МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Опорний конспект лекцій з курсу **"Аналіз вимог до програмного забезпечення"**

для студентів напрямку підготовки "Програмна інженерія"

Тернопіль – 2011

Козак О.Л. Опорний конспект лекцій з курсу “Аналіз вимог до програмного забезпечення” для студентів напрямку підготовки “Програмна інженерія” / О.Л. Козак. – Тернопіль, 2011. – 56 с.

Анотація. У навчальному посібнику висвітлено теоретичні основи інженерії вимог до програмного забезпечення. Розглянуті теми актуальні для фахівців на шляху до створення працездатного і якісного програмного продукту. Викладений матеріал повинен сприяти формуванню висококваліфікованих фахівців у галузі програмного забезпечення.

Укладач: **Козак Олександра Леонідівна**, к.т.н., викладач кафедри комп’ютерних наук ТНЕУ.

Відповідальний за випуск: **Дивак Микола Петрович**, д.т.н., професор, завідувач кафедри комп’ютерних наук ТНЕУ

Затверджено на засіданні кафедри комп’ютерних наук ТНЕУ.
Протокол № 15 від 18.05.2011 р.

ЗМІСТ

ТЕМА 1. АНАЛІЗ ВИМОГ – ЕТАП РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Інженерія ПЗ

Програмне забезпечення (ПЗ) – набір комп'ютерних програм і пов'язана із ним документація та данні (ISO/IEC 12207).

Життєвий цикл ПЗ (ЖЦ) - неперервний процес з моменту прийняття рішення про створення ПЗ до вилучення його з експлуатації.

Основні процеси та етапи ЖЦ:

- Специфікація вимог − опис вимог до програми, які обов'язкові для виконання, опис того, що програма повинна виконувати.
- Проектування програми − опис того, як програма буде працювати.
- Кодування − вихідний код і файли конфігурації.
- Тестування − контроль відповідності програми вимогам.
- Документування − документація до програми.
- Супровід.

Мета розробки ПЗ – у відведений час і бюджет, розробити якісне ПЗ, таке, що задовольняє реальні потреби користувача.

На рис.1 приведені статистичні дані проектів в області інформаційних технологій, зібрані групою The Standish Group International, Inc. (так званий CHAOS Report). Дані відносяться до проектів в США.

Рис.1.

По термінології The Standish Group:

Успішними проектами вважаються ті, які виконані без порушення часових і бюджетних обмежень, з функціональністю, яка співпадає з планованою в специфікації.

До *проблемних проектів* відносять ті, які завершені, і справно функціонують, але при їх виконанні виникли перевищення часового графіка, бюджету, і в них реалізовані не всі заявлені в специфікації функції.

До *провальних проектів* відносять ті, які припинені до завершення, або ніколи не будуть завершені.

Як видно з діаграми, за 8 років існує прогрес в створенні успішніших проектів, проте, частка проблемних проектів залишається великою.

Основні причини успіхів і провалу проекту:

Причини провалів:

- Недостатньо вихідної інформації від клієнта (12,8%)
- Неповні вимоги і специфікації (12,3%)
- Зміни вимог і специфікацій (11,8%)

Причини успіхів:

- Підключення до розробки користувача (15,9%)
- Підтримка з сторони виконавчого керівництва (13,9%)
- Чітка постановка вимог (13,0%).

1.2. Рівні вимог.

Аналізуючи причини успіхів та провалів, робимо висновок про важливість визначення та управління вимогами при розробці проектів, тому інженерія (аналіз) вимог потребує додаткового розгляду.

Наведемо основні визначення якими користуються в процесі аналізу вимог при розробці програмних систем (ПС)

Вимоги до ПЗ(software requirement)

- це умови або можливості, необхідні користувачам для вирішення проблем або досягнення цілей;
- це умови або можливості, якими повинна володіти ПС або системні компоненти, щоб виконати контракт або задовільнити стандартам, специфікаціям або іншим формальним документам;
- задокументоване представлення умов або можливостей для пунктів 1-2 (IEEE Standart Glossary of Software Engineering Terminology,1990).

Вимоги до ПЗ – це специфікація того що повинно бути реалізовано. Це описи поведінки системи, властивості системи або її атрибути. Вони можуть бути обмеженні процесом розробки системи. (Sommervile,1997).

Вимоги до ПЗ складаються з 3-х рівнів:

бізнес-вимоги;

вимоги користувачів;

функціональні вимоги.

При цьому кожна система характеризується *нефункціональними вимогами*.

Модель на рис. 2 ілюструє спосіб представлення вимог, схематично показує організацію вимог. Овал – позначає тип інформації для вимог, прямокутник спосіб зберігання інформації (документи, діаграми, бази даних) [1].

Бізнес-вимоги(business requirements) містять високорівневі цілі організації або замовників ПЗ.

Як правило, їх висловлюють, ті хто фінансує проект, покупці системи, менеджери користувачів. В цьому документі пояснено для чого необхідна така система, тобто описано *цілі* які організація має намір досягнути. Часто бізнес вимоги записують в формі документа про образ і межі проекту, який називають *уставом проекту* (project charter) або *документом ринкових вимог* (market requirement document). Визначення меж проекту є першим етапом управління загальними проблемами «розповзання» меж.

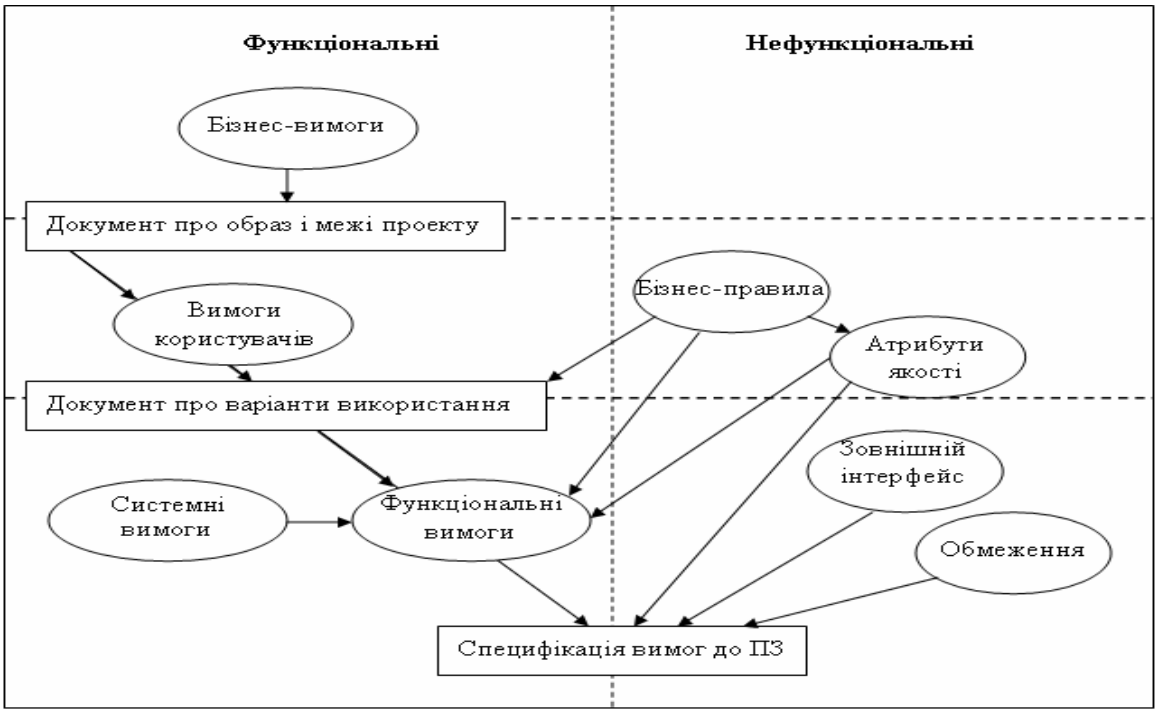


Рис.2. Взаємозв'язки кількох типів інформації для вимог

Вимоги користувачів (user requirements) описують цілі і задачі, які дозволить користувачу вирішувати дана система.

До способів представлення таких вимог відносяться варіанти використання, сценарії і таблиці «подія-відгук». В цьому документі вказано, що клієнти можуть робити з допомогою системи.

Функціональні вимоги (functional requirements) визначають функціональність ПЗ, яку розробники повинні забезпечити, щоб користувачі змогли виконати свої завдання в межах бізнес-вимог.

Інколи їх називають вимоги поведінки (behavioral requirements), вони містять положення з традиційним «повинна». Наприклад, «Система повинна по електронній пошті відправляти користувачу підтвердження замовлення». Функціональні вимоги описують, що розробнику необхідно реалізувати.

Терміном **системні вимоги** (system requirements) позначають високорівневі вимоги до ПС, які містять багато підсистем.

Бізнес-правила (business rules) включають корпоративну політику, постанови управління, промислові стандарти і чисельні алгоритми.

Бізнес-правила не є вимогами до ПЗ, тому що вони знаходяться зовні меж будь-якої системи ПЗ. Однак вони часто накладають обмеження, визначаючи хто може виконувати конкретні варіанти використання, або якими функціями повинна володіти система, яка підлягає відповідним правилам. Інколи бізнес-правила є джерелом атрибутів якості, які реалізуються в функціональності.

Функціональні вимоги документуються в **специфікації вимог до ПЗ** (software requirement specification, SRS), де описується очікувана поведінка системи.

Це може бути документ, база даних, таблиця з вимогами, сховище даних в комерційному інструменті управління вимогами, або навіть набір карток для невеликого проекту. Специфікація вимог до ПЗ використовується при розробці,

тестуванні, гарантії якості продукту, управлінні проектом і зв'язаних з проектом функціях. Окрім функціональних в специфікації містяться нефунціональні вимоги, де описані цілі і атрибути якості.

Атрибути якості (quality attributes) є додатковим описом функцій продукту, опис характеристик важливих для користувачів або розробників.

До таких характеристик відносяться легкість і простота використання, простота переміщення, цілісність, ефективність і стійкість до збоїв. Інші не функціональні вимоги описують зовнішні взаємодії системи і середовища, а також обмеження дизайну і реалізації.

Обмеження(constraints) стосуються вибору можливості розробки зовнішнього вигляду і структури продукту.

Характеристика (feature) − це набір логічно зв'язаних функціональних вимог, які забезпечують можливості для користувача і відповідають бізнес-цілям.

В області комерційного ПЗ характеристика є групою вимог, яку вирізняють всі зацікавленні особи, які важливі при прийняті рішення про покупку. Характеристики продукту, які перераховує клієнт, не еквівалентні тим які необхідні для вирішення задач користувача.

Хоча в моделі на рис.1 потік вимог показано в напрямку зверху вниз, проте слід очікувати і циклів, і ітерацій між бізнес-вимогами, вимогами користувачів та функціональними вимогами.

1.3. Розробка і управління вимогами

Деякі автори називають цілу область розробки технічних умов (requirements engineering) Аналіз вимог (АВ) або інженерія вимог (рис. 3) включає в себе етап *розробки вимог* (requirements development) та етап *управління вимогами* (requirements management).

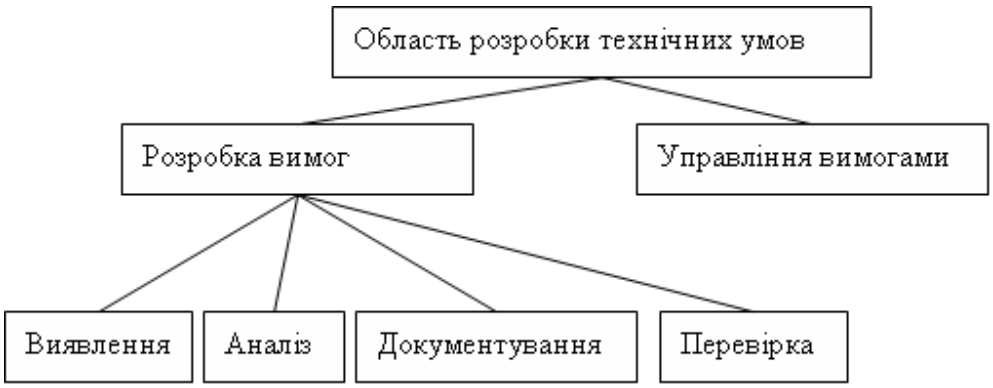


Рис.3 Компоненти інженерії вимог

В свою чергу *Розробка вимог* включає етап *виявлення* (elicitation); *аналізу* (analysis); *документування* (specification) та *перевірки* (validation).

Розробка вимог − це:

- ідентифікація класів користувачів для даного продукту;
- з'ясування потреб тих, хто представляє кожен клас користувачів;
- визначення завдань і цілей користувачів, а також бизнес-цілей, з якими пов'язані завдання ;

- аналіз інформації, отриманої від користувачів, щоб відокремити завдання від функціональних і нефункціональних вимог, бізнес-правил, передбачуваних рішень і зовнішніх даних;
- розподіл високорівневих вимог по компонентах ПЗ, визначеним в системній архітектурі;
- встановлення відносної важливості атрибутів якості;
- встановлення пріоритетів реалізації;
- документування зібраної інформації і побудова моделей;
- перегляд специфікації вимог, який дозволяє упевнитися в тому, що запити користувачів всіма розуміються однаково, і усунення проблем до передачі документа розробникам.

Поступовість процесу – ключ до успіху розробки вимог. Плануйте циклічність досліджень, деталізуйте високо рівневі вимоги і уточнюйте коректність у користувачів.

Управління вимогами – набуття і підтримка взаємозгоди з замовниками про вимоги до створюваного ПЗ. До дій по управлінню вимогами відносяться:

- визначення основної версії вимог (миттєвий зріз вимог для конкретної версії продукту);
- перегляд пропонованих змін вимог і оцінка вірогідності дії кожної зміни до її ухвалення;
- включення схвалених змін вимог в проект встановленим способом;
- узгодження плану проекту з вимогами;
- обговорення нових зобов'язань, заснованих на оціненому впливі зміни вимог;
- спостереження за окремими вимогами до їх дизайну, початкового коду і варіантів тестування;
- спостереження статусу вимог і дій при зміні впродовж всього проекту.

Основні наслідки проблем з вимогами − переробка того, що на вашу думку, вже готово. На це витрачають від 30 до 50% загального бюджету розробки, помилки в вимогах вартують від 70 до 85% вартості переробки.

Ризики при розробці ПЗ:

- недостатнє залучення користувачів;
- «розростання» вимог користувачів;
- двозначність вимог;
- «позолочення» продукту;
- мінімальна специфікація;
- пропуск класів користувачів;
- недбале планування.

Переваги високоякісного процесу розробки вимог

- менше дефектів у вимогах;
- менше переробок;
- менше непотрібних функцій;
- нижча вартість модифікації;
- швидша розробка;
- менше розповзання меж;
- менше безладу в проекті;

- точніші оцінки тестування;
- задоволення замовників і розробників.

1.4. Характеристики хороших вимог

Найкращий спосіб визначити чи вимоги мають бажані атрибути – попросити кількох зацікавлених в проекті осіб уважно переглянути специфікацію. Вони можуть виявити різні типи проблем. Наприклад, аналітики і розробники не можуть точно визначити повноту або коректність документу, тоді як користувачам не вдасться оцінити технічні характеристики.

Характеристики окремих положень специфікації вимог:

Повнота. Кожна вимога повинна повно описувати функціональність, яку слід реалізувати в продукті. Тобто вона повина містити, всю інформацію, необхідну для розробників, щоб їм вдалося створити цей фрагмент функціональності. Якщо ви розумієте, що даних певного роду не вистачає, використовуйте позначку «TBD» (to be determined - необхідно доозначити) на полях як стандартний прапор для виділення такого місця. Заповните всі пропуски в кожному фрагменті вимог, перш ніж приступати до конструювання цієї функції.

Коректність.Кожна вимога повинна точно описувати бажану функціональність. Для дотримання коректності необхідний зв'язок з джерелами вимог, наприклад з побажаннями користувачів або високорівневими системними. Вимоги до ПЗ, які конфліктують з «батьківськими» вимогами, не можна вважати коректними. Проте основна оцінка - за представниками користувачів, от чому їм необхідно надавати вимоги для перегляду.

Здійсненність. Необхідна можливість реалізувати кожну вимогу за відомих умов і обмежень системи і операційнго середовища. Щоб не придумувати недосяжні положення, забезпечте взаємодію розробників з маркетологами і аналітиками вимозі на період всього витягання вимог. Розробники реально оцінять, що можна виконати технічно, а що немає, або що зробити можна, але при додатковому фінансуванні.

Необхідність. Кожна вимога повинна відображати можливість, яка дійсно необхідна користувачам або яка потрібна для відповідності зовнішнім системним вимогам або стандартам. Відстежте кожну вимогу аж до стадії збору думок користувачів, коли виявлялися варіанти використання, бізнес-правила або інші джерела.

Визначення пріоритетів. Визначте пріоритети кожної функціональної вимоги, характеристики або варіанту використання, щоб визначити, що необхідне для кожної версії продукту. Якщо всі положення однаково важливі, менеджерові проекту буде важко справитися із зменшенням бюджету, порушенням термінів, втратою персоналу або додаванням нових вимог в процесі розробки.

Недвозначність. Всі читачі вимог повинні інтерпретувати їх однаково. Пишіть документацію просто, коротко і точно, застосовуючи лексику, зрозумілу користувачам. «ясність» — мета якості вимог, пов’язана з точністю. Занесіть всі спеціальні і заплутані терміни в словник.

Можливість перевірки. Спробуйте розробити декілька тестів або застосуйте інші прийоми для перевірки, наприклад експертизу або демонстрації, щоб встановити, чи дійсно в продукті реалізована кожна вимога.

Характеристики специфікації вимог

Недостатньо отримати прекрасні окремі положення. Набір вимог, що становить специфікацію повинен відповідати характеристикам.

Повнота. Ніякі вимоги або необхідні дані не повинні бути пропущені.

Узгодженість. Узгоджені вимоги не конфліктують з іншими вимогами такого ж типу або з високорівневими призначеними для користувача, системними або бізнес-вимогами. Неузгодженість документів слід усунути до початку процесу розробки. Ви не завжди знаєте, яке саме положення некоректне (якщо якесь некоректне), поки не виконаєте дослідження. Рекомендується записувати автора кожної вимоги, щоб дізнатися, хто її висловив, якщо конфлікт все-таки буде виявлений.

Здатність до модифікації. Необхідно забезпечити можливість переробки вимог, якщо знадобиться, і підтримувати історію змін для кожного положення. Для цього всі вони повинні бути унікально помічені і позначені, щоб ви могли посилатися на них однозначно. Кожна вимога повинна бути записана в специфікації тільки один раз, інакше ви отримаєте неузгодженість, змінивши тільки одне положення з двох однакових. Краще використовуйте посилання на первинні твердження, а не дублюйте положення. Модифікація специфікації стане набагато легшою, якщо ви складете зміст документа і покажчик. Збереження специфікації в базі даних комерційного інструменту управління вимогами зробить їх придатними для повторного використання.

Трасованість. Трасованість, або можливість для аналізу, можна реалізувати як в напрямі назад, до першоджерел, так і вперед, до елементів дизайну і початкового коду, який його реалізує, а також до варіантів використання, які дозволяють перевірити коректність реалізації. Трасовані вимоги помічають відповідними ідентифікаторами. Вони записані в структурованій, деталізованій формі, в протилежність параграфам в довгій оповідній формі. Уникайте злипання безлічі вимог в одну, окремі вимоги можна трасувати до різних елементів дизайну і коду.

ТЕМА 2. ВИЯВЛЕННЯ ВИМОГ

2.1. Джерела вимог.

Основним джерелом вимог до інформаційної системи, безумовно, є міркування, висловлені представниками Замовника. У відповідність з ієрархічною моделю вимог дана інформація структурується як мінімум на 2 рівні: *бізнес-вимоги* і *вимоги користувачів*.

Проблема полягає в тому, що вимоги формулюються до створюваної, ще не існуючої системи, тобто по суті вирішується початкова підзадача завдання проектування ПС, а представники Замовника далеко не завжди бувають компетентні в даному питанні. Тому, разом з вимогами, висловленими Замовником, доцільно збирати і вимоги від інших співвласників системи: співробітників аналітичної групи виконавця, зовнішніх експертів і т. д.

Результуючий, часто достатньо сирий матеріал розглядається, як документ *«Вимоги співвласників»*. На вимоги співвласників зазвичай не накладається ніяких спеціальних обмежень.

Іншим важливим джерелом інформації, окрім виявлення вимог, є *артефакти*, що описують наочну область. Це можуть бути документи з описом бізнес-процесів підприємства, виконані консалтинговим агентством, або просто документи (посадові інструкції, розпорядження, зведення бізнес-правил) прийняті на підприємстві. Однією з небагатьох методологій, в якій спеціально виділяється робочий потік ділового моделювання, є Rational Unified Process (RUP).

Ще однією альтернативною методологією, яку використовують при виявленні вимог, є так звані «*кращі практики*» (best practices), які дуже популярні в даний час в бізнес-консалтінгу і при впровадженні корпоративних інформаційних систем. Кращими практиками є описи моделей діяльності успішних компаній галузі, які використовувались тривалий час в сотнях і тисячах компаній по всьому світу.

Відзначимо, що основними джерелами, процесу виявлення вимог, є вимоги, висловлені співвласниками або (і) артефакти, що описують об'єкт дослідження. Проте, це достатньо спрощений погляд: щоб дані поступили «на вхід», аналітики вимог повинні виконати чималу роботу, пов'язану з підбором респондентів і інформаційних матеріалів, організацією інтерв'ю і т. д.

2.2. Стратегії виявлення вимог

Розглянемо популярні стратегії виявленння вимог детальніше.

Інтерв'ю

Ключовою стратегією виявлення вимог було і залишається інтерв'ю з експертами.

У класичній монографії Д.Марко [2] в процесі проведення інтерв'ю пропонується виділити три складові процесу: *підготовку*, *проведення інтерв'ю* і *завершення*. Нижче приводиться короткий огляд рекомендацій Д.Марко з акцентом на виявлення вимог (у монографії дані рекомендації по інтерв'юванню з метою формування моделі об'єкту дослідження).

Підготовка

Підготовка дозволяє спланувати процес опитування і виробити стратегію управління цим процесом. Важливість підготовчого етапу зростає, якщо респондент є «дефіцитним» корисним ресурсом, наприклад – президентом великої компанії.

При підготовці Д.Марко рекомендує наступні кроки:

- виберіть потрібного співбесідника;
- домовтеся про зустріч;
- встановіть попередню програму зустрічі;
- вивчіть супутню інформацію;
- погоджуйте свої дії з групою проектувальників.

При виборі співбесідника для цілей збору вимог визначальними є дві речі:

- він дійсно є експертом з даного питання;
- його думка дійсно є цінною при формуванні цільового набору вимог

На практиці можливі ситуації, коли вимога, сформульована одним з представників Замовника, не підтверджується іншим представником, що має великі владні повноваження. Треба чітко розуміти, що кожна вимога зрештою транслюється, з одного боку, в компоненту інформаційної системи, а з іншого – може бути виражена в певній кількості грошових знаків, які Замовник повинен буде виплатити Виконавцеві при прийманні роботи. Тому право формулювати вимоги і область компетенції того або іншого експерта повинні бути формально обумовлені внутрішнім документом Замовника, з яким слідує ознайомиться до початку проведення інтерв'ю.

Важливо заздалегідь обговорити мету зустрічі і обмежити бесіду в межах години або менше. Практика показує, що активне спілкування в процесі інтерв'ю, як правило, обмежується годиною. Якщо цього часу недостатньо, можна спланувати декілька зустрічей.

Корисними прийомами є формування програми бесіди і ознайомлення з нею респондента, докладне планування бесіди аж до запису підготовлених питань. Підготовлене таким чином інтерв'ю називають *структурованим* [3]. На додаток до такого побудованого інтерв'ю автор [3] пропонує проводити *неструктуроване інтерв'ю,* що «є неформальною зустріччю, якій не властиві заготовлені про запас питання або заздалегідь поставлені цілі». Мета такого інтерв'ю – спонукати співрозмовника до творчості в області, в якій опитувач недостатньо добре орієнтується.

Проведення інтерв'ю

У проведенні опитування найважливіше – правильно організувати і підтримувати потік інформації від експерта до вас. Рекомендується витратити час на обдумування вірного початку інтерв'ю, при зборі інформації по можливості використовувати записи, закінчувати розмову плавно. Обговоримо докладніше кожен з цих пунктів.

Починаючи розмову, не забудьте представитися і сформулювати мету зустрічі. Це допоможе уникнути непорозумінь і дасть бесіді правильний напрям. Крім того, обговоріть можливість ведення записів.

Збирайте інформацію, роблячи записи про все (про спеціальні терміни, взаємозв'язки між частинами системи і т.д.) і обмежуючи час бесіди. Запишіть

функції і дані, спробуйте «накидати» діаграму. Ставте запитання, які уточнюють і підтверджують відповіді.

Перш за все, не заперечуйте.

Ніколи не ставте навідних питань або питань з короткими відповідями "та чи ні". Замість цього записуйте те, що вам говорять, і просіть підвести підсумок або дати пояснення.

Ви отримаєте від інтерв'ю більше, якщо ви дасте експертові можливість говорити те, що він хоче сказати, а не те що ви хочете почути [2].

Завершення

Стежте за виникненням наступних ситуацій:

- ви вже отримали достатньо інформації;
- ви отримуєте великий об'єм невідповідної інформації;
- велика кількість інформації вас пригнічує;
- експерт починає втомлюватися;
- у вас з експертом часто виникають конфлікти.

Будь-яка з цих причин − достатня підстава для завершення бесіди. Коли ви вважаєте потрібними закінчити опитування, завершуйте бесіду плавно. Коротко підсумуйте основні пункти і зробіть огляд отриманих відомостей, які можуть бути пропущені або невірно тлумачитись. Домовтеся про час наступної зустрічі, якщо вона потрібна, і отримаєте рекомендації для найближчих опитувань. Поінформуйте експерта, коли і як ви збираєтеся використовувати отриману інформацію і коли ви пришлете йому матеріал на рецензування.

Завжди оформляйте матеріали опитування одразу ж після зустрічі з експертом. В цьому випадку негайно виникає зворотний зв'язок, і ви мінімізуєте можливість втрати важливої інформації.

Що потрібно пам'ятати при опитуванні

Наступні рекомендації допомагають підтримувати безперервність потоку і достовірність інформації, що поступає від експерта:

- робіть паузи, поки експерт думає. Дайте експертові можливість вирішувати, що сказати далі. Ніколи не перебивайте, підказуючи відповідь або ставлячи інше питання.
- Прагніть не ставити навідних питань, питань-підказок, питань, що містять відповідь, тому що це не дозволяє експертові ділитися своїми знаннями. Прагніть не ставити контрольних питань, оскільки це перериває потік інформації.
- Робіть записи, щоб зосередитися на предметі розмови і щоб підготуватися до наступного питання, але не стаєте стенографом, інакше ви можете втратити контроль над інтерв'ю.

Анкетування

Анкетування – самий маловитратний для аналітика спосіб отримання інформації, він же – і найменш ефективний. Зазвичай застосовується як доповнення до інших стратегій виявлення вимог.

Недоліки анкетування очевидні: респонденти часто бувають нездібні, або слабо мотивовані в тому, щоб добре і інформативно заповнити анкету. Велика вірогідність отримати неповну або зовсім помилкову інформацію. Перевага – в тому, що підготовка і аналіз анкет вимагають невеликий ресурс.

Л.Мацяшек [3] рекомендує формулювати в анкетах *питання із замкнутим циклом відповідей* в одній з наступних трьох форм.

Багатоальтернативні питання. Ця форма анкети відома всім хто проходив тестування, може розширюватися коментарями респондента у вільній формі.

Рейтингові питання. Представляють зумовлений набір відповідей на сформульовані питання. Використовуються такі значення, як «абсолютно згоден», «згоден», «відношуся нейтрально», «не згоден», «абсолютно не згоден», «не знаю».

Питання з *ранжируванням*. Передбачає ранжирування (впорядковування) відповідей шляхом привласнення ним порядкових номерів, процентних значень і тому подібне

Спостереження

Спостереження за роботою модельованої організаційної системи - корисна стратегія отримання інформації (хоча за наслідками спостереження можна отримати модель системи, а не модель аналізу вимог).

Розрізняють *пасивне* і *активне* спостереження. При активному спостереженні аналітик працює, як учасник команди, що дозволяє поліпшити розуміння процесів.

Через спостереження, а можливо, і участь аналітики отримують інформацію про операції, що відбувається день за днем, з перших рук. Під час спостереження за роботою системи часто виникають питання, які ніколи б не з'явилися, якби аналітик тільки читав документи або розмовляв з експертами.

Недоліком цієї стратегії є те, що спостерігач, як і всякий «вимірювальний прилад», вносить перешкоди до результатів вимірювань: співробітники організації, знаходячись «під ковпаком» можуть почати поводитися принципово по іншому, чим зазвичай.

Самостійний опис вимог

Документи – хороше джерело інформації, тому що вони найчастіше доступні і їх можна "опитувати" в зручному для себе темпі. Читання документів – прекрасний спосіб отримати первинне уявлення про систему і сформулювати питання до експертів.

Якщо досвідчений аналітик вже дослідив велике число систем такого ж типу, що і на підприємстві для якого впроваджується система, він володіє фундаментальними знаннями у відповідній наочній області, щодо певного класу систем. Автори методології SADT рекомендують проводити *самоопитування* з тим, щоб отримати максимальну користь від своїх знань.

За наслідками аналізу документів і власних знань аналітик може скласти *опис* вимог і запропонувати його представникам Замовника як інформація до роздуму, або основа для формування технічного завдання.

Недолік цієї стратегії – небезпека упущення інформації, специфічних для об'єкту дослідження (у разі самоопитування), або – неформалізованих знань, емпіричних правил і процедур, широко використовуваних на практиці, але що не увійшли до документів.

Сумісні семінари – JAD – метод (Joint Application Development)

Окрім класичного інтерв'ю «тет–а-тет», існує значна кількість методик, що припускають широку участь представників Замовника і Виконавця. JAD підхід відносять до сучасних методів виявлення вимог, проте вперше він був введений в кінці 1970-х компанією IBM. Одна з таких методик проведення сумісних семінарів - використання прийомів мозкового штурму.

Правила *мозкового штурму* припускають повну свободу думок, навіть найхимерніших і на перший погляд «маревних». Перше правило мозкового штурму – «повна заборона на будь-яку критику». Всяка висловлена думка представляє цінність, а повна відсутність заборон дозволяє повноцінним чином підключити творчу фантазію. На другому етапі, всі висловлені думки ретельно обговорюються, свідомо неприйнятні варіанти відсіваються, формуються колективні пропозиції.

Учасники JAD-наради:

Ведучий – фахівець в області міжособових комунікацій. Повинен орієнтуватися в наочній області, але не обов'язково добре орієнтуватися в проблемах IT.

Секретар – стенографіст зустрічі. Фіксує її результати на комп'ютері. Можливе застосування CASE-засобів.

Замовники – користувачі або керівники, основні учасники, що формують, обговорюють вимоги і приймаючі рішення.

Розробники – аналітики і інші учасники проектної команди. Працюють в більшій частині в пасивному режимі з метою якнайкращого розуміння проблемної області.

JAD метод базується на груповій динаміці. Групові зусилия більш перспективні з точки зору отримання кращих вирішень проблем.

Сумісні семінари, зберігаючи всі переваги режиму інтерв'ю, привносять додаткові бонуси: робота в групі продуктивніша, групи швидше навчаються, більш схильні до кваліфікованих висновків, дозволяють виключити багато помилок.

Ця стратегія, очевидно, одна з самих витратних, проте вона окуповується за рахунок меншої кількості помилок і відмови від формалізації на користь живого спілкування, вироблення спільної мови і ін. Деякі методології ґрунтуються на постійному тісному контакті між Замовником і Виконавцем і, якщо такої можливості немає – проект просто не зможе відбутися.

“*Роз'яснюючі зустрічі”* [4] або «запланований мозковий штурм» – термін, що прийшов із загальної практики менеджменту і базується на ідеях співпраці зацікавлених осіб для сумісного аналізу шляхів вирішення проблем, визначення і попередження ризиків і т.д.

Прототипування

Прототипування – ключова стратегія виявлення вимог в більшості сучасних методологій. Програмний прототип – «дзеркало», в якому видно віддзеркалення того, як зрозумів Виконавець вимоги Замовника. Процес виявлення вимог шляхом прототипирования тим більше інтенсивний, чим це дзеркало кривіше. Документальний спосіб виявлення вимог завжди поступається живому

спілкуванню. Аналіз того, що зроблене у вигляді інтерфейсів користувача дає ще більший ефект.

Метод *RAD* – один з найбільш відомих способів швидко створювати прототипи.

RAD базується на наступних базових принципах:

- Еволюційне прототипирование;
- CASE-средства, як основний інструмент, включаючи можливості прямого і зворотного проектування і автоматичної генерації коди;
- Висококваліфіковані фахівці, що добре володіють розвиненими інструментальними засобами;
- Інтерактивний JAD-метод, в якому спілкування поєднується з розробкою в режимі online;
- Жорсткі часові рамки, як протиотрута від «розповзання меж» проекту: якщо команда не укладається в строк – функціонал звужується.

Використання RAD підходу може виявитися привабливим варіантом для багатьох проектів, особливо, для невеликих проектів, які не зачіпають сферу ключових бізнес процесів організації, і які, таким чином, не задають план рішення для інших проектів по розробці ПО. Маловірогідно, щоб швидкі рішення були оптимальними або довготривалими для ключових сфер діяльності організації. З використанням RAD методу зв'язаний ряд проблем: несумісний проект GUI інтерфейсу; замість загальних рішень, сприяючих багатократному використанню ПЗ, спеціалізовані рішення; неповна документація; важке для підтримки і масштабування ПЗ.

ТЕМА 3. КЛАСИФІКАЦІЯ І СПЕЦИФІКАЦІЯ ВИМОГ

3.1.Способи представлення вимог

- Документація, в якій використовується чітко структурована і акуратно використовувана природна мова.
- Графічні моделі, що ілюструють процеси трансформації стану системи і їх зміни, взаємодії даних, атак же логічні потоки, класи об'єктів і відношення між ними.
- Формальні специфікації, де вимоги визначені за допомогою математично точних, формальних логічних мов.

Актори і варіанти використання

Результатом виявлення вимог є реєстр вимог. Вимоги співвласників зазвичай оформляються в простій письмовій формі, без будь-якої особливої регламентації. Типовий приклад оформлення вимоги до програми електронної пошти – «Система повинна дозволяти набирати текст повідомлення з можливістю форматування тексту і вставки смайликів». Дані вимоги далеко не у всьому можуть задовольняти критеріям, сформульованим в лекції Властивості вимог вони можуть суперечити один одному, бути неясними, неточними і так далі. Проте, документ «Вимоги співвласників», не дивлячись на невисокий рівень формалізації, грає дуже важливу роль, тут зібрані думки всіх зацікавлених сторін і головна мета збору початкових вимог полягала в тому, щоб отримати по можливості якомога повніший набір вимог, не пропустивши чогось важливого.

Для того, щоб підвищити рівень інформативності вимог, усунути взаємні суперечності і добитися виконання їх інших основних характеристик, здійснюється перехід від повністю неформалізованих текстів до частково регламентованих (наприклад, шаблонами MS Word) текстів, класифікація, привласнення наборів атрибутів, побудова моделей, прототипування.

Найпопулярнішим і вельми ефективним способом підвищення інформативності вимог є оформлення їх у вигляді варіантів використання (use case), запропонований І.Якобсоном.

Перш, ніж приступити власне до специфікації вимог у формі варіантів використання, RUP рекомендує виявити реєстр акторів (actors) і варіантів використання.

Актор – це хтось або щось, що є активним по відношенню до ПС. Якщо ви розробляєте простий текстовий редактор, то, швидше за все, вибір актора не складе особливих труднощів: це буде користувач, що набирає текст. Проте не завжди все так просто. Окрім користувача як актор може розглядатися інша ПС, апаратний пристрій, у багатьох випадках – активна компонента самої системи. Пошук акторів корпоративної інформаційної системи зазвичай зводиться до аналізу ролей різних користувачів. Менеджер з продажу, старший менеджер і начальник відділу продаж – один актор, два або три? Це залежить від їх функціональних обов'язків, розмежування доступу, способів використання інформаційної системи. Пошук акторів може здійснюватися, наприклад методом мозкового штурму. Надалі при необхідності знайдені актори можуть узагальнюватися, переглядатися і об'єднуватися.

Варіант використання в першому наближенні можна розглядати, просто, як функцію, що реалізовується системою. Проте, сучасний погляд на організацію бізнесу говорить про те, що всяка функція повинна мати цінність для кінцевого споживача продукту або послуги. Філософія варіанту використання припускає виділення серед всіх функцій системи підмножини, корисної конкретному кінцевому користувачеві (точніше кажучи, типу кінцевого користувача). Інша сторона – варіант використання повинен не тільки бути корисний, а ще і дозволяти отримувати конкретні закінчені результати. Так, однією з функцій текстового редактора, очевидно, є створення порожнього файлу. Але навряд користувач використовуватиме редактор з метою виготовлення порожніх файлів. Отже, створення порожнього файлу – функція, але не варіант використання системи. Варіантом використання може бути, наприклад, підготовка в текстовому редакторові службової записки. Варіант використання реалізується через функції системи.

3.2. Глосарій

Окрім формування вимог співвласників іншим результатом початкової фази виявлення вимог є концептуальний аналіз проблемної області. Найпершим результатом його є формування глосарію (словника) основних використовуваних термінів. Значення глосарію важко переоцінити: він є основою, ключем для одноманітного розуміння описів вимог Замовником і Розробником.

Крім того, глосарій є відправною крапкою для побудови більш розгорнених моделей проблемної області, які, на стадії реалізації інформаційної системи, лягають в основу об'єктної моделі (для об'єктно-орієнтованих застосувань) і моделі даних (для генерації схеми бази даних).

Глосарій оформляється, як текст, що складається з абзаців, кожен з яких визначає значення одне з термінів проблемної області. Термін зазвичай виділяють напівжирним кеглем. Іноді проблемну область доцільно сегментувати на ряд «підобластей» (subject areas). Тоді кожній з них в глосарії виділяється окремий параграф.

3.3.Специфікація варіанту використання

Існують різні шаблони опису варіантів використання. Розглядаються наступні основні стилі опису:

- Вільний формат
- Повний формат (запропонований А. Коберном)
- Таблиця в дві колонки
- Таблиця в три колонки
- Стиль RUP.

Крім того, іноді доцільно використовувати:

- Псевдокод
- Діаграму активності UML
- Інші графічні моделі.

Вільний формат

Вільний формат припускає опис дій користувача і системи в описовому стилі, наприклад: «Менеджер запрошує у ПС список замовлень за період. Система

відображає на екрані знайдені замовлення даного Менеджера з вказівкою їх основних атрибутів». Вільний стиль допускає використання конструкцій «**Якщо то**». «**Якщо** Менеджер має повноваження Начальника Відділу, **то** Система надає можливість проглядання замовлень всіх менеджерів цього відділу».

Шаблон повного опису варіанту використання по А. Коберну

Назва <коротка фраза у вигляді дієслова в невизначеній формі доконаного виду, що відображає мету>

Контекст використання <уточнення мети, при необхідності – умови її нормального завершення>.

Зона дії <посилання на рамки проекту>. Наприклад – підсистема бухгалтерського обліку.

Рівень <один з трьох: узагальнений, цілі користувача, підфункції>. Автор задає зумовлену трирівневу класифікацію вимог, в цілому відповідну класифікації вимог на бізнес-вимоги, вимоги користувачів і функціональні вимоги.

Основна дійова особа <ім'я ролі основного актора або його опис>.

Учасники і інтереси <список інших акторів-учасників прецеденту з вказівкою їх інтересів>.

Передумова <те, що очікується, вже має місце>.

Мінімальні гарантії <що гарантується акторам-учасникам>. Наприклад – у разі невдалої транзакції всі дані, що були в системі до її початку, зберігаються незмінними.

Гарантії успіху <що отримають актори-учасники у разі успішного досягнення мети>.

Тригер <те, що «запускає» варіант використання, зазвичай – подія в часі>.

Основний сценарій <тут перераховуються кроки основного сценарію, починаючи від трігера і аж до досягнення гарантії успіху>.

Формат опису: <Номер кроку> <Опис дії>

Розширення <тут послідовно описуються всі альтернативні сценарії>. Кожна з альтернатив прив'язана до кроку основного сценарію.

Формат опису:

*<Номер кроку**.**Номер розширення> <Умова>***:***<Дія або посилання на підлеглий варіант використання>.*

Будь-який з кроків основного сценарію може мати 1 або більше розгалужень. Кожне розгалуження оформляється у вигляді розширення. У блоці «Розширення» всі розширення описуються послідовно.

У випадку, якщо альтернативний сценарій не вдається описати одним рядком – застосовується наступний формат.

Починаючи з рядка, наступного після опису розширення, йде опис його дій у форматі основного сценарію:

<Номер кроку.Номер розширення.Номер кроку розширення> <Дія>

Опис розширення закінчується описом виходу з розширення. Основні варіанти виходу з розширення: повернення до чергового по номеру кроку основного сценарію, закінчення прецеденту, перехід до іншого кроку основного сценарію.

Список змін в технології і даних <що гарантується акторам-учасникам>. Наприклад – у разі невдалої транзакції всі дані, що були в системі до її початку, зберігаються незмінними.

Допоміжна інформація <додаткова інформація, корисна при описі варіанту використання>.

Табличні представлення варіанту використання

Іноді представляється зручним поміщати сценарії варіантів використання в таблицю, як це показано нижче. Інформація при цьому набирає більш структурованого вигляду.

Таблиця 1.

Таблиця в 2 колонки

Актор	*Дія*
Користувач	Формує запит на пошук замовлень
Система	Відображає список замовлень
Користувач	Вибирає необхідне замовлення
Система	Показує докладну інформацію за замовленням

Таблиця 2.

Таблиця в 3 колонки

№ кроку	***Користувач***	***Система***
1	Робить запит на пошук замовлень	Відображає список замовлень
2	Вибирає необхідне замовлення	Показує докладну інформацію за замовленням

Шаблон варіанту використання RUP

З шаблоном опису варіанту використання RUP і прикладами можна ознайомитися в інтерактивній версії RUP.

Нижче приведений короткий огляд його розділів.

1. **Найменування і короткий опис.** У цьому розділі вказується: найменування варіанту використання, актори варіанту використання, короткий (у один абзац) опис варіанту використання.

2. **Потік подій**

Основний потік подій

Тут перераховуються кроки основного сценарію, починаючи від трігера і аж до досягнення гарантії успіху

Альтернативні потоки подій

Кожен з альтернативних сценаріїв описується в окремому параграфі, в тому ж стилі, що і основний потік подій. Альтернативні сценарії описують поведінку системи при будь-яких відхиленнях від основного сценарію, а також поведінка у виняткових ситуаціях.

3. **Спеціальні вимоги**

Тут перераховуються нефункціональні вимоги, що мають безпосереднє відношення саме до цього варіанту використання.

4. **Передумови**

Події, що описуються передумовами або умовами поста, повинні бути станами, які користувач може спостерігати. Передумову описує стан, в якому система повинна знаходитися до початку виконання прецеденту.

5. **Постумови**

Постумова RUP по суті описує те ж, що і мінімальна гарантія у Коберна. Л.Новіков акцентує увагу на тому, що коректно сформульована постумова повинна бути істинною при будь-якому можливому сценарії прецеденту, а не описаному в основному потоці.

6. **Точки розширення**

Даний параграф визначає положення точок, що розширюють потік подій.

Вибір форми опису варіанту використання

При виборі форми і ступеня подробиці варіанту використання слід враховувати такі чинники, як:

- розміри проекту;
- важливість проекту і варіанту використання;
- традиції, що склалися в колективі «Замовник-Розробник».

Для невеликого проекту навряд чи буде доцільним застосовувати описи варіантів використання в розгорненому форматі, досить використовувати коротку форму вільного стилю. Для проекту, в якому задіяно більше десяти учасників, в якому виникають проблеми розбиття на мікро-колективи, координації учасників, слід вибрати більш формалізований і докладніший варіант.

Ступінь деталізації залежить також від критичності проекту в цілому і критичності варіанту використання в даному проекті. А.Коберн ділить всі програмні проекти по ступеню критичності на 4 категорії, виходячи з ціни помилок, проекти, помилки в яких можуть привести до:

- небезпеки для життя людей;
- непоправним фінансовим витратам;
- фінансовим витратам в обмеженому об'ємі;
- зниженню комфортності кінцевого користувача.

Очевидно, що військові системи, або системи управління складними технічними об'єктами вимагають більш скрупульозного документування, зокрема на рівні опису варіантів використання.

Крім того, в одному і тому ж проекті можуть зустрічатися важливіші, з позицій частоти і масовості використання, складності для розуміння, технічних ризиківк і т. д. і менш важливі прецеденти. В цьому випадку для різних прецедентів одного і того ж проекту цілком допустимий опис з різним ступенем деталізації.

Нарешті, специфікація варіантів в стилі Коберна, стилі RUP, в табличній формі, з використанням псевдокодів або графічних конструкцій визначається суб'єктивним вибором автора прецедентів і досвідом роботи, що склався, із замовником проекту.

3.4.Специфікація нефункціональних вимог

Опис нефункціональних вимог зазвичай здійснюється у формі, близькій до вільного формату опису варіанту використання. RUP рекомендує концентрувати нефункціональні вимоги в документі, що описує варіант використання в усіх випадках, коли це можливо. У випадку, якщо нефункціональні вимоги носять загальний характер і не можуть бути прив'язані до конкретного прецеденту – вони виносяться в документ «*Додаткова специфікація*».

Атрибути вимог. Описи вимог повинні бути операбельні. Для цього всі вимоги повинні враховуватися в тій або іншій обліковій системі, будь то електронна таблиця MS Excel, спеціалізована база даних або інтегроване середовище управління змінами. При реєстрації вимоги вона проходить класифікацію відповідно до певної системи ознак. Для оперативного управління вимогами буває корисно призначити їм такі властивості, як *проект*, *відповідальну особу*, *статус*, *ризик*, *ступінь закінченості* і т. п. У RUP для управління атрибутами вимог передбачений артефакт «*Атрибути вимог*».

Артефакт «Атрибути вимог», пропонований RUP, є репозиторій (місце, де зберігаються і підтримуються будь-які дані) текстів вимог, їх атрибутів і трасованості.

Атрибути вимог представлені *матрицею атрибутів вимог*, де для кожного типу вимог перераховуються вимоги по одній осі і атрибути вимог цього типу по іншій. Для кожної вимоги вказуються значення її відповідних атрибутів. Приклади атрибутів: статус в часі, пріоритет, важливість, ризик № ітерації (етапу) в плані.

Трасованість описується у вигляді дерева, що показує в графічному вигляді вхідні і (або) витікаючі зв'язки трасованості.

ТЕМА 4. РОЗШИРЕНИЙ АНАЛІЗ ВИМОГ. МОДЕЛЮВАННЯ

4.1. Які моделі використовувати

Вербальні описи варіантів використання системи на сьогодні є стандартом де-факто для формулювання угоди між Замовником і Виконавцем. Якщо обидві сторони готові виділити достатню кількість часу на уважний і всесторонній аналіз вимог до системи і на початковій фазі її створення сформулювали 80% всіх можливих сценаріїв використання системи на зрозумілій сторонам мові – це означає, що ключові ризики створення системи – ризик різного розуміння проблеми і ризик розмиття меж - багато в чому подолані.

Проте, далеко не кожен Замовник готовий скрупульозно обговорювати нудні томи опису варіантів використання, які навіть для систем середнього розміру можуть досягати сотні сторінок.

Щоб полегшити процес формулювання і розуміння вимог для Замовника, існує ряд прийомів.

Вимоги можна формулювати на різних рівнях абстракції. Так, рівень ***опису вимог***, підтримуваний в документі «Концепція», є достатньо збалансованим. Тож можна сказати і про короткі (у один абзац) описи ключової функціональності системи. Діючи таким чином, ми, очевидно, вирішимо проблему залучення Замовника в аналіз завдань, проте вказані вище ризики будуть знижені недостатньо: концептуальні описи функціональності залишають багато свободи для тлумачення в ту або іншу сторону.

Хорошою допомогою в рішенні задачі є застосування візуальних засобів опису вимог. Як відомо, у більшості людей образне мислення дозволяє сприймати інформацію в рази і порядки більш прискореному темпі, ніж вербальне. На сьогодні в арсеналі аналітика існують десятки методик, мов, візуальних уявлень, що дозволяють моделювати вимоги до системи. При створенні інформаційних систем стандартом де-факто є універсальна мова моделювання, UML.

Процес аналізу вимог тісно зв'язаний, з одного боку, з аналізом проблемної області, з іншої – з архітектурним аналізом і проектуванням. Часто на практиці буває важко вирізнити межі компетенції цих потоків робіт. Так, модель аналізу потоків даних, широко використовувана в аналізі проблемної області, згадується багатьма авторами, як модель, корисна в аналізі вимог. Ряд дослідників вважає за доцільне ілюструвати описи вимог діаграмами класів, хоча виділення класів відноситься не до аналізу вимог, а до архітектурного аналізу.

Як визначити доцільність використання тих або інших прийомів, методик, мов моделювання при аналізі вимог? Тут можна запропонувати три практичні рекомендації.

По-перше, аналіз вимог покликаний вивчати взаємодії програмної системи і її середовища, тобто користувачів, мережевих і системних компонент, таких, що знаходяться поза системою. Отже, *бізнес-моделі*, що описують взаємодії між компонентами організаційної системи можна розглядати лише як «*сировину*» для виявлення вимог, але не як моделі, що описують вимоги.

По-друге, аналіз вимог повинен знаходити відповідь на те, **ЩО** робить система, абстрагуючись від деталей реалізації, тобто того, **ЯК** вона це робить. Тому, припустимо, діаграму взаємодії об'єктів, що реалізовують той або інший

варіант використання, можна розглядати швидше, як ілюстрацію внутрішнього устрою системи, корисну для програміста, чим модель для Замовника.

Проте, найбільш важливим є **третє** міркування, в чомусь «опозиційне» по відношенню до перших двох. Для моделювання аналізу вимог слід застосовувати моделі, що *найадекватніше прояснюють функціональність системи і особливості її використання*. Проте, аналітик може вибирати ті мови і методики, які дозволять досягнути цільової функції: **зниження ризику нерозуміння між Виконавцем і Замовником і розмиття меж**.

4.2. Моделі UML, що пояснюють функціональність системи

Діаграма варіантів використання

Діаграма варіантів використання UML, Use Case Diagram – одне з найпростіших представлень системи. Її базові «будівельні елементи» – **актори** і **варіанти використання**. Діаграма задумана так, щоб дати найбільш загальне уявлення про функціональність системи (її компоненти), не вдаючись до деталей взаємозв'язків функцій. Тому основний вид відношення, використовуваний в діаграмі, – асоціація між актором і варіантом використання.

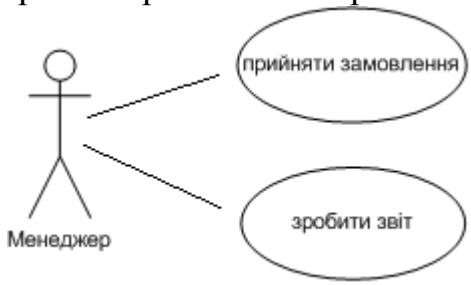


Рис.4. Use Case Diagram

Інші види відносин – відношення включення (include), розширення (extend) і узагальнення/генералізувати.

Включення служить для позначення підлеглих варіантів використання (коли один або більш за варіанти використання містять виклики однієї і тієї ж функціональності).

Рис. 5. Use Case Diagram

Розширення в точності відповідає точці розширення, яку використовую при описі варіанту використання.

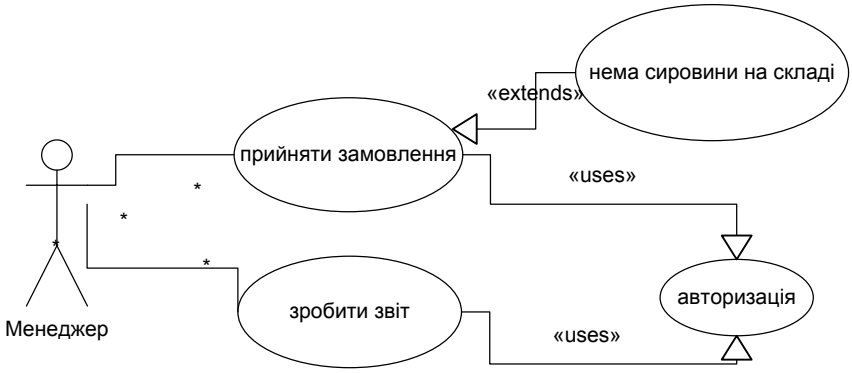


Рис. 6. Use Case Diagram

Відношення узагальнення може застосовуватися як до акторів, так і до варіантів використання, з метою вказівки спеціалізації одні щодо інших.

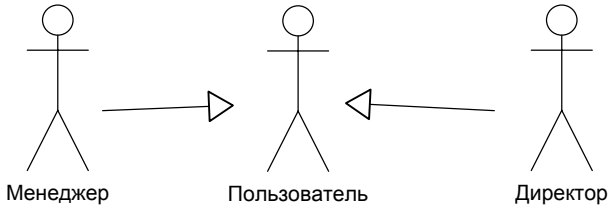


Рис. 7. Use Case Diagram

Діаграма дій

Якщо діаграма варіантів використання дає «вигляд зверху» на функціональність системи, **діаграма дій** UML, навпаки, дозволяє детально ілюструвати окремий варіант використання і його сценарії.

Основні компоненти опису системи:

- Функції (дії)
- Символи «старт» і «стоп»
- Потоки управління
- Розгалужувачі
- Лінійки синхронізації.

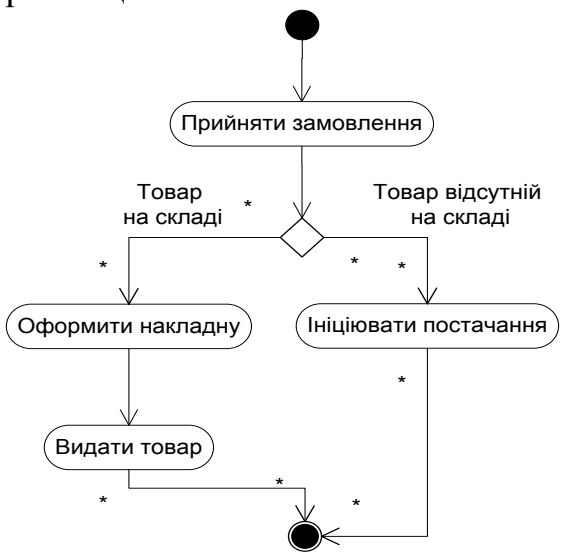


Рис.8. Діаграма дій

Діаграма дій дозволяє проілюструвати варіант використання з необхідним ступенем деталізації. Лінійний варіант використання приводить до діаграми дій з лінійним потоком управління між діями. Дії варіанту використання з альтернативними сценаріями реалізується через розгалужувачі. Лінійки синхронізації дозволяють описувати такі складні конструкції, як синхронізацію початку (закінчення) паралельних в часі процесів.

Окрім стандартного формату опису, UML пропонує варіант з «плаваючими доріжками». Цей формат зручний для опису випадку, коли у варіанті використання беруть участь декілька акторів.

Діаграма станів

Діаграма станів в аналізі вимог використовується, коли потрібно досліджувати поведінку системи, як кінцевого автомата. Це уявлення прийшло в UML з теорії систем.

У загальному випадку діаграма станів описує, як система поводиться в більш, ніж одному варіанті використання. Синтаксис діаграм станів багато в чому співпадає з синтаксисом діаграм дій.

Основні компоненти опису системи:

- Прості стани
- Складені стани
- Символи «старт» і «стоп»
- Переходи
- Лінійки синхронізації.

У мові UML під станом розуміється абстрактний метакласс, використовуваний для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути заданий у вигляді набору конкретних значень атрибутів класу або об'єкту, при цьому зміна їх окремих значень відображатиме зміну стану модельованого класу або об'єкту.

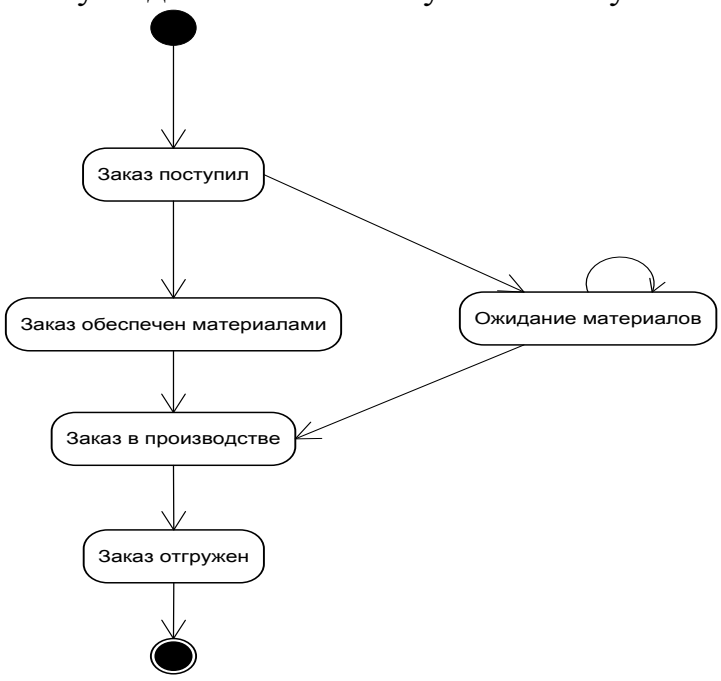


Рис.9. Діаграма станів

Перехід системи із стану в стан здійснюється при настанні *подій*. При цьому кажуть, що перехід *спрацьовує.* Перехід може бути безальтернативним, або містити альтернативи. У другому випадку перехід обумовлений настанням межових *умов.* Нарешті, подія може супроводжуватися виразом дії, яка відбувається у випадку, якщо спрацьовує перехід. Повний синтаксис опису переходу (написи на стрілці) наступний:

Подія [сторожова умова] / вираз дії

Іноді буває корисним об'єднати частину станів в один мета-стан. Графічно це виглядає, як символ стану (прямокутник з кутами, що округляють), що містить усередині себе декілька символів станів. При цьому можливі переходи між підлеглими станами, переходи між підлеглим і зовнішнім станами і переходи між складеним і зовнішнім станом.

4.3. *Діаграми UML, що пояснюють внутрішній устрій системи*

Деякі автори рекомендують використовувати при описі вимог діаграми UML, що описують створювану систему через її компоненти (класи, об'єкти), відносини і взаємодії між ними. Даний підхід має свої обмеження

Діаграма класів.

Для створення діаграми класів необхідно:

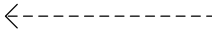
1. Здійснити пошук класів (ключових компонент проблемної області).
2. Для кожного знайденого класу визначити його ім'я, основні атрибути, операції і (або) відповідності.
3. Дослідити взаємозв’язки знайдених класів.

Класи на діаграмі представляються у вигляді прямокутників, відносини – у вигляді ліній, що зв'язують прямокутники. Лінії різного типу графічно відрізняються різним штрихуванням і стрілками.

Прийнято виділяти [5] 3 рівні абстракції класів: концептуальний рівень, рівень специфікації, рівень реалізації. Аналіз вимог розумно супроводжувати діаграмами, що відображають концептуальний рівень. На даному рівні при описі класів доцільно вказати їх найменування і відповідальності. атрибути і операції можна не вказувати, або ввести тільки найосновніші, відклавши їх дослідження на пізніші стадії деталізації.

Відносини, що підлягають аналізу на концептуальному рівні, – це:

асоціація (іменований зв'язок)


залежність (зміни в одному класі приводять до змін в іншому)


узагальнення / генералізує (родо-видове відношення)

агрегація (відношення «частина-ціле»)

композиція (відношення «частина-ціле», однозначно регламентуюча кількість і склад частин цілого).

Діаграма класів показує статичну структуру проблемної області. Для аналізу взаємодії об'єктів – екземплярів класу в ході реалізації варіанту використання в UML передбачено дві діаграми взаємодії: **діаграма кооперації** і **діаграма послідовності**.

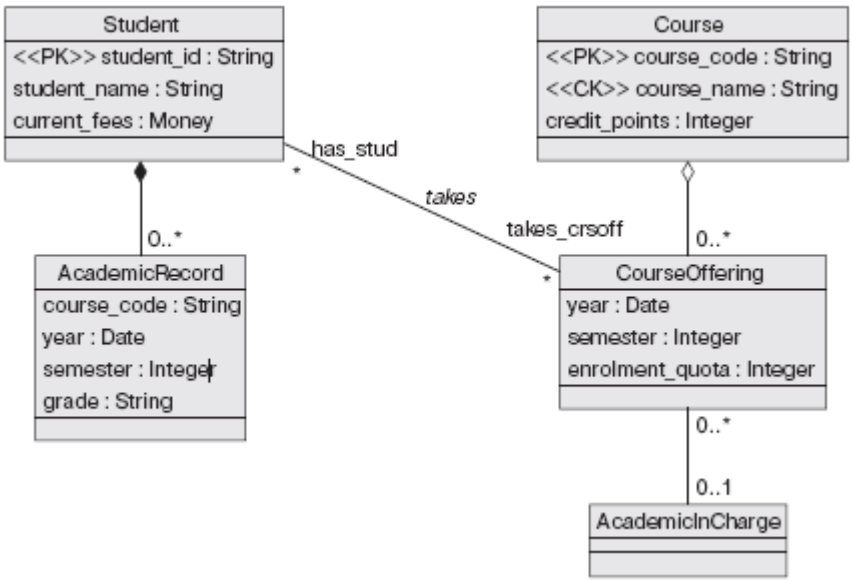


Рис. 10. Специфікація агрегацій (Запис на університетські курси)

Якщо діаграма класів у багатьох випадках і може розглядатися, як артефакт, що пояснює структуру проблемної області, то діаграми взаємодії навряд чи варто розглядати як виразний мовний засіб, що ілюструє вимоги до системи в діалозі «Замовник-Виконавець». Проте, у відповідності з Принципом 3 свобода вибирання мовних засобів, аналітик, що вирішив використовувати дані діаграми, може ознайомитися з ними в спеціальній літературі.

4.4. *Альтернативні мови моделювання*

Діаграма потоків даних

Діаграма потоків даних (data flow diagram, DFD) – один з основних інструментів структурного аналізу і проектування інформаційних систем, що існували в «доюмеельну» епоху. Не дивлячись на місце, що має, в сучасних умовах зсув акцентів від структурного до об'єктно-орієнтованого підходу до аналізу і проектування систем, «старовинні» структурні нотації як і раніше широко і ефективно використовуються як в бізнес-аналізі, так і в аналізі інформаційних систем.

Історично склалося так, що для опису діаграм DFD використовуються дві нотації – Йодана (Yourdon) і Гейна-Сарсона (Gane-Sarson), що відрізняються синтаксисом. На приведеній нижче ілюстрації використана нотація Гейна-Сарсона.

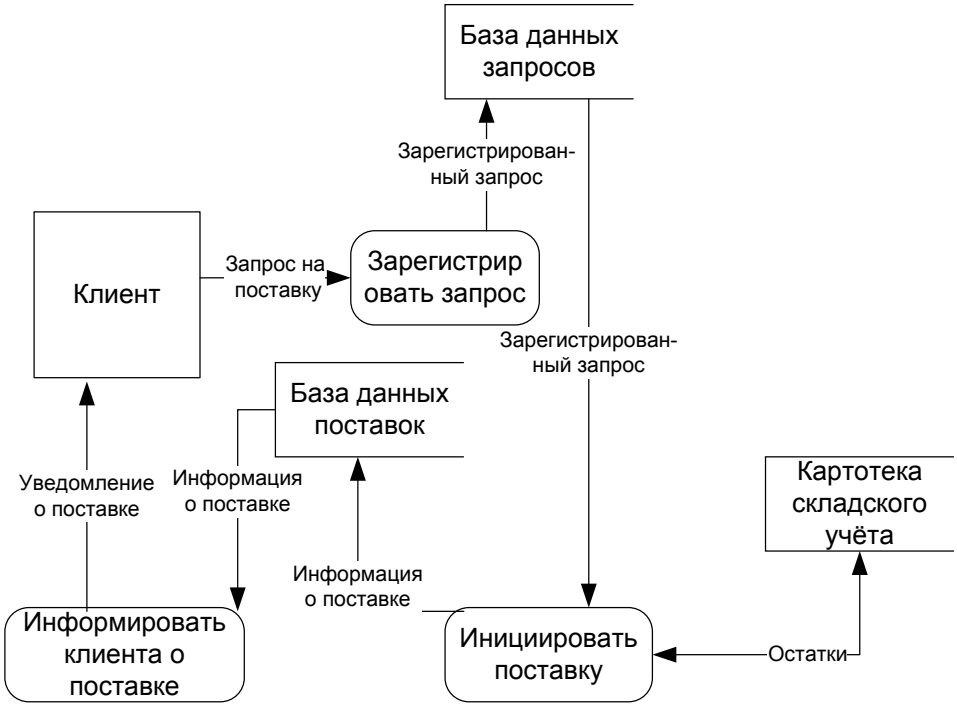


Рис.11. DFD

Інформаційна система приймає ззовні *потоки даних*. Для позначення елементів середовища функціонування системи використовується поняття *зовнішньої суті*. Усередині системи існують *процеси* перетворення інформації, що породжують нові потоки даних. Потоки даних можуть поступати на вхід до інших процесів, поміщатися (і витягуватися) в *накопичувачі даних*, передаватися до зовнішньої суті.

Модель DFD, як і більшість інших структурних моделей – ієрархічна модель. Кожен процес може бути підданий декомпозиції, тобто розбиттю на структурні складові, відносини між якими в тій же нотації можуть бути показані на окремій діаграмі. Коли досягнута необхідна глибина декомпозиції – процес нижнього рівня супроводжується *міні-специфікацією* (текстовим описом).

Крім того, нотація DFD підтримує поняття *підсистеми* – структурною компоненти системи, що розробляється.

Нотація DFD – зручний засіб для формування *контекстної діаграми,* тобто діаграми, що показує ПС, що розробляється, в комунікації із зовнішнім середовищем. Це - діаграма верхнього рівня в ієрархії діаграм DFD. Її призначення – обмежити рамки системи, визначити, де закінчується система, що розробляється, і починається середовище. Інші нотації, часто використовувані при формуванні контекстної діаграми, – діаграма **IDEF0(SADT)**, діаграма Ошибка! Источник ссылки не найден..

Серед різноманіття моделей, що використовуються в аналізі систем, хочеться особливо відзначити ще дві нотації, що дозволяють описати складні багато альтернативні взаємодії компонент інформаційної системи – нотацію IDEF3 [4] і EPC-діаграму ARIS [10].

Для моделювання вимог до систем з розгалуженою логікою К.Вігерс рекомендує використовувати таблиці і дерева рішень [1]. Часто на практиці бувають корисні діаграми суть-зв'язок і SADT-діаграми [9].

ТЕМА 5. АТРИБУТИ ЯКОСТІ ПЗ

Атрибути якості ПЗ – є додатковим описом функцій продукту, опис характеристик важливих для користувачів або розробників.

До атрибутів якості можна віднести декілька дюжин характеристик продукту (Charette, 1990), хоча для більшості проектів вистачило б всього декілька. Якщо розробникам відомо, які характеристики найбільш важливі для успіху проекту, вони можуть вибрати відповідні прийоми, працюючи над архітектурою, дизайном і програмним наповненням ПЗ, які дозволять досягти певної якості (Glass, DeGrace і Stahl). Для класифікації атрибутів якості застосовуються різні схеми (Boehm, Brown і Lipow, 1976; Cavano і McCall, 1978; IEEE 1992; DeGrace і Stahi, 1993).

У табл. 3 перераховано декілька атрибутів якості ПЗ, які необхідно брати до уваги в будь-якому проекті.

Таблиця 3.

Атрибути якості ПЗ

Важливі для користувача	Важливі для розробника
• Доступність • Ефективність • Гнучкість • Цілісність • Здатність до взаємодії • Надійність • Стійкість до збоїв • Зручність і простота використання	• Простота в експлуатації • Простота переміщення • Можливість повторного використання • Тестованість

Деякі мають велике значення для вбудованих систем (ефективність і надійність), тоді як інші особливо важливі для Інтернет-застосувань і застосувань для мейнфреймів (доступність, цілісність і легкість в експлуатації) або для настільних систем (здібність до взаємодії і зручність і простота використання). Для вбудованих систем часто враховують і інші важливі атрибути якості, наприклад безпека, легкість і, простота встановлення і обслуговування. Для Інтернет-застосувань також важливий ще один атрибут - масштабованість.

5.1. Атрибути, важливі для користувачів

• **Доступність.** Під доступністю розуміється запланований *час доступності* (uptime), протягом якого система доступна для використання і повністю працездатна.

Формально доступність дорівнює *середньому часу до збою* (mean time to failure, MT-TF) системи, діленому на суму середнього часу до збою і очікуваного часу до відновлення системи після збою.

На доступність впливають періоди планового технічного обслуговування.

Доступність як сукупність надійності, легкості в експлуатації і цілісності (Gilb, 1988).

- **Ефективність -** називається показник того, наскільки ефективно система використовує продуктивність процесора, місце на диску, пам'ять або смугу пропускання з'єднання (Davis, 1993).

Якщо система витрачає дуже багато доступних ресурсів, користувачі відмітять зниження продуктивності.

Недостатня продуктивність дратує користувачів ставить під удар безпеку, наприклад, при перевантаженні системи контролю процесів реального часу.

Визначіть мінімальну конфігурацію устаткування, при якій вдається досягти заданих ефективності, пропускній спроможності і продуктивності. Щоб дозволити нижню межу у разі непередбачених умов і визначити подальше зростання, ви можете скористатися таким формулюванням:

Приклад. Як мінімум 25% пропускної спроможності процесора і оперативної пам'яті, доступної застосуванню, не повинно використовуватися в умовах запланованого пікового навантаження.

Типові користувачі не формулюють вимоги до ефективності, вони можуть задати час відгуку або заповнення простору на диску.

Справа аналітика – поставити питання, які виявлять очікування користувачів про прийнятне зниження продуктивності, можливі піки навантаження і очікуване зростання.

- **Гнучкість –** показує з якою легкістю в продукт вдається додати нові можливості.

Якщо очікується, що при розробці доведеться вносити безліч поліпшень, варто вибрати такі рішення, які дозволять збільшити гнучкість ПЗ.

Цей атрибут важливий для продуктів, як модель розробки яких вибрано поліпшення і повтор успішних випусків, або розвиток прототипу.

Приклад. Програміст по технічному обслуговуванню, що не менше шести місяців працює з продуктом, повинен уміти підключати новий пристрій для створення друкарських копій, що передбачає зміну коду і тестування, не більше ніж за годину робочого часу.

Проект не можна вважати невдалим, якщо програмістові вимагається 75 хвилин, щоб встановити новий принтер, отже, це вимога допускає деяку міру свободи.

Якби ми не вказали цю вимогу, можливо, розробники вибрали б такий варіант дизайну, при якому установка нового пристрою в системі зайняла б дуже багато часу.

Записуйте вимоги до якості у форматі, який передбачає можливість їх вимірювання.

- **Цілісність –** включає безпеку пов'язану з блокуванням неавторизованого доступу до системних функцій, запобіганням втраті інформації, антивірусним захистом ПЗ і захистом конфіденційності і безпеки даних, введених в систему.

Цілісність дуже важлива для Інтернет-застосувань. Користувачі систем електронної комерції хочуть забезпечити дані своїх кредитних карток.

Відвідувачі Web-сайтів не бажають, щоб приватна інформація про них або список відвідуваних ними сайтів використовувалися не за призначенням, а

постачальники послуг доступу до Інтернету хочуть захиститися від атак типу «відмова в обслуговуванні» і інших хакерських атак.

У вимогах до цілісності немає місця помилкам. Використовуйте наступні точні терміни для формулювання вимог до цілісності:

перевірка ідентифікації користувача

рівні привілеїв користувача, обмеження доступу або певні дані, які повинні бути захищені.

Приклад. Тільки користувачі, що володіють привілеями рівня Аудитор, повинні мати можливість проглядати транзакції клієнтів.

Уникайте вказувати вимоги до цілісності у вигляді обмежень дизайну, як, скажімо, вимоги до пароля для контролю доступу. Реальна вимога повинна обмежувати доступ до системи неавторизированих користувачів; паролі - всього лише один із способів (хоча і найпоширеніший) виконання цього завдання, Засноване на вибраному підході до ідентифікації користувачів, це базова вимога до цілісності вплине на певні функціональні вимоги, які реалізують функції аутентифікації в системі.

- **Здатність до взаємодії** показує, яким чином система обмінюється даними або сервісами з іншими системами.

Щоб оцінити здатність до взаємодії, вам необхідно знати, які застосування клієнти застосовуватимуть спільно з вашим продуктом і обмін яких даних передбачається.

Користувачі Chemical Tracking System звикли малювати хімічні структури, за допомогою декількох комерційних інструментів, тому вони висунули наступну вимогу до здатності взаємодії:

Приклад. Chemical Tracking System повинна мати можливість імпортувати будь-які допустимі хімічні структури з пакетів ChemiDraw (версії 2.3 або ранішою) і Chem-Struct (версії 5 або ранішою).

Ви також могли б вказати дану вимогу як вимогу до зовнішнього інтерфейсу і визначити стандартні формати файлів, які здатна імпортувати Chemical Tracking System.

Як альтернативу можна визначити декілька функціональних вимог, що відносяться до операції імпорту. Проте іноді, розглядаючи систему з погляду атрибутів якості, ви з'ясовуєте, що деякі певні вимоги не задані. Клієнти не вказали, дану потребу при обговоренні зовнішнього інтерфейсу або функціональності системи. Як тільки аналітик поставив питання про інші системи, з якими доведеться взаємодіяти Chemical Tracking System, прихильник продукту негайно згадав два пакети для малювання хімічних структур.

- **Надійність –** називається вірогідність роботи ПЗ без збоїв протягом певного періоду часу (Musa, lannino і Okumoto, 1987).

Іноді однією з характеристик надійності вважають стійкість до збоїв.
Вимірюють надійність ПЗ:

· як відсоток успішно завершених операцій;

· середній період часу роботи системи до збою.

Визначають кількісні вимоги до надійності, ґрунтуючись на тому, наскільки серйозними виявляться наслідки збоїв і чи виправдана ціна підвищення надійності.

Системи, для яких потрібна висока надійність, слід проектувати з високим ступенем можливості тестування, щоб полегшити виявлення недоліків, що негативно впливають на надійність.

Приклад: Для ПЗ управління лабораторним устаткуванням, призначеним для дослідів з рідкісними дорогими хімікатами, що тривають цілий день, користувачам був потрібний програмний компонент, який забезпечив би надійність проведення експериментів, інші системні функції, такі як періодичний запис в журнал даних про температуру, були не такими важливими. *Умова надійності* – Не більше п'яти з тисячі початих експериментів можуть бути втрачені через збої в ПЗ.

- **Стійкість до збоїв (**відмовостійкість, fault tolerance) **–** розуміють рівень, до якого система продовжує коректно виконувати свої функції, не дивлячись на не вірне введення даних, недоліки підключених програмних компонентів або компонентів устаткування або несподівані умови роботи.

Стійке до збоїв ПЗ легко відновлюється після різних проблем і «не помічає» помилок користувачів.

З'ясувавши вимоги до стійкості роботи ПЗ, запитаєте користувачів, які помилкові ситуації можливі при роботі з системою і як система повинна на них реагувати.

Приклад 1. Якщо при роботі з редактором відбувся збій і користувач не встиг зберегти файл, то редактор повинен відновити всі зміни, внесені раніше, ніж за хвилину до збою, при наступному запуску програми даним користувачем.

Приклад 2. Для всіх параметрів, що описують графіки, повинні бути вказані значення за умовчанням, які ядро Graphics Engine використовуватиме у випадку, якщо дані вхідного параметра вказані невірно або відсутні.

Виконання цієї вимоги дозволить уникнути збою програми, якщо, наприклад, застосування запрошує колір, який плотеру не вдається відтворити. Graphics Engine використовує значення за умовчанням - чорний колір - і продовжить роботу. Проте це можна розглядати як збій ПЗ, оскільки кінцевий користувач не отримав бажаний колір. Проте такий підхід понизив серйозність наслідків збоїв - замість краху програми отриманий неправильний колір, що є прикладом відмовостійкої.

5.2. Атрибути, важливі для розробників

- **Легкість в експлуатації.** Цей атрибут показує, наскільки зручно виправляти помилки або модифікувати ПЗ. Легкість в експлуатації залежить від того, наскільки просто розібратися в роботі ПЗ, змінювати його і тестувати, і тісно пов'язано з гнучкістю і тестованістю. Цей показник украй важливий для продуктів, які часто змінюють, і тих, що створюються швидко (і, можливо, з економією на якості). Легкість в експлуатації вимірюють, використовуючи такі терміни, як *середній час потрібний для дозволу проблеми* і *відсоток коректних виправлень*.

Приклад. Для Chemical Tracking System одна з вимог до легкості в експлуатації сформульовано таким чином:

Легкість в експлуатації 1. Програміст, що займається технічним обслуговуванням ПЗ, повинен модифікувати існуючі звіти, щоб привести їх в .соответствие із змінами в

положеннях федерального уряду в області хімії, витративши на розробку не більше 20 робочих годин.

При роботі над проектом Graphics Engine ми розуміли, що нам доведеться часто вносити виправлення ПЗ, щоб воно відповідало потребам користувачів. Ми вказали приблизно такі критерії, щоб розробникам вдалося створити ПЗ, більш легке в експлуатації:

Легкість в експлуатації-2. Вкладеність функцій, що викликаються, не повинна перевищувати два рівні.

Легкість в експлуатації-3. Для кожного програмного модуля непорожні коментарі в співвідношенні до початкового коду повинні складати як мінімум 0,5. Ставте такі завдання розробникам обережно, інакше ті, позбувшись самовладання, почнуть діяти формально, виконуючи букву, але не суть завдання. Попрацюйте з програмістами, що займаються технічним обслуговуванням, щоб зрозуміти, які якості початкового |коди полегшать їм внесення змін або виправлення недоліків.

Для апаратних пристроїв з вбудованим ПЗ часто є вимоги до легкості в експлуатації. Одні з них відносяться до вибору дизайну ПЗ, тоді як інші впливають на дизайн устаткування. Наприклад останнє можна сформулювати так:

Легкість в експлуатації-4. Дизайн принтера дозволяє сертифікованому ремонтникові замінити кабельний шнур друкуючої головки не більше ніж за 10 хвилин, датчик стрічки не більше ніж за 5 хвилин і привід стрічки не більше ніж за 5 хвилин.

- **Легкість переміщення.** Мірою її вимірювання можна вважати зусилля, необхідні для переміщення ПЗ з одного операційного середовища в іншу. Деякі практики вважають можливість інтернаціоналізації і локалізації продукту вищим ступенем його мобільності Прийоми розробки ПЗ, які роблять легким його переміщення,, дуже схожі з тими, що застосовують, щоб зробити ПЗ багатократного використовуваним (Glass, 1992). Зазвичай мобільність продукту або не має ніякого значення, або украй важлива для успіху проекту. Важливо визначити ті частини продукту, які необхідно легко переміщати в інші середовища, і описати ці цільові середовища. Потім розробники виберуть способи розробки і кодування, які збільшать мобільність продукту.

- **Можливість повторного використання.** Постійне завдання розробки ПЗ - можливість повторного використання - показує зусилля, необхідні для перетворення програмних компонентів; з метою їх подальшого застосування в інших застосуваннях. Витрати на розробку ПЗ з можливістю повторного використання значно вище, ніж на створення компоненту, який працюватиме тільки в одному застосуванні. Воно повинне бути модульним, добре задокументованим, не залежати від конкретних застосування і операційного середовища, а також володіти деякими універсальними можливостями. Цілі багатократного використання складно кількісно зміряти. Вкажіть, які елементи нової системи необхідно спроектувати так, щоб спростити їх повторне застосування, або вкажіть бібліотеки компонентів багатократного використання, які необхідно створити додатково до проекту. Наприклад:

Можливість повторного використання-1. Функції введення хімічних структур повинні бути спроектовані так, щоб їх вдавалося повторно використовувати на рівні об'єктної коди в інших застосуваннях, побудованих з урахуванням міжнародних стандартів для представлення хімічних структур.

- **Тестованість.** Цей атрибут також називають *проверяемостью,* він показує легкість, з якою програмні компоненти або інтегрований продукт можна перевірити на предмет дефектів. Такий атрибут украй важливий для продукту, в якому використовуються складні алгоритми і логіка або є тонкі функціональні взаємозв'язки. Тестіруємость також важлива в тому випадку, якщо продукт необхідно часто модифікувати, оскільки передбачається піддавати його частому регресивному тестуванню, щоб з'ясувати, чи не погіршують внесені зміни існуючу функціональність

Оскільки я і команда розробників твердо знали, що нам доведеться тестувати Graphics Engine багато раз в період його неодноразових удосконалень, ми включили в специфікацію наступну директиву, тестируемости, що стосується, ПЗ:

Тестованість-1. Максимальна цикломатическая складність модуля не повинна перевищувати 20.

Цикломатична складність (cyclomatic complexity) - це кількість логічних відгалужень в модулі початкового коду (McCabe. 1982).

Чим більше відгалужень і циклів в модулі, тим важче його тестувати, розуміти і підтримувати. Звичайно, проект не буде провалений. якщо значення цикломатической складності одного з модулів досягне 24, проте наша директива вказала розробникам рівень якості, до якої слід прагнути. Якби її (вона представлена як вимога до якості) не було, не факт, що розробники при написанні своїх програм взяли б до уваги таку характеристику, як цикломатическая складність. В результаті код програми міг опинитися такий заплутаний, що його ретельне тестування і доповнення виявилося б практично неможливим, а відладка взагалі стала б кошмаром.

5.3. Визначення нефункціональних вимог за допомогою мови Planguage

Деякі з атрибутів якості, перерахованих в цьому розділі, є неповними або неспеціалізованими - дуже важко виразити їх одной-двумя короткими фразами. Вам не вдасться оцінити, чи відповідає продукт неточно сформульованим вимогам до якості чи ні. Крім того, спрощені завдання, пов'язані з якістю продуктивністю, можуть виявитися нездійсненними. Максимальний час відгуку, рівний двом секундам для запиту до БД, чудово працює при нескладному пошуку в локальній БД, але абсолютно нездійсненно при з'єднанні шести зв'язаних таблиць, размещенных на серверах, які розташовані в різних місцях.

Для вирішення проблеми неявних і неповних нефункціональних вимог консультант Tom Gilb (1988; 1997) розробив *Planguage* |мова планування з великим набором ключових слів, який дозволяє точно встановлювати атрибути якості і другу завдання проекту (Simmons, 2001). Далі показано, як виразити вимогу до продуктивності за допомогою всього лише декілька з безлічі ключових слів Planguage.

Приклад є версією вимог на мові Planguage до продуктивності з «95% запитів БД каталога повинні бути виконані протягом 3 секунд на комп'ютері Intel Pentium 1,1 Ггц під управлінням Microsoft Windows XP, коли доступні як мінімум 60% системних ресурсів».

TAG. Продуктивність. Час відгуку на запит.

AMBITION. Швидкий відгук на запити БД на призначеній для користувача платформі.

SCALE. Час (у секундах) між натисненням клавіші Enter і клацанням ОК для відправки запиту і початком відображення результатів запиту

METER. Тестування з використанням секундоміра, виконане на 250 тестових запитах, що представляють певний операційний профіль використання.

MUST. Не більше *10* секунди на 98% запитів. (Фахівець виїзний служби підтримки).

PLAN. Не більше 3 секунд для запитів категорії 1. 8 секунд для всіх запитів

WISH. Не більше двох секунд для всіх запитів.

*Основна призначена для користувача платформа DEFINED***.** Процесор Intel Pentium 4 1,1 Ггц, оперативна пам'ять 128 Мб, під управлінням ОС Microsoft Windows XR зі встановленим пакетом QueryGen версії 3.3, однопользовательский комп'ютер, вільно як мінімум 60% системних ресурсів, інші застосування не виконуються

У кожній вимозі присутній унікальний **Тег** (tag).

Мета (ambition) – встановлює мету або завдання для системи, яка породжує дану вимогу.

Масштаб (scale) – визначає одиниці вимірювання.

Лічильник (meter) – описує точно, як виконати вимірювання. Всі зацікавлені сторони повинні однаково добре розуміти, яким чином вимірюється продуктивність. Припустимо, користувачу простіше сприймає систему вимірювань, яка заснована на часі від натиснення клавіші Enter до виведення всіх результатів запиту, чим до початку відображення результатів, як вказано в прикладі, Розробник може заявити, що вимога задоволена, а користувач стверджуватиме зворотне. Точно виражені вимоги до якості і системи вимірювань допоможуть запобігти такого роду непорозумінням.

Ви можете вказати декілька бажаних кількісних величин, які необхідно вимірювати.

Критерій **must** (зобов'язання) – визначає найменший досяжний рівень. Вимога не задоволена до тих пір, поки не будуть задоволені всі умови must, таким чином, ця умова повинна бути обґрунтована в бізнес-термінах. Можна вказати вимогу must іншим способом. Для цього потрібно визначити умову **fait** (невдача) (ще одне ключове слів мови Planguage), наприклад: «Більше 10 секунд очікування для більше 2% всіх запитів». Величина **plan** (план) указує номінальне значення, **wish** (побажання) є ідеальним результатом. Крім того, покажіть джерело необхідної продуктивності, наприклад, згадану вище умову must (зобов'язання) вказав фахівець виїзної служби підтримки. Для полегшення читання будь-які спеціалізовані терміни, використовувані в операторах мові Planguage, задані як **defined** (визначувані).

Planguage визначає безліч додаткових ключових слів, за допомогою яких вдається добитися гнучкішою і точнішою специфи-кации вимог. Оскільки мова Planguage, його словник і синтаксис ще розвиваються, рекомендую звернутися за свіжою інформацією на Web-сайт *http://www.gilb.com*. Planguage є могутнім засобом для точного формулювання атрибутів якості і вимог до продуктивності. Вказавши декілька рівнів для очікуваного результату, ви отримаєте ширше уявлення про вимоги до .качеству, чим за допомогою простих конструкцій, таких, як «чорне -біле» і «так- ні».

5.4. Реалізація нефункціональних вимог

Дизайнери і програмісти повинні визначити якнайкращий спосіб задоволення вимоги для кожного атрибуту якості і продуктивності. Хоча атрибути якості відносяться до нефункціональних вимог, вони допомагають сформулювати функціональні вимоги, визначити напрями розробки або технічну інформацію, яка дозволить реалізувати продукт бажаної якості. У табл. 4 перераховані деякі категорії технічної інформації, які можуть бути виведені за допомогою атрибутів якості.

Таблиця 4.

Перетворення вимог до якості в технічну документацію

Типи атрибутів якості	**Категорія технічної інформації**
Цілісність, здібність до взаємодії, стійкість до збоїв, легкість і простота використання, безпека	Функціональні вимоги
Доступність, ефективність, гнучкість, производительность, надійність	Архітектура системи
Здібність до взаємодії, легкість і простота використання	Обмеження дизайну
Гнучкість, легкість в експлуатації, легкість перемещения. надійність, можливість повторного використання, тестируемость, легкість і простота використання	Керівництво ПЗ дизайну
Легкість переміщення	Обмеження реалізації

Наприклад, в медичний пристрій із строгими вимогами до доступності може входити джерело резервного живлення (архітектура) і функціональні вимоги для візуального або звукового сповіщення, коли продукт працює на резервному живленні. Це перетворення вимог до якості, що мають значення для користувачів або для розробників, у відповідну технічну інформацію є частиною процесу розробки вимог і дизайну високого рівня.

ТЕМА 6. ВІД РОЗРОБКИ ВИМОГ - ДО НАСТУПНИХ ЕТАПІВ

Досвідчені менеджери проекту і розробники розуміють цінність перетворення вимог до ПЗ в надійний дизайн і раціональні плани. Вони необхідні в тих випадках, коли наступна версія представляє 1 або 100% всього кінцевого продукту. У цьому розділі ми розглянемо, як подолати пропасти, яка відокремлює розробку вимог від успішного випуску продукту: декілька варіантів впливу вимог на плани проекту, дизайн, код і тестування, як показано на рис. 12.

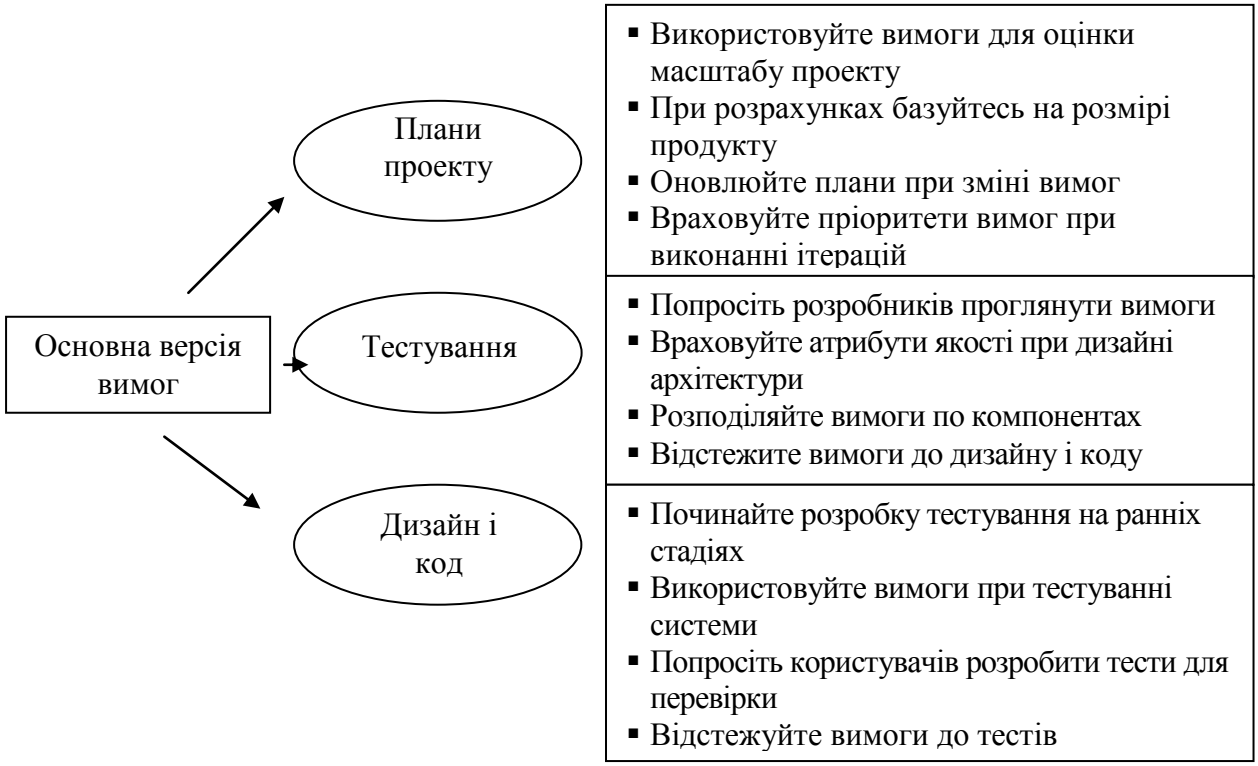


Рис.12. Вплив вимог на планування проекту, дизайн, написання коду і тестування

Оскільки саме вимоги визначають передбачуваний результат проекту, плани, кошториси і графіки слід розробляти на основі вимог. Проте необхідно пам'ятати, що найбільш важливий результат проекту - це та система, яка відповідає бізнес-цілям, а не обов'язково та, в якій реалізовані всі первинні вимоги згідно первинного плану проекту.

У вимогах і планах вказана первинна оцінка витрат на проект, визначена членами команди. Проте межі проекту можуть вийти за первинні рамки, та й плани можуть виявитися нездійсненними. Крім того, бізнес-потреби, бізнес-правила і обмеження проекту можуть змінитися. Успіх проекту сумнівний, якщо ви не оновлюватимете ваші плани відповідно до цілей, що змінюються, і обставин. вимоги є фундаментом для визначення планів, кошторису та графіку розробки ПЗ (рис.13).

Рис. 13.

Менеджерів проекту часто цікавить, скільки часу і зусиль знадобиться для розробки вимог. Для невеликих проектів, якими зазвичай займалася наша команда, цей етап зазвичай коштував від 12 до 15% всіх витрат (Wegers, 1996), проте показник залежить від об'єму і складності проекту. Не дивлячись на побоювання, що робота над вимогами уповільнить створення продукту, доведено, що зрозумілі вимоги прискорюють процес розробки, що і показують наступні приклади:

- вивчення 15 проектів у сфері телекомунікацій і банківської сфери показало, що в найбільш успішних проектах приблизно 28% ресурсів витрачалося на розробку вимог (Hofmann і Lehner 2001): 11%-на збір інформації по вимогах, 10% - на моделювання, а 7% на перевірку і затвердження. На розробку вимог для середнього проекту необхідно 15,7% всіх ресурсів і 38,6% часу;
- у проектах NASA, в яких витрачалося більше 10% всіх ресурсів на розробку вимог, витрати і відхилення від графіка виявилися істотно нижчими, ніж в проектах, де на вимоги витрачалися менше ресурсів (Hooks і Farry, 2001);
- дослідження, проведені в Європі, показали, що команди, які розробляють продукти швидше, присвятили більше часу і зусиль вимогам, ніж повільніші команди (Blackburn, ScuddernVanWassenhove, 1996).
- Не всі ресурси на розробку вимог повинні витрачатися на початку проекту, як в моделі «водопаду» або послідовної моделі життєвого циклу. У ітеративних моделях певний час на вимоги витрачатиметься в кожному повторному циклі розробки. Мета таких проектів - реалізовувати функціональність кожні декілька тижнів, тому вимогами доведеться займатися часто.

- **Вимоги і розрахунки**

Перший крок при оцінці проекту - оцінити об'єм продукту ПЗ. Це роблять виходячи з текстових вимог, моделей аналізу, прототипів або елементів призначеного для користувача інтерфейсу. Хоча не існує ідеального мірила розміру ПЗ, найчастіше використовуються наступні:

- кількість окремо тестованих вимог (Wilson, 1995);
- функціональні крапки і характерні крапки (Jones, 1996b) або тривимірні функціональні крапки, що включають дані, функції і елементи управління (Whitmire, 1995);
- кількість, тип і складність елементів графічного призначеного для користувача інтерфейсу (graphical user interface, GUI);
- оцінка рядків коду, необхідного для реалізації спеціальних вимог;
- кількість класів об'єктів або інших об'єктно-орієнтованих метрик (Whitmire, 1997).

Вимоги і графік

Для складних систем, в яких ПЗ є лише частиною кінцевого продукту, детальні графіки, як правило, складаються після розробки вимог на рівні продукту (системи) і попереднього визначення архітектури. На цьому етапі ключові дати постачання можуть бути визначені і узгоджені на основі інформації з таких джерел, як відділ маркетингу, відділ продажів, відділ по роботі з клієнтами і розробники.

Розглянете можливість поетапного планування і фінансування проекту. На стадії первинного вивчення вимог ви, отримаєте достатньо інформації, щоб скласти реалістичні плани і розрахунки для одного або більш за етапи збірки. Проекти, вимоги до яких сформульовані нечітко, виграють у разі поетапного циклу розробки. Ця модель дозволить розробникам приступити до створення якісного ПЗ задовго до повного з'ясування вимог. Розставивши пріоритети вимог, ви зможете визначити черговість включення функціональності на кожному етапі.

Часто причина невдачі проекту по розробці ПЗ криється в тому, що розробники і інші учасники проекту погано складають плани, а не в тому, що вони погані фахівці. До головних помилок планування відносяться пропуск стандартних завдань, недооцінка витрат або термінів, а також проектних рисок, необґрунтований оптимізм. Крім того, не забудьте врахувати можливі переробки. Ефективне планування проекту припускає наявність наступних елементів:

- оцінку розміру продукту;
- інформацію про продуктивність фахівців, засновану на минулому досвіді;
- список завдань, які необхідні для повної реалізації або перевірки функції або варіанту використання;
- вимоги з прийнятним рівнем стабільності;
- досвід, який дозволить менеджерові проекту врахувати нематеріальні чинники і індивідуальні особливості кожного проекту.

Від вимог до тестування

Тестування і розробка вимог зв'язані синергетично. Згідно думці консультанта Dorothy Graham: «Чим краще за вимогу. тим якісний тести; чим якісний аналіз тестування, тим краще вимоги» (Graham, 2002). Вимоги забезпечують основу для тестування системи. Продукт слід тестувати на відповідність тому, що він, як записано у вимогах, повинен робити, а не на предмет дизайну або коду. Тестування системи, виконане на основі коду. Продукт може з поводитися саме так, як описано у варіантах тестуванні, створених на основі коду, але це не означає, що він відповідним чином реалізує призначені для користувача або функціональні вимоги. Подключіть тестерів до експертизи вимог, щоб переконатися, що вимоги піддаються перевірці і можуть служити основою для тестування системи.

Тестери проекту повинні визначити, як вони перевірятимуть кожну вимогу. Можливих методів декілька:

- тестування (виконання ПЗ з метою пошуку дефектів);
- експертиза (перевірка коду на предмет його відповідності вимогам);
- демонстрація (демонстрація того, що продукт працює, як очікувалося);
- аналіз (обґрунтування того, як система повинна працювати за певних умов).

Обдумування того, як перевірити кожну вимогу, є цінним прийомом роботи над якістю. Використовуйте такі прийоми аналізу, як графіки типу «причина-результат», щоб скласти варіанти тестування на основі логіки, описаної у вимозі (Myers). При цьому виявляються неясності, пропуски або припущення і інші

проблеми. Необхідно, щоб кожну функціональну вимогу можна було прослідкувати принаймні до одного варіанту тестування в системному тестовому наборі, для того, щоб жодна очікувана реакція системи не залишилася неперевіреною. Ви можете зміряти прогрес тестування по частинах, обчислюючи відсоток вимог, які пройшли перевірку. Досвідчені тестери можуть збагатити тестування, засноване на вимогах, додатковими тестами, заснованими на історії продукту, передбачуваних сценаріях використання, загальних характеристиках якості і випадковостях.

У тестуванні на основі вимог застосовуються різні стратегії тестування: залежні від виконуваних дій, відданих (включаючи аналіз значень меж і розбиття на еквівалентні класи), від логіки, від подій і від стану. Можна генерувати варіанти тестування автоматично, використовуючи офіційну специфікацію, проте якщо ви використовуватимете поширеніші специфікації до вимог, складені на природній мові, вам доведеться розробляти варіанти тестування вручну.

У міру просування розробки команда конкретизуватиме вимоги від високих рівнів абстракції, як, наприклад, у варіантах використання, через деталізовані функціональні вимоги до ПЗ до специфікацій для окремих модулів коду. Авторитетний фахівець в області тестування Boris Beizer (1999) підкреслює, що тестування на основі вимог повинне виконуватися на кожному рівні конструювання ПЗ, а не тільки на кінцевому призначеному для користувача рівні. Маса коду в застосуванні не доступно користувачам безпосередньо, проте він необхідний для внутрішніх взаємодій. Кожен модуль повинен задовольняти власній специфікації, навіть якщо функція цього модуля невидима для користувача. Як наслідок, тестування системи на основі призначених для користувача вимог - необхідна, але не достатня стратегія тестування системи.

ТЕМА 7. ПРИНЦИПИ І ПРИЙОМИ УПРАВЛІННЯ ВИМОГАМИ ДО ПЗ

Розробку технічних умов можна розділити на розробку вимог і управління вимогами. Стадія розробки має на увазі збір інформації, аналіз, уточнення і затвердження вимог. Як правило, вона закінчується створенням пакету документів; документа про образ і межі продукту, документацію до варіантів використання, специфікації вимог до ПЗ, словника даних і відповідних моделей аналізу. Після перевірки і схвалення цей пакет визначає базову версію вимог, угоду між розробниками і клієнтами. Ймовірно, вході розробки полягають і додаткові угоди, що стосуються готового продукту, обмежень, графіків, бюджету або контрактних зобов'язань; але ці теми виходять за рамки даної книги.

Угода по вимогах є сполучною ланкою між розробкою і управлінням вимогами. У членів команди повинен бути доступ до поточних вимог впродовж всього проекту, можливо, через Web-додатки за допомогою інструментів управління вимогами. Під управлінням вимогами мають на увазі всі дії, що підтримують цілісність, точність і своєчасність оновлення угоди про вимоги в ході проекту. Як показано на рис.14, це:.

- управління змінами базової версії вимог;
- підтримка планів проекту актуальними відповідно до змінних вимог;
- управління версіями окремих вимог і документації вимог;
- контроль стану вимог в базовій версії;
- управління логічними зв'язками між окремими вимогами і іншими матеріалами для роботи з проектом.

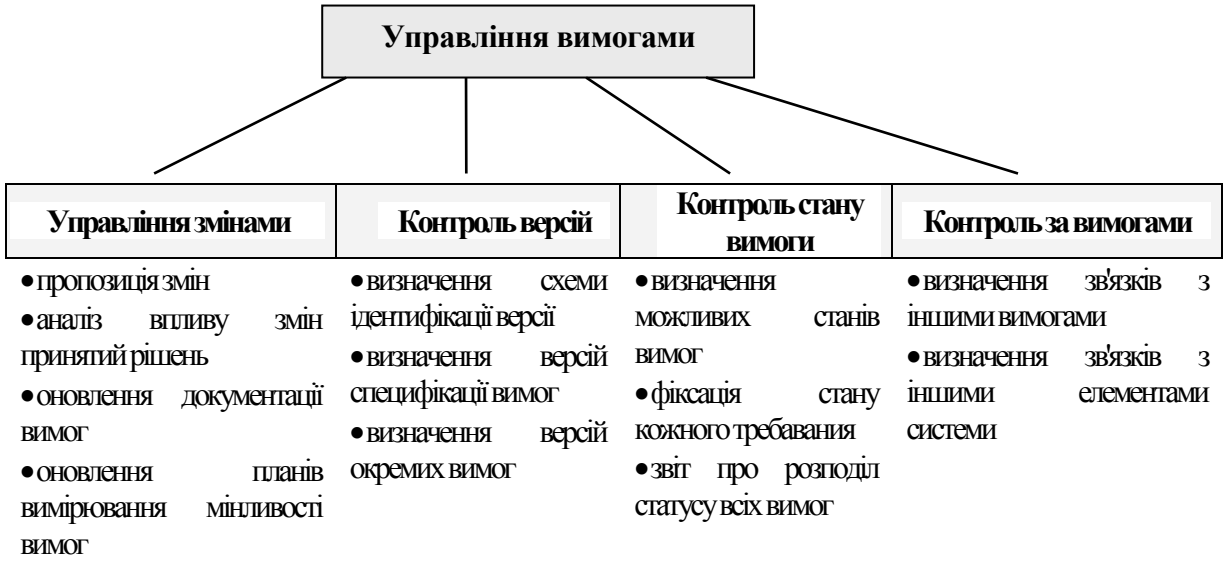


Рис. 14. Основні складові управління вимогами

Розробники, що приймають пропоновані зміни вимог, не завжди здатні дотримувати поточний графік і виконувати зобов'язання, що стосуються якості. Менеджер проекту повинен обговорити зміни цих зобов'язань з іншими менеджерами, що працюють над проектом, клієнтами і всіма зацікавленими в

процесі особами. Введення нових або зміна існуючих вимог впливає на проект таким чином:

- відкладається реалізація вимог нижчих рівнів;
- запрошуються нові фахівці;
- на короткий період часу вводиться режим наднормової роботи, по можливості оплачуваної;
- відповідно до нової функціональності змінюється графік;
- страждає якість, оскільки необхідно укластися в графік (дуже часто це реакція за умовчанням).

Жоден із цих способів не можна назвати ідеальним для всіх випадків, оскільки гнучкість проектів залежить від функцій, досвіду спеціалістів, бюджету, графіка і якості (Wiegers, 1996). При ухваленні рішень покладайтеся на пріоритети, встановлені при плануванні проекту ключовими фігурами з тих, хто зацікавлений в проекті. Незалежно від того, яка буде ваша реакція на коректування вимог, при необхідності прийміть як даність зміни очікувань і зобов'язань. Це набагато краще, ніж сподіватися, що якимсь чином до прийнятої на початку проекту дати випуску все нові функції з'являться в продукті, причому без таких наслідків, як перевитрата бюджету або перевтома команди.

У цьому розділі розглядаються основні принципи управління вимогами.

7.1. Базова версія вимог

Базова версія (baseline) - це набір функціональних і нефункціональних вимог, які розробники зобов'язалися реалізувати в певній (вибраній) версії. Визначення базової версії дозволяє зацікавленим в проекті особам виробити загальне розуміння можливостей і властивостей, які вони хочуть бачити в даній версії. Прийнята базова версія вимог - як правило, версія стає базовою після офіційної експертизи і затвердження - передається для управління конфігурацією. Подальші зміни дозволяється вносити до неї тільки через визначений в проекті процес управління змінами. До ухвалення базової версії вимоги все ще змінюються, тому не має сенсу обмежувати модифікацію якимись процедурами. Проте починайте контролювати версії, унікальним чином ідентифікуючи різні версії кожного документа вимог, відразу після того, як зробите попередній нарис цього документа

З практичної точки зору, положення в базовій версії повинні відрізнятися від запропонованих, але не прийнятих вимог, Необхідно, щоб основна версія специфікації вимог до ПЗ містила тільки ті вимоги, які заплановані для конкретного випуску. Цей документ повинен бути ясно визначений як базовий, щоб відрізняти його від серії виконаних раніше нарисів версій, які ще не затверджені. Збереження вимоги в засобі управління вимогами полегшує ідентифікацію вимог, що належать певній базовій версії, і управління змінами в ній. У цьому розділі описуються деякі способи управління вимогами в базовій версії.

7.2.Процедури управління вимогами

Необхідно визначити дії, які, як очікується, проектна команда виконуватиме для управління вимогами. Задокументуйте ці дії і організовуйте тренінг для фахівців, щоб співробітники організації могли виконувати їх послідовно і ефективно.

Зверніть увагу на наступне:

- на інструменти, прийоми і угоди для управління версіями різних документів вимог і окремих вимог;
- на те, як складається базова версія вимог;
- настатус вимог, які використовуватимуться, і особи, котс-рые мають право змінювати їх;
- на процедури контролю за статусом вимоги;
- на способи, за допомогою яких нові вимоги і зміни існуючих вимог пропонуються, обробляються, обговорюються і передаються всім зацікавленим особам;
- на методи аналізу впливу запропонованої зміни;
- на те, як на планах і зобов'язаннях проекту відіб'ються зміни вимог.

Ви можете включити всю цю інформацію в один опис процесу управління вимогами. Або ж розробити окремі процедури для контролю за змінами, аналізу впливу і контролю за станом. Ці процедури слід довести до зведення кожного співробітника організації, оскільки вони представляють загальні функції, які повинні виконуватися кожним проектною командою.

Хтось повинен виконувати дії при управлінні вимогами, тому в описах процесу також слід визначити ролі учасників, відповідальних за виконання кожного завдання. Проектний аналітик вимог, як правило, несе основну відповідальність за управління вимогами. Він вибирає механізм збереження вимог (наприклад, засіб управління вимогами), визначає атрибути вимог, скоординує оновлення даних про стан і трасованість вимог і згенерує звіти про зміну дій .

Якщо ніхто з власників проекту не несе відповідальності за виконання етапів управління вимогами, не слід чекати їх виконання.

7.3.Контроль версій

Люди турбуються, коли витрачають даремно час, працюючи над застарілими або непослідовними вимогами. Контроль версий- це важливий аспект управління специфікаціями вимог і іншими проектними документами. Кожній версії слід привласнити унікальний ідентифікатор. У кожного члена команди повинен бути доступ до поточної версії вимог, а зміни необхідно ясно документувати і передавати всім зацікавленим сторонам. Щоб мінімізувати конфлікти і недорозуміння, призначте право оновлення вимог певним особам і переконаєтеся, що ідентифікатор версії змінюється при кожній зміні вимоги.

Кожна версія документа вимог повинна містити історію переробки, де вказуються внесені зміни, дата кожного з них, особа, що внесла зміну, а також причина. Можливо, слід додавати номер версії до назви кожної окремої вимоги, який можна послідовно збільшувати при модифікації вимог, наприклад *Print.ConfirmCopies-1*.

Простий механізм управління версіями - іменувати уручну кожну версію специфікації вимог ПЗ згідно угоді. Спроба розрізняти версії документів по даті зміни або даті друку часто приводить до помилок і плутанини; тому не рекомендують їх використання. Деякі команди використовують такі записи: перша версія будь-якого нового документу називається «Версія 1.0, нарис 1»,

наступна чернетка називається «Версія 1.0, нарис 2», автор послідовно збільшує номер нарису при кожній зміні і так до тих пір, поки документ не буде затверджений базова версія документа. Тоді назва змінюється на «Версія 1.0, затверджена». Наступні версії називаються «Версія 1 : нарис 1» при невеликій зміні або «Версія 2.0, нарис 1» при значній зміні. (Зрозуміло, терміни «невеликий» і «значний» суб'єктивний або, принаймні, залежать від контексту.) При застосуванні цієї схеми ясно розрізняються накидання і версії базового документа, проте необхідно дотримувати строгий порядок у веденні документації.

Якщо ви зберігаєте вимоги в текстовому редакторові, ви можете відстежити корективи за допомогою режиму змін тексту. Ця функція виділяє внесену правку, закреслюючи видалені фрагменти, підкреслюючи додані і позначаючи вертикальними лініями на полях положення кожної зміни. Оскільки відкоректований документ важко читати, в текстовому редакторові передбачена можливість проглянути і роздрукувати документ з позначками або його остаточну версію з прийнятою правкою. Після того, як ви складете -базовую версію документа з такими позначками, спочатку створіть архів версії з позначками, потім прийміть всі корективи, а потім збережете очищену версію як нову базову версію для наступного раунду змін.

Складніший метод припускає збереження специфікації вимог в такому засобі контролю версій, як той, що застосовується для контролю початкової коди за допомогою системи контролю версій (check-in і check-out). Для цієї мети розроблена маса комерційних засобів управління конфігурацією. Я знаю про один проект. |коли декілька сотень написаних в Microsoft Word варіантів використання продукту зберігається в такому засобі контролю версій. Члени команди мають доступ до всіх попередніх версій кожного варіанту використання, для яких фіксується історія змін. Аналітикові вимог цього проекту і його заступникові, що відповідає за архівацію, наданий доступ на читання і запис; решті членів команди - доступ тільки на читання.

Найбільш надійний метод контролю версій полягає в зберіганні вимог в базі даних комерційного засобу управління вимогами ці засоби відстежують повну історію змін вимог, що особливо важливо, якщо потрібно повернутися до ранішої версії вимоги. Такий засіб дозволяє вносити коментарі, де можна розумно обгрунтувати рішення про додавання, зміну або видалення вимоги; ці коментарі можуть виявитися корисними при необхідності обговорення вимоги в майбутньому.

7.4. Атрибути вимог

Уявіть собі кожну вимогу у вигляді об'єкту, який володіє властивостями, що відрізняють його від інших вимог. На додаток до текстового опису, кожна функціональна вимога повинна мати декілька фрагментів інформації, що підтримують її, або *атрибутів* (attributes), пов'язаних з нею. Ці атрибути представляють вміст кожної вимоги, вони розташовуються за описом передбачуваної функціональності. Ви можете зберегти атрибути в таблиці, базі даних або, що найефективніше, в засобі управління вимогами. Останні надають декілька атрибутів, що згенерували системою, а також надають можливість визначити інші атрибути. Ці засоби дозволяють фільтрувати, сортувати вибрані підмножини вимог на підставі значень їх атрибутів і запрошувати базу даних для

перегляду. Наприклад, ви можете викликати список всіх високопріоритетних вимог, які Шерп повинна реалізувати у версії 2.3 і які мають статус Approved (Схвалено).

Великий набір атрибутів особливо важливий для величезних і складних проектів. Можливо, вам слід вказати для кожної вимоги такі атрибути, як:

- дата створення вимоги;
- номер його поточної версії;
- автор вимоги;
- особа, відповідальна за задоволення вимоги;
- відповідальний за вимогу або список зацікавлених осіб (щоб ухвалювати рішення про запропоновані зміни);
- стан вимоги;
- походження або джерело вимоги;
- логічне обґрунтування вимоги;
- підсистема (або підсистеми), для яких призначене требование;
- номер версії продукту, для якого призначена вимога;
- використовуваний метод перевірки або критерій тестування прийнятності;
- пріоритет реалізації;
- стабільність (показник того, яка вірогідність зміни требования в майбутньому; нестабільність вимог може показувати погано певні або мінливі бізнес-процеси або бизнес-правила).

Менеджер продукту в компанії, що випускає електронні вимірювальні прилади, хотів просто записати включені вимоги, тому що продукт конкурентів володів тими ж функціями. Такі функції добре відзначати за допомогою атрибуту Rationale (Логічне обґрунтування), що дозволяє вказати, чому конкретна вимога включена в продукт.

Виберіть найменший набір атрибутів, за допомогою якого вам вдасться ефективно управляти проектом, Наприклад, навряд чи вам одночасно буде потрібно і ім'я того, хто відповідає за вимогу, і назву підсистеми, в якій розташована вимога. Якщо яка-небудь інформація про атрибут зберігається у іншому місці, наприклад в загальній системі контролю розробки, не слід копіювати його в базі даних вимог.

Безліч атрибутів вимог можуть настільки спантеличити розробників що вони просто не зможуть надати всі значення для всіх атрибутів і ефективно використовувати інформацію про атрибути. Почніть з трьох або чотири основних атрибутів. Додайте інші атрибути, коли ви знатимете, яким чином вони поліпшать ваш проект.

Основні версії вимог динамічні. Набір вимог, запланований для певної версії, змінюватиметься у міру додавання нових положень і видалення або відстрочення до пізніших версій існуючих вимог. Розробники можуть «жонглювати» окремими документами вимогами для поточного проекту і майбутніх версій. Управління цими базовими версіями, що змінюються, і специфікацією вимог на їх основе- нудна справа. Якщо відкладені вимог або ті, від яких ви вирішили відмовитися, залишаються в специфікації вимог, читачі специфікації можуть заплутатися в тому, які вимоги включені в конкретну базову версію. Проте вам навряд чи захочеться

витрачати багато часу даремно, перетягуючи вимоги з однієї специфікації вимог в іншу. Один із способів вирішення цієї проблеми - зберегти вимоги в засобі управління вимогами і визначити атрибут Release Number (Номер версії). Відстрочення вимоги означає зміну його запланованого випуску, тому просто відновивши номер версії, ви пересунете вимогу в іншу базову версію. Тими вимогами, від яких ви вирішили відмовитися, краще всього управляти за допомогою атрибуту статусу, як описано в наступному розділі.

На визначення і оновлення цих атрибутів потрібна частина витрат, відведених на управління вимогами, проте ці інвестиції в значній мірі окупаються. У одній компанії періодично генерувався звіт про вимоги, в якому показувалося, яка з 750 вимог в трьох зв'язаних специфікаціях призначено кожному дизайнерові. Один з дизайнерів виявив декілька вимог, які він не вважав своїми. Він розрахував, що заощадив від одного до двох місяців, відведених на переробку дизайну, які б були необхідні, якби він з'ясував ситуацію з цими вимогами пізніше.

7.5. Контроль статусу вимог

Іноді розробники ПЗ дуже оптимістичні, коли повідомляють про закінчення завдання. Часто вони видають самі собі кредит, вважаючи виконаним ті завдання, до яких вони приступили, але ще не закінчила. Вони можуть зробити спочатку певну роботу на 90%, а потім зіткнутися з непередбаченими труднощами. Тенденція переоцінювати виконання роботі приводить до ситуації, типової для всіх проектів по розробці ПЗ або крупних завдань, коли протягом довгого часу повідомляється, що виконане 90% об'ємів. Контроль статусу кожної функціональної вимоги впродовж всієї розробки дозволяє точніше оцінювати готовність проекту. Очікування ж «виконання завдань» означає тільки повтори. Наприклад, ви плануєте реалізувати в наступній версії тільки частина варіанту використання, залишивши повну реалізацію до майбутніх версій. В цьому випадку ви відлежуватимете стан тих функціональних вимог, які заплановані для найближчої версії, оскільки цей набір вимог повинен бути виконаний на 100%до випуску продукту.

Існує старий жарт, що на першу половину проекту по розробці витрачається 90% ресурсів, а на решту половини - що залишилися 90% ресурсів. Занадто оптимістична оцінка і відношення потурання до контролю статусу - вірний шлях до перевитрат в проекті.

У таблиці 5 перераховано декілька можливих станів вимог. (Альтернативна схема показана в Caputo, 1998.) Деякі фахівці додають стан Designed (Розроблено) (якщо елементи дизайну, в яких відбиті функціональні вимоги, створені і перевірені) і Delivered (Випущено) (ПЗ віддано в руки користувачів, як для **бета-тестування).** Корисно відстежувати вимоги, від яких ви **відмовилися,** і причини цього, тому що знехтувані вимоги мають звичай «спливати» в ході розробки. Статус Rejected (Відхилено) дозволяє залишити запропоновану вимогу доступною для майбутнього використання, не створюючи безладу в наборі затверджених вимог для певного випуску.

Розподіл вимог по декількох категоріях стану має більший сенс, чим спроба контролювати відсоток завершення кожної вимоги або всієї базової версії. **Визначите,** хто може змінити стан вимоги, і оновлюйте статус, тільки коли задоволені певні умови переходу. Зміна стану також веде до оновлення даних

про трасованість вимог, коли указується, в яких елементах дизайну, коди і тестування відбита вимога (табл. 5).

Таблиця 5.

Рекомендовані стани вимог

Стан	Визначення
Proposed (Запропоновано)	Запит вимоги авторизованим джерелом
Approved (Схвалено)	Вимога проаналізована, її вплив на проект прораховано, і воно було розміщене в базовій версії певної версії. Ключові зацікавлені в проекті особи погодилися з цією вимогою, а розробники ПЗ зобов'язалися реалізувати його
Implemented (Реалізовано)	Код, що реалізовує вимогу, розроблений, написаний і протестований. Вимога відстежена до відповідних елементів дизайну і коду
Verified (Перевірено)	Коректне функціонування реалізованої вимоги підтверджене у відповідному продукті. Вимога перевірена до відповідних варіантів тестування. Тепер вимога вважається завершеною.
Deleted (Видалено)	Затверджена вимога видалена з базової версії. Опишіть причини видалення і назвіть того, хто ухвалив це рішення Вимога запропонована, але не заплановано для реалізації ні в одній майбутніх версій.
Rejected (Відхилено)	Опишіть причини відхилення вимоги і назвіть того, хто ухвалив це рішення

7.6. Управління змінами вимог

Як і при розробці вимог, у ваш план проекту повинні бути включені завдання і ресурси для виконання завдань по управлінню вимогами, описані в цьому розділі. Якщо ви з'ясуєте, скільки зусиль витрачається на управління вимогами, ви зможете оцінити - мало їх, як треба або дуже багато - і змінити робочі процеси і майбутні плани відповідним чином. У організації, де ніколи не вимірювалася жодна із сторін роботи, як правило, важко контролювати, як члени команди витрачають їх робочий час. Для вимірювання безпосередніх зусиль на розробку і управління проектом необхідно зайнятися корпоративною .культурой і дисципліною співробітників - тільки так вдасться фіксувати щоденну роботу. На це

йде не так багато часу, як багато хто боїться, Команда зможе отримати цінні ідеї, зрозумівши, як насправді витрачаються зусилля на різні завдання проекту (Wegers). Звернете увагу, що робочі зусилля - це не те ж саме, що минулий календарний час. Виконання завдань може уриватися або виникає необхідність переговорити з іншими людьми, що веде до затримок. Зусилля, що додаються, необхідні для виконання завдання і вимірювані в людино-годинах, не обов'язково зміняться із-за цих чинників, проте тривалість виконання завдання збільшиться.

Контроль зусиль також дозволяє з'ясувати, чи виконують розробники передбачувані завдання для управління вимогами. Невдача в управлінні вимог збільшує ризик появи некерованих змін і вимог, по необережності пропущених в ході реалізації.

Врахуйте зусилля для виконання наступних дій зусиллями для управління вимогами:

- пропозиція зміни вимог і нових вимог;
- оцінка запропонованих змін, включаючи оцінку впливу зміни;
- зміна роботи радого з управління;
- оновлення документації вимог або бази даних;
- повідомлення про зміни вимог зацікавленим групам, окремим особам;
- контроль і звіт про стан вимог;
- збір інформації про трасованість вимог.

Управління вимогами допомагає переконатися, що зусилля, витрачені на збір, аналіз і документацію вимог не витрачені даремно. Наприклад, якщо ви не зможете своєчасно оновлювати набір вимог відповідно до змін, що вносяться, то зацікавленим в проекті особам буде важко представити функціональність в кожній версії. Ефективне управління вимогами може зменшити невірні очікування: зацікавлені особи будуть проінформовані про поточний стан вимог в ході процесу розробки.

ТЕМА 8. ТЕСТУВАННЯ ВИМОГ

Вимоги - основа майбутньої системи

Вимоги це -

- функціональна характеристика системи, необхідна замовникові для того щоб вирішити проблему або досягти поставлених цілей
- функціональна характеристика, якою повинна володіти система для того того щоб бути відповідною контракту, стандарту, специфікації або іншому формальному документу
- документ, що описує функціональні характеристики позначені вище

Існують різні типи вимог:

- **Бізнес вимоги -** цілі і завдання підприємства
- **Вимоги користувачів -** потреби окремих (груп) замовників
- **Функціональні вимоги -** описують **що** повинна робити система
- **Нефункціональний вимоги -** описують **як** повинна працювати система
- **Припущення і обмеження -** аспекти наочної області які обмежують або впливають на дизайн системи

8.1. Методи тестування вимог

Перед тестуванням вимог

- Вимоги проаналізовані і задокументовані аналітиком
- Аналітик самостійно провів перевірку

Під час тестування вимог

- Зацікавлені особи підтверджують, що вимоги коректні і зрозуміли
- Користувачі підтверджують, що вимоги відображають їх потреби

Критерій готовності вимог

- У вимогах міститься достатньо інформації для початку розробки

Учасники тестування вимог

- Замовник, користувачі, проектна команда

Тестування вимог грає важливу роль в процесі розробки ПЗ

Зменшення кількості доопрацювань і змін

Скорочення ризиків

Ознайомлення і узгодження завдань між розробниками

Методи тестування вимог

- Перевірка документації
- Аналіз поведінки системи
- Прототипування

8.2. Перевірка вимог

Перевірка вимог найпоширеніший метод тестування вимог

*Опис методу:*Послідовний перегляд і перевірка всіх вимог

Дозволяє перевірити на: правильність, повноту, простежуваність, важливість, зрозумілість, однозначність і вимірність

Застосовується: замовниками, аналітиками, проектними менеджерами, тестувальниками

Сильні сторони

- Простота використання

- Відсутність спеціальних вимог до перевіряючого
- Покриває багато критеріїв якості
- Менші витрати часу

Слабкі сторони

- Якість перевірки залежить від перевіряючої
- Залучення різних фахівців
- Наявність документа з вимогами

8.3. Аналіз поведінки системи

Опис методу

- Формування вимог у форматі вхід-вихід, подія- наслідок, умова-відповідь

Застосовується найчастіше

- Тестувальниками (test cases)
- Аналітиками (use cases)

Дозволяє перевірити на

- повноту, зрозумілість, однозначність

Сильні сторони

- Добре перевіряє вимоги
- Представляє вимоги в структурованому і зрозумілому вигляді
- Результати методу легко використовуються для створення сценаріїв тестування

Слабкі сторони

- Вимагає більшої кількості часу
- Вимагає спеціальної підготовки

8.4. Прототипування

Опис методу

- Створення моделі майбутньої системи

Застосовується частіше для перевірки

- Повноти, правильності, реалізовуваності, зручності використання

Використовується

- Аналітиками, архітекторами

Приклад використання

- Горизонтальний – модель якомога ширшого набору функціональності
- Вертикальний – докладна вузьконаправлена модель невеликої частини системи
- Throw-away – не використовується в подальшій розробці
- Еволюційний – поступово допрацьовується і перетворюється на готову систему

Сильні сторони

- Користувачі дістають можливість перевірити рішення
- Наочна допомога для розробників і тестувальників

- Перевірка вимоги на ту, що реалізовується і на зручність призначеного для користувача інтерфейсу

Слабкі сторони

- Вимагає значного часу
- Спеціальна підготовка для створення еволюційного прототипу

Вимоги важливо тестувати для того що б гарантувати високу якість системи
У вимог є багато критеріїв, яким вони повинні задовольняти
Тестування вимог виконується широким колом фахівців

ТЕМА 9. УЗГОДЖЕННЯ ВИМОГ ТА КЕРУВАННЯ РИЗИКАМИ

9.1.Узгодження вимог

Зміна вимог. Ви можете зменшити розповзання меж проекту, використовуючи документ про його образ і межі як контрольні точки для затвердження змін. Об'єднання виявлення вимог з істотною участю користувачів може скоротити розповзання вимог зразково уполовину (Jones). Методи контролю якості, що визначають помилки у вимогах на ранніх стадіях, скорочують кількість подальших модифікацій. Щоб зменшити вплив зміни вимог, відкладайте реалізацію тих з них, які, швидше за все, зміняться, поки вони не будуть визначені, а також проектуйте систему, закладаючи можливість легкої модернізації.

Процес зміни вимог. Ризик, пов'язаний з тим, як відбувається зміна вимог, залежить і від відсутності визначеного процесу змін, використання неефективних механізмів змін і внесення коректівів без урахування цього процесу. Розвиток культури і дисципліни управління змінами вимагає часу. Дуже важливо, щоб при зміні вимог застосовувався аналіз дії пропонованих змін, рішення ухвалював радий з управління змінами і використовувався інструментальний засіб для підтримки певної процедури.

Нереалізовані вимоги. Матриця трасованості вимог допомагає уникнути пропуску яких-небудь вимог під час проектування, конструювання або тестування.

Збільшення об'єму проекту. Якщо вимоги спочатку погано визначені, подальше їх виявлення може збільшити об'єм проект!. Реалізація нечітко певних областей продукту зажадає більше зусиль, чим очікувалося. Ресурси проекту, розподілені відповідно до початковим неповним вимогам, можуть виявитися недостатніми для задоволення повного спектру потреб користувача. Для пом'якшення цієї риски плануйте життєвий цикл, що складається з поетапних або розроблених засобами инкрементального моделювання версій. Реалізуйте базову функціональність в першому випуску і нарощуйте здібності системи далі

9.2. Керування ризиками.

Менеджери проектів по розробці ПЗ повинні виявляти ризик і управляти ним, починаючи з чинників, пов'язаних з вимогами.

Ризик (risk) - це умова, яка може спричинити будь-які втрати або іншим способом поставити під загрозу успіх проекту. Ця умова ще не породила проблему, і ви хочете, щоб так воно і залишалося. Ці потенційні проблеми можуть надати несприятливі дії на вартість, терміни, технічний успіх, якість продукту або ефективність роботи команди.

Управління ризиком - краща практика індустрії ПЗ (Brown, 1996) - це процес виявлення, оцінки і управління ризиком до того, як він завдасть збитку вашому проекту.

Оскільки ніхто не може упевнено передбачати майбутнє, *управління ризиком* - це спосіб мінімізації вірогідності потенційних проблем або збитку від них. Управління ризиком означає роботу над потенційною небезпекою до того, як вона перейде в кризову фазу. Це покращує вірогідність успіху проекту і зменшує фінансові або інші наслідки риски, якої і,е вдасться уникнути. Ризик, поза сферою компетенції команди слід передати керівництву відповідного рівня.

Оскільки вимоги такі важливі в проектах по створенню ПЗ, передбачливий менеджер вже на ранніх стадіях проекту виявить ризик, пов'язаний з вимогами, і буде досить агресивно контролювати його. Поширені чинники ризику, пов'язані з вимогами, включають невірне розуміння вимог, недостатнє залучення користувачів, неточності або зміни в масштабах і цілях проекту, і вимоги, що постійно змінюються. Менеджери проектів можуть управляти ризиком, пов'язаним з вимогами, тільки в співпраці з клієнтами або їх представниками. Сумісне документування чинників ризиків, пов'язаних з вимогами і планування дій із зменшення їх наслідків укріплює партнерство клієнта і розробника.

Проекти підстерігає маса ризиків, окрім тих. що пов'язані з межами проекту і вимогами. Один з них - залежність від зовнішніх елементів, таких, як субпідрядник або інший проект, що надає компоненти для повторного використання. При управлінні проектом на перше місце виходять такі види ризиків, як неточні оцінки, неприйняття менеджерами точних оцінок, недостатня прозорість статусу проекту і текучка кадрів. Технологічний ризик представляє загрозу для дуже складних або таких, що використовують передові розробки проектів.

Управління ризиком (risk management) − це застосування інструментальних засобів і процедур для обмеження чинників ризиків в проекті прийнятними рамками. Управління ризиком надає стандартний підхід до виявлення і документування чинників ризику., оцінці їх потенційного збитку і пропонує стратегії їх пом'якшення (Williams, Walker, і Dorofee, 1997). Управління ризиком включає а себе дії, показані на рис. 15.

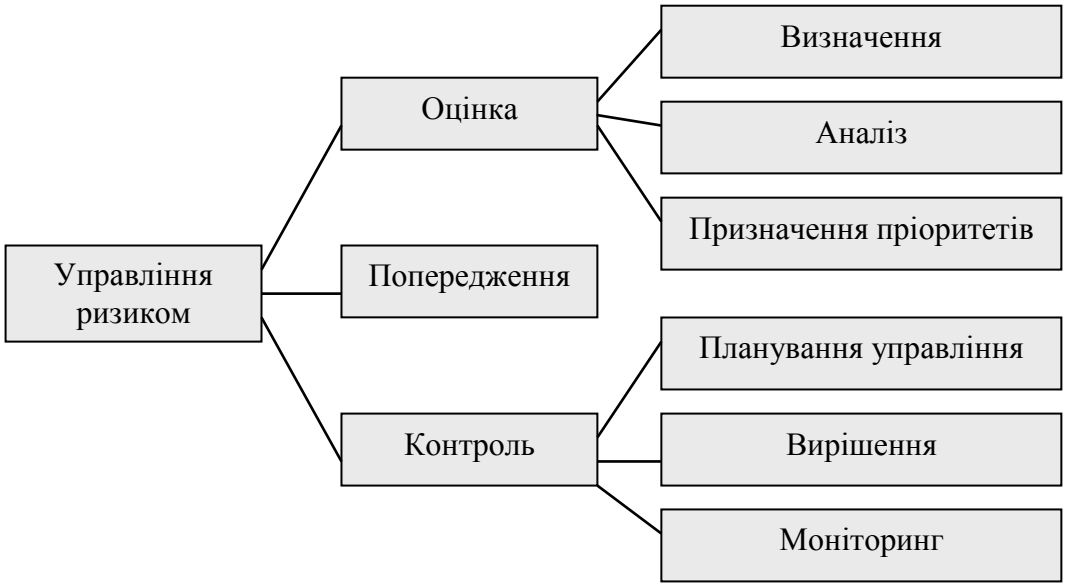


Рис. 15. Елементи управління ризиками

Оцінка ризиків (risk assessment) - це процес дослідження проекту для виявлення областей потенційної ризиків. Складаючи списки чинників ризиків, типових для розробки ПЗ, ви полегшите завдання виявлення чинників ризиків. При аналізі, ризиків досліджують потенційні наслідки конкретних чинників ризиків для вашого проекту. Визначення пріоритетів чинників ризиків допоможе вам концентруватися на найбільш небезпечних, оцінюючи потенційну схильність кожного з них. Схильність - це функція, що включає як вірогідність потерпіти утрату із-за ризиків, так і масштаби цієї утрати.

Запобігання ризику (risk avoidance) - це один із способів вирішення питання: не робіть того, що ризиковано. Ви можете уникнути ризиків, не починаючи певні проекти, покладаючись на випробувані, а не самі передові технології або виключаючи функції, які буде особливо важко реалізувати вірно.

Велику частину часу вам доведеться виконувати *контроль ризиків* (risk control), що дозволить управляти виявленими чинниками ризиків з високим пріоритетом. Планування управління ризиком має на увазі створення плану дій для кожного окремого чинника, включаючи методи пом'якшення, плани на випадок непередбачених обставин, відповідальних осіб і терміни виконання. Мета дій з пом'якшення дії ризиків - або не дозволити ризику стати проблемою, або зменшити його шкідливу дію. Ризик не контролюватиме сам себе, тому дозвіл ризиків має на увазі реалізацію планів ПЗ забороні кожної ризиків. І, нарешті, трасуйте своє просування до дозволу кожної ризиків за допомогою моніторингу ризиків, яка повинна стати частиною вашого стандартного трасування статусу проекту. Відстежуйте, наскільки добре працюють ваші дії з пом'якшення ризиків, шукайте нові чинники ризиків, що виникли недавно, прибирайте ризики, загроза яких минула, і періодично оновлюйте пріоритети у вашому списку чинників ризиків.

Документування ризиків проекту. Недостатньо просто знати ризики, загрозливі вашому проекту. Вам потрібно управляти ними так, щоб мати можливість повідомляти про області і статус ризиків зацікавленим в проекті особам впродовж проекту. На рис. 15 показаний шаблон документування окремої області ризиків. Іноді зручно зберігати його в електронній таблиці, що дозволяє легко сортувати список областей ризиків. Замість того щоб включати його в свій план управління проектом або в специфікацію вимог до ПЗ, ведіть список областей ризиків як окремий документ - так його легко оновлюватиме впродовж проекту.

Ідентифікатор:
<порядковыйномер>
Дата відкриття:
<дагаг коли елемент ризиків було виявлено>
Дата закриття:
<дата, коли елемент ризику був закритий>
Опис:
<опис елементу ризиків у формі «причина - слідство» >
Вірогідність:
<ймовірність того, що елемент ризиків стане проблемой'>
Вплив:
<потенційні втрати, які можуть бути нанесені, якщоелемент ризику стане проблемою>
Схильність:
<вірогідність, помножена на втрати>
План пом'якшення ризиків:
<один або декілька методів управління, уникнення, зменшення
наслідків інших способів мінімізації ризиків>
Відповідальна особа:
< людина, що відповідає за дозвіл елементу ризиків >
Термін виконання
<дата, до якої дії з пом'якшеннюяризиків повинні бути виконанні>

Рис. 16. Шаблон трасування елементу ризиків

Використовуйте формат «причина - наслідок», документуючи чинники ризиків. Тобто спочатку формулюйте причину ризиків, а потім її потенційний несприятливий результат-слідство. Часто люди, що говорять про ризик, приводять тільки умову («клієнти не прийдуть до єдиної думки щодо вимог до продукту») або тільки слідство («ми зможемо задовольнити, лише одного з наших основних клієнтів»). Одна причина може спричинити декілька наслідків, і декілька причин можуть привести до одного наслідку.

Визначите декілька загрозливих вашому поточному проекту ризиків, пов'язаних з вимогами. Не позначайте поточні проблеми як чинники ризику, відзначайте тільки те, що ще не трапилося. Документуйте ризики, використовуючи шаблон. Запропонуєте принаймні один можливий метод помякшення для кожного ризику.

Проведіть «мозковий штурм» ризиків з ключовими особами, зацікавленими в проекті. Виявите якомога більше чинників ризиків, пов'язаних з вимогами. Оцініте кожен чинник ПЗ вірогідності його реалізації і відносному впливу і перемножте ці величини для обчислення схильності цьому ризику.. Відсортуйте список ПЗ зменшенню схильності, щоб визначити п'ять найсерйозніших чинників ризиків, пов'язаних з вимогами. Призначте для кожного з них відповідального за виконання дій з пом'якшення

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вигерс Карл Разработка требований к программному обеспечению. Пер, с англ. − М.:Издательско-торговый дом «Русская Редакция», 2004. − 576с.
2. Марка Д.А. Методология структурного анализа и проектирования – С.-Пб.: Питер, 1995. – 235 с.
3. Лешек А. Мацяшек Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ. - М.: Издательский дом "Вильямс". − 2002. − 432 с.
4. Орлик С., Булуй Ю. Введение в программную инженерию и управление жизненным циклом ПЗ Программная инженерия. Программные требования. Режим доступу: http://www.sorlik.ru/swebok/3-1-software_engineering_requirements.pdf
5. Фаулер Скотт До. UML в короткому викладі. Застосування стандартної мови об'єктного моделювання: Пер. з англ. – М.:Мир, 1999. – 191 с.
6. Алістер Коберн. Сучасні методи опису функціональних вимог до систем
7. Л.Новіков. Введення в Rational Unified Process. - Режим доступу: http://www.interface.ru/rational/interface/151199/rup/main.htm
8. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с.
9. Маклаков С.В. Bpwin, Erwin, Case-средства разработки информационных систем. – Москва “ДиалогМифи ” – 2000
10. Орлов С. Технологии разработки программного обеспечения: Учебник. – СПб.: Питер, 2002. – 464 с.
11. Каменова, Громов. Моделирование бизнеса. Методология ARIS. — М.: Весть-МетаТехнология, 2001.