш об'єктів blob, в якому кешуються найчастіше викликаємі зі сховища blob. Для кожного кешованого blob записується час його останньої зміни. Коли клієнтський кеш приймає рішення оновити об'єкти blob зі сховища, він може використовувати умовний GET на підставі часу зміни (з умовою «якщо змінений з моменту X»). Таким чином, зі сховища будуть завантажуватися тільки ті об'єкти blob, які

були змінені в період часу з моменту X і відрізняються від своєї керованої копії.

Система Blob забезпечує інтерфейс для перечислення об'єктів *blob* контейнера. Підтримується ієрархічний перелік об'єктів *blob* контейнера і механизм продовження, що забезпечує виконання великої кількості об'єктів *blob*.

Интерфейс *ListBlobs* підтримує параметри *«prefix»* (префікс) і *«delimiter»* (розділювач), які забезпечують можливість побудови ієрархічного переліку об'єктів *blob*. Наприклад, хай в обліковому запису *«sally»* є контейнер *«movies»* объектів *blob* з такими іменами:

Action/Rocky1.wmv
Action/Rocky2.wmv
Action/Rocky3.wmv
Action/Rocky4.wmv
Action/Rocky5.wmv
Drama/Crime/GodFather1.wmv
Drama/Crime/GodFather2.wmv
Drama/Memento.wmv
Horror/TheBlob.wmv

Як бачите, «/» використовується як роздільник для створення подібної каталогу ієрархії імен blob. Щоб отримати список всіх папок , задаємо в запиті ListBlobs «delimiter = /». І ось як буде виглядати запит і частина відповіді:

запит:

GET http://sally.blob.windows.net/movies?comp=list&delimiter=/

Відповідь:

<BlobPrefix>Action</BlobPrefix>
<BlobPrefix>Drama</BlobPrefix>
<BlobPrefix>Horror</BlobPrefix>

Зверніть увагу, тег «BlobPrefix» вказує на те, що відповідний запис є префіксом імені blob, а не повним ім'ям blob. Також слід зауважити, що один і той же префікс повертається в результаті тільки один раз.

Наступним етапом можна поєднувати префікс і роздільник для отримання списку вмісту підпапки. Наприклад, задаючи «prefix = Drama /» і «delimiter = /», отримуємо список всіх підпапок і файлів каталогу «Drama»:

запит:

GET http://sally.blob.windows.net/movies?comp=list &prefix=Drama/ &delimiter=/

відповідь:

<BlobPrefix>Drama/Crime</BlobPrefix>
<Blob>Drama/Memento.wmv</Blob>

«Drama / Memento.wmv» - це повне ім'я blob, отже, воно так і позначено.

Інтерфейс ListBlobs забезпечує можливість задавати «maxresults», тобто максимальне число результатів, яке повинно бути повернуто в цьому виклику. Більш того, система визначає верхню межу для максимального числа результатів, які можуть бути повернуті одним викликом . Після досягнення меншого з цих двох граничних значень виклик повертається з відповідною кількістю результатів і непрозорим «NextMarker» (маркер наступного). Наявність цього маркеру свідчить про те, що даний запит не забезпечив повернення всіх можливих результатів. «NextMarker» може використовуватися для продовження складання списку для наступної сторінки результатів.

У попередньому прикладі припустимо, що потрібно скласти список всіх об'єктів blob каталогу «Action», повертаючи кожен раз максимум по 3 результату. У цьому випадку перший набір результатів був би таким:

запит:

*GET http://sally.blob.windows.net/movies?comp=list &prefix=Action &maxresults=3 *

віповідь:

<Blob>Action/Rocky1.wmv</Blob>
<Blob>Action/Rocky2.wmv</Blob>
<Blob>Action/Rocky3.wmv</Blob>
<NextMarker> OpaqueMarker1</NextMarker>

З першим набором об'єктів blob повертається і непрозорий маркер, який може бути переданий в другій виклик ListBlobs. Тоді цей виклик забезпечить повернення таких результатів:

запит:

GET http://sally.blob.windows.net/movies?comp=list &prefix=Action &maxresults=3
&marker=OpaqueMarker1

відповідь:

<Blob>Action/Rocky4.wmv</Blob>
<Blob>Action/Rocky5.wmv</Blob>
<NextMarker></NextMarker>

Як показано вище, повернуті залишені об'єкти blob каталогу; «NextMarker» порожній, це свідчить про те, що отримані всі результати.

Azure Queue Services

Windows Azure Queue надає надійний механізм доставки повідомлень. Вона пропонує простий алгоритм диспетчеризації асинхронних завдань, який забезпечує можливість підключення до різних компонентів додатку в хмарі. Черги Windows Azure Queue характеризуються високою надійністю, постійністю і продуктивністю. Поточна реалізація гарантує одноразову

обробку повідомлення. Більш того, Windows Azure Queue має REST-інтерфейс, таким чином, додатки можуть створюватися на будь-якій мові програмування і виконувати доступ до черги через веб в будь-який час з будь-якої точки Інтернету.

Розглянемо створення додатків в хмарі з використанням Azure Queue. Windows Azure Queue дозволяє розділити різні частини програми в хмарі, що робить можливим використання різних технологій для створення цих додатків і їх масштабування відповідно до потреб трафіку. Рис. 9. ілюструє простий і поширений сценарій для додатків в хмарі.

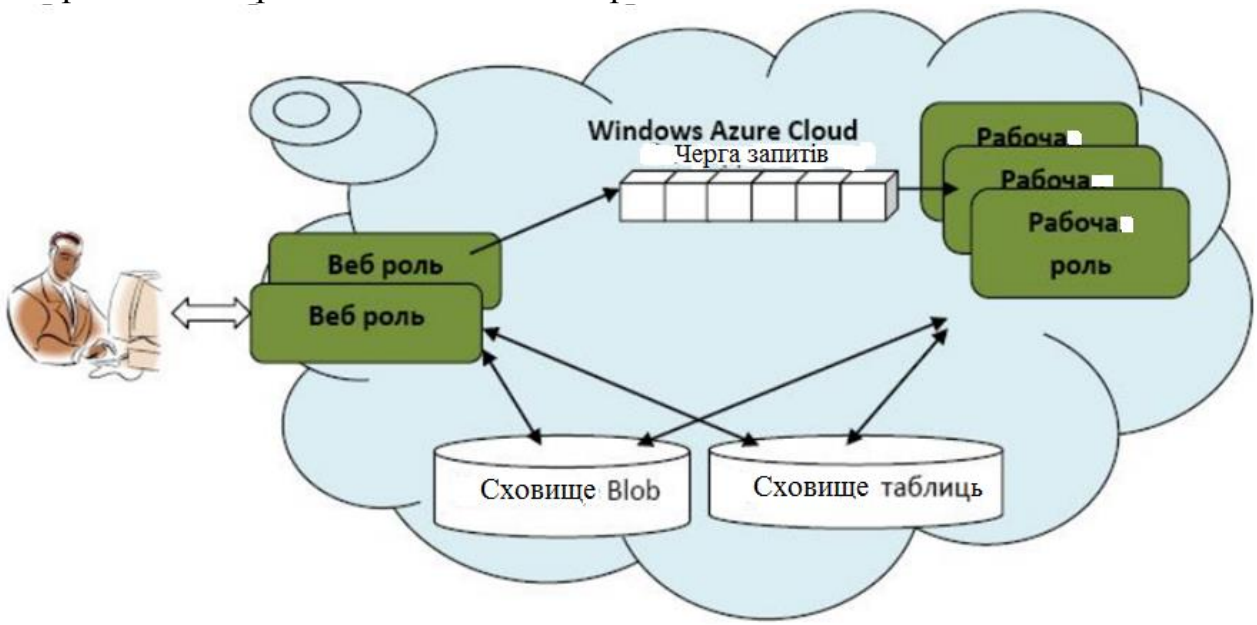


Рис. 9. Побудова додатків для хмари з використанням Azure Queue

Є ряд веб ролей, на яких розміщується інтерфейсна логіка обробки веб-запитів. Також існує ряд робочих ролей, які реалізують бізнес-логіку програми. Веб ролі обмінюються інформацією з робочими ролями за допомогою наборів запитів.

Розглянемо, як приклад, додаток онлайн сервісу відеохостингу. Користувачі можуть завантажувати відео в цей додаток; після цього додаток може автоматично перетворювати і зберігати цей відеофайл в різних форматах мультимедіа; крім того, додаток автоматично індексує дані опису відео для спрощення пошуку (наприклад, за ключовими словами опису, іменами акторів, режисерів, назвою і т.д.).

Такий додаток може використовувати описану раніше архітектуру. Веб-ролі реалізують рівень представлення і обробляють веб-запити користувачів. Користувачі можуть завантажувати відео через веб-інтерфейси. Відео-файли можуть зберігатися як великі двійкові об'єкти в сховище Azure Blob. Додаток може також обслуговувати ряд таблиць для обліку наявних відеофайлів і зберігання індексів, використовуваних для пошуку. Робочі ролі виконують перетворення вхідних відеофайлів в різні формати і збереження їх в сховищі Azure Blob. Робочі ролі також відповідають за оновлення таблиць цього додатку в Azure Table. При отриманні запиту користувача (наприклад, запиту на завантаження відео) веб-ролі створюють робочий елемент і розміщують його

в чергу запитів. Робочі ролі витягають ці робочі елементи з черги і оброблюють відповідним чином. У разі успішної обробки робоча роль повинна видалити робочий елемент з черги, щоб уникнути повторної його обробки іншою робочою роллю.

Завдяки застосуванню Windows Azure Queue така архітектура має ряд переваг:

1. Масштабованість - Додаток може легше масштабуватися відповідно до потреб трафіку. Переваги, пов'язані з масштабуванням: По-перше, довжина черги безпосередньо відображає наскільки добре робочі ролі справляються з загальним робочим навантаженням. Збільшення черги свідчить про те, що робочі ролі не можуть обробляти дані з необхідною швидкістю. В цьому випадку додатку може знадобитися збільшити кількість робочих ролей, щоб забезпечити більш швидку обробку. Якщо довжина черги незмінно залишається близькою до нуля, це означає, що серверна частина програми має більші обчислювальні потужності, ніж потрібно. У цьому випадку програма може скоротити кількість робочих ролей для забезпечення раціонального витрачання ресурсів. Відстежуючи розміри черги і налаштовуючи кількість внутрішніх вузлів, додатки можуть ефективно масштабуватися відповідно до обсягів трафіку. По-друге, застосування черг дозволяє розділити частини програми і виконувати їх масштабування незалежно один від одного, що набагато спрощує це завдання. Додаток може вільно масштабувати критично важливі компоненти, додаючи для них більше ресурсів / комп'ютерів. По-третє, застосування черг забезпечує гнучкість для ефективного використання ресурсів в додатку, що підвищує ефективність масштабування. Тобто для робочих елементів з різними пріоритетами і / або різних розмірів можуть використовуватися різні черги, які будуть оброблятися різними пулами робочих ролей.
2. Поділ ролей - Використання черг дозволяє розділити різні частини програми, що забезпечує істотну гнучкість і розширюваність з точки зору побудови програми. Повідомлення в черзі можуть бути в стандартному або розширюваному форматі, такому як XML, що забезпечує незалежність компонентів на обох кінцях черги один від одного, оскільки вони можуть розуміти повідомлення в черзі. Різні частини системи можуть бути реалізовані з використанням різних технологій і мов програмування. Наприклад, компонент на стороні черги може бути написана на .NET Framework, а інший компонент – на Python. Більше того, зімни, проходящі в середині компонента, не мають ніякого впливу на остальну систему. Наприклад, компонент може бути зіменим з використанням зовсім іншої технології чи мови програмування і при цьому система буде продовжувати працювати і для цього не прийдеться змінювати інші компоненти так як використання черг забезпечує розділення компонентів. Крім того використання черг забезпечує співіснування в системі різних реалізацій одного і того ж компоненту, отже додаток може вільно переходити до нових технологій: стара і нова реалізації можуть виконуватися одночасно на різних серверах і

опрацьовувати робочі елементи однієї черги. При цьому інші компоненти додатку ніяк не будуть зачеплені.

3. Сплески трафіку - Черги забезпечують буферизацію, що компенсує сплески трафіку і скорочує вплив, який чиниться збоями окремих компонентів. У розглянутому раніше прикладі можливі випадки надходження великої кількості запитів в короткий проміжок часу. Робочі ролі не можуть швидко обробити всі запити. У цьому випадку запити не відхиляються, а буферизуются в чергу, і робочі ролі отримують можливість обробляти їх у власному нормальному темпі, поступово повертаючись до звичайного режиму роботи. Це дозволяє додатку обробляти нерівномірний трафік без зниження надійності.

Завдяки моделі черги додатки застраховані від втрати даних і зниження надійності навіть в умовах систематичних збоїв компонентів додатку.

Windows Azure Queue має наступну модель даних.

• Облікова запись сховища - Будь-який доступ до Windows Azure Storage здійснюється через обліковиу запись сховища.

• Це найвищий рівень простору імен для доступу до черг і їх повідомлень. Для використання Windows Azure Storage користувач має створити облікову запись сховища. Виконується це через веб-інтерфейс порталу Windows Azure Portal.

При створенні облікового запису користувач отримує 256-розрядний таємний ключ, котрий надалі використовується для аутентифікації запитів цього (за допомогою цього таємного ключа створюється підпис HMAC SHA256 для запиту). Цей підпис передається з кожним запитом для аутентифікації через перевірку достовірності підпису HMAC.

• Облікова запись може мати багато черг.

Черга – черга має багато сповіщень. Область дії імені черги обмежена обліковим записом.

1. Кількість сповіщень в черзі не обмежено.
2. Сповіщення зберігаються максимум тиждень. Система видаляє сповіщення, поступивші більше тиждня назад .
3. З чергами можуть бути асоційовані метадані. Метадані представляються в формі пар <*имя, значение*>,їх розмір може складати максимум 8КБ на чергу.

• Повідомлення

Повідомлення зберігаються в чергах. Кожне повідомлення може бути розміром не більше 8Кб. Для зберігання даних більшого розміру використовуються сховища Azure Blob або Azure Table, а в повідомленні вказується ім'я великого двійкового об'єкта / сутності. Зверніть увагу, що коли повідомлення поміщається в сховище, його дані можуть бути двійковими. Але при добуванні повідомлень зі сховища відповідь формується в форматі XML, і дані повідомлення повертаються base64-кодованими. Повідомлення можуть повертатися з черги в будь-якому порядку, і повідомлення може бути

повернуто кілька разів. Розглянемо деякі параметри, які використовуються Azure Queue Service:

1. MessageID: Значення GUID, яке ідентифікує повідомлення в черзі.

2. VisibilityTimeout: Ціле значення, яке визначає час очікування видимості повідомлення в секундах. Максимальне значення - 2 години. Значення за замовчуванням - 30 секунд.

3. PopReceipt: Рядок, який повертається для кожного витягнутого повідомлення. Цей рядок, разом з MessageID, необхідний для видалення повідомлення з черги (Queue). Даний параметр слід розглядати як непрозорий, оскільки в майбутньому його формат і вміст можуть бути змінені.

4. MessageTTL: Визначає термін життя (time-to-live, TTL) повідомлення в секундах. Максимально допустимий термін життя - 7 днів. Якщо цей параметр опущений, термін життя за замовчуванням - 7 днів. Якщо протягом терміну життя повідомлення не буде видалене з черги, воно буде видалено системою зберігання в процесі очищення.

URI конкретної черги структуровано таким чином:

http: // <облікова запись> .queue.core.windows.net / <Імя черги>

Перша частина імені хоста утворена ім'ям облікового запису сховища, за яким слідє ключове слово «queue». Це забезпечує направлення запиту до частини Windows Azure Storage, яка обробляє запити черги. За ім'ям хоста йде ім'я черги. Існують обмеження іменування облікових записів і черг . Наприклад, ім'я черги не може включати символ «/».

Будь-який доступ до Windows Azure Queue виконується через HTTP-інтерфейс REST. Підтримуються як HTTP, так і HTTPS протоколи.

До командам HTTP / REST на рівні облікового запису відносяться:

• List Queues - Представити список всіх черг цього облікового запису.

До командам HTTP / REST на рівні черги відносяться:

• Create Queue - Створити чергу під даним обліковим записом.

• Delete Queue - Видалити зазначену чергу і її вміст без можливості відновлення.

• Set Queue Metadata - Задати / оновити визначені користувачем метадані черги. Метадані асоційовані з чергою в вигляді пар ім'я / значення. Ця команда забезпечить перезапис всіх існуючих метаданих новими.

• Get Queue Metadata - Витягти визначені користувачем метадані черги, а також приблизну кількість повідомлень в заданій черзі.

Операції рівня черги можуть виконуватися з використанням наступного URL:

http: // <обліковий запис> .queue.core.windows.net / <ІмяЧерги>

До командам HTTP / REST, підтримуваним для реалізації операцій на рівні повідомлення, відносяться:

• PutMessage (QueueName, Message, MessageTTL) - Додати нове повідомлення в кінець черги. MessageTTL визначає термін життя даного повідомлення. Повідомлення може зберігатися в текстовому або двійковому форматі, але повертається base64-кодованим.

• GetMessages (QueueName, NumOfMessages N, VisibilityTimeout T) - Витягти N повідомлень з початку черги і зробити їх невидимими протягом заданого VisibilityTimeout проміжку часу T. Ця операція поверне ID повідомлення, повернутого разом з PopReceipt. Повідомлення можуть повертатися з черги в будь-якому порядку, і повідомлення може бути повернуто кілька разів.

• DeleteMessage (QueueName, MessageID, PopReceipt) - Видалити повідомлення, асоційоване з даними PopReceipt, повернутим раніше викликом GetMessage. Зверніть увагу, що якщо повідомлення не буде видалене, воно повторно з'явиться в черзі після закінчення VisibilityTimeout.

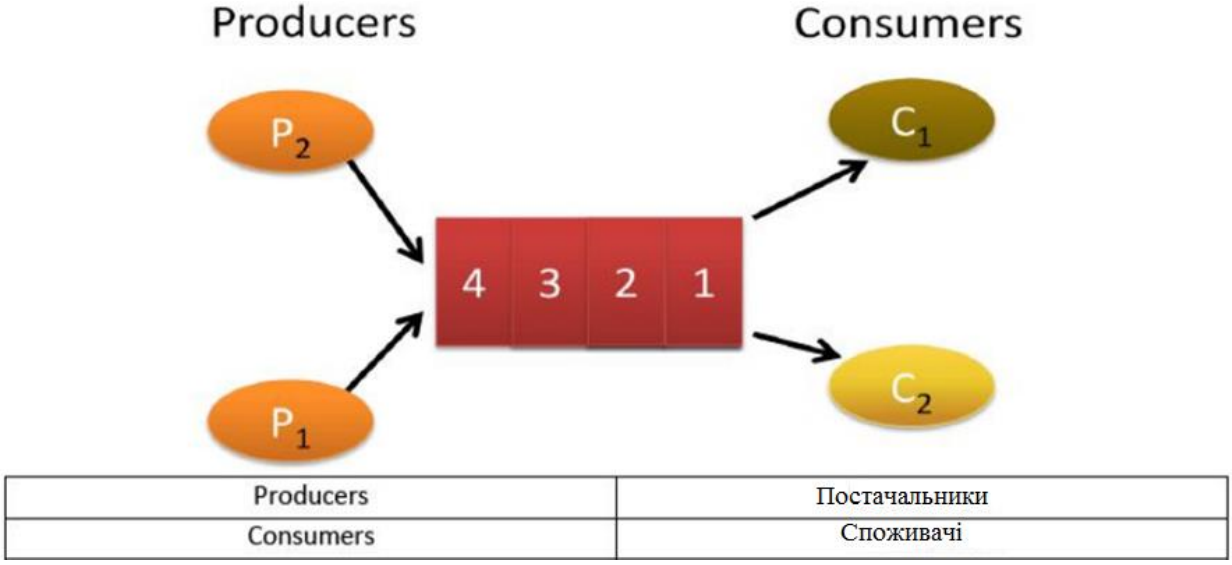
• PeekMessage (QueueName, NumOfMessages N) - Витягнути N повідомлень з початку черги, не роблячи повідомлення невидимими. Ця операція поверне ID повідомлення для кожного повернутого повідомлення.

• ClearQueue - Видалити всі повідомлення з заданої черги. Зверніть увагу, що викликаюча сторона повинна повторювати цю операцію до тих пір, поки отримує повідомлення про успішне її виконання, це забезпечить видалення всіх повідомлень черги.

Операції рівня повідомлення можуть бути виконані з використанням наступного URL:

http: // <учетнаязапісь> .queue.core.windows.net / <ІмяОчереді> / messages

На рис. 10 наведено приклад, який ілюструє семантику Windows Azure Queue.



Producers	Постачальники
Consumers	Споживачі

рис. 10 Приклад використання черги

У цьому прикладі постачальники (P1 і P2) і споживачі (C1 і C2) обмінюються інформацією через подану вище чергу. Постачальники формують робочі елементи і поміщають їх у вигляді повідомлень в чергу. Споживачі вилучають повідомлення / робочі елементи з черги і обробляють їх. Може існувати багато постачальників і багато споживачів. Розглянемо послідовність дій:

1. C1 витягує повідомлення з черги. Ця операція повертає повідомлення 1 і робить його невидимим в черзі на 30 секунд (приймаємо в даному прикладі, що використовується VisibilityTimeout за замовчуванням, що складає 30 секунд).

2. Потім з'являється C2 і витягує з черги ще одне повідомлення. Оскільки повідомлення 1 як і раніше невидиме, ця операція не побачить повідомлення 1 і поверне C2 повідомлення 2.

3. По завершенні обробки повідомлення 2 C2 викликає Delete, щоб видалити повідомлення 2 з черги.

4. Тепер уявімо, що C1 дає збій в ході обробки повідомлення 1, таким чином, C1 не видаляє це повідомлення з черги.

5. Після завершення часу VisibilityTimeout для повідомлення 1, воно знову з'являється в черзі.

6. Після того, як повідомлення 1 повторно з'явилося в черзі, воно може бути вилучено наступним викликом від C2. Тоді C2 повністю обробить повідомлення 1 і видалить його з черги.

Як показано в цьому прикладі, семантика API черги гарантує кожному повідомленні в черзі шанс бути обробленим повністю, в крайньому разі, один раз. Тобто, якщо виникає збій споживача в період після вилучення ним повідомлення з черги і до його видалення, повідомлення знову з'явиться в черзі після закінчення часу VisibilityTimeout. Це забезпечує можливість цьому повідомленню бути обробленим повністю іншим споживачем.

Розглянемо REST-запити, які використовуються Windows Azure Queue.

Нижче показаний приклад REST-запиту для операції постановки в чергу. Зверніть увагу, що використовується HTTP-команда PUT. Задається необов'язкова опція «messagettl», яка визначає термін життя повідомлення в секундах. Максимальний термін життя - 7 днів. Якщо цей параметр опущений, значення за замовчуванням - 7 днів. Після закінчення терміну життя повідомлення буде видалено системою . Параметр Content-MD5 може бути заданий для захисту від помилок передачі по мережі і забезпечення цілісності. В даному випадку, Content-MD5 - це контрольна сума MD5 даних повідомлення в запиті. Параметр Content-Length (Довжина вмісту) визначає розмір вмісту повідомлення. Також в заголовку HTTP-запиту є заголовок авторизації. Зверніть увагу, що дані повідомлення розташовуються в тілі HTTP-запиту. Повідомлення може зберігатися в текстовому або двійковому форматі, але при добуванні воно повертається base64-кодованим.

PUT http://sally.queue.core.windows.net/myqueue/messages
? messagettl = 3600
HTTP / 1.1 Content-Length 3900
Content-MD5: HUXZLQLMuI / KZ5KDcJPcOA ==
Authorization: SharedKey sally: F5a + dUDvef +
PfMb4T8Rc2jHcwfK58KecSZY + l2naIao =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
Message Data Contents .....

Нижче показаний приклад REST-запиту для операції вилучення з черги. Зверніть увагу, що використовується HTTP-команда GET. Задані два необов'язкові параметри. «Numofmessages» визначає, скільки повідомлень

повинно бути вилучено з черги; максимальне число - 32. За замовчуванням витягується одне повідомлення. У прикладі нижче буде вилучатись по 2 повідомлення. Параметр «visibilitytimeout» визначає час очікування видимості; повідомлення буде залишатися невидимим в черзі, протягом цього проміжку часу, в секундах, і знову з'явиться в черзі, якщо не буде видалено до завершення періоду очікування видимості. Максимальне значення цього часу очікування - 2 години, і значення за замовчуванням - 30 секунд. У прикладі нижче час очікування видимості задано рівним 60 секундам. Також в заголовку HTTP-запиту є елемент авторизації. Зверніть увагу, що відповідь надходить в XML-форматі, і ці повідомлення у відповіді будуть base64-кодованими (в прикладі нижче розташовуються між тегами <MessageText> </ MessageText>).

GET http://sally.queue.core.windows.net/myqueue/messages
?numofmessages=2 &visibilitytimeout=60
HTTP/1.1
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw=
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT

Відповідь на цей вклик буде аналогічним, отримуємому в наступному прикладі:

HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 22fd6f9b-d638-4c30-b686-519af9c3d33d
Date: Thu, 13 Nov 2008 21:37:56 GMT
<?xml version="1.0" encoding="utf-8"?>
<QueueMessagesList>
<QueueMessage>
<MessageId>6012a834-f3cf-410f-bddd-dc29ee36de2a</MessageId>
<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD/////AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWUuU
2VydmljZ
S5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN1bHR1cmU9
bmV1dHJhbCwg
UHVibGljS2VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5DaXMuU2VydmljZXM
uTmVwaG9zLlF1
ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVldWVNYW5hZ2VyK
1JYVpcHQCAAAA
FjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWN
raW5nRmllbGQ
DAAtTeXN0ZW0uR3VpZA0CAAAABP3///8LU3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9j
Al9kAl9lAl9mAl
9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAIBwcCAgICAgICAjSoEmDP8w9Bvd3cKe
423ipfNapL7xPL

SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>
<MessageText>......</MessageText>
</QueueMessage>
<QueueMessage>
<MessageId>2ab3f8e5-b0f1-4641-be26-83514a4ef0a3</MessageId>
<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD/////AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWUuU
2VydmljZ
S5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN1bHR1cmU9
bmV1dHJhbCwg
UHVibGljS2VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5DaXMuU2VydmljZXM
uTmVwaG9zLlF1
ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVldWVNYW5hZ2VyK
1JYVpcHQCAAAA
FjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWN
raW5nRmllbGQ
DAAtTeXN0ZW0uR3VpZA0CAAAABP3///8LU3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9j
Al9kAl9lAl9mAl
9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAAIBwcCAgICAgICAuX4syrxsEFGviaDUU
pO8KNfNapL7xPL
SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>
<MessageText>.....</MessageText>
</QueueMessage>
</QueueMessagesList>

Нижче представлений приклад REST-запиту для операції видалення повідомлення. У цьому випадку використовується HTTP-команда DELETE. Параметр «popreceipt» визначає повідомлення, яке повинно бути видалено. «Popreceipt» виходить в результаті попередньої операції вилучення з черги, як показано в прикладі раніше.

DELETE /sally/myqueue/messages/6012a834-f3cf-410f-
bddddc29ee36de2a?popreceipt=AAEAAAD%2f%2f%2f%2f%2fAQAAAAAAAAAMA
gAAAFxOZXBob3Mu
UXVldWUuU2VydmljZS5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4
wLjAsIEN1bHR1c

mU9bmV1dHJhbCwgUHVibGljSV5VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5
DaXMuU2Vyd
mljZXMuTmVwaG9zLlF1ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWFs
UXVldWVNYW5
hZ2VyKJlY2VpcHQCAAAAFjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWx
pdHlTdGFydD5rX
19CYWNraW5nRmllbGQDAAtTeXN0ZW0uR3VpZA0CAAAABP3%2f%2f%2f8LU3lz
dGVtLkd1aWQL
AAAAAl9hAl9iAl9jAl9kAl9lAl9mAl9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAIBwc
CAgICAgICAjSoE
mDP8w9Bvd3cKe423ipfNapL7xPSAsAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA%3d&timeou
t=30
HTTP/1.1
Content-Type: binary/octet-stream
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT
Authorization: SharedKey
sally:M/N65zg/5hjEuUS1YGCbVDHfGnI7aCAudkuTHpCDvZY=

Інтерфейс програмування додатків Windows Azure SDK .

Windows Azure SDK надає розробникам інтерфейс програмування додатків, необхідний для розробки, розгортання та управління масштабованих сервісів в Windows Azure.

В даній лекції ми розглянемо основні можливості Windows Azure SDK. Мета даної лекції - ознайомитися з комплектом засобів розробки Windows Azure SDK.

Azure Cloud Fabric і служби Azure Storage не підтримують розробку або налагоджувальні операції в хмарі, тому Azure SDK дозволяє робити це локально у вигляді додатків *Development Fabric (DF)* і *Development Storage (DS)*, які встановлює Windows Azure SDK. Разом з SDK також встановлюються колекція додатків прикладів і бібліотеки упакованих класів для полегшення програмування додатків.

Повинні бути встановлені .NET Framework 3.5 SP1 і SQL Express 2005 or 2008, також необхідно включити ASP.NET і WCF HTTP Activation для IIS 7.0 для Windows Server 2008, Windows Vista SP2, or Windows 7 RC або пізніші для встановлення і запуску SDK.

Нотатки до випуску включають інструкції для налаштування цих опцій. Використання SDK не є обов'язковим, тому що є можливість користуватися будь-якими операційними системами і мовами програмування, які підтримують HTTP запити і відповіді. Однак, використання SDK інтерфейсів прикладного

програмування .NET і бібліотек для додатків і сховищ дозволяє найбільш просто працювати з HTTP безпосередньо.

Після того, як Ви встановили Azure SDK, Ви повинні завантажити та встановити інструменти Windows Azure для Visual Studio для додавання шаблонів проектів *Web Cloud Sevice*, *Worker Cloud Service*, *Web and Worker Cloud Service Workflow Service*. Ви можете завантажити поточну версію Windows Azure SDK і Windows Azure Tools для Visual Studio з головної сторінки Windows Azure за посиланням *www.microsoft.com/azure/windowsazure.mspx*.

Після встановлення Windows Azure Tools в Visual Studio з'являються шаблони *Cloud Service* в діалозі створення нового проекту (рис.11.). При виборі вузла Cloud Service відкриваються New Cloud Service, який дозволяє додати *ASP. NET Web Roles, Worker Roles or CGI Web Roles* для нового проекту. Windows Azure SDK дозволяє додати більше, ніж одну роль для кожного типу Cloud Service. Кожна роль використовує окремий екземпляр Windows Azure CPU.

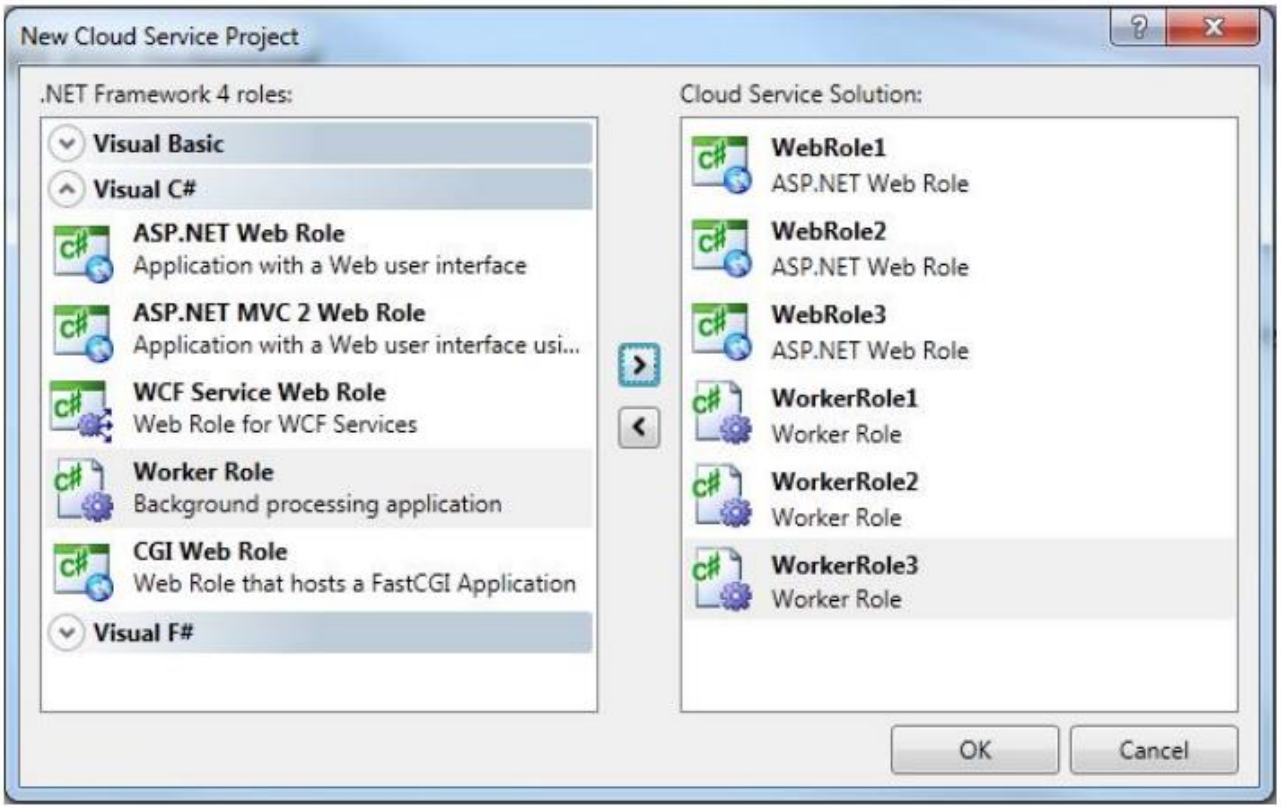


Рис.11. Створення нового проекту Cloud Service в Visual Studio

Додавши зазначені ролі і натиснувши ОК, відкриється нове рішення з проектами *WebRole і Worker-Role* в Solution Explorer як показано на рис.12.

Вузол *Roles* містить елементи *WebRole*, які вказують на кожну *WebRole*, що забезпечує користувацький інтерфейс ASP.NET для додатку і кожен *WorkerRole* для обчислювальних операцій, які не вимагають користувацького інтерфейсу, або використовують сторіноку *ASP. NET WebRole* замість цього.

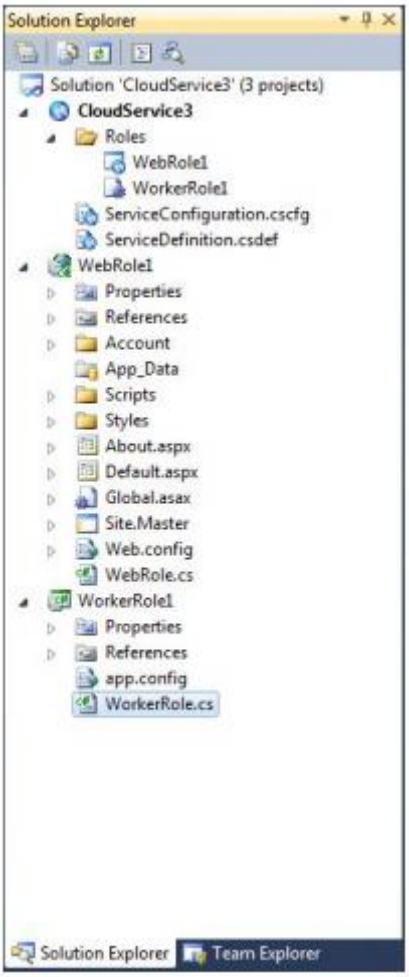


Рис.12 Solution Explorer Cloud Service

Залежно від типу Cloud Service проекти включають простір імен *Microsoft. ServiceHosting. ServiceRuntime*, який містить класи, зазначені в таблиці нижче.

Клас	Опис
RoleEntryPoint	Забезпечує методи для керування ініціалізацією, запуском і зупинкою методів сервісу і використовується для мониторингу стану сервісу
RoleException	Повідомляє про помилки, коли виникають недопустимі операції в середині ролі
RoleManager	Забезпечує методи журналювання повідомлень і надходячих застережень, витягує налаштування конфігурації сервісу та повертає місцезнаходження ресурсу
RoleStatus	Інформує про поточний статус ролі: Healthy, NonExistent, Started, Starting, Stopped, Stopping чи Unhealthy

Проекти, які використовують шаблон *WebRole* визначають веб сторінку ASP.NET Default.aspx як початкову точку для призначеного для користувача інтерфейсу додатку в хмарі.

Цей сервіс об'єднує бібліотеку класу *Common* з додатку-зразку *HelloFabric* для сприяння журналювання проблем додатку. Журнали додатку - це практичні засоби налагодження додатків, запущених в Cloud Fabric. Для

читання журналів Ви повинні скопіювати їх в Blob- масив, використовуючи інструментарій порталу. Зразок проекту *StorageClient* включає бібліотеку класу *StorageClient*, яка забезпечує в поєднанні з бібліотекою .NET Client для сервісу даних ADO.NET, інтерфейсний клас Microsoft .NET для HTTP операцій над *Azure Blob, Queue і Table Storage* сервісами. Цей проект також включає консольний додаток, який дозволяє тестувати можливості бібліотеки. Консольний додаток C # запускається в *Development Fabric* с *Development Storage*.

При встановленні Windows Azure SDK не встановлюються зразки додатків, які включені в *Program Files \ Microsoft Windows Azure SDK \ v1.0 \ samples.zip*. Встановіть зразки, розпакував samples.zip в директорію, де Ви маєте права на запис. Нижче можна знайти опис деяких зразків додатків.

Для запуску прикладу *CloudDrive* необхідний PowerShell.

Директорія, в яку було вилучено вміст архіву samples.zip, також містить наступні три пакетних файли (cmd), які можна запустити з командного рядка:

• *buildall.cmd* будує всі зразки проектів без використання Visual Studio:
• *createtables.cmd* викликає *buildall.cmd* і створює базу даних і таблиці, необхідні для зразків, які використовують Table Storage.
• *rundevstore.cmd* викликає *createtables cmd* і запускає розробку сховища, розміщуючи його в базі даних, створеної *createtables.cmd*.

До складу Development Fabric входять наступні виконувані файли: *DFAgent. exe, DFLoadBalancer. exe, DFMonitor. exe і DFService. exe*, які за замовчувнням встановлюються програмою установки Azure SDK в каталог *\ Program Files \ Windows Azure SDK \ v1.0 \ bin \ devfabric*. Після запуску Development Fabric в диспетчері завдань Ви можете побачити ці чотири процеси. Зробити це можна виконавши:

• Виберіть *Програми \ Windows Azure SDK \ Development Fabric* для запуску служби Development Fabric і його користувацького інтерфейсу *DFUI. exe*
• Правий клік мишею по значку Development Fabric в області повідомлень панелі задач і вибрати запуск служби Development Fabric (рис.13)
• Скомпілювати та запустити пдодаток Azure в Visual Studio.

Рис.13 Повідомлення, які відображаються при натисканні правою кнопкою миші по значку Development Fabric в області повідомлень панелі завдань.

Рис.14 показує користувацький інтерфейс DFUI. Коли Ви запускаєте або зупиняєте налагодження, відповідні додатки з'являються, або пропадають з користувацького інтерфейсу DFUI.

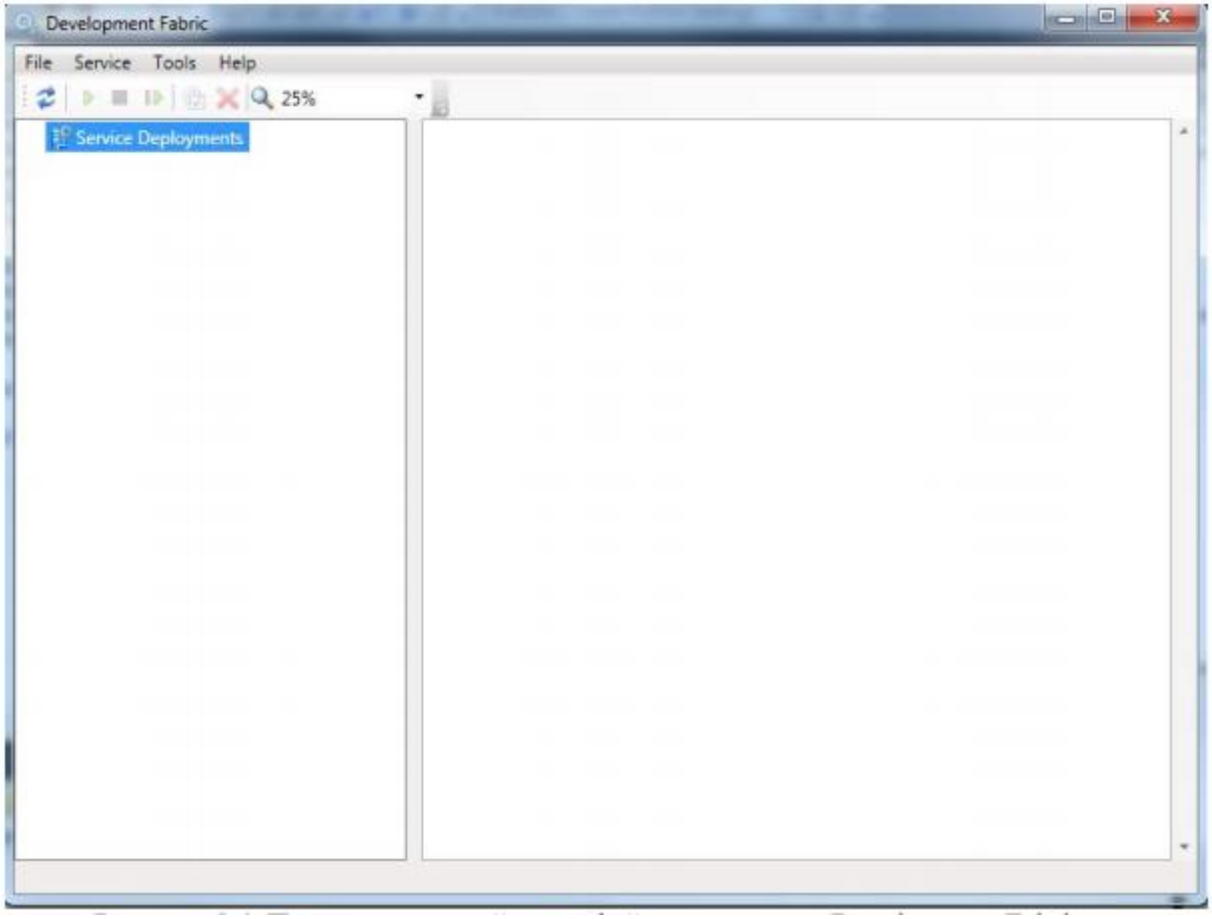


Рис.14. Користувацький інтерфейс додатку Development Fabric.

Платформа Windows Azure підтримує три типи масштабованих сховищ:
• Неструктуровані дані (blob)
• Структуровані дані (таблиці)
• Сполучення між додатками і сервісами (черги)

Запускаючи *rundevstore.exe,* або збираючи і запускаючи користувацький код Azure в Visual Studio, запускаються всі три сервіси, навіть якщо Ваш проект вимагає тільки один сервіс і відображається в користувацькому інтерфейсі Development Storage.

Для захисту від втрати даних, хмара Azure зберігає блоби, таблиці і черги в мінімум трьох роздільних контейнерах в одному центрі обробки даних. Інструмент геолокації Azure дозволяє дублювати дані в декількох центрах обробки даних Microsoft для зменшення наслідків відновлення після катастроф і для підвищення продуктивності в специфічних географічних регіонах.

Додатки Azure, які Ви запускаєте в Development Framework, можуть мати доступ до локальних даних в Development Storage, або до даних, завантажених в хмару Azure. Додаток звертається до певного порту і даних, розташованих в визначених місцях в файлі конфігурації проекту *ServiceConfiguration. cscfg*. Файл конфігурації проекту *Azure ServiceDefinition.csdef* визначає стандартні точки входу і налаштування конфігурації, які зберігаються в файлі *ServiceConfiguration.cscfg*. У роздруківці 1 показано вміст файлу *ServiceDefinition.csdef* за замовчуванням, коли Ви створюєте проект Azure,

використовуючи один з стандартних шаблонів Windows Azure Tools для Visual Studio (відзначені важливі значення).

Роздруківка 1 Вміст файлу ServiceDefinition.csdef

```
<ServiceDefinition name="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
<WebRole name="WebRole">
<InputEndpoints>
<!-- Must use port 80 for http and port 443 for https
when running in the cloud -->
<InputEndpoint name="HttpIn" protocol="http" port="80" />
</InputEndpoints>
<ConfigurationSettings>
<Setting name="AccountName"/>
<Setting name="AccountSharedKey"/>
<Setting name="BlobStorageEndpoint"/>
<Setting name="QueueStorageEndpoint"/>
<Setting name="TableStorageEndpoint"/>
</ConfigurationSettings>
</WebRole>
</ServiceDefinition>
```

Значення *InputEndpoint* використовується тількио для сховищ в хмарі.

Роздруківка 2 показує вміст файлу *ServiceConfiguration. cscfg* для веб додатку *SampleWebCloudService* з конфігурацією по замовчуванню для Development Storage (виділено):

Роздруківка 2 Вміст файлу ServiceConfiguration.cscfg

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration
">
<Role name="WebRole">
<Instances count="1"/>
<ConfigurationSettings>
<Setting name="AccountName" value="devstoreaccount1"/>
<Setting name="AccountSharedKey"
value="Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ
1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==" />
<Setting name="BlobStorageEndpoint" value="http://127.0.0.1:10000/"/>
<Setting name="QueueStorageEndpoint" value="http://127.0.0.1:10001/"/>
<Setting name="TableStorageEndpoint" value="http://127.0.0.1:10002/"/>
<!--<Setting name="AccountName" value="oakleaf"/>
<Setting name="AccountSharedKey" value="3elV1ndd . . . Coc0AMQA==" />
<Setting name="BlobStorageEndpoint"
value="http://blob.core.windows.net" />
```

<Setting name="QueueStorageEndpoint"
value="http://queue.core.windows.net" />
<Setting name="TableStorageEndpoint"
value="http://table.core.windows.net" />
-->
</ConfigurationSettings>
</Role>
</ServiceConfiguration>

Описи елементів файлу конфігурації ServiceConfiguration.csfg:

• Instances count - кількість примірників Вашого застосування, яке буде створено в хмарі, коли Ви розгорнете його.
• AccountName- ім'я, асоційоване з Вашим Hosted Service, з яким Ви створювали обліковий запис, для Development Storage це devstoreaccount1.
• AccountSharedKey шифрує кілька елементів в HTTP запиті.
• BlobStorageEndpoint- це публічний постійний Universal Resource Identifier (URI). Для Developer Storage це адреса інтерфейсу комп'ютера loopback (localhost = 127.0.0.1) з TCP портом за замовчуванням 10000.
• QueueStorageEndpoint для сховища в хмарі це публічний постійний URI. Для Developer Storage це адреса інтерфейсу комп'ютера loopback з TCP портом за замовчуванням 10001.
• TableStorageEndpoint публічний постійний Universal Resource Identifier (URI). Для Developer Storage це адреса інтерфейсу комп'ютера loopback з TCP портом за замовчуванням 10002.

Ви можете налаштувати власні номери TCP портів, якщо при використанні стандартних налаштувань виникає конфлікт з поточною конфігурацією.

В даній лекції ми отримали початкові відомості про роботу з Windows Azure SDK. Розглянули процедуру створення Cloud Service, користувацький інтерфейс Development Fabric.

Питання для самоконтролю.

1. Охарактеризуйте програмне забезпечення хмарних обчислень.
2.Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація IT-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Що являє собою Платформа корпорації Майкрософт Windows Azure?
7. Що Ви отримуєте при використанні Azure?
8. Назвіть компоненти Платформи Windows Azure.
9. Назвіть інтерфейси прикладного програмування.
10. Що таке Windows Azure Storage?
11. Охарактеризуйте комплект засобів розробки Windows Azure.
12. Що таке Windows Azure Table?
13. Що таке Windows Azure Blob?

14. Що таке Windows Azure Queue?
15. Що таке «Інфраструктура як сервіс» (Infrastructure as a Service, або IaaS)?
16. Що таке «Платформа як сервіс» (Plaatform as a Service, або PaaS)?
17. Що таке «Програмне забезпечення як сервіс» (Software as a Service , SaaS)?
18. Що таке Windows Azure SDK?
19. Назвати обчислювальні ресурси «хмар».
20. Які елементи містить вузел Roles?
21. Які типи масштабованих сховищ підтримує Платформа Windows Azure?
22. Охарактеризуйте користувацький інтерфейс Development Fabric.
23. Наведіть процедуру створення Cloud Service.
24.Перерахуйте види послуг, які надаються хмарними системами.

Література:

1.Windows Azure Platform Training Kit - January 2011 Update
2. http://microsoft.com/faculty
3. Аарон Сконнард (Aaron Skonnard) http://msdn.microsoft.com
4. http://windows.azure.com
5. http://microsoft.com/faculty
6. Практичні матеріали Azure SDK
7. http://help.live.com
8. http://www.ixbt.com/soft/msolive-1.shtml
9. http://www.ixbt.com/soft/msolive-2.shtml
10.http://www.microsoft.com/rus/newscenter/news/read/?id=post/2010/10/19/Microsoftd0bfd180d0b5d0b4d181d182d0b0d0b2d0bbd18fd0b5d182-Office-365.aspx
11. http://www.google.com/apps

7.3. *Розподілені бази даних*

Сучасні інформаційні системи мають складну багаторівневу організацію. Її основу складають інтегровані та розподілені реляційні та об'єктні бази даних, інтегровані та розподілені об'єктно-орієнтовані програмні компоненти та засоби віддаленого доступу до інформації.

В сучасних додатках проблема масштабованості перемістилася переважно в область керування даними і ми досягли того рівня, коли вибір мови програмування, операційної системи і навіть обладнання і порядку експлуатації (дякуючи хостингу віртуальних машин і "хмарам") настільки спростилися і здешевилися, що практично перестали бути проблемою і визначаються особистими уподобаннями і необхідністю. Якщо хочите створювати масштабований додаток, то Ваш вибір перш за все – **база даних**, бо саме від того, яку базу даних виберете, залежить простота збору, зберігання, обробки, аналізу і видалення даних. Визначитися з вибором бази даних часто буває складніше, ніж вибрати жанр для даних із конкретної предметної області.

Ви вчили і добре обізнані з реляційними базами. Їх дуже багато, більше сотні, вони домінуючі і будуть залишатись ними в і майбутній перспективі, але існують і інші, які все більше і більше привертають увагу. Їх теж вже немало і їх поява обумовлена пошуком альтернатив і зокрема: структур даних без схем,

нетрадиційних структур даних, простої аплікації, високої доступності, горизонтального масштабування та нових способів формулювання запитів. Всі ці технології отримали збиральну назву *NoSQL*. Так як ці питання досить обширні і не можуть бути осягнуті навіть за допомогою кількох лекцій, тому розглянемо їх в вигляді тез. Детальніше їх освойте самотужки в рамках самостійної роботи студента.

Фрагментально розглянемо сім баз даних: реляційні, ключі і значення, стовпцеві, документо-орієнтовані, графові. Оглянемо бази, покриваючи всі категорії: одну реляційну (Postgres), два сховища ключів і значень (Riak, Redis), стовпцеву (HBase), дві документо-орієнтованих (MongoDB, CouchDB) та одну графову (Neo4J).

Бази призначені для вирішення конкретних практичних задач. Реляційні з'явились, коли гнучкість запитів вважалась важливішою за гнучкість схеми. Стовпцеві добре пристосовані для зберігання великих об'ємів на декількох машинах і зв'язок між даними відходить на інший план. Майже всі задачі можна вирішити за допомогою названих СУБД і питання лише наскільки ефективне використання їх в конкретній предметній області при певних сценаріях використання і наявних ресурсах.

1.***Реляційні СУБД*** **.**

Реляційні системи управління базами даних (РСУБД) основані на теорії множин, в основі їх реалізації лежать таблиці із рядків і стовпців. Спосіб взаємодії з РСУБД – запити на мові Structured Query Language (SQL).

Дані типізовані, це можуть бути числа, рядки, дати, неструктуризовані двійкові об'єкти.

PostgreSQL – найстаріша і провірена часом СУБД. PostgreSQL – одна із кращих РСУБД с відкритим кодом.

Сильні сторони PostgreSQL Багато років досліджень і промислової експлуатації практично у всіх областях, де використовуються комп'ютери, гнучкі засоби запитів, найвищий рівень непротиріч та довговічності даних. Для Postgres написані та вивірені драйвери большості мов програмування. Можна писати власні мовні разширення, вводити нові типи індексів, створювати власні типи даних і навіть переписувати плани виконання запитів. Якщо для інших СУБД з відкритим початковим кодом існують хитрі лицензійні домовленості, то PostgreSQL – квінтэсенція відткритості. У коду взагалі нема володаря. Всякий може робити з проектом практично все що завгодно, тільки не кладучи при цьому відповідальності на авторів.

Слабкі сторони PostgreSQL

Сегментування – не найсильніша сторона реляційних баз і PostgreSQL в тім рахунку. Якщо вимоги масштабування по горизонталі, а не по вертикалі (декілька паралельних сховищ даних, а не одна суперпотужна машина чи кластер), то краще використати щось інше.

2. **Riak**

Riak –це розподілене сховище ключів і значень, в якому значенням може бути що завгодно – простий текст, документ в форматі JSON чи XML,

зображення чи відеокліп. Для доступу до сховища надається простий HTTP-інтерфейс.

Riak зможе зберегти будь-які дані. Riak також може похвалитися відмовоздатністтю. Любий сервер може бути зупинений чи запущений в любий момент, точки загальної відмови не існує. Кластер продовжує працювати при видаленні, добавлені чи аварійній відмові серверів. Відмова одного вузла – не критична ситуація. Riak не має хорошої підтримки довільних запитів, а сховище ключів і призначень, по самій своїй природі, погано з'вязуються одне з одним (іншими словами, поняття зовнішнього ключа відсутнє). Riak «розмовляє на мові веб» краще, ніж всі інші. Riak –кращий вибір для центрів обробки даних – таких, як Amazon, – які повинні обслуговувати багато запитів з низькою затримкою. В усмовах, коли кожна лишня мілісекунда очікування означае згубити потенціального клієнта, Riak найкраще. Вона проста в налаштуванню та адмініструванні і може рости разом з вимогами.

Riak – перша із розглянутих нами баз даних NoSQL. Це розподілене репліцуємe сховище ключів і значень з доповненими функціями, не маюче точки загальної відмови.

Для працюючого тільки з реляційними базами данх, Riak може здатися дивним створінням. В нім нема ні транзакцій, ні SQL, ні схеми. Є ключі, але з'вязування сегментів зовсім не схоже на з'єднання таблиць. А технологія mapreduce взагалі покажеться чимось малозрозумілим.
Однак для вирішення певного класу задач всі ці компроміси виправдані. Здатність Riak масштабуватися шляхом збільшення кількості серверів (а не нарощування потужності одного сервера) та простота використання –кращий приклад спроби вирішити специфічні проблеми масштабування в веб. Riak опирається на уже існуючу структуру HTTP, даючи максимальну гнучкість для побудови каркасів чи систем з підтримкою веб.

Сильні сторони Riak

При проектуванні широкомасштабної системи замовлень нашталкт Amazon, або коли ставиться на перше місце висока доступність, то Riak, безумовно, заслуовує на увагу. Один із беззаперечних плюсіів Riak – увага до усунення точок загальної відмови з метою забезпечити безперебійну роботу та нарощування чи скрочення кластеру у відповідності зі змінюючимися вимогами. Якщо структура даних не дуже складна, то працювати з Riak буде просто. Але при цьому зберігається можливість маніпуляцій з даними, якщо це необхідно. На даний час підтримуються десяток мов (повний перелік можна знайти на сайті Riak), але, при готовності писати на Erlang, можна разширювати ядро в будь-яку сторону. А якщо потрібна більша швидкодія, ніж здатен забезпечити протокол HTTP, то можете спробовати обмінюватися даними по більш ефективному двійковому транспортному протоколу Protobuf19.

Слабкі сторони Riak

Якщо потрібні прості засоби формулювання запитів, складні структури даних або жорстка схема чи нема потреби в горизонтальному масштабуванню, то Riak, мабуть, не потрібний.

Один з основних недоліків Riak – відсутність простих і надійних засобів формулювання довільних запитів. Технология mapreduce дає фантастичну функциональність, але хотілось би бачити більше вбудованих дій, основаних на URL чи PUT-запитах. Добавлення індексування було вагомим кроком в правильному напрямку, але хотілось би, щоб ці ідеї отимали подальший розвиток. І накінець, при бажанні писати на Erlang, можна зіштовхнутися з деякими обмеженнями JavaScript – неможливості написати функції, визиваємі в точці підключення після фіксації, – і недостатньою швидкодією mapreduce. Однако команда Riak працює і над цими питаннями.

Riak и теорема CAP

(CAP - Consistency, Availability, partition Tolerance (узгодженість, доступність, стійкість до втрати зв'язаності). Слід зазначити , що слово *consistency* в абревіатурах ACID і CAP означає різні речі). Теорема CAP розкриває неприглядну правду про те, як розподілені системи ведуть себе в умовах нестабільної мережі. CAP стверджує, що можна створити розподілену систему, яка буде узгодженою (consistent) (тобто операції записі атомарні і всі наступні операції читання бачуть нові значення), доступною (available) (база даних повертає значення поки працює хоч один сервер) і сітйка до втрати зв'язності (partition tolerant) (система продовжує функціонувати, навіть якщо зв'язок між серверами тимчасово відсутній), але одночасно можна гарантувати тільки дві із цих трьох властивостей. Іншими словами, можна створити розподілену систему, яка буде узгодженою і стійкою до втрати зв'язаності, або систему, яка буде узгодженою і стійкою (але при цьому не буде стійкою до втрати зв'язаності, тобо, по суті, не буде розподіленою). Але не можливо створити розподілену базу даних, яка була би однчасно узгодженою, доступною і стійкою до втрати зв'язаності.

Теорема CAP суттєва при проектуванні розподіленої бази даних, так як Ви повинні вирішити, чем готові пожертвувати.

Яку би базу даних Ви не вибрали, вона зможе гарантувати або доступність, або узгодженість. Стійкість до втрати зв'язаності – чисто архітектурне рішення (від неї залежить, чи буде система взагалі розподіленою). Важливо зрозуміти сенс теореми CAP, щоб реалістично оцінювати можливості. Компроміси, прийняті в різних описаних базах даних, спираються саме на цю теорему.

Riak гарно обходить обмеження , які накладає теорема CAP на всяку разподілену базу даних. І це дивовижно при порівнянні з PostgreSQL, яка здебільшого підтримує тільки строгу погодженість записі. В Riak використовується ідея про те, що умови CAP можна задавати на рівні сегментів чи запитів. Це значний крок в бік надійної і гнучкої СУБД з відкритим початковим кодом. Знайомлячись з іншими базаи даних, згадуйте про Riak – вас постійно буде дивувати його гнучкість. При необхідності

зберігати гігантський каталог даних Riak може здатися непоганим вибором.

Якщо необхідна база даних, яка вбудовується в пристрій, чи потрібно оброблювати фінансові транзакції, то Riak не годиться. Якщо стоїть задача забезпечити горизонтальне масштабування чи обслуговувати інтенсивний потік запитів через веб, то до Riak слід придивитися.

2 HBase

Система Apache HBase створена для серйозних робіт. Якщо об'ем даних не вимірюється багатьма гігабайтами, то варто взяти щось простіше.
На перший погляд, HBase дуже схожа на реляційну базу даних – настільки, що, не знаючи, з чим маєте справу, можна сплутати одну з іншою. Сама складна частина в вивченні HBase – не технологія; проблема в тім, що багато термінів, використовуваних в HBase, оманливо знайомі. Наприклад, HBase зберігає дані в контейнерах, які називаються *таблицями*. Таблиці же складаються із *комірок*, які знаходяться на перетині *рядків* і *столпців*. Таблиці HBase ведуть себе зовсім не так, як відношення, рядки зовсім не схожі на записі, а склад стовпців може бути змінним (схема його не контролює).

В HBase вбудований ряд функцій, відсутній в інших базах даних, напрклад, версіонування, стиснення, прибирання сміття (для даних з вичерпаним терміном зберігання) і таблиці в памяті. Так як ці можливості мають початкову присутність, значить, прийдеться писати меньше кодів, коли вони знадобляться. Крім того, HBase дає строгі гарантії непротирічності, що спрощує перехід від реляційних баз даних. При обробці трудомістких запитів HBase часто обжене інші СУБД. Цим пояснюється, чому HBase так часто використовується в великих компаніях для реалізації систем аналізу журналів та пошуку.

HBase – це *стовпчикова* база даних , яка може похвалитися підтримкою непротирічності та горизонтальною масштабованністю. Вона влаштована по зразку BigTable , високопродуктивної закритої бази даних, яка розроблена Google та описана в статті «Bigtable: A Distributed Storage System for Structured Data» (http://research.google.com/archive/bigtable.html) .

HBase – це сімбіоз простоти і складності. Сама модель зберігання даних відносно проста, в схему встроюються лише деякі обмеження. Однако при вивченні системи значно мішає, що термінологія запозичена із реляційних баз даних (наприклад, це відноситься до слів *таблиця та стовчик*). При проектуванні схеми HBase наріжним каменем є спроможність таблиць і стовбців.

Сильні сторони HBase

Серед особливостей HBase уваги заслуговує архітектура горизонтальної масштабованості і вбудовані засоби версіононування та стиснення. Для деяких задач вбудоване версіонування – надзвичайно корисна функція. Наприклад, зберігання історії версій сторінок вікі-сайта необхідно для виробітки політик і обслуговування. HBase позволяє не запобігати ніяких спеціальних мір для

реализації історії сторінок – ми отримуємо це безкоштовно.

HBase задумана для масштабування по горизонталі.Якщо об'ем даних вимірюється гігабайтами чи терабайтами, то HBase – те,що потрібно. В HBase є механізми обліку конкретних серверних стоєк; реплікація даних між вузлами, встановленими в одній чи в різних стойках ЦОД, забезпечує швидке відпрацювання відмов без помітного зниження функціональності.

Слабкі сторони HBase

Хотя HBase проектувалась з урахуванням горизонтальної масштабованості, існує нижній поріг кількості вузлів. Для надійної работи необхідно не менше п'яти вузлів. Поскільки система призначена для обработки великих массивів даних, адмініструвати її досить важко. Для вирішення невеликих задач HBase навряд чи годиться, а документації для неспеціалістів практично не існує, що ускладнює її вивчення. Крім того, HBase майже ніколи не розгортається в ізоляції. Вона частина екосистеми масштабіруємих компонентів, до числа яких належить Hadoop (незалежна реаізація каркасу MapReduce, вперше запропонованого Google), розподілена файлова система Hadoop (HDFS). Один із найважливіших вкладів Google в інформатику – популяризація алгоритмичного каркасу mapreduce для параллельного виконання завдань на декількох вузлах. Він описаний в основоположній статті Google і став корисним інструментом для виконання запитів в цілому класі збереження даних, стійких до втрати зв'язаності (http://research.google.com/archive/mapreduce.html).

Zookeeper (служба баз головного серверу, координуюча роботу вузлів). У цієї системи є як плюси, так і мінуси; вона забезпечує високу стабільність, але покладає на адміністратора груз відповідальності за обслуговування.

Слід відмітити, що HBase не дає ніяких засобів сортування чи індексування, крім ключів рядків. Рядки зберігаються відтсортованими по ключах, але ні по якому іншому полю: по імені і значенню стовпця сортування не проводиться. Тому для пошуку рядка по любому критерію, кром її ключа, прийдеться сканувати всю таблицю, або будувати і супроводжувати власний індекс. Відсутнє також поняття типу даних. Значення всіх полів в HBase трактуются як неінтерпретуємі масиви байтів. Нема ніякого розрізнення між цілим числом, рядком та датою. Для HBase все це просто байти, а їх інтерпретація покладається на додаток. .

HBase і теорема CAP

В термінології CAP HBase, безумовно, відноситься до класу CP. HBase дає строгі гарантії погоджувальності. Якщо один клієнт успішно записав якесь значення, то всі інші клієнти побачуть його при наступному запиті. Деякі бази даних, наприклад Riak, дозволяють змінити параметры CAP на рівні окремої операції. Але тільки не HBase. При не дуже серйозній втраті зв'язаності, наприклад, при виході з ладу одного вузла, HBase залишиться доступною – відповідальність за обслуговування запитів буде передана іншим вузлам. Але

в патологічній ситуації, наприклад, коли працездатність зберіг лише один вузол, у HBase не буде іншого вибору, як відмовити в обслуговуванні запиту.

Обговорення властивостей CAP дещо ускладниться, якщо взяти до уваги міжкластерну реплікацію. В типовій багатокластерній конфігурації кластери можуть бути територіально разнесені. В такому випадку для любого заданого сімейства стовпців запись проводиться тільки на один кластер, а інші просто надають доступ до реплікованих даних. Така система являється погодженою в кінечному рахунку , поскільки репліковані кластери повертають останнє значення, про яке знають, а воно може і не співпадати з останнім значенням, записаним на головному кластері.
Термінологія може здатися обманливо знайомою, а установка і налаштування – заняття не для слабких. З іншого боку, деякі можливості HBase, наприклад, версіонування і стискання, просто унікальні і можуть зробити HBase дуже приваблевим вибором для рішення деяких задач. Крім того, не можна не відмітити можливість горизонтального масштабування на вузли зі стандартним обладнанням. Взагалі, HBase серйозний інструмент, який вимагає відповідного поводження.

4. MongoDB

Перша публічна версія MongoDB була випущена в 2009 році.
Система задумувалась як масштабіруєма база даних – назва Mongo походить від слова «humongous», отриманим об'єднанням «huge» (гігантський) і «monstrous», а в якості основних проектних цілей були поставлені висока працездатність і простота доступу до данх. Це документна база даних, яка забезпечує не тільки збереження, але і опитування вкладених даних, пред'являя довільні запити. Схема бази даних не нав'язується (в цім MongoDB схожа на Riak, але відрізняється від Postgres), тому один документ може мати поля чи типи, відсутні у всіх інших документах колекції. Mongo – сховище JSON-документів (хотя, строго кажучи, дані зберігаються в двійковому варіанті JSON, який називається BSON). Документ Mongo можно уподобити рядку реляційної таблиці без схеми, в якій допускається довільна глибина вкладеності значень.

Сильні сторони Mongo

Головна перевага Mongo – здатність оброблювати гігантські масиви даних (і величезній потік запитів) за рахунок реплікації та горизонтального масштабування. Але вона також пропонує гнучку модель даних, так як не потрібно визначати схему, а дані можуть бути вкладеними (для досягнення аналогічного эффекту в РСУБД прийшлось би використовувати з'єднання). Накінець, при проектуванні MongoDB закладувалась простота використання. Існує схожість між командами Mongo та деякими концепціями баз даних на основі SQL. Це не випадково, і саме по цій причині Mongo має багато прибічників із кола бувших користувачів объектно-реляційних моделей (ORM).

Слабкі сторони Mongo

Те, що Mongo заохочує денормалізацію схеми (хоча самі схеми і відсутні), деяким може здатися неприйнятним. Є розробки, в яких жорсткі обмеження, які накладаються реляційною СУБД з її холодною логікою, вселяють впевненість. Можливість вставки в будь-яку колекцію значення довільного типу викликає побоювання. Єдина опечатка може стати причиною довготривалих мук, якщо Ви не здогадаєтеся шукати причину в іменах полів і колекцій. Гнучкість Mongo - не така вже велика перевага, якщо модель даних вже досить зріла і давно зафіксована. Оскільки Mongo орієнтована на великі набори даних, то використати її має сенс у великих кластерах, для проектування і обслуговування яких потрібно деякі зусилля.Навідміну від Riak, де додавання нових вузлів робиться прозоро і майже непомітно в процесі експлуатації, налаштування кластера Mongo має на увазі попереднє планування. Mongo - відмінний вибір для тих, хто вже звично використовує для зберігання даних реляційну базу даних і звертається до неї за допомогою якої-небудь системи об'єктно-реляційного відображення. ЇЇ часто рекомендують працюючим на платформах Rails, Django і взагалі усім, хто користується патерном модель-представлення-контролер (MVC), оскільки вони все одно реалізують перевірку даних і управління полями за допомогою моделей на рівні додатка і оскільки з міграцією схеми можна буде розпрощатися (здебільшого). Для додавання нових полів в документ досить просто додати поле в модель даних, і Mongo спокійно сприйме нові поняття. В порівнянні з реляційними базами даних Mongo дає набагато природніше рішення багатьох типових завдань, в яких набір даних визначається додатком.

5.CouchDB

CouchDB - типова документо-орієнтована база даних на основі JSON і REST. Вже перша версія, випущена в 2005 році, була спроектована з розрахунку на мережу веб і усі незліченні помилки, відмови і "глюки", які її супроводять. Тому по стабільності з CouchDB не може порівнятися майже жодна з існуючих баз даних. Якщо інші системи здатні пережити випадкові пропажі мережі, то CouchDB прекрасно себе почуває навіть тоді, коли поява мережі - рідкісна подія. Як і MongoDB, CouchDB зберігає документи - JSON-об'єкти, що складаються з пар ключ-значення, причому значеннями можуть бути дані різних типів, у рахунку і інші об'єкти з необмеженою вкладеністю. Проте довільні запити не підтримуються; основний спосіб пошуку документів - це індексовані представления, що породжуються інкрементною процедурою mapreduce. Ця база даних може масштабуватися вгору і вниз, а тому легко адаптується до завдань будь-якого розміру і складності. Вона розрахована не лише на велетенські кластери з дорогих серверів, але і на інші сценарії розгортання : від центру обробки даних до смартфону. Запустити CouchDB можна на телефоні Android, на персональному MacBook або в ЦОДі. CouchDB - документо-орієнтована база даних, в якій форматом зберігання і взаємодії із зовнішнім світом є JSON. Як і в Riak, усі звернення до CouchDB робляться за допомогою REST- інтерфейсу. Реплікація може бути як одно-, так і двосторонньою, за запитом або безперервною. CouchDB забезпечує високу гнучкість при ухваленні рішень про структуру, захист і розподіл даних.

Сильні сторони CouchDB

CouchDB - надійний і стабільний представник сімейства баз даних категорії NoSQL. Припускаючи, що мережі принципово ненадійні, а апаратні збої неминучі, CouchDB пропонує півностттю децентралізований підхід до організації сховищ даних. Досить мала, щоб уміщатися в смартфоні, і досить велика для підтримки корпоративних рішень, CouchDB може бути розгорнута на самих різних платформах. CouchDB - в рівній мірі база даних і API. (Changes API – інтерфейс, позволяючий відслідковувати зміни в базі даних і миттєво отримувати нові значення). Крім канонічного проекту Apache CouchDB існують альтернативні реалізації і постачальники служб CouchDB, побудованих на базі гібридних серверних рішень, причому їх кількість постійно росте. Оскільки CouchDB з'явилася "з веб і для веб", то вона досить просто вбудовується як окремий шар у веб- технології - подібно балансувальникам навантаження і розподіленим кешам, - але при цьому зберігає унікальність своїх API.

Слабкі сторони CouchDB

Зрозуміло, CouchDB годиться не для будь-якого завдання. Основані на технології mapreduce представлення, при усій своїй новизні, не можуть забезпечити такі ж гнучкі засоби вибірки даних, як реляційні СУБД. Насправді, у виробничій системі взагалі не рекомендується прибігати до довільних запитів. Крім того, прийнята в CouchDB стратегія реплікації не завжди являється відповідним вибором. У CouchDB в основу реплікації покладений принцип "все або нічого", тобто на усіх реплікуємих серверах зберігаються одні і ті ж дані. Не існує механізму сегментації, що дозволяє розподілити дані по різних серверах в ЦОД. Головна причина додавання нових вузлів в CouchDB - не стільки розподілити дані, скільки збільшити продуктивність операцій зчитування і запису. Прагнення CouchDB забезпечити надійність в умовах неоднозначності робить її відповідною системою для протистояння жорсткій реальності дикого світу Інтернету. Спираючись на стандартні веб-технології, зокрема HTTP/REST і JSON, CouchDB відмінно вбудовується у будь-яке рішення, де такі технології активно використовуються, тобто - з урахуванням сучасних тенденцій - скрізь. Але і у відгородженому від зовнішнього світу садку ЦОД CouchDB теж знайде застосування, якщо Ви готові вирішувати виникаючі конфлікти, або не проти альтернативної реалізації типу BigCouch, але не чекаєте знайти готовий механізм сегментації. У CouchDB є немало інших особливостей, що роблять її унікальною базою даних, що заслуговує на увагу. Не можливо розглянути їх всі, але хоч би перерахуємо деякі: простота резервного копіювання, двійкові вкладення в документи і CouchApps - система для розробки і розгортання веб-додатку прямо через CouchDB без додаткового ПЗ проміжного шару.

6. Neo4J

Neo4j - новий тип сховищ даних NoSQL, що отримав назву графова база даних. Як випливає з назви, дані в ній зберігаються у вигляді графу (у тому сенсі, в якому він розглядається в математиці). Відмітною особливістю таких

баз даних є можливість малювати їх структуру на дошці у вигляді прямокутних блоків і ліній, що сполучають їх. Все, що можна зображувати у такому вигляді, можна зберегти в Neo4j. У Neo4j упор робиться швидше на зв'язку між значеннями, ніж на загальні характеристики значень (як в колекціях документів або в рядках таблиці). Таким чином, зовсім різнорідні дані можна зберігати просто і природньо. Розмір Neo4j невеликий - настільки, що її можна впровадити практично у будь-яке застосування. З іншого боку, в Neo4j можна зберігати десятки мільярдів вузлів і стільки ж ребер. А враховуючи підтримку реплікації на багато серверів, ця СУБД здатна впоратися із завданням будь-якого розміру. Neo4j - краща реалізація графської бази даних (досить рідкісний клас) з відкритим початковим кодом. У графових базах даних упор робиться на зв'язках між даними, а не на спільності значень. Моделювати дані у вигляді графів просто. Треба лише створити вузли і зв'язки між ними і за бажання асоціювати з ними пари ключ-значення. Запити зводяться до опису обходу графа, вирушаючи з початкового вузла.

Сильні сторони Neo4j

Neo4j - один з кращих зразків графових баз даних з відкритим початковим кодом. Такі бази прекрасно підходять для зберігання неструктурованих даних - іноді навіть краще, ніж документні сховища. Мало того що в Neo4j немає ні типів, ні схеми, вона ще і не накладає ніяких обмежень на взаємозв'язки між даними. Це "звалище" в хорошому значенні слова. Поточна версія Neo4j підтримує 34,4 мільярда вузлів і стільки ж зв'язків - більш ніж достатньо для більшості завдань (у одному графі Neo4j можна було б зберігати більше 42 вузлів для кожного з 800 мільйонів користувачів Facebook). У дистрибутив Neo4j входять інструменти для прискорення пошуку (Lucene) і прості у використанні (хоча, можливо, не цілком звичні) мовні розширення, зокрема Gremlin і REST- интерфейс. Але Neo4j не лише проста у використанні, але і працює швидко. На відміну від операцій з'єднання в реляційних базах даних, або операцій mapreduce в інших базах, обхід графа вимагає постійного часу. Пов'язані дані знаходяться всього в одному кроці - не треба робити масивне з'єднання з подальшою фільтрацією результатів, як у більшості розглянутих вище СУБД. Яким би великим не був граф, перехід з вузла А у вузол В вимагає усього одного кроку, якщо між цими вузлами є зв'язок. Нарешті, у видання Enterprise включені засоби для створення високодоступних сайтів з великою кількістю запитів на читання - шляхом створення HA-кластеру Neo4j.

Слабкі сторони Neo4

Neo4j не позбавлена недоліків. Вибрана термінологія (вузол замість вершини і зв'язок замість ребра) тільки утрудняє взаєморозуміння. HA-кластер прекрасно справляється з реплікацією, але реплікувати дозволено тільки граф цілком. Неможливо сегментувати граф, що накладає обмеження на його розмір. Нарешті, якщо ви шукаєте продукт з відкритим початковим кодом і ліцензією, відповідною для корпоративного використання (наприклад, MIT), то Neo4j, мабуть, не для вас. Видання Community поставляється за ліцензією GPL, але якщо вам потрібні засоби, що входять тільки у видання Enterprise

(високодоступний кластер і резервне копіювання), для побудови виробничої системи, то, ймовірно, за ліцензію доведеться заплатити.

Neo4j і теорема CAP

Для тих, хто збирається будувати розподілену систему, стратегія Neo4j має бути зрозуміла з фрази "високодоступний кластер". Neo4j має властивості доступності і стійкості до втрати зв'язаності. Кожен підпорядкований сервер повертає тільки ті дані, які має в розпорядженні в даний момент, і вони впродовж деякого періоду часу можуть відрізнятися від тих, що зберігаються в головному вузлі. Затримку оновлення можна скоротити, зменшивши інтервал між опитуваннями на підпорядкованому сервері, але технічно система все одно забезпечує тільки узгодженість. Тому HA-кластер Neo4j рекомендується використати тільки в додатках, орієнтованих здебільшого для читання. Простота Neo4j може бентежити, якщо Ви не звикли моделювати дані у вигляді графів. Незважаючи на потужний API з відкритим початковим кодом і роки промислової експлуатації, у цієї СУБД все ще відносно мало користувачів. Пояснюється це тільки недостатньою інформованністю, тому що графові бази даних природньо відбивають процес інтерпретації даних людиною. Ми представляємо сім'ї у вигляді дерев, друзів - у вигляді графів; мало хто розглядає зв'язки між людьми як самовідносні типи даних. Для деяких класів завдань, наприклад, соціальних мереж, вибір Neo4j напрошується сам собою. Але і в не таких очевидних випадках має сенс придивитися до цієї СУБД уважніше. .

7. Redis

Redis (REmote DIctionary Service - віддалена служба словників) - просте в роботі сховище ключів і значень з розвиненим набором команд. Перша версія системи вийшла в 2009 році. І в швидкодії у неї практично немає суперників. Читання робиться швидко, а запис ще швидший - на деяких еталонних тестах продемонстрировано до 100 000 операцій SET в секунду. Творець Redis Сальваторе Санфилиппо (Salvatore Sanfilippo) називає свій проект "Сервером структур даних", щоб підкреслити закладені в нього можливості роботи із складними типами даних і інші особливості. Вивченням цього надшвидкого "не просто сховища ключів і значень" ми і завершимо огляд сучасного ландшафту баз даних. Redis подтримує складні структури даних, хоча і не до такої міри, як документо-орієнтовані бази даних. Вона підтримує запити, що повертають множини, але не на такому рівні детальності, як реляційні СУБД, і без підтримки типів. І, зрозуміло, вона швидка, хоча для досягнення цієї мети приносить довговічність даних в жертву швидкодії. Окрім серверу структур даних, Redis є ще і блокуючою чергою (чи стеком), а також системою публікації-підписки. У ній можна настроювати політики терміну зберігання, рівні довговічності і параметри реплікації. Усе це робить Redis швидше комплектом інструментальних засобів, в який входять корисні алгоритми роботи із структурами даних і процеси, ніж базою даних якогось певного жанру. Великий список клієнтських бібліотек для Redis дозволяє використати її

у багатьох мовах програмування. Робота з нею не лише проста, але і приносить задоволення.

Резюме

Redis - компактне сховище ключів і значень (чи структур даних), споживаюче мало ресурсів. Воно схоже на багатофункціональні інструменти. Як і вони, Redis придатна для вирішення самих різних, часто несподіваних завдань. До усього іншого, сховище Redis працює швидко, просте в експлуатації і забезпечує таку довговічність даних, яку Ви налаштуєте. Redis рідко використовується як автономна база даних, частіше вона входить до складу багатосторонньої екосистеми, де застосовується для трансформації даних, кешування запитів, або управління чергами повідомлень (завдяки вбудованим блокуючим командам). .

Сильні сторони Redis

Очевидна цінність Redis - швидкодія, чим можуть похвалитися багато подібних сховищ ключів і значень. Але, на відміну від більшості таких сховищ, Redis ще уміє зберігати складені значення - списки, хеши і множини - і застосовувати до них спеціальні операції. Мало того, в Redis ще можна тонко настроювати довговічність даних, віддаючи перевагу або надійності зберігання, або швидкості. Вбудована реплікація типу головний-підлеглий - ще один зручний спосіб підвищити довговічність, не приносячи швидкодію в жертву повільного дозапису у файл при кожній операції До того ж, реплікація підвищує продуктивність систем, в яких основне навантаження доводиться на операції зчитування.

Слабкі сторони Redis

Своєю швидкодією Redis зобов'язана, передусім, тому, що зберігає усі дані в пам'яті. Хтось назве це обманом, тому що база даних, яка взагалі не звертається до диску, природньо, працюватиме швидко. Будь-якому сховищу в оперативній пам'яті властива проблема довговічності даних; якщо зупинити базу, не скинувши дані на диск, то дані будуть втрачені. Навіть якщо встановити синхронний режим дозапису у файл при кожній операції, все одно при відтворенні файлу відновлення даних з простроченим терміном зберігання не цілком надійно - це характерно для будь-якого механізму відтворення подій, прив'язаних до часу. Втім, ця проблема швидше теоретична, ніж практична. Redis також не підтримує набори даних, розмір яких перевищує об'єм доступної оперативної пам'яті (підтримка віртуальної пам'яті з Redis буде виключена). Розроблявся кластер Redis, який дозволить вийти за межі обмеження на об'єм ОЗУ в одному комп'ютері, але назараз я не маю достовірної інформації про результат. Доки користувачі, що бажають організувати кластер, повинні самі написати відповідний клієнт (як драйвер на Ruby).

Redis не відчуває нестачі в командах - всього їх більше 120. Назви більшості команд пояснюють їх призначення, треба лише звикнути до думки, що деякі надмірні букви опущені (як, наприклад, в команді INCRBY). Redis вже став невід'ємною частиною багатьох систем. На його основі створені декілька

проектів з відкритим початковим кодом - від Resque (написана на Ruby служба асинхронних черг завдань) до SocketStream (система управління сеансами для Node.js).

Висновки

За способом зберігання бази даних можна приблизно розбити на п'ять жанрів: реляційні, сховища ключів і значень, стовпцеві, документні і графські. Чим вони відрізняються один від одного і для яких завдань підходять, або не підходять - коли їх має сенс **використати**, а коли треба триматися **чим далі**:

Реляційні бази даних

Це класичний і найпоширеніший варіант. Реляційні системи управління базами даних (РСУБД) грунтовані на теоріях множин, дані в них зберігаються у вигляді двовимірних таблиць, які складаються з рядків і стовпців. Реляційні бази строго контролюють типи даних і зазвичай допускають зберігання чисел, рядків, дат і двійкових об'єктів (BLOB'ов), що не інтерпретуються, хоча, PostgreSQL підтримує такі розширення, як масив і куб.

Хороші для:

Завдяки структурованій природі даних реляційні бази має сенс використати, коли структура даних заздалегідь відома, а способи їх можливого використання - ні. Іншими словами, Ви готові авансом заплатити за складність організації, сподіваючись, що в майбутньому це окупиться гнучкістю запитів. Багато завдань бізнесу зручно моделюються саме так: замовлення, постачання, облік складських запасів, кошики закупівель і інші. Заздалегідь невідомо, як згодом потрібно буде витягати дані (наприклад, скільки замовлень ми обробили в цьому місяці?), але дані за своєю природою регулярні і контролювати цю регулярність корисно.

Не дуже добрі для:

Якщо структура даних змінна або характеризується глибокою вкладеністю, то реляційна СУБД - не кращий помічник. У ній необхідно заздалегідь визначити схему, що дуже важко зробити, якщо усі записи структурно відрізняються один від одного. Подумайте, як би Ви стали розробляти базу даних для опису усіх живих істот в природі. Спроба створити повний список ознак ("має волосся", "кількість ніг", "відкладає яйця" і т. д.) приречена на провал. У такому разі потрібна база даних, яка не накладає таких строгих обмежень на дані, які зберігаються.

Сховища ключів і значень

Сховище ключів і значень - проста з усіх розглянутих моделей. У нім прості ключі відображаються на, можливо, складніші значення, як у велетенській хеш-таблиці. Завдяки відносній простоті бази даних цього жанру забезпечують максимальну гнучкість реалізації. Пошук в хеш-таблиці

здійснюється швидко, тому у разі Redis швидкодія була поставлена основною задачею. Крім того, пошук в хеш-таблицях легко розподіляється на декілька машин, і в Riak цей факт використаний для побудови простих в управлінні кластерів. Зрозуміло, простота може виявитися мінусом, якщо до моделювання даних пред'являються складні вимоги.

Хороші для:

Оскільки необхідність підтримувати індекси відсутня або невелика, то при проектуванні сховищ ключів і значень часто закладається горизонтальна масштабованість, дуже висока швидкодія, або те і інше разом. Особливо хороші вони для задач, де між даними немає великої кількості зв'язків. Наприклад, у веб-застосуванні цьому критерію відповідають сеанси користувачів; дії одного користувача в сеансі практично ніяк не пов'язані з діями інших користувачів.

Не дуже добрі для:

Через відсутність індексів і засобів сканування КЗ-сховища мало чим допоможуть, якщо вимагається пред'являти запити, відмінні від простих операцій CRUD (створення, читання, оновлення, видалення).

Стовпцеві бази даних

Стовпцеві бази даних (також бази даних з сімействами стовпців) мають багато загальних рис з КЗ-сховищами і РСУБД. Як і в КЗ-сховищах, значення просяться по ключу. Як і в реляційних базах, значення групуються в нуль або більше за стовпці, хоча в кожному рядку може бути заповнене довільне число стовпців. Але на відміну від того і іншого, в стовпцевих базах дані зберігаються по стовпцях, а не по рядках. Додавання нових стовпців обходиться дешево, версіонування реалізується тривіально, і на зберігання відсутніх значень місце не витрачається. Розглянута нами база даних HBase - класичний приклад цього жанру.

Хороші для:

Столвцеві бази даних традиційно розроблялися з метою забезпечити горизонтальну масштабованість. Тому вони особливо хороші для завдань, пов'язаних з "великими даними", коли база розміщується в кластері з десятків, сотень, а то і тисяч вузлів. Крім того, в них зазвичай вбудовується підтримка стискування і версіонування. Канонічний приклад завдання, орієнтованого на зберігання даних по стовпцях, - індексування веб-сторінок. Сторінки у веб містять багато тексту (і тут дуже корисне стискування), в якійсь мірі взаємозв'язані і змінюються з часом (тут на допомогу приходить версіонування).

Не дуже добрі для:

У різних стовпцевих баз даних різні можливості і, відповідно , різні недоліки. Але усім їм властива одна загальна риса: краще проектувати схему з урахуванням майбутніх запитів. Це означає, що треба апріорі уявляти собі, як

використовуватимуться дані, а не тільки, що вони містять. Якщо способи використання даних заздалегідь невідомі - наприклад, потрібні довільні звіти, - то стовпцева база даних - не оптимальний варіант.

Документні бази данных

У документній базі даних об'єкт, що зберігається, може містити довільний набір полів і навіть допускає вкладеність будь-якого рівня. Зазвичай такі об'єкти представляються в нотації JavaScript Object Notation (JSON), прийнятій як в MongoDB, так і в CouchDB, хоча ця вимога аж ніяк не є концептуально значимою. Оскільки документи не пов'язані один з одним так тісно, як в реляційних базах даних, їх порівняно просто сегментувати і реплікувати на декілька серверів, тому часто зустрічаються розподілені реалізації. Система MongoHQ ставить метою підвищити доступність за рахунок створення ЦОД, здатних підтримувати гігантські набори даних масштабу веб. З іншого боку, CouchDB орієнтована на простоту і довговічність даних, а доступність досягається за рахунок реплікації типу головний-головний на вузли з високим рівнем автономності. Ці проекти значною мірою перекриваються.

Хороші для:

Документні бази даних хороші для завдань, де предметна область характеризується високою мірою мінливості. Якщо зарані невідомо, як виглядають дані, то документна база - якраз те, що треба. Крім того, документи по самій своїй природі часто добре лягають на об'єктно-орієнтовані моделі програмування. А, значить, зменшується розузгодження інтерфейсів при переході від моделі бази даних до об'єктної моделі додатку.

Не дуже добрі для:

Якщо ви звикли до складних запитів із з'єднаннями в реляційній базі даних з добре нормалізованою схемою, то в документній базі цих можливостей бракуватиме. Документ зазвичай містить усю або велику частину інформації про об'єкт. Якщо в реляційній базі природнє прагнення нормалізувати дані з метою усунення дублювання, яке може привести до розузгодження, то в документних базах денормалізоване зберігання – це норма.

Графові бази даних

Графові бази даних - новий, швидкорозвиваємий клас систем, в яких упор робиться швидше на встановлення довільних зв'язків між даними, ніж на фактичні значення. Прикладом може служити проект з відкритим початковим кодом Neo4j, який все частіше використовується у багатьох соціальних мережах. На відміну від інших стилів баз даних, де колекції схожих об'єктів групуються в загальних контейнерах, в графових базах дані зберігаються у вільнішій формі; обробка запиту зводиться до дотримання по ребрах, що сполучають дві вершини, інакше кажучи, до обходу вузлів. У міру використання у все більшому числі проектів графові бази даних починають застосовуватися не лише для побудови простих соціальних застосувань, але і

для більш специфічних цілей, наприклад: системи рекомендації, списки управління доступом і географічні дані.

Хороші для:

Графовіі бази даних неначе спеціально придумані для додатків, які характеризуються наявністю мережі даних. Типовий приклад - соціальна мережа, де вузли представляють користувачів, між якими існують різні зв'язки. Моделювання таких даних за допомогою інших засобів частенько є складною проблемою, але графові бази даних справляються з цим завданням на ура. До того ж, вони прекрасно лягають на об'єктно-орієнтовану систему. Все, що можна змоделювати на дошці, можна змоделювати і за допомогою графа.

Не дуже добрі для:

Із-за наявності великї кількості зв'язків між вузлами графові бази даних як правило погано пристосовані до розміщення на декількох машинах. Швидкий обхід графа був би неможливий, якби довелося відправляти по мережі запити іншим вузлам, тому графові бази даних масштабуються погано. Швидше за все, графова база буде лише однією з частин вашої системи, в якій зберігаються тільки зв'язки, тоді як основні дані знаходяться у іншому місці. Визначитися з вибором бази даних нерідко виявляється складніше, ніж просто вирішити який жанр краще всього підходить для даних з конкретної предметної області. Очевидно, що соціальний граф найпростіше зберігати в графовій базі даних, але якщо йдеться про Facebook, то даних надто багато - в такій базі вони просто не помістяться. Швидше, ви зупинитеся на якій-небудь системі, пристосованій до "великих даних", наприклад, HBase або Riak. Таким чином, ви будете вимушені вибрати стовпцеву базу або сховище ключів і значень. Чи узяти інший приклад: реляційна база даних начебто ідеально підходить для реалізації банківських транзакцій, але ж і Neo4j підтримує ACID- транзакції, так що вибір є. Ці приклади показують, що при виборі бази даних - або декількох баз даних, - оптимальних для предметної області, слід брати до уваги не лише жанр. Взагалі кажучи, у міру зростання об'єму даних, гранична місткість деяких баз стає обмеженням. Стопцеві бази даних часто розподіляються по декількох ЦОД і підтримують "великі дані" максимального розміру, тоді як місткість графових баз найменша з усіх. Взагалі, це правило не універсальне. Так, Riak - великомасштабне сховище ключів і значень, призначене для сегментації на сотні і тисячі вузлів, тоді як Redis проектувалася для роботи в одному вузлі, але з можливістю реплікації типу головний-підлеглий і сегментації, реалізованої на рівні клієнта. При виборі бази даних треба враховувати і такі характеристики, як довговічність, доступність, узгодженість, масштабованість і безпека. Ви повинні вирішити, чи потрібна можливість пред'являти довільні запити, або буде досить технології mapreduce. Ви віддаєте перевагу HTTP/REST-інтерфейсу, або потребуєте драйверу для спеціалізованого двійкового протоколу? Навіть такі, на перший погляд, другорядні питання, як існування програми масового завантаження даних, можуть виявитися важливими в конкретному випадку.

Питання для самоконтролю

1. Охарактеризуйте роль і значення баз даних для Grid і хмарних обчислень
2. Перерахуйте особливості використання баз даних в розподілених обчисленях
3. Що таке розподілена база даних?
4. З чого складається система керування розподіленими базами даних ?
5. Що таке гомогенні та гетерогенні розподілені СКБД?
6. Що таке фрагментація, розподіл, реплікація?
7. Охарактеризуйте централізоване розміщення, фрагментоване розміщення
8. Що таке розміщення з повною реплікацією?
9. Що таке розміщення з вибірковою реплікацією?
10. Що таке розподілена обробка?
11. Що таке паралельна СКБД?
12. Охарактеризуйте переваги, недоліки і властивості СКБД.
13. Що лежить в основі реалізації реляційних СУБД ?
14. СУБД PostgreSQL – переваги і недоліки
15. Riak – розподілене сховище ключів і значень – переваги і недоліки
16. HBase – стовпчикова база даних – переваги і недоліки
17. MongoDB – переваги і недоліки
18. CouchDB - документо-орієнтована БД на основі JSON і REST– переваги і недоліки
19. Neo4j - тип сховищ даних NoSQL, отримав назву графова база- переваги і недоліки
20. Redis (віддалена служба словників) - сховище ключів і значень з розвиненим набором команд – переваги і недоліки.

Література

1. Eric Redmond, Jim R. Wilson. A Guide to Modern Databases and the NoSQL Movement– 2012 Pragmatic Programmers, LLC.– 384p.
2. http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks
3. http://allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
4. http://www.erlang.org/
5. http://ruby-lang.org
6. http://rubygems.org
7. http://rubygems.org/gems/riak-client
8. http://research.google.com/archive/mapreduce.html
9. http://wiki.basho.com/Replication.html
10. http://code.google.com/p/leveldb/
11. http://www.mongodb.org/downloads
12. http://rdoc.info/github/couchrest/couchrest/master/
13. http://download.freebase.com/datadumps/latest/browse/film/performance.tsv
14. http://neo4j.org/download/

15.http://redis.io

8. Система проектування SolidWorks

8.1. Система проектування SolidWorks. Основні принципи.

Система **SolidWorks** — система автоматизованого інженерного проектування та підготовки виробництва виробів будь-якої складності та призначення. SolidWorks є ядром інтегрованого комплексу автоматизації підприємства, за допомогою якого здійснюється підтримка життєвого циклу виробу згідно з концепцією CALS — технологій, включаючи двонаправлений обмін даними з іншими Windows-застосунками та створення інтерактивної документації.

Розробник - SolidWorks Corporation.

Програма з'явилась в 1993 році та склала конкуренцію таким продуктам як AutoCAD та Autodesk Mechanical Desktop, SDRC I-DEAS і Pro/ENGINEER, Solid Edge.

В залежності від класу розв'язуємих задач замовникам пропонується три базових конфігурації системи: SolidWorks, SolidWorks Professional та SolidWorks Premium.

Існує велика кількість версій. Назви версій:

Назва/Версія	**Version History Value**	**Дата релізу**
SolidWorks 95	440000	1995
SolidWorks 96	270	1996
SolidWorks 97	483	1996
SolidWorks 97Plus	629	1997
SolidWorks 98	817	1997
SolidWorks 98Plus	1008	1998
SolidWorks 99	1137	1998
SolidWorks 2000	1500	1999
SolidWorks 2001	1750	2000
SolidWorks 2001Plus	1950	2001
SolidWorks 2003	2200	2002
SolidWorks 2004	2500	2003
SolidWorks 2005	2800	2004
Solid Works 2006	3100	2005
SolidWorks 2007	3400	2006
SolidWorks 2008	3800	1 липня 2007
SolidWorks 2009	4100	28 січня 2008
SolidWorks 2010	4400	9 грудня 2009
SolidWorks 2011	4700	17 серпня 2010

SolidWorks 2012	5000	вересень 2011
SolidWorks 2013	6000	вересень 2012
SolidWorks 2014	7000	7 жовтня 2013
SolidWorks 2015	8000	9 вересня 2014
SolidWorks 2016	9000	1 жовтня 2015
SolidWorks 2017	10000	19 вересня 2016
SolidWorks 2018	11000	26 вересня 2017

Стабільний випуск SolidWorks 2017 SP0.0 (19 вересня 2016)
Версії SolidWorks 2021 SP1 (листопад 2020)[2][3].

Операційна система - Windows. Ліцензія Proprietary software.

SolidWorks – САПР і є конструкторською системою твердотільного параметричного моделювання машинобудівних конструкцій спеціально розробленою для використання на персональних комп'ютерах під керуванням операційної системи Windows. Стандартний графічний користувальницький інтерфейс Windows і засоби твердотільного параметричного моделювання дозволяють швидше і легше ніж будь-коли створювати тривимірні моделі деталей, складальні одиниці, генерувати креслення, значно знижуючи терміни проектування і зменшуючи час виходу виробів на ринок. При цьому слід зауважити, що всі машинобудівні підприємства в Україні приймають конструкторську документацію, створену тільки в системі SolidWorks. Якщо Ви створите документацію в іншій ситстемі, наприклад, в AutoCAD, то при необхідності передати її на виробниче підприємсво будете змушені переробити її в системі SolidWorks. Запам’ятайте це! Саме тому проводиться ознайомлення з системою SolidWorks.

Система SolidWorks є дуже потужною системою. Вона має окремі ресурси, наприклад:

SolidWorks Motion - призначений для розрахунку руху механізмів. Модуль використовує інформацію, що міститься в збірках SolidWorks з можливістю уточнення розрахункової моделі за допомогою його процедур. SolidWorks Motion є третім, найбільш функціональним інструментом SolidWorks, для імітації руху. Перші два рівні: рух збірки і фізичне моделювання, присутні в базовій конфігурації SolidWorks Standard, можуть бути використані для створення кінематичної моделі збірки, імітації руху без отримання чисельних характеристик. Після цього інформація без будь-яких додаткових дій сприймається на рівні SolidWorks Motion.

SolidWorks Routing - модуль проектування трубопроводів. Часто при проектуванні приладів і обладнання виникає завдання створення трубопроводів і комунікацій, які б об'єднали компоненти збірок і зробили тривимірну модель завершеною. Включення трубопровідної обв'язки в тривимірну модель виробу дозволяє вирішити багато проблем уже на етапі проектування і уникнути ситуації, коли на етапі монтажу виявляється, що труби неправильно зігнуті і заважають роботі інших систем або в існуючій конструкції недостатньо вільного місця для прокладки всіх необхідних комунікацій.

Завдання створення тривимірних моделей трубопроводів виникає при проектуванні приладів і обладнання різних галузей машинобудування, при створенні гідравлічних і пневматичних систем, в нафтогазовій промисловості при створенні трубопровідної обв'язки, а також при проектуванні різних інженерних комунікацій, підведень і шлангів.

Бібліотеки стандартних виробів **Toolbox SolidWorks** використовуються для забезпечення автоматичного сполучення стандартних виробів при вставлянні в збірку та надають можливість групових операцій. Toolbox дозволяє проводити проектувальні розрахунки балок і підшипників. Бібліотеки Toolbox редагуються і налаштовуються під конкретні завдання будь-якого підприємства.

SolidWorks Simulation (COSMOSWorks) – універсальний інструмент для аналізу методом кінцевих елементів.

SolidWorks Simulation існує в трьох конфігураціях: власне SolidWorks Simulation, SolidWorks Simulation Professional та SolidWorks Simulation Premium. Однак, навіть в мінімальній конфігурації модуля міцнісного аналізу забезпечується повноцінний статичний аналіз, як деталі, так і збірки з використанням кінцевих елементів твердого тіла, поверхонь і балок.

МодульSolidWorks Simulation дозволяє проводити інженерні розрахунки і моделювати різних впливи навколишнього середовища на виріб.

Основними особливостями SolidWorks Simulation є:

- лінійний аналіз;
- втомний аналіз металу;
- нелінійний аналіз;
- тепловий аналіз;
- частотний аналіз;
- аналіз виробів з пластмаси і гуми;
- динамічний аналіз і ін.

Лінійний аналіз напруг дозволяє швидко і ефективно перевірити якість, продуктивність і безпеку ще при їх створенні.

Лінійний аналіз напруг за допомогою SolidWorks Simulation може бути невід'ємною частиною процесу розробки, що знижує потребу в дорогих прототипах, виключає доопрацювання і затримки, а також економить часу і витрати на розробку. За допомогою такого аналізу можливо обчислювати напруги і деформації геометрії, такі як:

- деталь або збірка під навантаженням, яка деформується з невеликими поворотами і переміщеннями;
- статичні навантаження виробу (не враховуючи інерції) і постійні навантаження;
- матеріал під постійною напругою деформації.

Моделювання методом аналізу кінцевих елементів (FEA) – це дискретизація проектованих компонентів в тверде тіло, оболонку або балковий елемент, що використовує лінійний аналіз напружень для визначення реакції деталей і вузлів під впливом:

- сили;
- тиску;

- прискорення;
- температури.
- контакту між компонентами.

Для проведення аналізу напружень повинні бути відомими дані матеріалів компонентів. Стандартна база даних **SolidWorks CAD** попередньо завантажена матеріалами, які можуть бути використані **SolidWorks Simulation.**

Моделювання методом скінченних елементів використовується для розрахунку переміщень компонентів, деформацій і напружень при внутрішніх і зовнішніх навантаженнях.

Так як промислові компоненти виконуються переважно з металу, аналіз металевих компонентів може бути виконано за допомогою лінійного або нелінійного аналізу напружень. Аналіз неметалевих компонентів (наприклад, пластмаси або гумових деталей) повинен здійснюватися з використанням методів нелінійного аналізу напружень через їх складний взаємозв'язок деформації і навантаження.

SolidWorks Flow Simulation Electronic Cooling Module Add-In – додатковий модуль для теплового розрахунку електронних пристроїв. Він включає розширену базу даних по віртуальних вентиляторах, матеріалах електротехнічного призначення, термоелектричних охолоджувачах (елементи Пельтьє). Модуль забезпечує імітацію проходження постійного струму та нагріву ним, теплових трубок, багатошарових друкованих плат.

SolidWorks Flow Simulation HVAC Module Add-In – додатковий модуль **SolidWorks Flow Simulation** для розрахунку систем вентиляції, опалення та кондиціонування. Він включає: розширену базу даних по будівельних матеріалах і вентиляторах; уточнену модель теплообміну випромінювання з урахуванням відображення, заломлення і спектральних характеристик; розрахунок параметрів комфорту – середньої прогнозованої оцінки, середньої температури та ін.

SolidWorks Flow Simulation є модулем гідрогазодинамічного аналізу в середовищі **SolidWorks**. Для модуля Flow Simulation немає різниці між геометричними сутностями, створеними в SolidWorks або імпортованими в базовий модуль.

Забезпечується підтримка для 64-розрядних операційних систем з доступом до всієї доступної оперативної пам'яті. Також використовується багатопроцесорність.

Модуль Flow Simulation програмного середовища SolidWorks дає можливість моделювання процесів:
- стаціонарні і нестаціонарні течії;
- стискувані і нестискувані (рідини або гази) течії, включаючи до-, транс- і надзвукові режими;
- ідеальні і реальні гази;
- неньютонівські рідини;
- одно і багатокомпонентні течії без хімічних взаємодій і розділення фаз;
- спільні розрахунки течії рідини або газу та теплопередачі всередині твердого тіла без наявності границі розділення газ – рідина;
- ламінарні і турбулентні течії, враховуючи ламінарний/турбулентний перехід;

- «заморожування» течій для розділення «швидких» і «повільних» процесів;
- течії в пористих середовищах з урахуванням теплопровідності стінок;
- урахування шорсткості стінок;
- зовнішні і/або внутрішні течії;
- конвекційний теплообмін, вільна, вимушена або змішана конвекція;
- радіаційний теплообмін з управлінням прозорістю стінок і розділенням властивостей стінок для теплообміну випромінюванням і сонячною радіацією;
- розрахунок траєкторій твердих частинок і крапель в потоці та ін.

Початковими і граничними умовами можуть задаватися наступні вихідні параметри:
- швидкість, тиск (статичний, динамічний, оточуючого середовища), масові та об'ємні витрати;
- температура, концентрація компонентів, параметри турбулентності;
- витратно-напірні характеристики віртуальних вентиляторів;
- різноманітні типи стінок, включаючи жорсткість, коефіцієнт тепловіддачі і параметри умовного середовища на стінках, що не межують з реальним середовищем;
- джерела тепла (об'ємні і поверхневі), віртуальні тепловентилятори;
- можливості вказати залежність граничних умов та параметрів від часу та координат;
- симетрія відносно базових площин і періодична симетрія.

Керування обчислювальними операціями виконуються безпосередньо по згенерованій розрахунковій сітці моделі SolidWorks, що створюється автоматично в області твердого тіла. Сітка адаптується в залежності від геометричних характеристик моделі і поля вирішення.

Результати досліду виводяться у вікні SolidWorks. Існує можливість виводу функції у вигляді кольорових епюр, векторів та ізоліній, відображення результатів за допомогою ізоповерхонь.
За результатами розрахунків можна створювати тривірневі траєкторії; виводити характеристики розрахунків, розподіл будь-якої характеристики вздовж будь-якої кривої в MS Excel.

За той обмаль часу, який відводиться для дисципліни Ви, звичайно, не зможите глибоко зануритись в освоєння названих питань, але саме знання про них допоможе Вам впоратися при їх виникнені. Я це знаю з власного досвіду.

Основне призначення SolidWorks – забезпечення наскрізного процесу проектування, інженерного аналізу та підготовки виробництва виробів будь-якої складності і призначення, включаючи створення інтерактивної документації та забезпечення обміну даними з іншими системами.

Основні принципи SOLIDWORKS

Основою геометричної моделі будь-якого проекту в SolidWorks є твердотільна деталь. Потім, внаслідок застосування операцій композиції вони складаються у збірки (складання). Геометричні моделі виробів створюються за допомогою методів параметричного моделювання. Деталі створюються як у незалежному режимі, так і в контексті збирання, прив'язуючись до елементів

збирання, і зберігаються в окремих файлах. Одні й ті самі деталі можуть використовуватися у різних збірках. Проектування моделі може здійснюватися як "згори донизу", так і "знизу вгору", а також в результаті комбінації цих підходів. Зміни в моделі можуть проводитись у реальному масштабі часу.

Модель SolidWorks складається з трьох основних видів уявлень - деталей, складання та креслення. Зв'язок між деталями, складаннями та кресленнями гарантує, що зміни, зроблені в одному вигляді, автоматично виконуються у всіх інших видах. Усі операції зі створення ескізів, деталей, складання суворо документуються і відображаються у дереві конструювання та менеджері властивостей. "Дерево моделі" дозволяє здійснювати "навігацію" за структурою виробу, виконувати операції редагування елементів, приховування та відображення елементів (у деталях), компонентів (у збірках) та видів (у кресленнях), а також копіювання елементів та компонентів. Елементи моделі можна тиражувати ("розмножувати") з використанням кругових та лінійних масивів. Можливі масиви як окремих елементів, так груп елементів. Система передбачає виконання режимів автоматичного проставлення лінійних та кутових розмірів, взаємозв'язків, а також сполучення. В останньому випадку система сама за типом вибраних елементів визначає можливий спосіб і тип сполучень (співвісність, збіг). Накладення взаємозв'язків приводить до того що компонент втрачає деякі ступені свободи.

Система SolidWorks ефективна для вирішення задач аналізу та синтезу проектно-конструкторських рішень. Складання можуть реструктуризуватися, тобто "розпускатися" на окремі підзбірки, компоненти та деталі, останні знову можуть об'єднуватися в інші підзбірки, компоненти можуть перетягуватися з одних підзбірок в інші і т. д.

SolidWorks дозволяє створювати розрізи та перерізи (прості, ступінчасті, розгорнуті), місцеві види та ізометрію. При створенні розрізів та перерізів система автоматично виконує штрихування. Тип (крок та нахил) штрихових ліній можна змінювати. Замикаючі об’єм поверхні автоматично конвертуються в тверде тіло.

Проектування виробів супроводжується обчисленням масово-геометричних характеристик, до яких належать площі поверхонь, об’єм тривимірних тіл, маси, центри тяжкості, моменти інерції.

При компонуванні виробів важливу роль відіграє перевірка елементів конструкції на інтерференцію. Наприклад, при розкритті панелей сонячних батарей космічного апарату неприпустиме їхнє зіткнення з іншими елементами конструкції. Інший приклад: антенні пристрої космічного апарату повинні розміщуватися таким чином, щоб при їх розворотах також не було зіткнень і, крім того, в зону огляду антенних пристроїв не попадали інші елементи конструкції. Це виявляється перевіркою елементів конструкції на інтерференцію.

Важливо також мати можливість оцінювати геометричні конструкції з погляду впливу на їх працездатність значень допусків (наприклад, точності установки чутливих елементів приладів космічного апарату на його поверхню). Усі ці завдання ефективно вирішуються з використанням системи SolidWorks.

У системі SolidWorks використовують тривимірний підхід до проектування. При проектуванні деталі від початкового ескізу до кінцевої моделі створюється тривимірний об'єкт.

Модель SolidWorks складається з деталей (рис. 1), збірок (рис. 2) та креслень (рис. 3). Кожна деталь, збірка або креслення у системі SolidWorks називається документом і зберігається в окремому файлі. Зазвичай спочатку малюється ескіз, створюється основа, а потім у модель додаються численні елементи. За допомогою елементів можна додавати матеріал у деталь або збирання, або видаляти його. Навіть при видаленні матеріалу з моделі елемент, як і раніше, вважається доданим. Саме так слід сприймати модель. Розглянемо для прикладу отвір - це елемент, а не звичайний отвір. Елемент має власні властивості (ім'я, розміри, обмеження, колір), які можна завжди редагувати. Ескіз, який використовується для побудови елементу також можна відредагувати. Для створення деталі можна використовувати один або кілька елементів. Потім можна об'єднати та виконати сполучення відповідних деталей для створення збірок. Потім з документів деталей або збирання можна створити креслення. Креслення генеруються системою автоматично з тривимірного об'єкту. Можна скільки завгодно вдосконалювати модель, додаючи, змінюючи елементи та їх порядок. Зв'язок між деталями, складаннями та кресленнями гарантує, що зміни, зроблені в одному виді, автоматично виконуються у всіх інших видах.

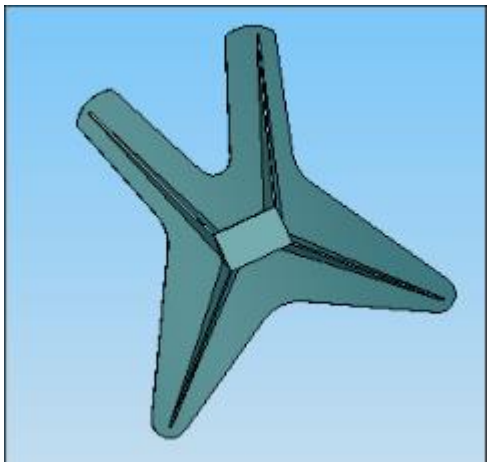


Рис.1. Тривимірна деталь

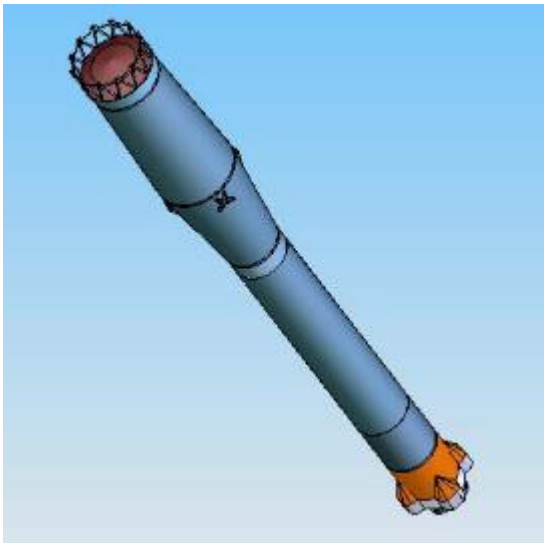


Рис.2. Тривимірна збірка

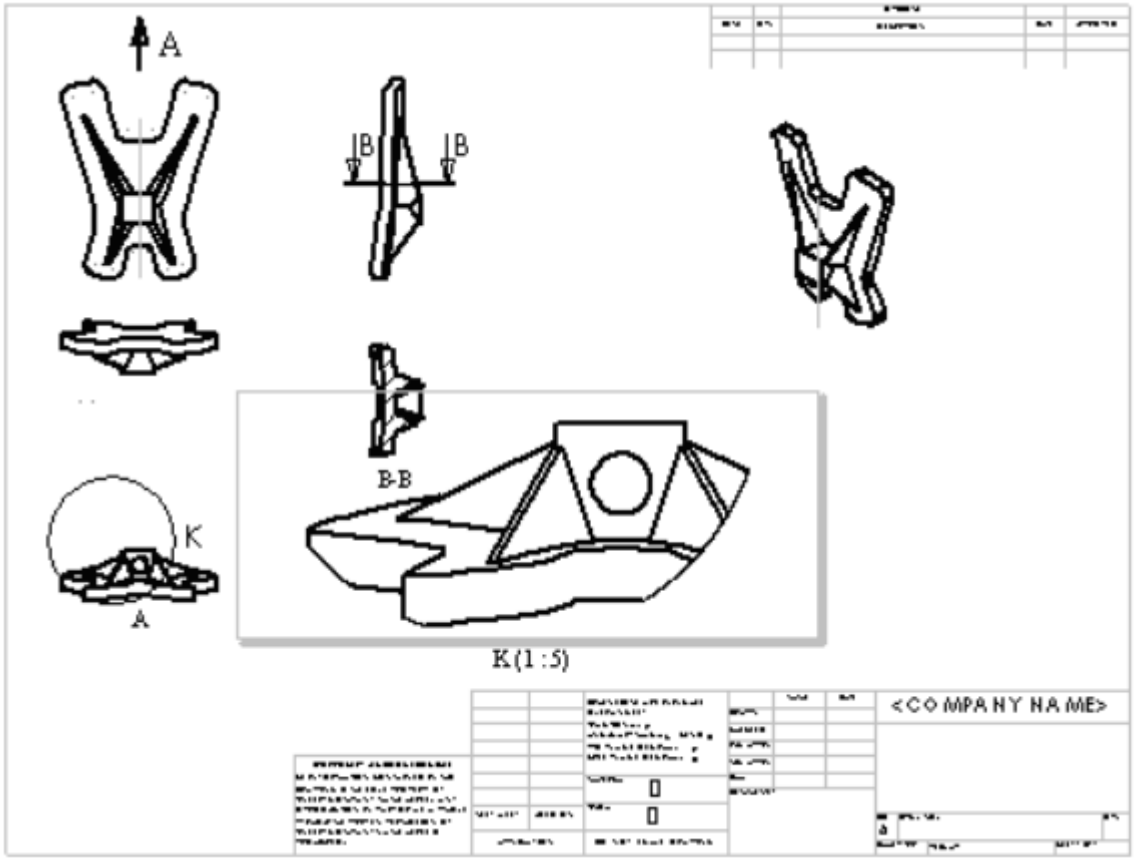


Рис. 3. Креслення, згенерований із тривимірної моделі

Деталі являються основними стандартними блоками програмного забезпечення SOLIDWORKS.

Збірки включають деталі або інші збірки, називаємі вузлами .

Модель SOLIDWORKS складається із тривимірної геометрії , яка визначає її кромки, грані і поверхні.

Програма SOLIDWORKS позволяє швидко і точно проектувати моделі. Моделі SOLIDWORKS:

• визначаються тривимірним проектуванням

• засновуються на компонентах.

Креслення чи збірки можно створювати на будь-якому етапі в процесі проектування.

Вікна документів в SOLIDWORKS

У вікні документів SolidWorks міститься (рис.4)*:

- FeatureManager (дерево конструювання), в якому відображена структура деталі, збірки (складання) або креслення;
- PropertyManager (мененджер властивостей);
- ConfigurationManager (менеджер конфігурацій);
- графічна область;
- панелі інструментів;
- строка меню;
- строка стану;
- панель задач.

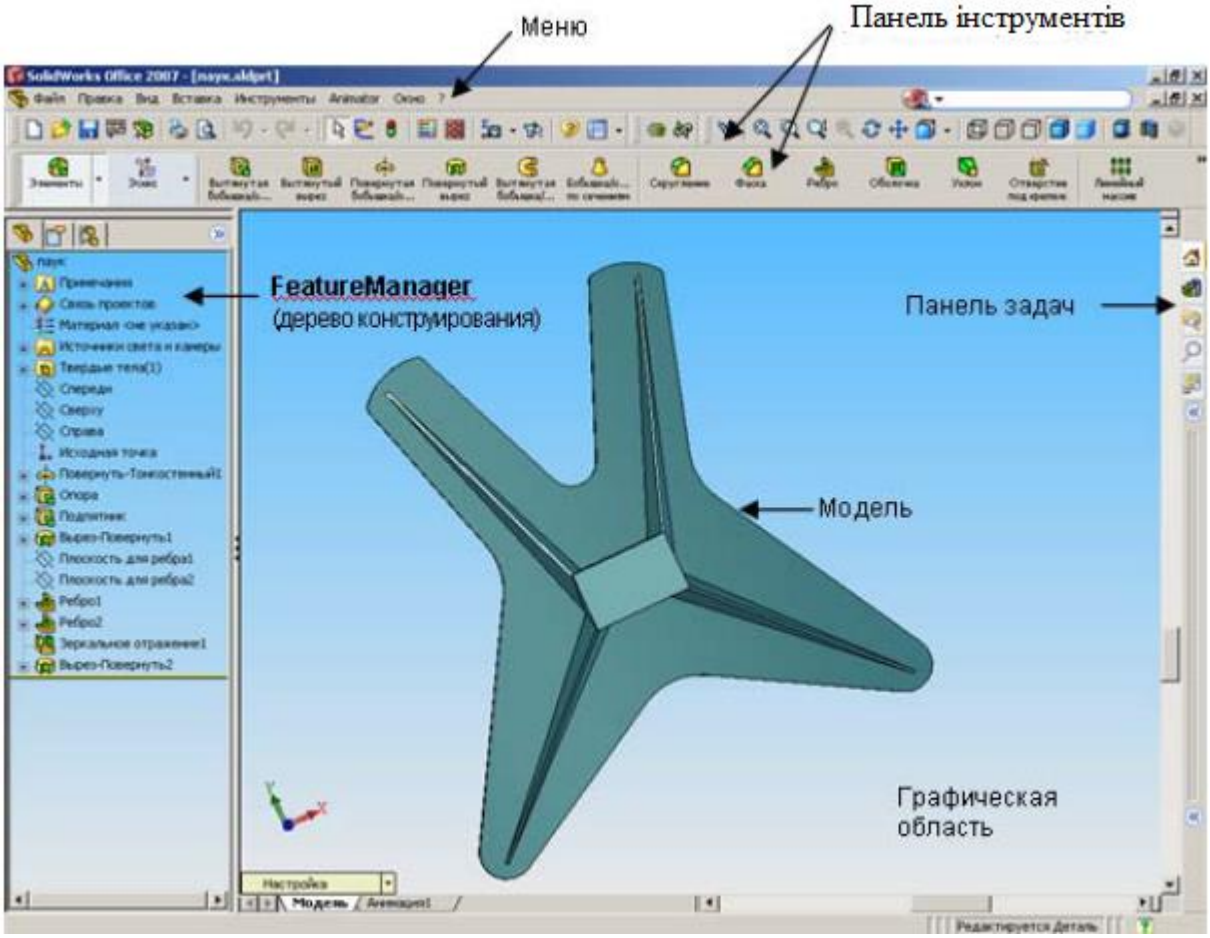


Рис. 4. Вікно SolidWorks

* SolidWorks повністтю русифікований і для кращого сприйняття всім загалом будемо це використовувати при відтворенні вікон.

*FeatureManager (*Дерево конструювання)

У дереві конструювання FeatureManager у лівій частині вікна SolidWorks відображається контурний вигляд активної деталі, складання або креслення (див. рис. 4). Можна легко побачити побудову моделі або складання або переглянути різні листи та види креслення.

Дерево конструювання FeatureManager і вікно графічної області динамічно пов'язані. Можна вибирати елементи, ескізи, креслярські види та допоміжну геометрію у будь-якій частині вікна.

Дерево конструювання FeatureManager дозволяє:

- вибрати елементи деталі або складання по імені;
- визначити і змінити послідовність, в якій створюються елементи;
- переупорядкувати елементи шляхом їх перетягування у списку дерева конструювання FeatureManager;
- відобразити розміри елемента;
- змінити ім'я елемента.

*PropertyManage*r (менеджер властивостей)

PropertyManager (менеджер властивостей) з'являється автоматично замість дерева конструювання FeatureManager, коли ми користуємося будь-яким інструментом (рис. 5). Менеджер властивостей дозволяє керувати графікою, не перекриваючи її діалоговими вікнами. Тут ми можемо простежити взаємозв'язки ескізу, побачити та змінити координати початкових і кінцевих точок наших об'єктів, радіуси дуг та кути нахилу прямих щодо осі X.

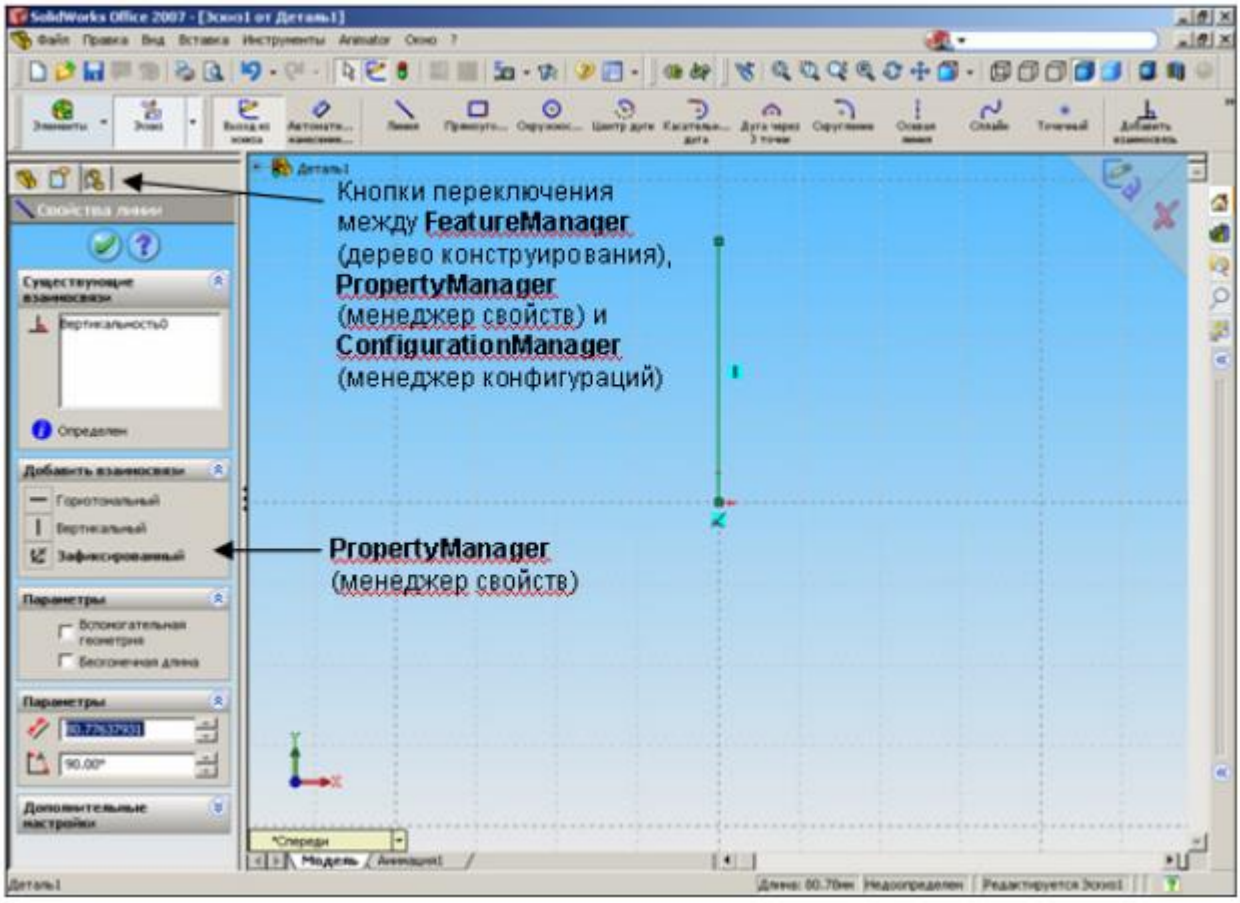


Рис. 5

ConfigurationManager (менеджер конфігурацій)

Конфігурації потрібні у тому, щоб у одному документі створювати кілька різних варіантів однієї й тієї ж моделі.У кожній конфігурації деталі можна задавати різні розміри ескізів, використовувати різні елементи, змінювати властивості.

У кожній конфігурації збірки можна використовувати різні пари та

різні деталі, наприклад, можна створити кілька конфігурацій збирання.

Панель задач

Панель завдань надає додаткові можливості управління

файлами під час роботи у SolidWorks. Наприклад, за допомогою неї можна відкрити документ (вкладка *Ресурси SolidWorks*), скористатися бібліотечними елементами (вкладка *Бібліотека проектування*) або знайти файли

за допомогою вбудованого провідника (Вкладка *Провідник файлів*).

Панель завдань з'являється автоматично під час відкриття документу SolidWorks і містить такі вкладки:

- **Ресурси SolidWorks;**
- **Библиотека проектування**
- **Провідник файлів**
- **Пошук;**
- **Відобразить палитру.**

Для прикріплення панелі задач в разгорнутому виді потрібно натиснути на кнопку в стічці заголовку, а для зворотньої процедури — на (рис. 6).

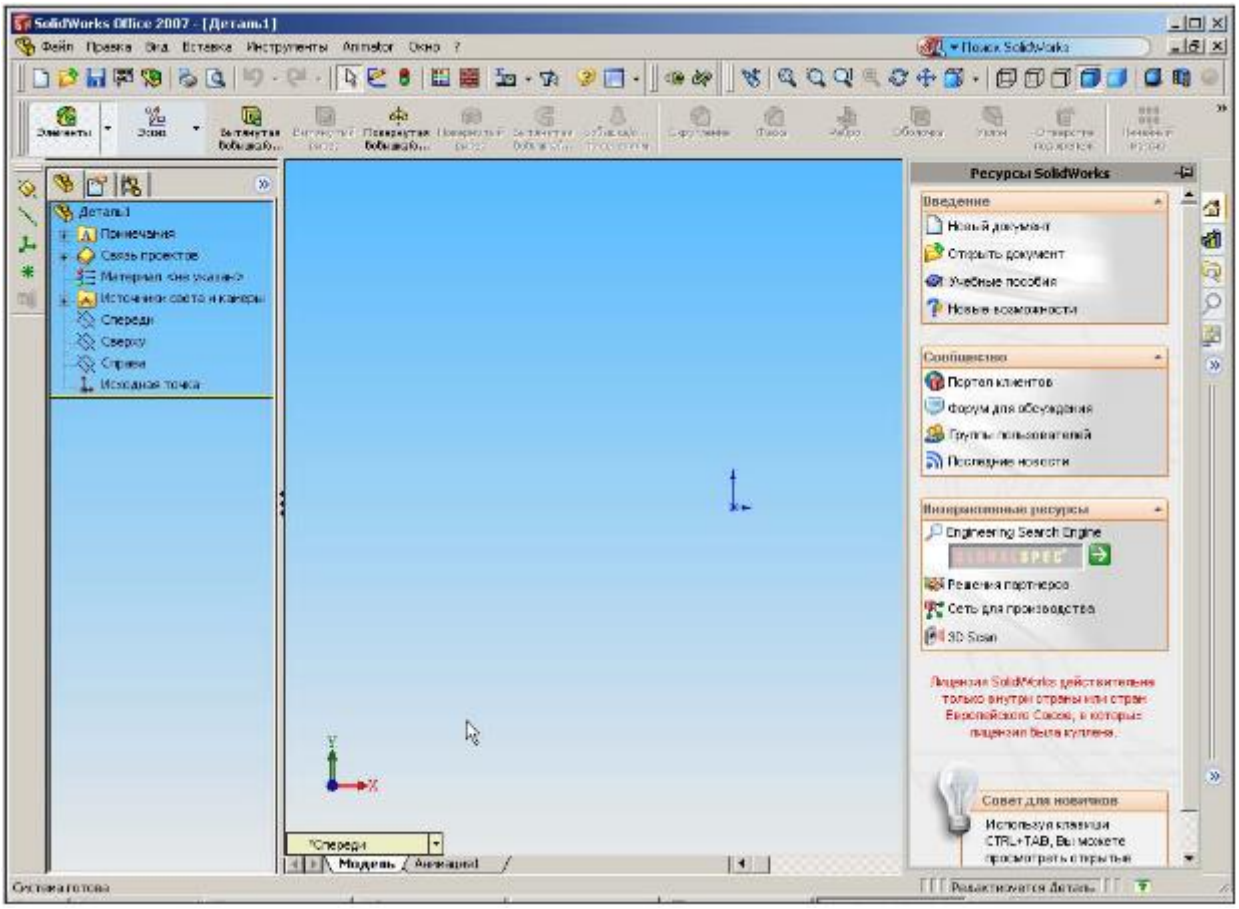


Рис.6. Відображення панелі задач

Тривимірне проектування

В програмі SOLIDWORKS використовується тривимірний підхід до проектування. При проектуванні деталі від первинного эскизу до кінцевого результату створюється тривимірниа модель. На основі цієї моделі можна створювати 2-мірні креслення або сполучати компоненти, які складаються з деталей або вузлів для створення 3-мірних збірок. Можна також створювати двомірні креслення тривимірних складань (збірок). При проектуванні моделі за допомогою програми SOLIDWORKS можна надати їй велику наочність за всіма

трьома вимірами, тобто уявити модель у тому вигляді, в якому вона буде під час виробництва.

Однією з найбільш зручних особливостей програми SOLIDWORKS є відображення будь-яких змін, які виконуються в деталі, у всіх пов'язаних кресленнях чи складаннях (збірках).

В проектуванні в SOLIDWORKS використовується наступна *термінологія*:

Початкова (вихідна) точка

Елементи зустрічаються по всій програмі SOLIDWORKS та документації. Відображається у вигляді двох кольорових стрілок та представляє (0,0,0) координату моделі. Коли ескіз стає активним, вихідна крапка (точка) ескізу відображається червоним і представляє (0,0,0) координату ескізу. Розміри та взаємозв'язки можуть бути додані до вихідної точки моделі, але не ескізу.

Площина

Плоска допоміжна геометрія. Можна використовувати площини, наприклад, для додавання двомірного ескізу, для розрізу моделі, а також як нейтральні площини для ухилу.

Вісь

Пряма лінія, яка використовується для створення геометрії моделі, елементів або шаблонів. Вісь можна створити різними способами, включаючи перетин двох площин. SOLIDWORKS неявно створює тимчасові вісі для кожної конічної чи циліндричної грані моделі.

Грань

Межі, що дозволяють визначити форму моделі чи поверхні. Грань - це область моделі або поверхні (плоска або неплоска), яку можна вибрати. Наприклад, прямокутна твердотільна деталь має шість граней.

Кромка

Місце, де дві або кілька граней перетинаються та з'єднуються. Краї можна вибирати, наприклад, для створення ескізу та розмірів.

Література

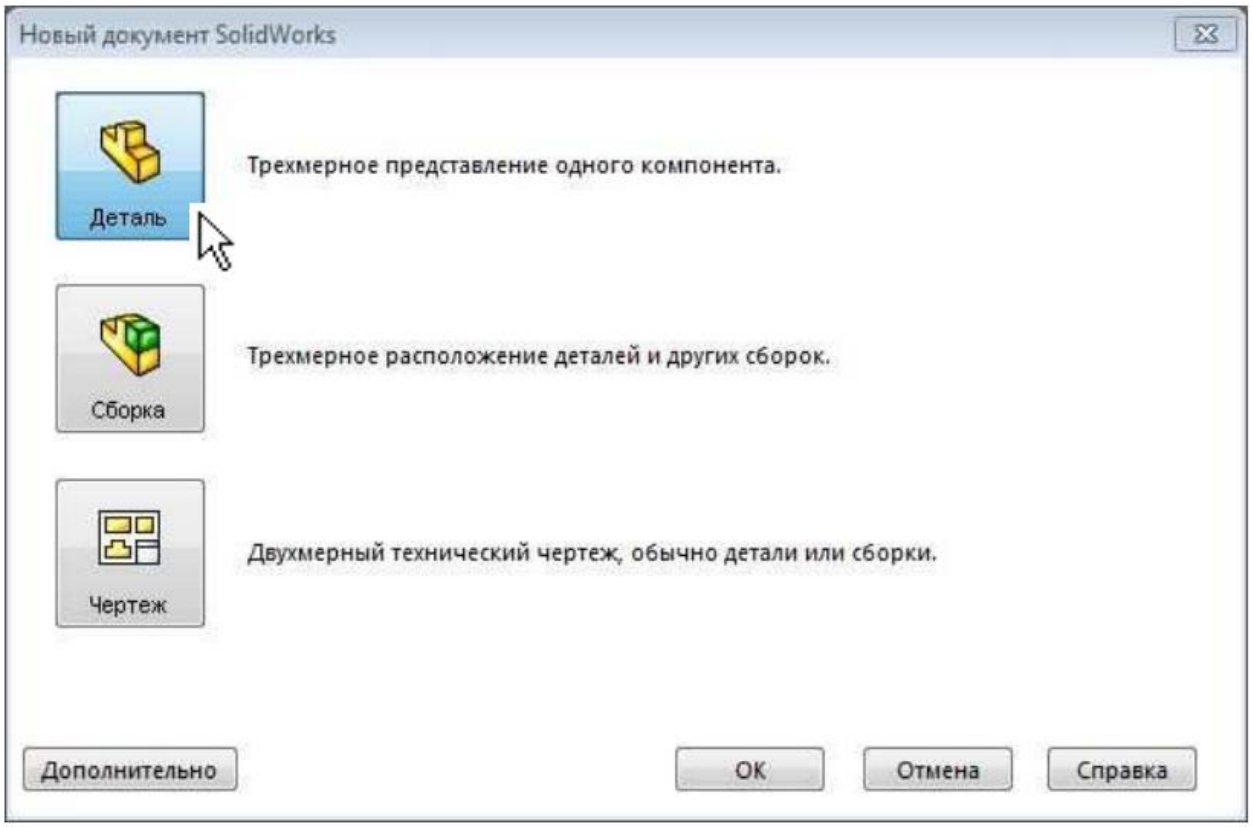
1. Вебсайт www.solidworks.com (http://www.solidworks.com)
2. Notes de version de SOLIDWORKS 2019 SP0 (https://files.solidworks.com/Supportfiles/Release_Notes/2019/French/relnotes.htm) — 2018.
3. Notes de version de SOLIDWORKS 2019 SP3 (https://files.solidworks.com/Supportfiles/Release_Notes/2019/French/relnotes.htm) — 2019.

4. Бібліотеки для Solidworks (http://www.leninsw.com)

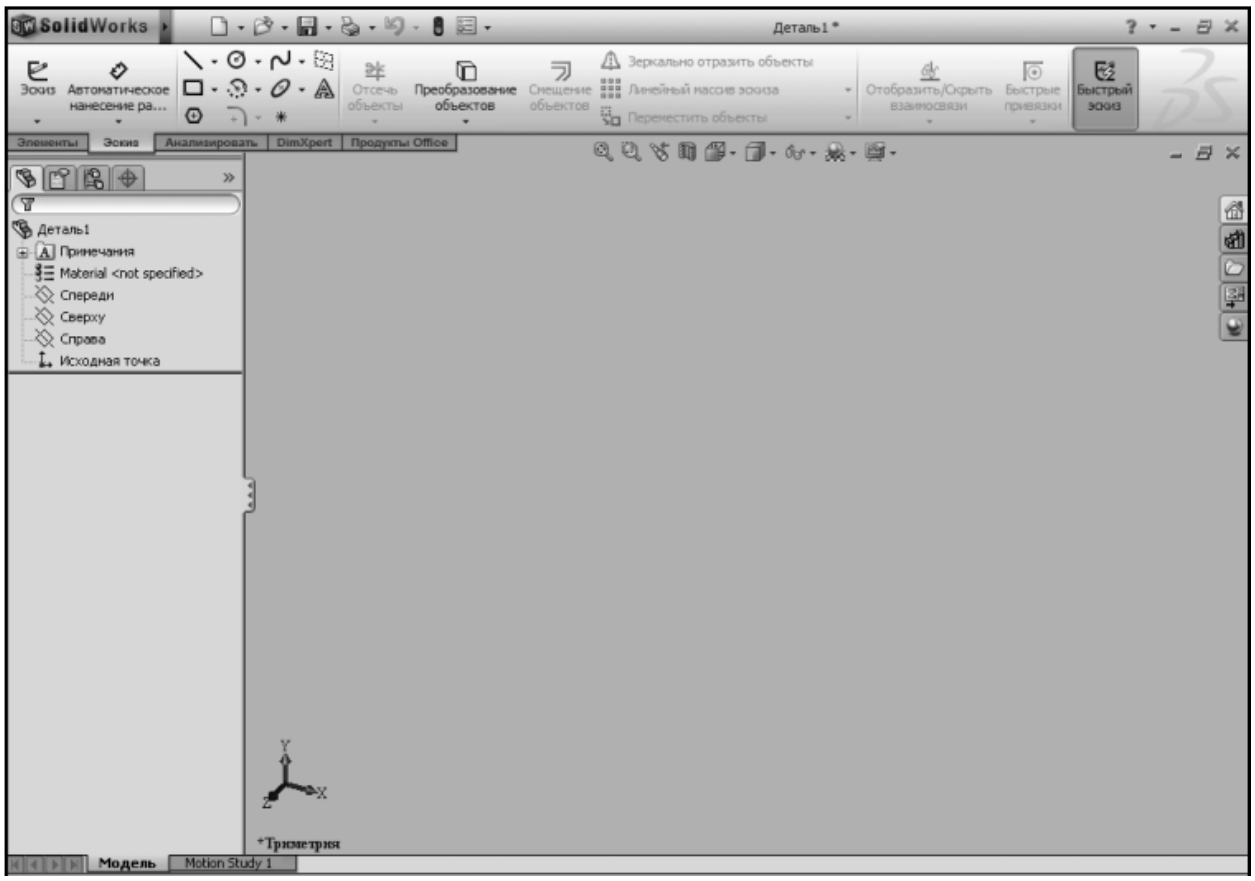
8.2. Інтерфейс користувача. Процес проектування.

Запуск *SolidWorks* здійснюється клацанням миші по значку  на робочому столі або вибором команди *Пуск* | *Програми* | *SolidWorks* | Після запуску програми будуть доступні два значки у лівому верхньому куті екрана:  - Відкриття існуючих файлів деталей, збірок або креслень та —  створення нових файлів. Для створення нового файлу клацнути на другому значку лівою кнопкою миші. Якщо вже є будь-які файли SolidWorks, то можна скористатися першою піктограмою. У з'явившомуся вікні шаблонів, показаному на малюнку нижче*, оберемо шаблон *Деталь*, вибравши значок *Деталь*, та натиснемо кнопку OK.

* SolidWorks повністтю русифікований і для кращого сприйняття всім загалом будемо це використовувати при відтворенні вікон.



Далі Ви бачите перед собою інтерфейс користувача *SolidWorks* , який знаходиться в режимі *Ескіз* (див. малюнок нижче). Програма написана відповідно до стандартів *Windows*, тому всі елементи інтерфейсу розміщені на звичних місцях. У верхній частині знаходиться рядок меню, з якого викликаються команди програми. Щоб викликати меню, необхідно навести курсор миші на напис SolidWorks. Наприклад, у меню *Файл* згруповано такі команди, як *Створити*, *Відкрити*, *Закрити*, *Зберегти* та ін, тобто працюючими з файлами. Меню *Правка* дозволяє вирізати, копіювати, вставляти та видаляти елементи побудови, а також скасовувати введені команди. Меню *Вид* об'єднує команди, які задають орієнтацію моделі та виду проектованої деталі чи складання. Меню *Вставка* призначена для додавання різних елементів побудови. Коли меню неактивне, його замінює меню часто вживаних команд. Через меню (*Допомога*) можна отримати доступ до великої електронної довідкової системи, за допомогою якої користувач *SolidWorks* може швидко освіжити у пам'яті призабуті команди.



В наступній таблиці перераховані деякі інструменти, використовуємі при проектуванні, і їх розташування в меню, на панелях інструментів і в *CommandManage*r

Інструмент	Значок	Меню	Панель інструментів	Диспетчер команд
Создать		**Файл > Создать**	Стандарт	Строка меню
Сохранить		**Файл > Сохранить**	Стандарт	Строка меню
Параметры		**Сервис > Параметры**	Стандарт	Строка меню
Эскиз		**Вставка > Эскиз**	Эскиз	Эскиз
Автоматическое нанесение размеров		**Сервис > Размеры > Авто**	Эскиз	Эскиз
Прямоугольник		**Сервис > Объекты эскиза > Прямоугольник**	Эскиз	Эскиз
Вытянутая бобышка/Основание		**Вставка > Бобышка/Основани > е Вытянуть**	Элементы	Элементы
Оболочка		**Вставка > Элементы > Оболочка**	Элементы	Элементы
Вставить компоненты		**Вставка > Компонент > Существующая деталь/Сборка**	Сборка	Сборка
Сопряжение		**Вставка > Сопряжение**	Сборка	Сборка

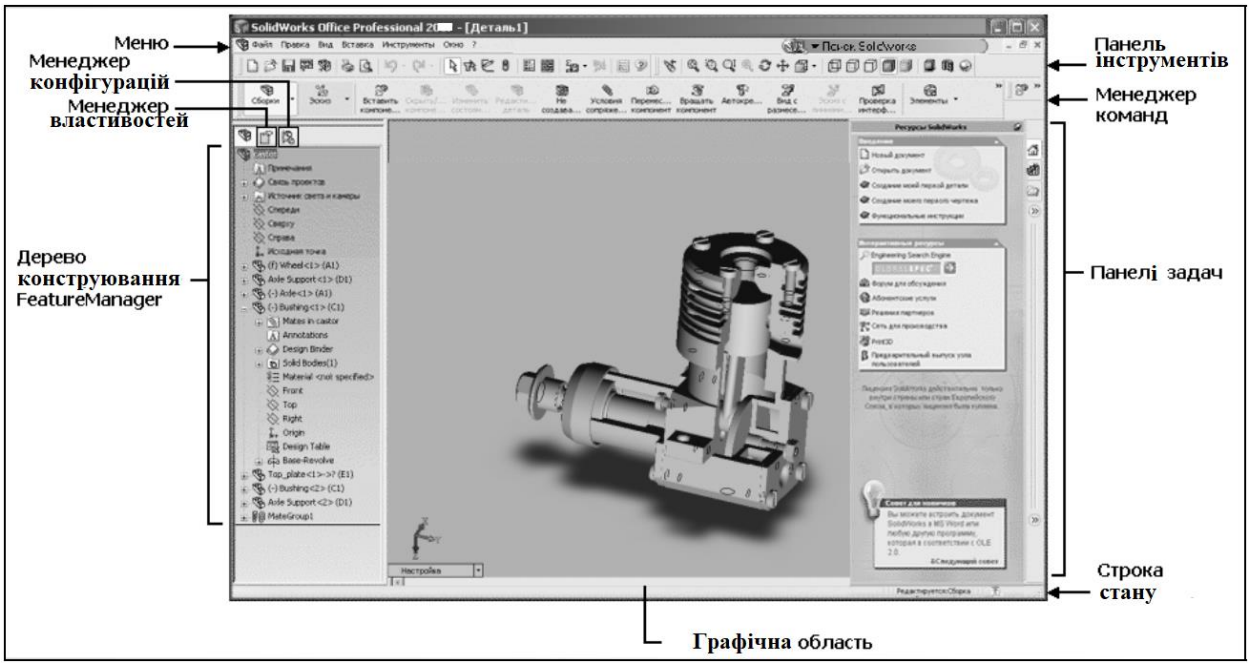
При цьому звертаю увагу, що в різних версіях значки можуть мати незначні відмінності, наприклад, значок ескізу в попередніх версіях має вигляд 

SOLIDWORKS включає інструменти інтерфейсу користувача та функції, які дозволяють ефективно створювати та редагувати моделі, включаючи знайомі функції системи Windows, як перетягування та зміна розмірів вікон. У програмі SOLIDWORKS використовується багато таких же значків, як і в Windows, наприклад, для друку, відкриття, збереження, вирізання та вставки.

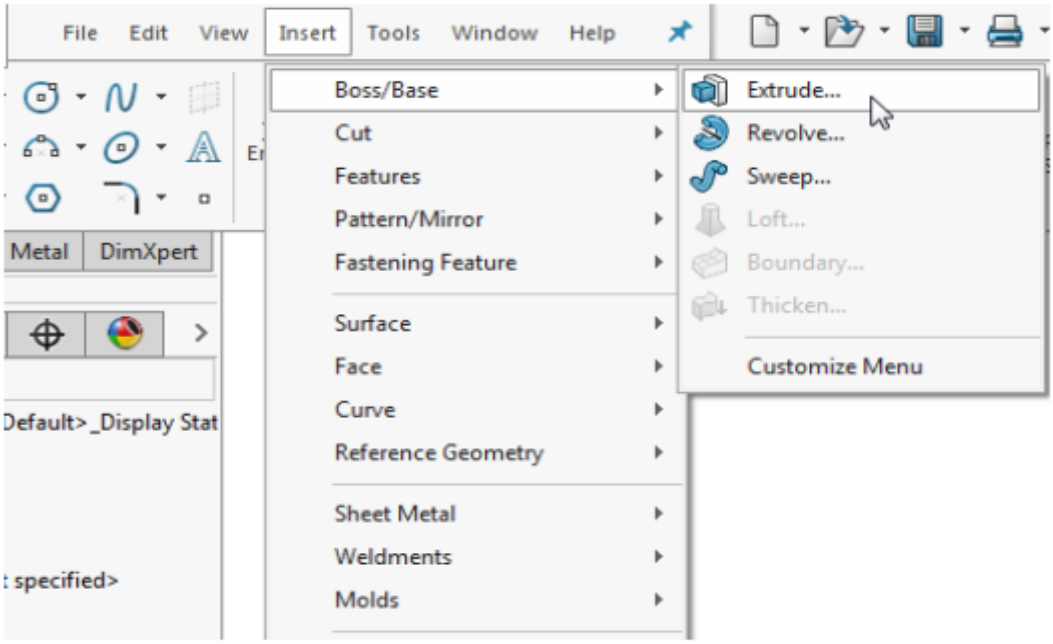
Вікна документів SOLIDWORKS містять дві панелі.

Ліва панель або область менеджера відображає структуру деталі, збірки або креслення. Вибравши елемент із дерева конструювання FeatureManager, можна , наприклад, відредагувати ескіз, відредагувати елемент, а також погасити або висвітлити елемент чи компонент.

Права панель є графічною областю, в якій виконуються різні операції над деталлю, збиранням або кресленням.



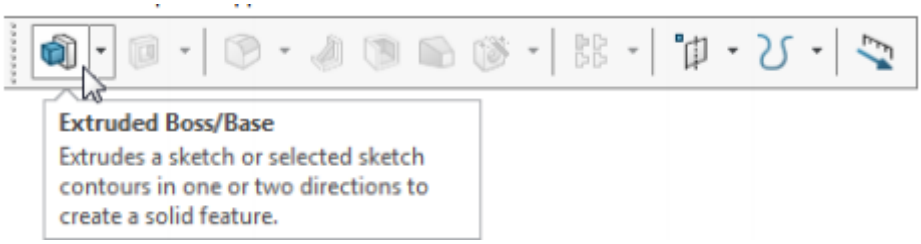
До всіх команд програми SOLIDWORKS можна отримати доступ за допомогою меню. У меню SOLIDWORKS використовуються угоди Windows, включаючи вкладені меню та прапорці, які вказують активність пункту. Також можна використовувати контекстні меню, доступні за допомогою правої кнопки миші.



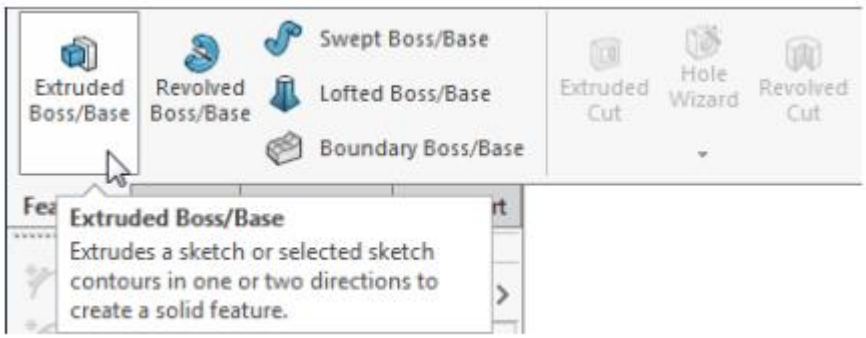
Доступ до функцій SOLIDWORKS можна отримати за допомогою панелі інструментів. Панелі інструментів організовані за своїми функціями, наприклад панель інструментів *Ескіз* або *Збірка*. Кожна панель інструментів містить

окремі значки, які позначають певні інструменти, наприклад, ***обертати вид, круговий масив і круг (Вращать вид , Круговой массив і Окружность*)**.

Панелі інструментів можна відображати та приховувати, розташовувати по всіх чотирьох сторонах вікна SOLIDWORKS або переміщати по всьому екрану. У SOLIDWORKS стан панелей інструментів зберігається від сеансу до сеансу. Крім того, можна додавати або видаляти інструменти для налаштування панелей інструментів. При наведенні вказівника на кожну піктограму відображаються спливаючі підказки.



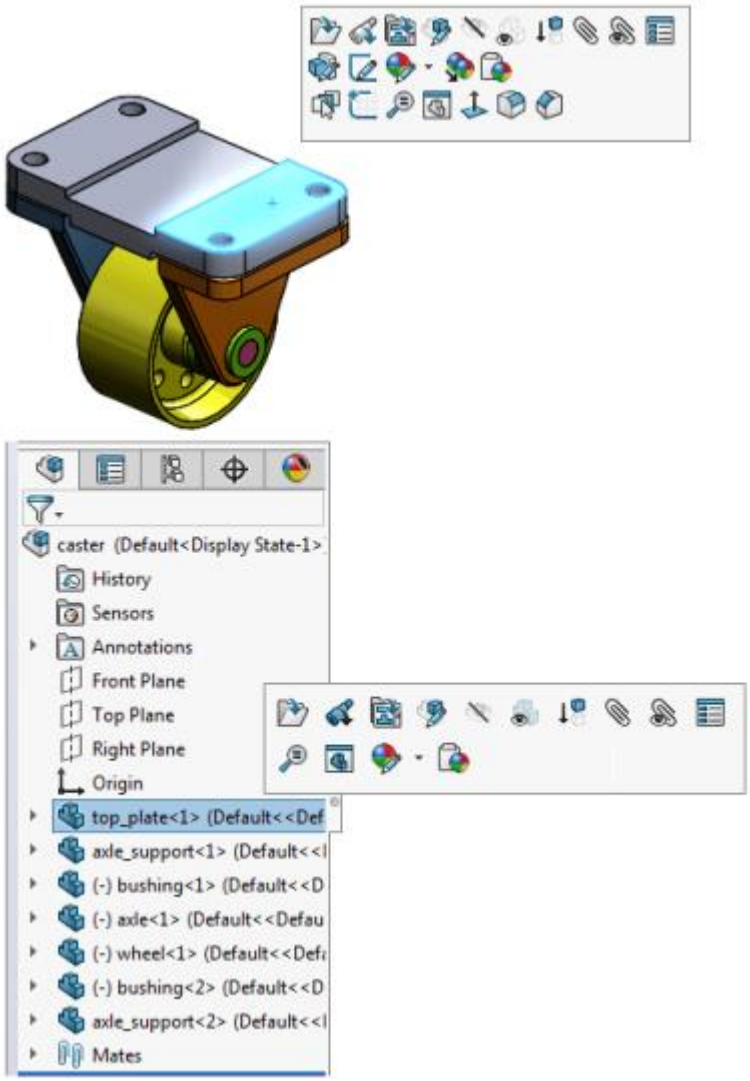
CommandManager (диспетчер команд) – це контекстна панель інструментів, яка оновлюється автоматично, залежно від типу активного документу. При натисканні на вкладку під *CommandManager* вона оновлюється для відображення відповідних інструментів. Кожному типу документа, наприклад деталі, складання або креслення, відповідають різні вкладки, визначені щодо його завдань. Вміст вкладок можна налаштовувати подібно панелей інструментів. Наприклад, при виборі вкладки *Елемент* відображаються інструменти, пов'язані з елементами. Також можна додавати або видаляти інструменти для налаштування *CommandManager*. При наведенні вказівника на кожну піктограму відображаються спливаючі підказки.



Панелі меню, які налаштовуються, дозволяють створювати власні набори команд для режиму деталі, складання, креслення та ескізу.

Контекстні панелі інструментів відображаються під час вибору елементів у графічній області або у дереві конструювання *FeatureManager*. Вони надають

доступ до часто виконуваних діям для відповідного контексту. Контекстні панелі інструментів доступні для деталей, збірок та ескізів.



Процес проектування.

Процес проектування зазвичай включає наступні кроки:

• Ідентифікація вимог моделі.

• Створення концепції моделі на основі певних потреб.

• Розробка моделі на основі цієї концепції.

• Аналіз моделі.

• Створення прототипу моделі.

• Конструювання моделі.

• Зміна в моделі (за потреби).

Спосіб проектування

Перед фактичним проектуванням моделі корисно запланувати спосіб, яким вона буде створена. Після того, як Ви визначили потреби та виділили відповідні концепції, можна розпочати розробку моделі.

Ескізи

Створіть ескізи та визначте спосіб нанесення розмірів та місця взаємозв'язків.

Елементи

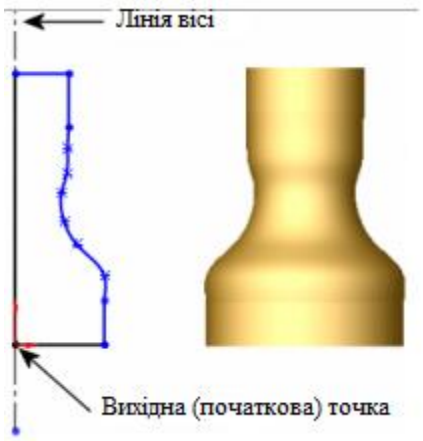
Виберіть відповідні елементи, наприклад витяжки та заокруглення, визначте найкращі елементи для використання та вирішіть, у якому порядку застосовувати ці елементи.

Складання (Збірки)

Виберіть компоненти для об'єднання та типи пар. Модель майже завжди включає один або кілька ескізів, а також один або кілька елементів. Однак не всі моделі включають складання. *Ескіз* є основою більшості тривимірних моделей. Створення моделі зазвичай починається з ескізу. В ескізі можна створити елементи. Для створення *Деталі* можна використовувати один або кілька елементів. Далі можна об'єднати та виконати поєднання відповідних деталей для створення збірок. Потім із документів деталей або складання можна створити креслення.

Ескіз – це двомірний профіль або поперечний переріз. Для створення двомірного ескізу можна використовувати площину чи плоску грань. Крім двомірних ескізів, можна також створювати тривимірні ескізи з віссю Z, а також з осями X та Y. Існує кілька способів створення ескізу. Усі ескізи містять такі елементи:

Вихідна (початкова) точка. У багатьох випадках ескіз починається в вихідній точці, яка служить якорем для ескізу.

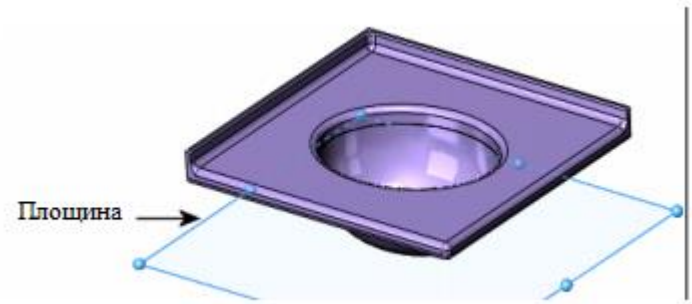


Осьова лінія проводиться через вихідну точку та використовується для створення елементу ***повернути***. Незважаючи на те, що осьова лінія не завжди

потрібна в ескізі, вона дозволяє забезпечити симетрію. Крім того, можна використовувати осьову лінію для використання взаємозв'язку "*дзеркально*", а також для забезпечення рівних і симетричних взаємозв'язків між об'єктами ескізу. Симетрія – це важливий інструмент, який допомагає швидше створювати осесиметричні моделі.

Можна створити площини в документах деталей або збірок. Можна створювати ескізи на площинах за допомогою таких інструментів ескізу, як *Лінія* або *Прямокутник*, а також створювати перерізи моделі. В деяких моделях площина ескізу впливає тільки на спосіб відображення моделі у стандартному ізометричному вигляді (3-мірному). Вона не впливає на задум проекту. Для інших моделей вибір правильної початкової площині, на якій створюватиметься ескіз, дозволяє створити більш ефективну *Модель*.

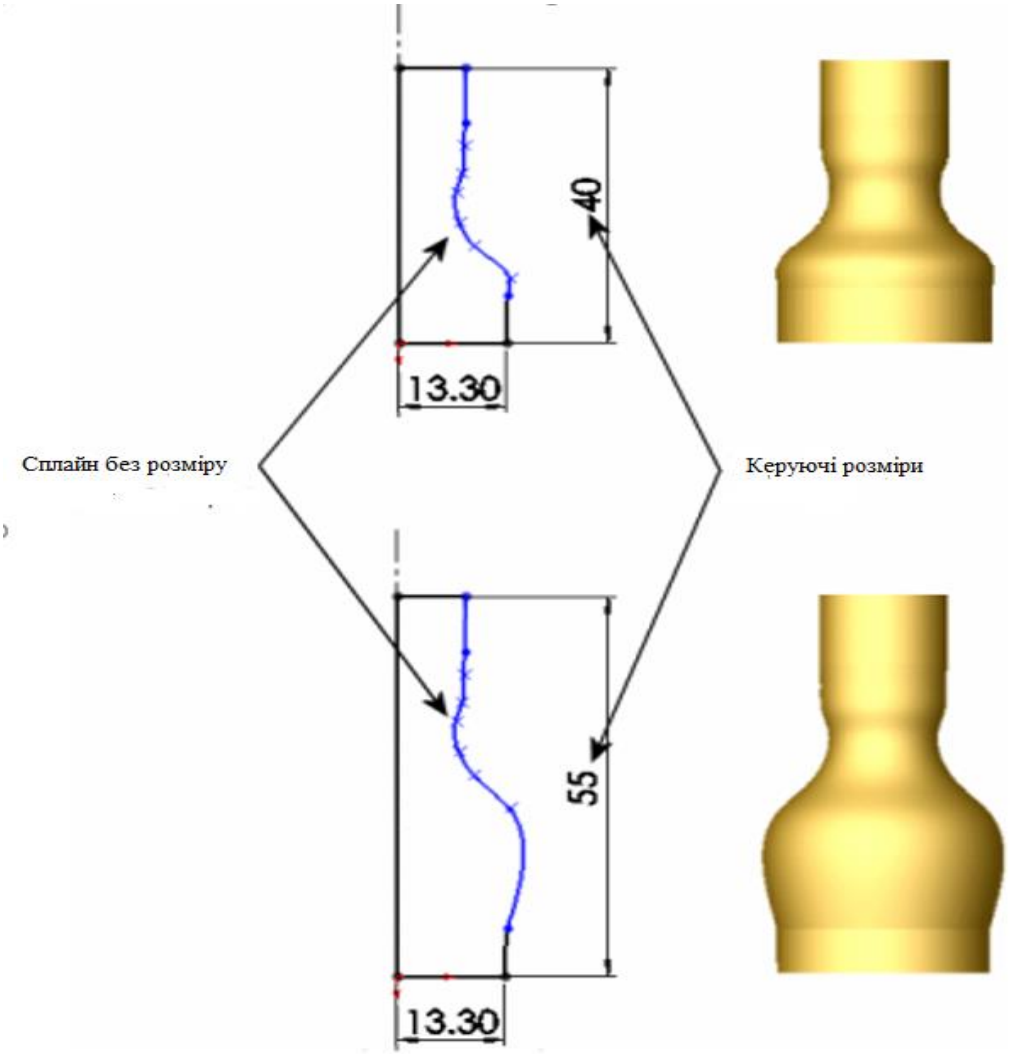
Стандартними площинами є *передня*, *верхня* та *права* орієнтації. Можна також додавати і позиціонувати площини в міру необхідності. У даному прикладі використовується верхня поверхня.



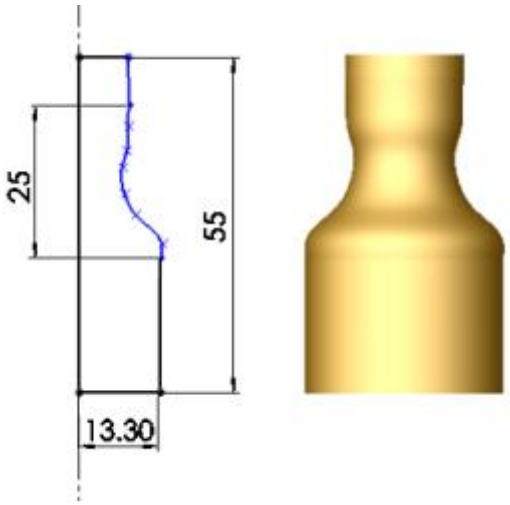
Додаткові відомості про площини отримати у розділі довідки « *Де розпочати створення ескіз*у».

Розміри

Можна вказати розміри між елементами, наприклад довжини та радіуси. При зміні розмірів змінюється розмір та форма деталі. У програмі використовується два типи розмірів: керуючі та керовані. Керуючі розміри створюються за допомогою інструменту *Розмір*. Керуючі розміри змінюють розмір моделі при зміні їх значень. Наприклад, у наступній деталі можна змінити висоту з 40 мм до 55 мм. Зверніть увагу на те, як змінюється форма поверненої деталі, так як на сплайн не нанесені розміри.



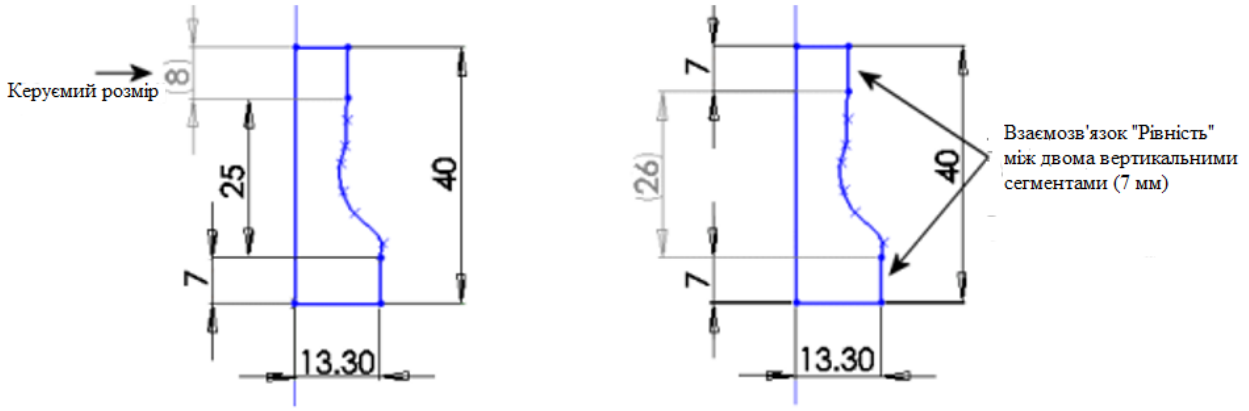
Щоб зберегти однакову форму, створену сплайном, необхідно вказати розмір сплайну.



Керовані розміри

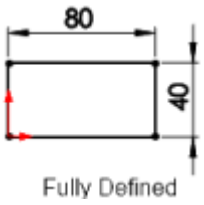
Деякі розміри, пов'язані з моделлю, є керованими. Керовані чи довідкові *Розміри* можна створювати в інформаційних цілях за допомогою інструменту *Розмір*. Значення керованих розмірів змінюється при зміні керуючих розмірів або взаємозв'язків у моделі. Змінити керовані розміри безпосередньо не можна,

якщо не перетворити їх у керуючі. Якщо, напиклад, вказана загальна висота 40 мм, вертикальна частина під сплайном - 7 мм, а сегмент сплайну 25 мм, вертикальний сегмент над сплайном обчислюється в 8 мм (як показано керованим розміром). Розміщення керуючих розмірів та взаємозв'язків у тому чи іншому місці дозволяє контролювати задум проекту. Наприклад, якщо вказати загальну висоту 40 мм та створити взаємозв'язок "рівність" між верхнім та нижнім вертикальними сегментами, верхній сегмент стане у розмірі 7 мм. Вертикальний розмір 25 мм конфліктує з іншими розмірами та взаємозв'язками (бо 40-7-7=26, а не 25). Зміна розміру 25 мм на керований усуває конфлікт та показує, що довжина сплайне повинна дорівнювати 26 мм.



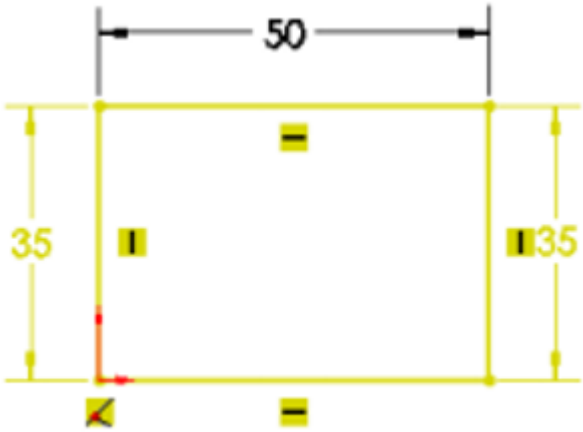
Визначення ескізів

Ескізи можуть бути повністю визначеними, недовизначеними чи перевизначеними. У повністю визначених ескізах всі лінії та криві , а також їх розташування описуються розмірами чи взаємозв'язками, або й тим і іншим одночасно. Не потрібно повністю визначати ескізи перед використанням до створення елементів . Однак слід повністю визначити ескізи підтримки задуму проекту. Повністю певні ескізи відображаються чорним кольором.



Шляхом відображення недовизначених об'єктів ескізу можна визначити, які розміри чи взаємозв'язки необхідно додати, щоб повністю визначити ескіз. Можно використовувати кольорові мітки, щоб визначити, чи є ескіз недовизначеним. Недовизначені ескізи відображаються синім кольором. Крім кольорових міток, об'єкти в недовизначених ескізах не зафіксовані і їх можна перетягувати.

Перевизначені ескізи містять надмірні розміри та взаємозв'язки, які знаходяться в конфлікті. Перевизначені розміри та взаємозв'язки можна видалити, але їх не можна змінити. Перевизначені ескізи відображаються жовтим кольором.

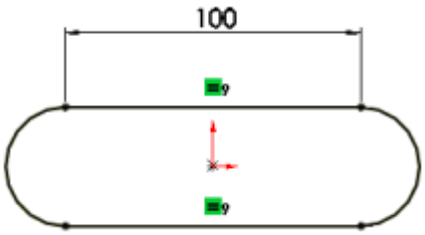


Перевизначено

Цей ескіз перевизначено, оскільки вказано розміри обох вертикальних ліній прямокутника. Прямокутник за визначенням має дві пари рівних сторін. Тому необхідний лише один розмір 35 мм.

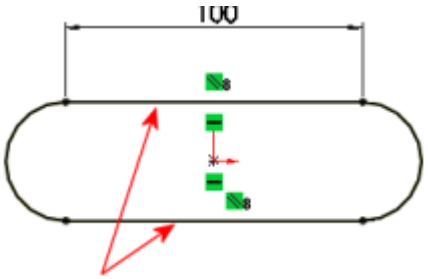
Взаємозв'язки

Взаємозв'язки встановлюють геометричні відносини, наприклад рівність або торкання, між об'єктами ескізу. Наприклад, можна встановити рівність між двома горизонтальними 100 мм лініями.



Можна окремо вказати розмір кожного горизонтального об'єкту, однак якщо встановити взаємозв'язок "рівність" між ними, то потрібно оновлювати лише один розмір при зміні довжини. Зелені символи вказують на відношення рівності між горизонтальними лініями.

Нижче показана концепція взаєзв’язків



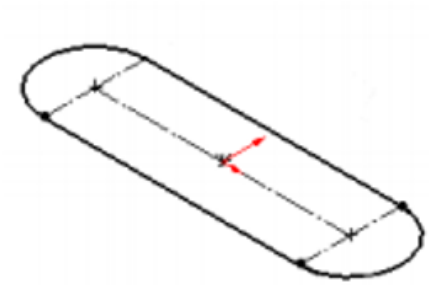
Можна використовувати інструмент *Додати взаємозв'язки*

Складність ескізу

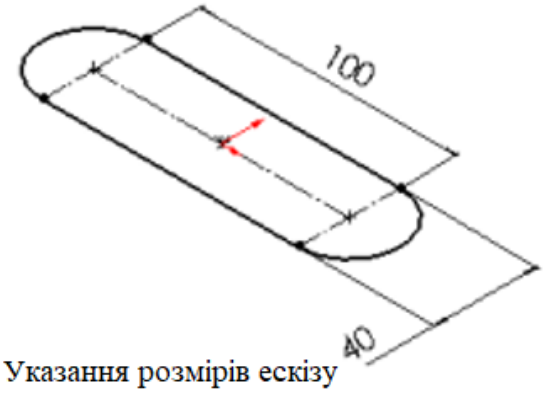
Простий ескіз простий у створенні та оновленні. Він швидше перебудовується. Одним із способів спрощення ескізу є застосування взаємозв'язків у міру його створення. Крім того, можна скористатися властивістю повторення та симетрії.

Елементи

Після створення ескізу можна створити тривимірну модель за допомогою, наприклад, елементу «Витягнути»

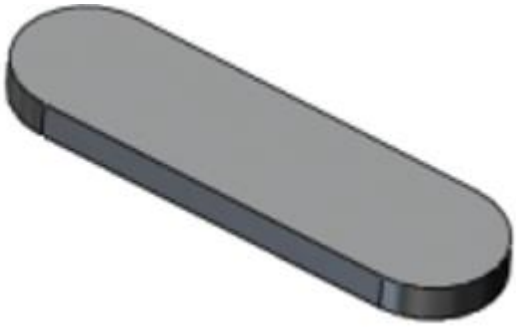


Створення ескізу



Указання розмірів ескізу

Далі витяжка ескізу на 10 мм:

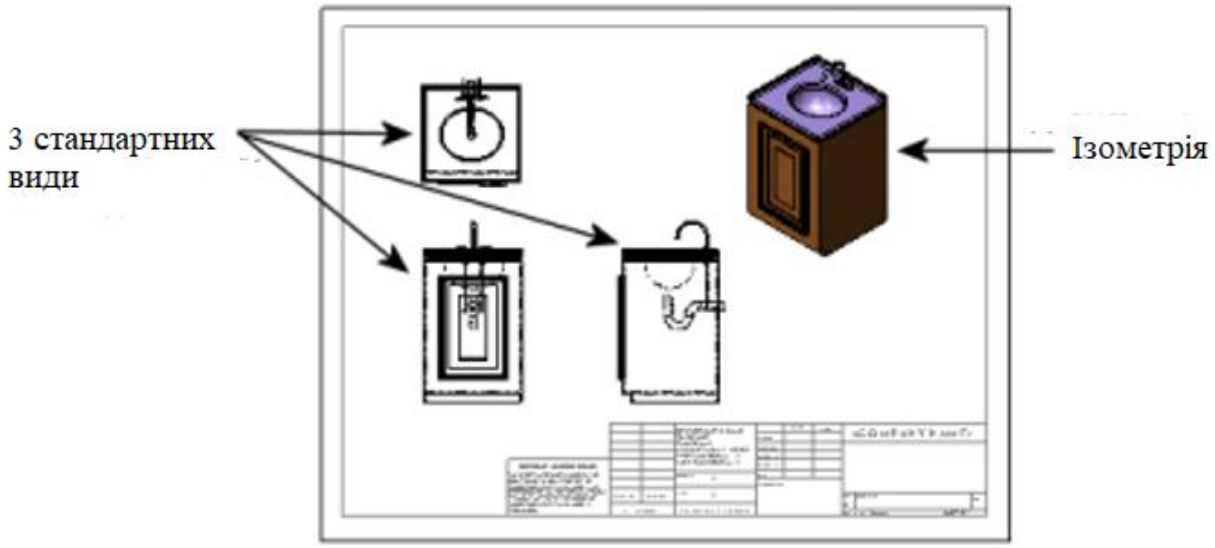


Складання (збірка)

Можна створити кілька деталей, які збираються у складання. Об'єднати деталі в складання можна за допомогою таких пар, як *Концентричність* або *Збіг*. Поєднання визначають дозволений напрямок руху компонентів.

Креслення

Креслення створюються з моделей деталі або складання. Креслення доступні в різних видах, наприклад у стандартних 3 видах та ізометричних (тривимірних). Можна імпортувати розміри з документу моделі та додавати примітки, наприклад, позначення бази.



Редагування моделі

Використовуючи дерево конструювання FeatureManager програми SOLIDWORKS та PropertyManager (Менеджер властивостей), можна редагувати ескізи, креслення, деталі та складання. Також можна редагувати елементи та ескізи, вибираючи їх безпосередньо з графічної області. Цей візуальний підхід усуває необхідність знати ім'я елементу. До можливостей редагування належать можливість вибрати ескіз у дереві конструювання FeatureManager та відредагувати його. Наприклад, можна редагувати об'єкти ескізу, змінювати розміри, переглядати чи видаляти існуючі взаємозв'язки, додавати нові взаємозв'язки між об'єктами ескізу, а також змінювати величину відображаємих розмірів. Також можна вибрати елемент для редагування безпосередньо з графічної області.

Редагування ескізу

Після створення елементу можна змінити більшість його значень. Функція *Редагування елементу* дозволяє відобразити відповідний PropertyManager. Наприклад, якщо для кромки використовується заокруглення з постійним радіусом, відображається вікно Заокруглення PropertyManager, в якому можна змінити радіус. Також можна відредагувати розміри двічі натиснувши на елемент або ескіз у графічній області, щоб відобразити розміри, а потім змінити їх на місці.

Редагування елементу

Для певної геометрії, наприклад, кількох тіл поверхонь в одній моделі, можна приховати або відобразити одне або кілька тіл поверхонь. Можна приховати або відобразити ескізи , площини та вісі у всіх документах, а також види, лінії та компоненти в кресленнях.

Приховати та відобразити

Можна вибрати будь-який елемент у дереві конструювання FeatureManager та погасити його, щоб переглянути модель без цього елемента. Коли елемент погашено, він тимчасово виключається з моделі (але не видаляється). Елемент зникає з поля зору моделі. Потім цей елемент можна висвітлити, щоб повернути модель до вихідного станом. Крім того, елементи можна погашати або висвічувати у збірках.

Погасити та висвітлити

При роботі з моделлю з кількома елементами можна виконати відкат дерева конструювання FeatureManager до попереднього стану. При переміщенні смуги відкату всі елементи моделі будуть відображатися в стані відкату, поки дерево конструювання FeatureManager не буде повернено до вихідного стану. Відкат корисний для вставлення елементів перед іншими елементами, прискорення часу перебудови моделі під час її редагування, а також визначення того, як модель була побудована.

Література

1. Вебсайт www.solidworks.com (http://www.solidworks.com)

2. Solidworks Tutorials [Електронний ресурс]. – Електрон. дані – Режим доступу: http://www.solidworks.com/sw/resources/solidworks-tutorials.htm. – Загол з екрану.