МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерних наук

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ

з дисципліни
"РОЗРОБКА AGILE ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ"

для студентів спеціальності

8.05010302 "Інженерія програмного забезпечення"

Тернопіль 2012

А.В. Пукас // Опорний конспект лекцій з дисципліни „РОЗРОБКА AGILE ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ” для студентів спеціальності 8.05010302 “Інженерія програмного забезпечення”. – Тернопіль, 2012. – 100 с.

Укладач: **Пукас Андрій Васильович, к.т.н.,** доцент кафедри комп’ютерних наук ТНЕУ;

Рецензент: доцент кафедри комп’ютерних технологій Тернопільського національного педагогічного університету ім. В.Гнатюка, к.т.н., доцент **Рак В.І.**

Затверджено на засіданні кафедри комп’ютерних наук ТНЕУ.
Протокол № __ від «___» _____ 2012 р.

Зміст

Лекція 1. Вступ в Agile.

1. Поняття Agile.
2. Extreme Programming.
3. Scrum.

1. Поняття Agile

"Гнучкі" (agile) методи розробки ПЗ з'явилися як альтернатива формальним і "поважним" методологіям, на кшталт CMM і RUP. Талановиті програмісти не бажаючи перетворення розробки ПЗ на рутину, хочуть мати максимум свобод і обіцяють натомість високу ефективність. З іншого боку, практика показує, що "поважні" методології в значній кількості випадків неефективні. Основними положеннями гнучких методів, закріплених в Agile Manifesto в 2007 році є наступні:

- індивідуали і взаємодія замість процесів і програмних засобів;
- працююче ПЗ замість складної документації;
- взаємодія із замовником замість жорстких контрактів;
- реакція на зміни замість того, щоб дотримуватися плану.

Фактично, гнучкі методології говорять про невеликі, такі, що само організуються, команди, що складаються з висококваліфікованих і енергійних людей, орієнтованих на бізнес, тобто, наприклад, які розробляють свій власний продукт для випуску його на ринок. У цього підходу є, очевидно, свої плюси і свої мінуси.

2. Extreme Programming

Найвідомішим гнучким методом є Extreme Programming (відома скорочена назва - XP). Він був створений талановитим фахівцем в області програмної інженерії Кентом Беком в результаті його роботи в 1996-1999 роках над системою контролю платежів компанії "Крайслер".

Модель процесу по XP виглядає як часта послідовність випусків (releases) продукту, таких частих, наскільки це можливо. Але при цьому обов'язково, щоб у випуск входила нова діюча функціональність. Нижче перераховані основні принципи організації процесу по XP.

1. Планування (Planning Game), засноване на принципі, що розробка ПЗ є діалогом між можливостями і бажаннями, при цьому змінюються і те і інше.
2. Простий дизайн (Simple Design) - проти надмірного проектування.
3. Метафора (Metaphor) - суть проекту повинна уміщатися в 1-2 містких фразах або в деякому образі.
4. Рефакторинг (Refactoring) - процес постійного поліпшення (спрощення) структури ПЗ, необхідний у зв'язку з додаванням нової функціональності.
5. Парне програмування (Pair Programming) - один програмує, інший думає над підходом в цілому, про нові тести, про спрощення структури програми і так далі.
6. Колективне володіння кодом (Collective Ownership).

7. Участь замовника в розробці (On-site Customer) - представник замовника входить в команду розробника.

8. Створення і використання стандартів кодування (Coding Standards) в проекті - при написанні коду (створюються і) використовуються стандарти на імена ідентифікаторів, складання коментарів і так далі.

9. Тестування - розробники самі тестують своє ПЗ, поділяючи цей процес з розробкою. При цьому рекомендується створювати тести до того, як буде реалізована відповідна функціональність. Замовник створює функціональні тести.

10. Безперервна інтеграція. Сама розробка представляється як послідовність випусків.

11. 40-годинний робочий тиждень.

Проте в повному об'ємі XP не була використана навіть її авторами і являється, швидше, філософією. Крім того, відомі і впроваджуються окремі практики XP, як, наприклад, парне програмування, колективне володіння кодом, і, звичайно ж, рефакторинг коду. Ідея простого, ненадмірного дизайну проекту також зробила значний вплив на світ розробників ПЗ.

Більш практичним "гнучким" методом розробки є Scrum.

3. Scrum

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

Загальний опис. Метод Scrum дозволяє гнучко розробляти проекти невеликими командами (7 чоловік плюс/мінус 2) в ситуації вимог, що змінюються. При цьому процес розробки є ітеративним і надає велику свободу команді. Крім того, метод дуже простий - легко вивчається і застосовується на практиці. Його схема зображена на рис. 1.1.

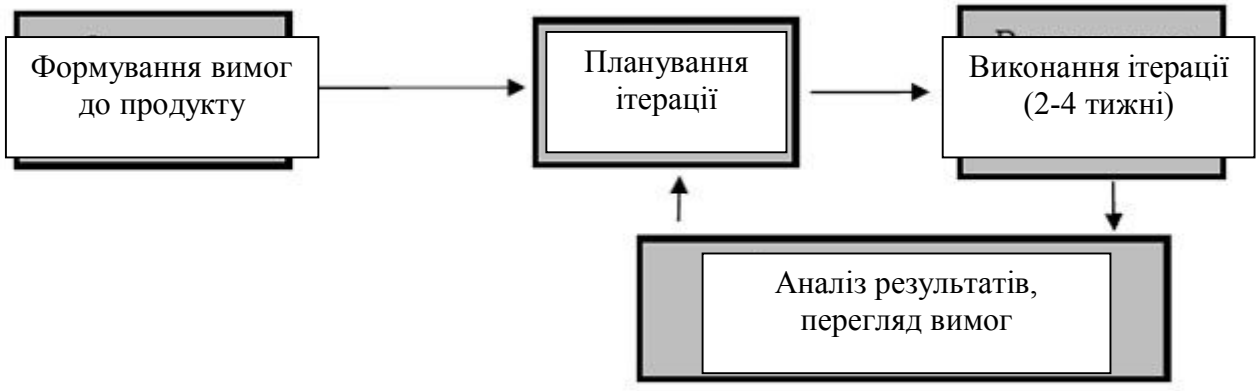


Рис. 1.1.

Спочатку створюються вимоги до усього продукту. Потім з них вибираються найактуальніші і створюється план на наступну ітерацію. Впродовж ітерації плани не міняються (цим досягається відносна стабільність розробки), а сама ітерація триває 2-4 тижні. Вона закінчується створенням працездатної версії продукту (робочий продукт), яку можна пред'явити замовникові, запустити і продемонструвати, хай і з мінімальними функціональними можливостями. Після цього результати обговорюються і вимоги до продукту коригуються. Це зручно робити, маючи після кожної ітерації продукт, який вже можна якось використати, показувати і обговорювати. Далі відбувається планування нової ітерації і усе повторюється.

Усередині ітерації проектом повністю займається команда. Вона є плоскою, ніяких ролей Scrum не визначає. Синхронізація з менеджментом і замовником відбувається після закінчення ітерації. Ітерація може бути перервана лише в особливих випадках.

Ролі. У Scrum є всього три види ролей.

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

Scrum-майстри (Scrum Master) забезпечують максимальну працездатність і продуктивну роботу команди - як виконання Scrum-процесу, так і рішення господарських і адміністративних завдань. Зокрема, його завданням є відмежування команди від усіх дій ззовні під час ітерації.

Scrum-команда (Scrum Team) - група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Першим завданням команди є постановка для ітерації реально досяжних і пріоритетних для проекту в цілому завдань (на основі Project Backlog і при активній участі власника продукту і Scrum-майстра). Другим завданням є виконання цього завдання щоб то не було, у відведені терміни і із заявленою якістю. Важливо, що команда сама бере участь в постановці завдання і сама ж його виконує. Тут поєднується свобода і відповідальність, подібно до того, як це організовано в MSF. Тут же "просвічує" дисципліна зобов'язань.

Практики. У Scrum визначені наступні практики.

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

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

зустрічі? У цій нараді може брати участь будь-яка зацікавлена особа, але тільки учасники Scrum -команди мають право приймати рішення. Правило обгрунтоване тим, що вони давали зобов'язання реалізувати мету ітерації, і тільки це дає упевненість в тому, що вона буде досягнута. На них лежить відповідальність за їх власні слова, і, якщо хтось з боку втручається і приймає рішення за них, тим самим він знімає відповідальність за результат з учасників команди. Такі зустрічі підтримують дисципліну зобов'язань в Scrum-команді, сприяють утриманню фокусу на цілях ітерації, допомагають вирішувати проблеми "у зародку". Зазвичай такі наради проводяться стоячи, впродовж 15-20 хвилин.

Sprint Review Meeting. Проводиться у кінці кожного Sprint. Спочатку Scrum -команда демонструє Product Owner зроблену впродовж Sprint роботу, а той у свою чергу веде цю частину мітингу і може запросити до участі усіх зацікавлених представників замовника. Product Owner визначає, які вимоги з Sprint Backlog були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в Sprint Backlog для наступної ітерації. У другій частині мітингу проводиться аналіз минулого спринту, який веде Scrum -майстер. Scrum -команда аналізує в останньому Sprint позитивні і негативні моменти спільної роботи, робить висновки і приймає важливі для подальшої роботи рішення. Scrum -команда також шукає шляхи для збільшення ефективності подальшої роботи. Потім цикл повторюється.

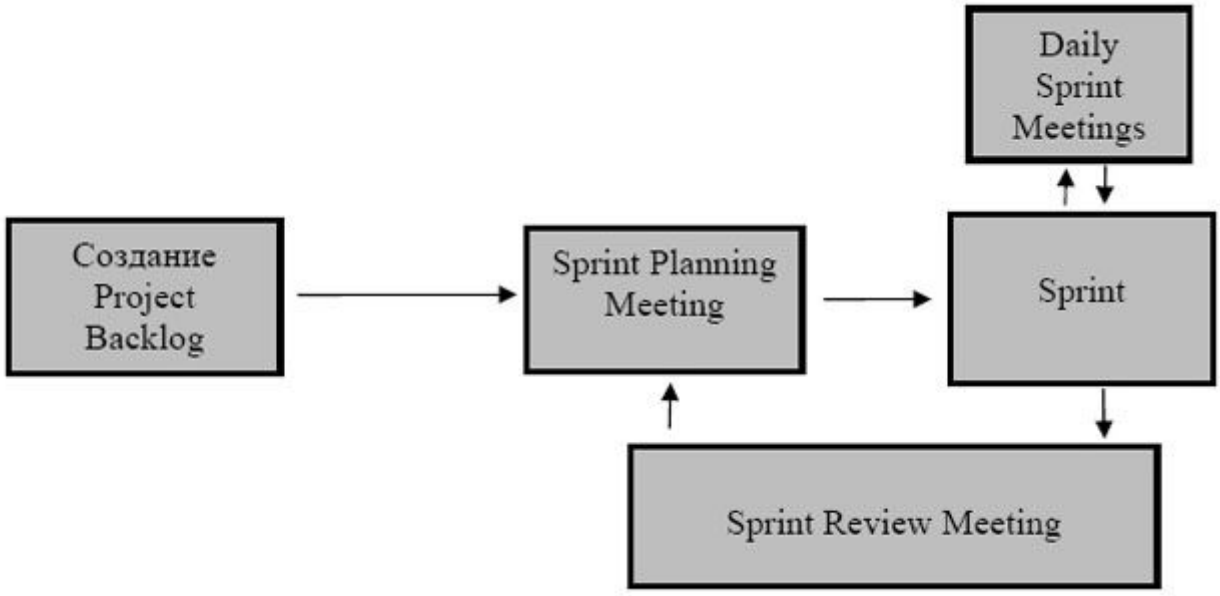


Рис. 1.2.

Лекція 2. Огляд методології Microsoft Solutions Framework для Agile

1. Загальні визначення.
2. Аспекти.
3. Принципи.
4. Основи роботи

1. Загальні визначення

Ролі .У гнучкій методології Microsoft Solutions Framework (MSF) розробки ПЗ усіх осіб, що беруть участь у виробництві, використанні і супроводі продукту, мають рівні повноваження. Учасники команди мають різні ролі, пов'язані з їх функціями, при цьому жодна з ролей не вважається важливіше за іншу : таке ділення гарантує реалізацію якісного рішення. Члени команди можуть виступати в одній або декількох ролях.

Дії і операції. Ролі виконують *операції* (activity), які можуть бути згруповані в *дії* (workstream). Іншими словами, дія — це набір операцій.

Операції призводять до виникнення кінцевих продуктів і можуть вимагати для свого виконання кінцеві продукти як результати попередніх операцій.

Кінцеві продукти. *Кінцеві продукти* (deliverables) — це документи, електронні таблиці, проектні плани, початкові тексти програм і інші результати операцій.

Цикли і ітерації. За допомогою *циклів* описується періодичність виконання різних дій, а також частота випуску і оновлення кінцевих продуктів. Виконання проектів і завдань, що входять в них, проводиться циклічно.

Тісна інтеграція гнучкої методології MSF розробки ПЗ з Visual Studio Team System забезпечує прискорену *ітеративну розробку* з постійним уточненням деталей і вдосконаленням кінцевого продукта. Визначення вимог до продукту, розробка і тестування — це повторюються дії, що перекриваються між собою, ведуть до поступового завершення проекту.

Рис. 2.1

Вклад різних ітерацій на завершення проекту різний. Наприклад, короткі ітерації дозволяють понизити погрішність попередніх оцінок і надають оперативні зведення про точність проектних планів. У результаті кожної ітерації повинна з'являтися стабільно працююча частина системи.

Якість. *Якість робіт* є одним із засадничих принципів в гнучкій методології MSF розробки ПЗ. Усі учасники проекту повинні усвідомлювати

важливість якісної роботи і керуватися цим принципом при проектуванні, реалізації, тестуванні і впровадженні системи. Особливу увагу слід приділяти якості в таких областях, як безпека, продуктивність і інтерфейс користувача.

Треки

Рис. 2.2

Треки служать для угрупування операцій, що завершуються контрольними точками, тобто що мають певний бюджет і графік. Треки не завжди пов'язані із завданнями або роботами, що виконуються в циклах. Треки можуть накладатися один на одного, а між операціями, що виконуються в різних треках, відбувається постійний обмін інформацією.

Дисципліни. *Дисципліни* — це що відносяться до проекту в цілому треки, в яких задіяні усі учасники команди.

Керівництво. *Керівництво* (governance) — це контроль бюджету і графіку виконання проекту в прив'язці до поточних результатів. У гнучкій методології MSF розробки ПЗ визначені п'ять контрольних точок керівництва, в кожній з яких дається відповідь на певне питання. Не слід плутати треки керівництва з впорядковуванням робіт за допомогою їх групування в ітерації з циклами усередині.

Описувачі. *Описувач* (work item) — це запис у базі даних, яка використовується в Visual Studio Team Foundation для відстежування призначення певної операції або дії виконавцям, а також для моніторингу стану цього завдання. У гнучкій методології MSF розробки ПЗ визначені п'ять типів описувачів для призначення і відстежування робіт. Дані, що зберігаються в загальній базі цих описувачів і в сховищі показників, дозволяють в реальному часі відстежувати стан проекту.

Описувачі характеризуються *станом* (state), наприклад стани «новий дефект» і «закритий дефект». При *переході* (transition) з одного стану в інший потрібно *основу* (reason), наприклад від стану «новий дефект» відразу до стану «закритий дефект» на підставі того, що це дублікат вже існуючого дефекту.

2. Аспекти

Методологія MSF — більше ніж набір правил, яким необхідно сліпо слідувати. Вона покликана прищепити культуру, що сприяє реалізації успішних проектів. *Аспект* — це сукупність показників, визначальна, як окремий учасник проекту оцінює ту або іншу ситуацію і реагує на неї. Учасники проекту від його початку до завершення повинні приймати до уваги аспекти при ухваленні рішень, розставлянні пріоритетів, представленні своїх інтересів і при взаємодії з іншими учасниками команди і іншими зацікавленими особами. У MSF представлено дев'ять аспектів.

Розуміння практичної цінності. Перший аспект акцентує увагу учасників проекту на його практичних результатах. Кожна програмна система створюється з конкретною метою, наприклад для вирішення деякої проблеми у бізнесі. При побудові програмних систем, особливо великих, вони випробовують на собі безліч дій, частина з яких вимагає у відповідь докладання значних зусиль. У кінцевому ж рахунку програмний проект оцінюється по його корисності для бізнесу. Кожен учасник повинен завжди пам'ятати про практичні результати і має бути націлений на їх досягнення.

Представлення інтересів

З програмними проектами зазвичай пов'язані багато зацікавлених осіб зі своїми інтересами, наприклад, замовникам потрібна добре працююча система. Можна також представити неживих «зацікавлених осіб», наприклад саму систему, що розробляється. Система може бути розрахована на роботу впродовж тридцяти років, а може використовуватися тільки тридцять хвилин. Системи, розраховані на тривале використання, розробляються довше. Кожна із зацікавлених сторін (будь то люди або ні) в проекті має бути представлена захисником своїх інтересів. Кожному учасникові проекту важливо розуміти, чиї інтереси він представляє. Слід пам'ятати, що модель представлення інтересів — це сукупність обмежень, тому для отримання оптимальних результатів переважно неминучі компроміси.

Гордість за участь в проекті

Почуття гордості за участь в проекті — важливий чинник для отримання якісного продукту. З почуття гордості виростає мотивація і відповідальність за виконання проекту. Розробка ПЗ схожа на мистецтво, і якісні системи — улюблені дітища їх творців. Комфортні умови праці, довіра, можливості росту, мотивовані, професійні люди — усе це що становлять, необхідні для створення якісного ПЗ. Підтримка почуття гордості за участь в проекті — завдання і учасників проекту, і організації. Застосовуйте визначення загальних витрат в проекті за принципом «від частки до загального», дайте проекту кодове ім'я і чітко ідентифікуйте команду проекту — це допоможе підтримувати почуття гордості. У MSF представники усіх ролей відповідальні за створення продукту найвищої якості.

Своєчасне виконання своїх обов'язків

Багато завдань при розробці ПЗ пов'язані між собою. Бувають моменти, коли ми не можемо виконати свою роботу самостійно, без участі колег. Для ефективної роботи треба своєчасно вирішувати завдання, від яких залежить діяльність інших учасників проекту, і тримати їх в курсі поточного стану справ. Ви маєте бути надійним і заслуговуючим довіри партнером для своїх колег.

Бачення системи в цілому

При роботі над конкретною частиною системи є ризик «не побачити за деревами ліси». Важливо не лише представляти свій відрізок роботи, але і розуміти, як ваша діяльність відбивається на кінцевому продукті. Потрібно приділяти більше уваги підсумку, а не процесу. Це не означає, що процес поганий або не важливий: процес не самоціль, а засіб досягнення кінцевого результату. Треба чітко розуміти, навіщо ви виконуєте те або інша дія і як результати вашої роботи інтегруються в загальне рішення. Думайте про те, що треба для реалізації системи, не вдаючись до дрібних деталей. Регулярно — не лише під час планових зустрічей — спілкуйтеся з іншими учасниками команди.

Рівнозначність учасників

Аспект рівнозначності учасників команди означає їх рівноправ'я і рівноцінність, незалежно від їх ролей і інтересів, що представляються ними. Він може бути забезпечений за рахунок відсутності обмежень в комунікаціях і прозорості проекту. Цей аспект також припускає взаємоповагу і турботу про кожного члена колективу. Атмосфера взаємоповаги — необхідна умова максимальної ефективності в роботі будь-якої команди: вона забезпечує високу колективну відповідальність, ефективні комунікації і командний дух. Для успішної роботи в команді рівнозначних учасників представники усіх ролей повинні піклуватися про якість створюваного продукту, діяти як представники інтересів замовника і розуміти вирішувану бізнес-проблему.

Господарський підхід

До ресурсів проекту, корпоративним і обчислювальним ресурсам треба відноситися по-господарському. Такий підхід треба застосовувати в усьому — від використання ефективних методів управління проектом до оптимізації системних ресурсів. Далекоглядним кроком буде постачання описувачів помилок детальними даними, які допоможуть комусь в подальшому. Також корисно давати точну оцінку трудовитрат на виконання завдань розробки і тестування. Існуючі ресурси повинні використовуватися скрізь, де це можливо.

Безперервне навчання

Готовність до навчання включає постійне самоудосконалення учасника команди шляхом накопичення знань і обміну ними з іншими. Потрібно вчитися на чужому досвіді: не повторювати помилок і застосовувати успішні підходи. У графіці проекту треба передбачити час для навчання членів команди, аналізу поточного стану справ і виконаної роботи. Крім того, важливою індивідуальною рисою кожного учасника команди має бути прагнення до обміну знаннями з іншими.

Прихильність якості

Аспект прихильності якості припускає такий підхід до планування і реалізації рішення, який враховує усі потреби кінцевого користувача. При цьому

якість в таких областях, як продуктивність або безпеку, повинна враховуватися на усіх етапах розробки. Без такого підходу до забезпечення якості систему, як правило, створюють, роблячи неявні допущення про її поведінку. В результаті функціонування системи не задовольняє замовника. Для забезпечення якості треба бачити систему в цілому і не покладатися на неявні допущення, а керуватися явно сформульованими вимогами до якості.

3. Принципи

Microsoft Solutions Framework (MSF) for Agile Software Development — це методологія побудови .NET-застосувань і іншого об'єктно-орієнтованого ПЗ. У її основі лежить гнучкий, керований, адаптивний до контексту проекту процес розробки. Безпосередньо у гнучку методологію MSF розробки ПЗ включені норми роботи з вимогами до якості в таких областях, як продуктивність і безпека. У цій методології також враховуються конкретні умови для реалізації кожного проекту. При такому підході створюється адаптивний процес, що забезпечує усунення обмежень більшості гнучких процесів розробки ПЗ і досягнення цілей, встановлених концепцією проекту.

Партнерство із замовниками

У моделі команди MSF, заснованої на представленні інтересів, ключова увага приділяється розумінню потреб замовника і участі представників замовника в реалізації проекту. Головний пріоритет для будь-якої професійної команди — діяти так, щоб клієнти були задоволені результатами. Орієнтуватися на клієнта — означає розуміти його проблеми. Розібравшись у вирішуваній проблемі клієнта, потрібно залучити його в роботу в тому об'ємі, який відповідає його очікуванням. На усіх фазах проекту треба підтримувати відкрите, активне, регулярне спілкування із замовником. Це важливо тому, що часто тільки замовник бачить різницю між дійсними і уявними проблемами бізнесу.

Єдина точка зору

У MSF наполегливо рекомендується виробити єдину точку зору на підходи до реалізації рішення. Загальний погляд усіх учасників команди гарантує, що вони однаково розуміють, який буде результат їх роботи; вони об'єднуються навколо єдиної мети і однаково трактують потреби замовника. Спільна робота однодумців завжди ефективніша, оскільки рішення приймаються не довільно, а грунтуючись на загальному баченні проблеми. Без єдиної точки зору учасники команди можуть мати протирічні уявлення про цілі роботи, а досягти потрібних результатів в цьому випадку складніше. Навіть після того, як результат отриманий, не усі учасники можуть погодитися з тим, що він виявився успішний. Розуміння переваг виробленого рішення і уміння їх сформулювати часто є ключовим чинником успіху.

Інкрементна видача результатів.

Ніщо так не завойовує довіру замовника, як часта видача результатів. Дуже вигідно постійно мати «практично готовий» продукт. Реагування на потреби замовника регулярною видачею невеликих працездатних доповнень наочно демонструє прогрес розробки. При частій видачі результатів для замовника існує гарантія працездатності команди і розвитку процесу й інфраструктури. При цьому ризики, помилки і упущені вимоги виявляються на ранніх стадіях. Інкрементний

підхід підтверджує правильність проектних рішень і забезпечує їх коректування завдяки ефективному зворотному зв'язку.

Для частої видачі результатів робота має бути розбита на невеликі фрагменти, результати повинні видаватися точно по графіку, а у разі декількох варіантів рішення повинні надаватися вони усі, а не один успішно вибраний.

Плануйте, виконуйте плани, оцінюйте прогрес і темп роботи команди на основі інкрементної видачі результатів — і ви збільшите рентабельність. Мінімізуйте діяльність, що не приносить зрозумілих замовникові конкретних результатів. Застосовуйте ітерації для підтримки ритму видачі тих результатів, які ваш клієнт здатний оцінити. Уважно оцінюйте ефективність передачі робіт від одного учасника команди іншому. Розробники повинні постійно перевіряти створюваний продукт, а ваша компанія — випробовувати нові версії.

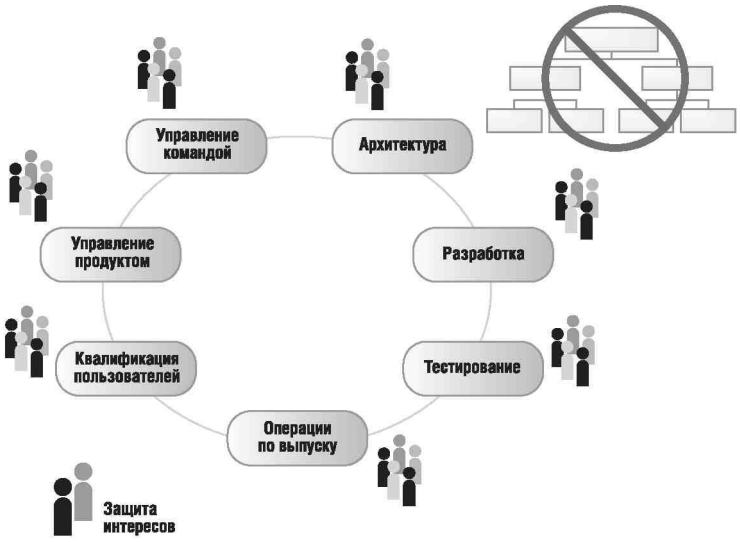
Інвестиції в якість

В успішній команді кожен учасник повинен відчувати свою відповідальність за продукт, що розробляється, і бути представником інтересів замовника, піклуючись про якість упродовж усього життєвого циклу розробки. Якість повинна враховуватися в планах і графіках. Використовуйте асигнування на виправлення дефектів (заплановані ітерації для усунення несправностей) для зниження загальних витрат на помилки. Таким чином дещо знижуються темпи розробки, проте забезпечується резерв часу в подальших ітераціях для зменшення кількості дефектів.

Широкі повноваження учасників проекту

Команда працює ефективно, коли кожному учасникові надані усі необхідні повноваження для виконання його обов'язків і він упевнений, що якщо його робота залежить від колег, вона буде виконана. У свою чергу, замовник має право вважати, що команда виконає свої зобов'язання, і може будувати свої плани, виходячи з цього припущення. У разі можливої затримки або зміни функцій необхідно своєчасно повідомити про це клієнта.

Модель команди



Модель команди MSF описує підхід компанії Microsoft до структуризації учасників команди і їх дій, що призводять до успіху проекту. Фундаментальними принципами моделі команди MSF є:

■ команда — це група рівнозначних співробітників з чіткою підзвітністю,

відповідальністю, що розділяється, і вільним спілкуванням;

■ захист інтересів кожного ключового представника, залученого в проект, голос якого може вплинути на успіх;

■ необхідна гнучкість в масштабуванні команди.

Чітка підзвітність

Модель команди MSF грунтується на передумові, що цілі усіх учасників рівноцінні, сегменти діяльності у рамках проекту унікальні і при цьому немає єдиного представника усіх різноманітних цілей. При такому підході в команді рівнозначних учасників доцільно поєднувати чітку підзвітність перед зацікавленими сторонами з колективною відповідальністю за кінцевий результат. Кожен учасник підзвітний колективу (і організації, яка за ним стоїть) за досягнення цілей, встановлених для його ролі. Іншими словами, представник кожної ролі зобов'язаний звітувати за свій вклад в кінцевий результат. Відповідальність же в команді рівнозначних учасників розподіляється рівномірно. На те є причини: по-перше, неможливо виділити результат роботи окремого учасника із загального рішення, і, по-друге, команда працює ефективніше, коли кожен її учасник, що виконує будь-яку роль, бачить картину в цілому. Така взаємозалежність членів колективу повинна стимулювати їх інтерес до областей, за які вони не підзвітні, забезпечуючи повноцінне використання усього спектру знань, компетенції і досвіду команди.

Досягнення проекту відносяться до усіх учасників: вони розділяють усю загальну повагу і заслужену винагороду у разі вдалих рішень. Разом з цим, кожен учасник проекту повинен замислитися над підвищенням свого професійного рівня, витягаючи уроки з менш вдалих проектів.

Врахування будь-якого досвіду

Кожен проект, середовище і команда унікальні, отже будь-який проект і всяка його ітерація — це потенційне джерело додаткового досвіду. Проте не можна упізнати нічого нового без зворотного зв'язку і відкритості в колективі. Якщо не піклуватися про учасників команди, не давати їм відчути себе розкутим, зворотний зв'язок буде обмеженим і неефективним. Якщо ж забезпечити людям психологічний комфорт, кожен учасник і команда в цілому будуть націлені на самоудосконалення, поглиблення своїх знань і обмін ними з колегами. Як вже відзначалося, у графіку проекту треба врахувати час на навчання, у тому числі на основі попереднього і чужого досвіду. Детальний аналіз зробленого в доброзичливій атмосфері — ключовий принцип MSF, що забезпечує ефективні комунікації між членами команди.

Вільне спілкування

Історично у багатьох організаціях і проектах застосовувався принцип необхідного знання, відповідно до якого співробітники діставали доступ тільки до тих відомостей, які були потрібні для виконання конкретних функцій. Часто такий підхід призводить до непорозумінь і знижує шанси команди на досягнення успіху.

У MSF пропонується відкритий і чесний обмін інформацією як всередині команди, так і з ключовими зацікавленими сторонами поза нею. Вільний обмін інформацією не лише скорочує ризик виникнення непорозумінь і невиправданих витрат, але і забезпечує максимальний вклад усіх учасників команди в зниження існуючої в проекті невизначеності.

Гнучкість і готовність до змін

Чим більшу віддачу для бізнесу від впровадження нових технологій хочуть отримати організації, тим більше вони мають бути готові інвестувати в нові області. Не можна розраховувати на успіх, не освоюючи нові території. У MSF передбачається, що усе навкруги безперервно міняється, і захистити проект від цих змін неможливо. Застосування моделі проектної групи MSF гарантує участь усіх ролей і їх залученість в процес ухвалення рішень, обумовлених змінами, що відбуваються. Ця модель заохочує *гнучкість* (agility) при роботі в середовищі, що міняється. Участь усіх ролей в процесі ухвалення рішень забезпечує розгляд питань з урахуванням повного спектру точок зору.

4. Основи роботи.

Після створення проекту на базі гнучкої методології MSF в Team Foundation Server слід вивчити послідовність подальших дій для організації роботи над проектом і знайти необхідні додаткові відомості, які дозволять оптимально використати процес MSF.

Перші завдання. При створенні проекту формуються описувачі завдань, які дозволяють розподілити завдання і відразу приступити до роботи. Ці завдання можна проглянути в електронній таблиці Project Checklist.xls на порталі проекту або за допомогою Team Explorer (Провідника команди). Project Checklist.xls знаходиться в теці Project Management (Управління проектом) порталу проекту або в теці Documents (Документи) при перегляді в Team Explorer. Ці завдання також можна побачити, виконавши запит Project Checklist (Контрольний список проекту) в Team Explorer.

Умови завершення. У контрольному списку проекту містяться усі описувачі, помічені як завершення, що містять умови, для цієї ітерації. Наприклад, в описувачі завдання є поле Exit Criteria (Умови завершення). Завдання, у яких в цьому полі міститься значення Yes (Так), представлені в контрольному списку. Контрольний список проекту використовується для відстежування усіх критично важливих робіт, які мають бути виконані для успішного завершення ітерації.

Початкові завдання. Початкові завдання в контрольному списку проекту — це завдання, які мають бути завершені до початку першої ітерації. В першу чергу йдуть підготовчі дії з інформування учасників проекту про його початок, а закінчується список завданнями планування першої ітерації.

Актуалізація даних проекту. Зазвичай проектні роботи починаються до створення проекту команди на Team Foundation Server. Якщо вже є концепція проекту, які-небудь вимоги, а також призначені або вимагаючі відстежування завдання, тоді треба відразу відновити проектні дані.

Кінцеві продукти. Прогляньте усі шаблони кінцевих продуктів в теці Documents в Team Explorer або на порталі проекту. Якщо у вас вже є кінцеві продукти або супутня інформація, ви можете актуалізувати проект команди, завантаживши на його портал зовнішні документи або відновивши існуючі документи MSF на порталі проекту.

Описувачі. У MSF використовуються описувачі дефектів, ризиків, завдань, сценаріїв і вимог до якості. Якщо у вас вже є описувачі, застосовні в цьому

проекті, імпортуйте їх. Наприклад, якщо ви вже визначили які-небудь ризики, задокументуйте їх у вигляді описувачів ризиків за допомогою Team Explorer.

Початок дій. Робота над проектом розпочинається з того, що бізнес-аналітик формулює концепцію проекту. Цей етап є частиною виконання дії «Формулювання концепції проекту» (Capture Project Vision). Крім того, менеджер проекту починає планувати першу ітерацію у рамках виконуваної їм дії «Планування ітерації» (Plan an Iteration). Після запуску цих двох дій, природним чином починаються і інші дії, в які залучені учасники проекту з іншими ролями.

Типи описувачів. У MSF використовується п'ять типів описувачів. У описувачах кожного типу містяться дані для формування різних звітів. Щоб звіти відповідали своєму призначенню і виконання робіт відстежувалося адекватно, учасники проекту повинні коректно використати типи описувачів.

Запити. Стан робіт над проектом можна дізнатися за допомогою запитів, що видаються з теки Work Items (Описувачі) в Team Explorer. Для отримання додаткових відомостей про запити клацніть вкладку Index (Покажчик), а потім — Queries (Запити).

Звіти. На початку роботи над проектом немає значимих показників для створення корисних звітів, але з кожним днем з'являється все більше за даних, по яких можна судити про прогрес проекту. Передусім, слід розібратися із звітом Remaining Work (Робота, що залишилася). У першій і подальших ітераціях цей звіт дозволить вам стежити за ходом проекту. Для отримання додаткових відомостей про звіти клацніть вкладку Index (Покажчик), а потім — Reports (Звіти).

Лекція 3. Ролі учасників групи розробників ПЗ згідно MSF.

1. Бізнес-аналітик
2. Менеджер проекту
3. Архітектор
4. Розробник
5. Тестувальник
6. Реліз-менеджер
7. Адміністратор баз даних
8. Розробник баз даних

1. Бізнес-аналітик

У рамках командної моделі MSF (MSF Team Model) *бізнес-аналітик* бере участь в управлінні продуктом. Основне завдання бізнес-аналітика — розібратися в можливостях системи, що відносяться до бізнесу і розкрити їх команді. Він працює з користувачами і іншими зацікавленими особами, щоб розуміти їх потреби і завдання, трансформувати їх в конкретні визначення, сценарії і вимоги до якості, які команда розробників використовуватиме для побудови застосування. Крім того, бізнес-аналітик визначає очікування від функціональних можливостей системи і управляє ними. У проекті він представляє користувачів і бере участь в управлінні продуктом в тому сенсі, що постійно відстежує інтереси користувачів і замовників проекту. І, нарешті, бізнес-аналітики відповідають за забезпечення взаємодії між розробниками і користувачами. Людина, призначена на цю роль, має бути додана в групу доступу «Співробітник» (Contributor). Це дозволить йому виконувати усі функції, необхідні у рамках його діяльності, наприклад, такі як створення і зміну документів, описувачів і кінцевих продуктів.

Дії і операції ролі

■ Дії:
□ формулювання концепції проекту;
□ розробка вимог до якості
□ створення сценарію;
□ планування ітерації.
■ Операції:
□ написання концепції;

□ визначення збірних образів;
□ уточнення збірних образів;
□ визначення вимог до якості шляхом «мозкового штурму»;
□ створення моментальних знімків діяльності;
□ визначення пріоритетів в списку вимог до якості
□ написання вимог до якості
□ визначення вимог безпеки;
□ робота над сценаріями методом «мозкового штурму»;
□ визначення пріоритетів в списку сценаріїв;
□ оцінка завдання по розробці бази даних;
□ виконання ретроспективного аналізу;
□ оцінка сценарію;
□ оцінка вимоги до якості
□ складання графіку сценарію;
□ складання графіку реалізації вимог до якості
□ огляд цілей.

2. Менеджер проекту

У рамках командної моделі MSF *менеджер проекту* бере участь в управлінні продуктом. Основне завдання менеджера проекту — домагатися виконання поставлених перед командою завдань відповідно до графіку і у рамках бюджету. На менеджерові проекту лежать зобов'язання по плануванню і складанню графіку робіт, що включають розробку проекту і планів ітерацій, відстежування стану справ і складання звітів, а також визначенню ризиків і виробленню заходів по їх зменшенню. Менеджер проекту також проводить консультації з бізнес-аналітиками по плануванню сценаріїв і виробленню вимог до якості для кожної ітерації, консультується з архітекторами і розробниками для оцінки об'ємів робіт, радиться з тестувальниками, щоб спланувати тестування, і, крім того, сприяє взаємодії учасників команди. Людина, призначена на цю роль, має бути додана в групу доступу «Адміністратор проекту» (Project Administrator). Це дозволить йому виконувати такі функції, як створення нового проекту, формування нової команди і додавання в неї учасників.

Дії і операції ролі

■ Дії:
□ контроль ітерації;
□ планування ітерації;
□ ведення проекту.
■ Операції:
□ оцінка завдання по розробці бази даних;
□ моніторинг ітерації;
□ зниження ризику;
□ виконання ретроспективного аналізу;
□ визначення тривалості ітерації;
□ оцінка сценарію;
□ оцінка вимоги до якості
□ розбиття сценаріїв на завдання;

□ розбиття вимоги до якості на завдання;
□ складання графіку сценарію;
□ складання графіку реалізації вимог до якості
□ оцінка ходу виконання;
□ оцінка порогових значень показників тестів;
□ визначення ризику;
□ огляд цілей;
□ класифікація дефектів.

3. Архітектор

У рамках командної моделі MSF *архітектор* відповідає за архітектуру проекту. Його основне завдання — забезпечити успіх проекту шляхом розробки основних принципів застосування, які включають як організаційну конфігурацію системи, так і фізичну структуру її розгортання. При цьому архітектор повинен прагнути до зниження складності шляхом розділення системи на зрозумілі і прості частини. Архітектура застосування надзвичайно важлива, оскільки вона не просто встановлює етапи побудови системи, а визначає, чи буде застосування мати властивості, властиві успішним проектам. До них відносяться: зручність використання, надійність, практичність супроводу, продуктивність і безпека, а також можливості модифікації у разі зміни вимог. Людина, призначена на цю роль, має бути додана в групу доступу «Співробітник» (Contributor). Це дозволить йому виконувати усі функції, необхідні у рамках його діяльності, такі як створення і зміну документів, описувачів і кінцевих продуктів.

Дії і операції ролі

■ Дії:
□ розробка архітектури рішення;
□ планування ітерації.
■ Операції:
□ розділення системи на підсистеми;
□ визначення інтерфейсів;
□ розробка моделі загроз;
□ розробка моделі продуктивності;
□ створення архітектурної моделі;
□ створення архітектури інфраструктури;
□ визначення вимог безпеки;
□ розбиття сценаріїв на завдання;
□ розбиття вимоги до якості на завдання.

4. Розробник

У рамках командної моделі *MSF розробник* виконує розробку застосування. Його основне завдання — реалізувати застосування згідно із специфікаціями і у встановлені терміни. Розробник також допомагає уточнювати фізичний дизайн, оцінювати час і зусилля для виконання конкретних елементів, виконує реалізацію функцій або керує нею, готує продукт до впровадження і є експертом команди в технологічних областях. Людина, призначена на цю роль, має бути додана в групу доступу «Співробітник» (Contributor). Це дозволить йому виконувати усі функції,

необхідні у рамках його діяльності, такі як створення і зміну документів, описувачів і кінцевих продуктів.

Дії і операції ролі

■ Дії:

□ зборка продукту;
□ усунення дефекту;
□ реалізація завдання по розробці;
□ планування ітерації.

■ Операції:

□ розділення системи на підсистеми;
□ визначення інтерфейсів;
□ розробка моделі продуктивності;
□ запуск зборки;
□ перевірка випуску;
□ виправлення зборки;
□ приймання зборки;
□ відтворення дефекту;
□ визначення причини виникнення дефекту;
□ перепризначення дефекту;
□ вибір стратегії усунення дефекту;
□ створення або зміна тесту модуля;
□ виконання тесту модуля;
□ рефакторинг кода;
□ огляд коду;
□ інтеграція змін;
□ оцінка завдання по розробці;
□ кодування;
□ аналіз коду;
□ оцінка завдання по розробці бази даних;
□ виконання ретроспективного аналізу;
□ визначення тривалості ітерації;
□ оцінка сценарію;
□ оцінка вимоги до якості;
□ розбиття сценаріїв на завдання;
□ розбиття вимоги до якості на завдання;
□ створення нотаток про випуск;
□ розгортання продукту.

5. Тестувальник

У рамках командної моделі MSF *тестувальник* виконує завдання тестування продукту. Основним завданням тестувальника є виявлення проблем в продукті, які можуть несприятливо вплинути на його якість. Тестувальник зобов'язаний розуміти контекст проекту і допомагати іншим членам команди розуміти рішення, засновані на цьому контексті. Ключова мета тестувальника — пошук серйозних дефектів в продукті шляхом його тестування і подальший їх опис. Кожен знайдений дефект тестувальник повинен супроводжувати точним

описом шкідливої дії і запропонувати спосіб обійти дефект, щоб зменшити цю дію. Він повинен як можна простіше описати дефект і послідовність, в якій його можна відтворити.

Тестувальник бере участь в діяльності команди за визначенням стандартів якості продукту. Мета тестування — переконатися, що відомі функції працюють правильно, і виявити можливі проблеми продукту. Людина, призначена на цю роль, має бути додана в групу доступу «Співробітник» (Contributor). Це дозволить йому виконувати усі функції, необхідні у рамках його діяльності, такі як створення і зміну документів, описувачів і кінцевих продуктів.

Дії і операції ролі

■ Дії:

□ планування ітерації;
□ закриття дефекту;
□ тестування вимоги до якості
□ перевірка сценарію.

■ Операції:

□ виконання ретроспективного аналізу;
□ розбиття сценаріїв на завдання;
□ розбиття вимоги до якості на завдання;
□ перевірка виправлення;
□ закриття дефекту;
□ визначення підходу до тестування;
□ створення тесту продуктивності;
□ створення тесту безпеки;
□ створення стресового теста;
□ створення теста навантаження;
□ вибір і запуск тестового завдання;
□ документування дефекту;
□ оцінка порогових значень показників тестів.

6. Реліз-менеджер

У рамках командної моделі MSF *реліз-менеджер* відповідає за операції по випуску продукту. Основна мета реліз-менеджера — забезпечення випуску готового продукту. Він координує випуск продукту з фахівцями з експлуатації, створює план випуску продукту і сертифікує підготовлені до випуску версії для постачання або розгортання. Людина, призначена на цю роль, має бути додана в групу доступу «Адміністратор проекту» (Project Administrator). Це дозволить йому виконувати такі свої функції, як створення нового проекту, формування нової команди і додавання в неї учасників.

Дії і операції ролі

■ Дії:

□ випуск продукту.

■ Операції:

□ виконання плану випуску;
□ створення нотаток про випуск;
□ розгортання продукту.

7. Адміністратор баз даних

Основне завдання *адміністратора баз даних* в контексті розробки бази даних — підтримка створення проектів баз даних, а також внесення змін проекту в робочу базу даних. На додаток до цього він повинен виконувати традиційні завдання, такі як щоденне адміністрування і підтримка серверів баз даних.

Дії і операції ролі

■ Дії:

□ створення проекту бази даних;
□ розгортання проекту бази даних.

■ Операції:

□ створення проекту бази даних;
□ імпорт існуючої бази даних;
□ установка параметрів зборки і розгортання;
□ зміна згенерованих сценаріїв;
□ перевірка проекту бази даних;
□ розміщення проекту бази даних в службі управління початковим кодом;
□ синхронізація проекту бази даних;
□ перевірка зборки;
□ виконання тестування модулів бази даних;
□ аналіз змін;
□ створення резервної копії робочої бази даних;
□ установка бази даних на тестовому сервері;
□ установка бази даних.

8. Розробник баз даних

Основне завдання *розробника баз даних* — виконання комплексу завдань по розробці бази даних у встановлені терміни. Крім того, він відповідає за оцінку вартості, контроль над реалізацією функцій і допомогу іншим учасникам команди з питань, пов'язаних з базами даних. Розробник баз даних спільно з адміністратором баз даних і прикладними розробниками бере участь в ітеративному життєвому циклі розробки бази даних.

Дії і операції ролі

■ Дії:

□ реалізація завдання по розробці бази даних;
□ планування ітерації.

■ Операції:

□ оцінка завдання по розробці бази даних;
□ оновлення локального середовища проекту;
□ виконання тесту модуля бази даних;
□ рефакторинг бази даних;
□ визначення тривалості ітерації;
□ оцінка сценарію;
□ оцінка вимоги до якості
□ розбиття сценаріїв на завдання;
□ розбиття вимоги до якості на завдання.

Лекція 4. Дії в MSF.

1. Контроль ітерацій
2. Планування ітерацій

1. Контроль ітерацій

Усі ітерації, що виконуються, необхідно контролювати. Час, що витрачається на рішення проблем, впливає на загальний темп проекту. При видачі запитів пошуку проблем зареєструвати заблоковані описувачі дозволяє прапор «Issue». Запити пошуку проблем повинні виконуватися хоч би один раз на початку робочого дня. Після закінчення ітерації потрібний аналіз зробленого для визначення шляхів вдосконалення процесу і робочого середовища. Якщо ітерація розпочинається зі зборів, на яких вона планується, закінчуватися вона повинна зборами, що закривають ітерацію, на якому проводиться ретроспективний аналіз виконаної роботи.

Операції:

Моніторинг ітерації	Огляд пріоритетів в описувачі
Зниження ризику	Зменшення вірогідності ризику
Виконання ретроспективного аналізу	Організація ретроспективного аналізу

Операція: Моніторинг ітерації

Розуміння поточного стану ітерації дуже важливе для її успішного завершення. Один з найважливіших моментів в оцінці ходу ітерації — аналіз описувачів високопріоритетних сценаріїв, вимог до якості і помилок в цій ітерації. Актуалізуйте план ітерації, щоб побачити незакриті завдання, пов'язані з описувачами перерахованих типів. Визначте завдання, що завершилися, і розподіліть пріоритети так, щоб спочатку закінчувалися найбільш важливі частини ітерації. Завдання менеджера проекту полягає в тому, щоб розпізнати потенційні проблеми, що призводять до зупинки роботи, а також розпізнати вузькі місця, що знижують темп виконання роботи, і забезпечити безперервне виконання проекту.

Огляд пріоритетів в описувачі	Якщо в процесі ітерації змінюються пріоритети описувачів сценаріїв або вимог до якості, менеджер проекту відповідає за те, щоб організувати коротку нараду за участю бізнес-аналітиків, архітекторів, розробників і тестиров-щиков для обговорення змін. Пріоритети можуть мінятися з багатьох причин: із-за виникнення технічних питань, зміни пріоритетів замовника або залежностей
Рішення проблем	Щодня виконуйте запити пошуку проблем і перевіряйте, чи не заблоковані ті або інші завдання розробки або тестування. При виникненні проблем залучайте до їх рішення інших учасників команди

Операція: Зниження ризику

Для зниження ризику необхідно зробити кроки з метою зменшення його вірогідності або пом'якшення його дії. Проактивне управління рисками ефективніше за реактивний. Щоб проактивне управління рисками було рентабельним, в першу чергу треба приділяти увагу найбільш ризикованим моментам, але витрачати на їх усунення не більше, ніж вони можуть принести збитків.

Зменшення вірогідності	Запропонуйте варіанти зниження вірогідності виникнення несприятливих подій
Пом'якшення дії	Якщо вірогідність виникнення несприятливої ситуації залишається високою, запропонуєте способи пом'якшення шкідливої дії
Вироблення плану на випадок надзвичайних обставин	Якщо ризик все ж залишається неприйнятно високим, переглянете усі зроблені допущення і вимоги до якості, що особливо відносяться до експлуатаційного середовища, доступності і спеціальних можливостей
Застосування рішень	Реалізуйте вибране рішення, що забезпечує зменшення вірогідності і пом'якшення дії несприятливих обставин

Операція: Виконання ретроспективного аналізу

Зворотний зв'язок є важливою складовою будь-якого гнучкого процесу розробки ПЗ. Оскільки всяка проектна група — як і кожна проблема, з якою вона стикається, — унікальна, створення середовища, що відповідає усім потребам її учасників, є ключовим чинником у збільшенні продуктивності їх праці. Ретроспектива дозволяє команді розробників проаналізувати свої правильні і помилкові дії після завершення чергової ітерації. Головне тут — побачити можливі шляхи вдосконалення процесу і відповідним чином перебудуватися. Щоб отримати максимальну користь від ретроспективи, відповідна нарада повинна тривати близько двох годин. Проте якщо члени команди бачать, що усе йде гладко, зустріч можна скоротити. Остання ітерація закінчується ретроспективним аналізом проекту.

Організація наради для ретроспективи	Запросіть учасників команди на нараду. Воно повинне проводитися після завершення ітерації за участю усіх членів колективу. Нарада має бути відносно короткою, але достатньою для того, щоб зробити необхідні висновки (звичайно близько двох годин). Остаточний ретроспективний аналіз (у кінці проекту) може зайняти більше часу, аж до декількох днів для масштабних проектів. Ці завершуючі збори доцільно організовувати так, щоб воно не заважало повсякденній діяльності
Проведення наради	Встановіть чіткі правила участі співробітників в нарадах. Використайте список з двох стовпців : «плюсів» і «мінусів» завершеної ітерації
Враховуйте результати аналізу в наступних ітераціях	Там, де це можливо, використайте результати виконаного ретроспективного аналізу для коригування планів наступних ітерацій

2. Планування ітерацій

При плануванні ітерації важливо заздалегідь визначити правильний баланс між сценаріями, вимогами до якості і асигнуваннями на виправлення помилок. За кожну ітерацію можна виконати обмежений об'єм роботи. У поточну ітерацію потрапляють сценарії і вимоги до якості, що мають найбільшу практичну цінність. Первинний план ітерації створюється на основі *елементів сценаріїв* (scenario entries) і вимог до якості, що мають доки приблизні оцінки. Потім елементи, що потрапили в план ітерації, передаються бізнес-аналітику для написання сценаріїв. Після цього розробники і тестувальники розбивають сценарії на завдання. Ці завдання розподіляються між розробниками, і робиться точніша оцінка витрат, яка використовується для рівномірного розподілу навантаження між розробниками. Завершення вироблення плану ітерації відбувається на зборах бізнес-аналітиків, що беруть участь в ітерації, менеджерів проекту і розробників.

Операції:

Визначення тривалості ітерації	При плануванні відштовхуйтеся від дати закінчення проекту
Оцінка сценарію	Визначте підмножину зі списку сценаріїв
Оцінка вимоги до якості	Визначте підмножину зі списку вимог до якості
Планування сценарію	Створіть нарис плану ітерації
Планування вимоги до якості	Створіть нарис плану ітерації
Планування ресурсів на виправлення дефектів	Зарезервуйте ресурси на усунення дефектів
Розбиття сценарію на завдання	Зберіть команду
Розбиття вимоги до якості на завдання	Зберіть команду

Операція: Визначення тривалості ітерації

Ітерація — це набір завдань, які мають бути виконані впродовж фіксованого проміжку часу. Загальний час, необхідний для завершення завдань, називається тривалістю ітерації. Графік проекту розбитий на послідовність ітерацій і завдань, які, у свою чергу, мають свій графік. Визначення тривалості ітерації включає розгляд усіх ключових чинників, включаючи дату постачання продукту (якщо вона визначена), розмір сценаріїв і їх загальний час. Зазвичай тривалість ітерації визначається в тижнях, проте можливі і менші одиниці часу. Тривалість ітерацій визначається на початку проекту, а потім на основі цих даних визначається загальна тривалість проекту.

Планування від дати закінчення	Якщо для проекту задана дата постачання кінцевого результату проекту, то вона визначає тривалість ітерацій. Створіть план проекту або за допомогою календаря з'ясуєте, скільки часу може бути відведено на розробку

Перевірка розміру сценарію	Після закінчення планування першої ітерації перевірте, чи не розбиваються сценарії на дрібніші із-за її тривалості. Збільште тривалість ітерації, якщо це необхідно, для того, щоб вона відповідала розміру сценарію
Перевірка витрат на інтеграцію	Інтеграція виконується упродовж усієї ітерації після закінчення кожного завдання розробки. Враховуються усі особливі вимоги або витрати на інтеграцію
Аналіз тривалості ітерації	Проводьте аналіз тривалості ітерації спільно з проектною групою. Оптимальна тривалість дозволяє виконати невелику кількість сценаріїв або вимог до якості. Кожна ітерація повинна закінчуватися випуском внутрішньої або зовнішньої версії з новими функціональними можливостями, виправленими помилками, реалізованими сценаріями або вимогами до якості

Операція: Оцінка сценарію

Оцінка сценарію виконується для створення загальної картини, яка допомагає зрозуміти, скільки зусиль знадобиться для реалізації сценарію. Подібні оцінки дозволяють бізнес-аналітику розставляти пріоритети і регулювати зовнішні очікування. Менеджер проекту спільно з розробниками виконує оцінки для найбільш пріоритетних сценаріїв зі списку.

Визначте підмножину зі списку сценаріїв	Відкрийте список сценаріїв на порталі проекту. Для кожного сценарію з максимальним пріоритетом призначте розробника або групу розробників, які зможуть виконати оцінку
Оцінка реалізації	Спільно з розробниками оцініть *приблизний порядок величини* (rough order of magnitude, ROM) складності кожного сценарію. Для багатьох проектів може бути застосоване наступне правило. Якщо необхідно виконати не більше шести завдань, що вимагають 1-2 дні, в цьому полі вказується значення 1. Якщо необхідно виконати від 6 до 12 завдань, що вимагають 1-2 дні, в цьому полі вказується значення 2. Якщо трудовитрати більші, в цьому полі вказується значення 3 і розглядається можливість розбиття сценарію на частини. Важливо дотримуватися зразкових співвідношень оцінок, т. е. сценарій з порядком складності 2 має бути приблизно в два рази більше сценарію з оцінкою складності 1
Визначення особливих вимог	Спільно з розробниками визначите ризики, які можуть затримати або завадити успішному завершенню робіт по реалізації сценарію, такі як недолік знань або залежність від застосувань сторонніх постачальників. Створіть описувач для кожного виявленого ризику

Операція: Оцінка вимог до якості

Оцінка вимог до якості служить для визначення зусиль, необхідних для їх реалізації. Подібні оцінки дозволяють бізнес-аналітику встановлювати пріоритети і регулювати зовнішні очікування. Менеджер проекту спільно з розробниками вибирає зі списку найбільш пріоритетні вимоги до якості і виконує для них оцінки.

Вибір вимог до якості зі списку	Відкрийте список вимог до якості на порталі проекту. Для кожного сценарію з максимальним пріоритетом призначте розробника або групу розробників, які зможуть виконати оцінку
Виконання оцінки реалізації	Спільно з розробниками оцініте *приблизний порядок величини* складності кожної вимоги до якості. Для багатьох проектів може бути застосоване наступне правило. Якщо необхідно виконати не більше шести завдань, що вимагають 1-2 дні, в цьому полі вказується значення 1. Якщо необхідно виконати від 6 до 12 завдань, що вимагають 1-2 дні, в цьому полі вказується значення 2. Якщо трудовитрати більші, в цьому полі вказується значення 3 і розглядається можливість розбиття на частини завдання реалізації вимоги до якості. Важливо дотримуватися зразкових співвідношень оцінок, т. е. сценарій з порядком складності 2 має бути приблизно в два рази більше сценарію з оцінкою складності 1
Визначення обмежень	Спільно з розробниками визначите ризики, які можуть завадити успішному і своєчасному завершенню робіт по реалізації вимоги, такі як недостатність знань або залежність від ПЗ сторонніх постачальників. Створіть описувач ризику і прикріпите його до відповідної вимоги до якості для відображення залежності

Операція: Розбиття сценаріїв на завдання

Сценарій перед реалізацією треба розбити на завдання. Таке розбиття дозволяє точніше виконати оцінку складності і розподілити завдання між розробниками. Деякі сценарії можуть зачіпати декілька підсистем застосування. В цьому випадку для кожної області, що зачіпає, створюється окреме завдання. Ключовим моментом в успішному розбитті сценарію є підключення до цього процесу розробників і тестувальників, а також заохочення їх вибирати собі завдання самостійно. Усі завдання мають бути розподілені.

Зберіть команду	Зберіть архітекторів, розробників і тестувальників, які працюватимуть над сценарієм, або якось впливатимуть на нього. Менеджер проекту організовує короткі збори або дискусію по електронній пошті і фіксує завдання

Визначення архітектури і завдань розробки	Починаючи з того місця, де новий сценарій відрізняється від вже реалізованих сценаріїв, простежите його до кінця. Звертайте увагу на видимі користувачеві елементи, зміни в інфраструктурі або структурі даних, а також на моменти, які впливають на усю архітектуру. Визначте зміни в компонентах системи, які стануться в результаті реалізації планованого сценарію
Вибір архітектури і завдань розробки	Нехай архітектори і розробники самі виберуть собі завдання
Призначення відповідального розробника сценарію	Передайте сценарій одному з розробників, працюючих над завданнями сценарію. Він відповідатиме за наскрізне тестування інтеграції цих завдань
Створення тестових завдань	Відредагуйте документ, що описує тестування для цієї ітерації, вказавши в нім тестові дані і заповнивши інші розділи. Створіть завдання для розробки необхідних тестів перевірки сценарію
Вибір тестових завдань	Нехай тестувальники самі виберуть собі завдання

Операція: Розбиття вимоги до якості на завдання

Перед реалізацією кожна вимога до якості має бути розбита на завдання. Таке розбиття дозволяє точніше виконати оцінку складності і сприяє кращому розумінню розробниками майбутньої роботи. У рамках реалізації вимог до якості зазвичай треба виконати декілька завдань. Ключовим моментом в успішному розбитті вимоги до якості являється підключення до цього процесу розробників і тестувальників, а також заохочення їх вибирати собі завдання для себе самостійно. Усі завдання мають бути розподілені.

Зберіть команду	Зберіть архітекторів, розробників і фахівців з тестування, які працюватимуть над вимогою до якості, або якось впливатимуть на нього. Менеджер проекту організовує короткі збори або дискусію по електронній пошті і фіксує завдання
Визначення архітектури і завдань розробки	По посиланнях знайдіть сценарії, що мають відношення до вимоги до якості, і проведіть їх огляд. Вимоги до якості розрізняються по області свого впливу. Частина з них впливає тільки на один сценарій, тоді як інші можуть чинити дію відразу на декілька сценаріїв. Якщо будуть виявлені сценарії, пропущені бізнес-аналітиком, додайте їх до відповідної вимоги до якості
Вибір архітектури і завдань розробки	Довірте архітекторам і розробникам самим вибрати собі завдання
Призначення відповідального розробника вимоги до якості	Передайте вимогу до якості розробникові, якому належать завдання за відповідним сценарієм. Він відповідатиме за наскрізне тестування інтеграції цих завдань

Створення тестових завдань	Відредагуйте документ, що описує тестування для цієї ітерації, вказавши в нім тестові дані і заповнивши інші розділи. Створіть набір завдань для розробки необхідних тестів
Вибір тестових завдань	Нехай тестувальники самі виберуть собі завдання

Операція: Планування ресурсів на виправлення дефектів

Складання графіку виділення ресурсів на виправлення дефектів припускає виправлення дефектів в порядку, визначуваному їх важливістю. Зазвичай ресурси на виправлення дефектів виділяються усій групі розробників, але можуть обмежуватися одним або декількома розробниками з команди. Асигнування на виправлення дефектів беруться із загального бюджету розробки.

Планування асигнувань на виправлення дефектів	Із загального бюджету розробки виділите частину, необхідну на виправлення дефектів (одиниця виміру — абстрактний людино-день). Якщо план ітерації виконується в Microsoft Excel, позначте ці асигнування як резерв бюджету. Якщо план ітерації виконується в Microsoft Project, то асигнування можуть бути заплановані умовно
Облік залежностей і інших чинників	Якщо сценарії або вимоги до якості сплановані з урахуванням асигнувань на виправлення дефектів, переконаєтеся, що ключові розробники, що мають велику кількість невиправлених помилок, мають достатній резерв часу для виправлення помилок з найвищим пріоритетом. Перевірте ще раз майбутні завдання, щоб переконатися в наявності резерву. Перепризначуйте завдання або дефекти для забезпечення виправлення дефектів
Проведення зборів, присвячених початку ітерації	Завершіть розробку плану ітерації проведенням зборів, на яких буде викладена інформація про майбутню ітерацію. На зборах мають бути присутніми усі учасники команди, щоб ще раз перевірити завдання ітерації

Операція: Складання графіку сценарію

Сценарій розробляється і проходить первинне тестування у рамках певної ітерації. У плані ітерації відбито поточне розуміння того, що в ній повинно бути виконано. План має бути завершений до початку ітерації. Первинний план створюється на основі оцінок, а потім уточнюється у міру того, як сценарії і вимоги до якості розбиваються на завдання. З кожним завданням пов'язані оцінка трудомісткості і відповідальні розробники. Якщо дозволяє бюджет ітерації, для завершення її планування можуть бути проведені присвячені цьому збори. Воно дозволяє усім зацікавленим учасникам команди зібратися разом для обговорення виниклих питань і завершення плану ітерації.

Створення плану початкової ітерації	Спільно з бізнес-аналітиком розробіть план початкової ітерації

Уточнення плану початкової ітерації	Коли сценарій написаний і завдання по розробці включені в план ітерації, загальна кількість запланованих робіт не повинна перевищувати середні можливості команди розробників
Проведення наради, присвяченої початку ітерації	Завершіть розробку плану ітерації проведенням зборів, на яких буде викладена інформація про майбутню ітерацію. На зборах мають бути присутніми усі учасники команди, щоб ще раз перевірити завдання ітерації

Операція: Складання графіку реалізації вимог до якості

Для реалізації вимог до якості має бути складений графік робіт. Вимоги до якості описують нефункціональні вимоги, такі як, наприклад, рівень продуктивності. З деякими вимогами не пов'язані завдання. Проте вимоги можуть бути пов'язані з сценаріями, в яких вони реалізуються. Таким чином, вимоги до якості мають бути заплановані одночасно з відповідними сценаріями, щоб забезпечити виконання завдань, пов'язаних з їх описувачами.

Створення плану початкової ітерації	Спільно з бізнес-аналітиком розробіть план початкової ітерації. Отримайте останню версію списку вимог до якості з розставленими пріоритетами і оціненими трудовитратами. Спільно з бізнес-аналітиком виберіть ті з вимог, які укладаються в план ітерації з урахуванням середньої продуктивності команди, показаної в попередній ітерації, і кількості сценаріїв і асигнувань на виправлення дефектів, запланованих на цю ітерацію
Уточнення плану початкової ітерації	Коли завдання по розробці будуть включені в план ітерації, переконаєтеся, що загальна кількість запланованих робіт не перевищує середні можливості команди розробників
Проведення зборів, присвячених початку ітерації	Завершіть розробку плану ітерації проведенням зборів, на яких буде викладена інформація про майбутню ітерацію. На зборах мають бути присутніми усі учасники команди, щоб ще раз перевірити завдання ітерації

Лекція 5. Розробка архітектури рішення

1. Основні операції при розробці архітектури рішень
2. Розділення системи на підсистеми
3. Визначення інтерфейсів
4. Розробка моделі загроз
5. Розробка моделі продуктивності
6. Створення архітектурної моделі
7. Створення архітектури інфраструктури

1. Основні операції при розробці архітектури рішень

Хороша архітектура — це ясна і проста внутрішня структура більшості елементів застосування. Проста архітектура знижує складність застосування. Архітектура може визначати такі структурні елементи, які дозволяють простіше модифікувати застосування у разі зміни вимог, і завдяки яким ділянки застосування можуть розроблятися незалежно. Крім того, в хорошій архітектурі використовуються переваги від розбиття по рівнях для збільшення надійності і зменшення часу виведення застосування на ринок. Якщо є технологічні ризики, для їх зменшення можна застосовувати розробку архітектурних моделей, що допомагає кращому розумінню системи. І, нарешті, безпека і продуктивність — це питання архітектури. Робота над ними повинна проводитися з урахуванням усієї системи.

Операції:

Розділення системи на підсистеми	Виберіть архітектурні шаблони
Визначення інтерфейсів	Розробіть інтерфейси
Розробка моделі загроз	Створіть огляд застосування
Розробка моделі продуктивності	Проведіть огляд вимог до якості
Створення архітектурної моделі	Вивчіть ризики
Створення архітектури інфраструктури	Розробіть архітектуру інфраструктури

2. Операція: Розділення системи на підсистеми

Застосування, що є частиною розподіленої системи, містять логічно пов'язані фрагменти коду і спільно реалізують поведінку цілої системи. Розділення системи на модулі дає масу переваг. Воно дозволяє понизити загальну складність, інкапсулювати функції, збільшити потенційні можливості повторного використання підсистем і створити логічні одиниці для розробки. При цьому відпадає необхідність реалізовувати усю систему відразу.

Для створення гнучкої архітектури в MSF застосовується *створення тимчасового прототипу* (shadowing). *Застосування-прототип* (shadow) дозволяє застосувати принцип низхідного проектування у будь-якій ітерації. На початку ітерації, коли існує лише проект, прототип випереджає робочий програмний код. У цей період проектні конструкції не синхронизовані з кодом. У рамках прототипу виконуються усі зміни архітектури або дизайну, які потрібні для оберігання основного коду від перетворення його на «димар», «спагеті-код» і інших проблем, пов'язаних з поганим проектуванням. У міру реалізації елементів *початкового упереджаючого прототипу* (leading shadow), архітектура все більше відповідає робочому коду. Ті частини системи, які були спроектовані, але не були реалізовані, тепер стають дійсними. Коли архітектура відповідає робочому коду, прототип називається *таким, що завершує* (trailing shadow). Це сума конструкцій з усіх попередніх ітерацій.

Щоб уникнути надмірної деталізації архітектури, рекомендується сконцентруватися на рівні компонентів і встановлюваних елементів. Додайте застосування, необхідні для підтримки функціональних можливостей, передбачених в запланованих сценаріях, вимогах до якості і в намічених для виправлення дефектах. На початку ітерації додайте завдання по переробці коду, щоб він відповідав зміненій архітектурі. Якщо необхідно, перевірте нову структуру за допомогою архітектурних моделей.

Вибір шаблону архітектури	Якщо передбачається розгортання системи у рамках існуючої інфраструктури, вивчите логічну діаграму центру обробки даних для того, щоб зрозуміти технологію розгортання. Якщо такої діаграми не існує, створіть її
Створення застосувань	У рамках діаграми застосування створіть упереджаючий прототип або його еквівалент для вибраної топології системи. Створіть діаграму системи на основі застосувань-прототипів і додайте проксі для усіх нереалізованих точок входу
Перевірка розгортання	Після того, як діаграма застосування стала відповідати архітектурі в цій ітерації і визначена цільова інфраструктура, пора перевірити розгортання. На діаграмі застосувань виберіть певний варіант установки і відобразіть нові застосування на логічні сервери, що відповідають їм
Створення завдань, що реалізовують зміни в архітектурі	Створіть нові завдання, у рамках яких буде реалізований упереджаючий прототип і виконані необхідні переробки існуючих функціональних можливостей в нове застосування

3. Операція: Визначення інтерфейсів

Розробку системи можна розпаралелювати, якщо до початку розробки визначені усі інтерфейси. Частенько на початку ітерації є ще недостатньо деталей, що заважає повному розумінню, необхідному для розробки елементів інтерфейсу. Тому розробку інтерфейсів варто відкладати до того моменту, поки вона не стане

абсолютно необхідною: адже навіть після визначення інтерфейсів можуть відбуватися зміни в проекті.

Опрацювання інтерфейсів допомагає при інтеграції систем, а також при зміні підходів в реалізації завдань розробки. У діаграмі застосувань слід відображати зовнішні інтерфейси системи, а також внутрішні інтерфейси вищого рівня.

Проектування інтерфейсу	Визначте інтерфейси для сценарію або вимоги до якості. Перевірте вимоги по взаємодії із зовнішніми системами
Визначення інтерфейсів веб-сервісів	Виберіть кінцеву точку веб-сервісу на діаграмі застосування. Визначте операцію і забезпечте її необхідною інформацією, такою як ім'я, повертаний тип і параметри. Переконайтеся, що у властивостях правильно налагоджені мова і шлях до генерованих файлів. Згенеруйте заглушку для нового веб-сервісу
Створення інтерфейсів класів	Спільно з розробниками визначите методи класів, які забезпечуватимуть інтерфейси між розробниками. Застосовуючи діаграму класів, перевірте узгодженість класів і нових методів

4. Операція: Розробка моделі загроз

У моделі загроз документуються відомі загрози безпеки, і описується, як на них слід реагувати. Моделювання загроз є частиною структурного підходу, що дозволяє визначити небезпеки, з якими найімовірніше зіткнеться система, а також міра їх дії. У рамках цілісної моделі загрози розглядаються в порядку убування небезпеки, що представляється, а також розглядаються необхідні заходи протидії ім. Модель загроз слід створювати якомога раніше, а потім, у міру розвитку архітектури, уточнювати її. Спільно з бізнес-аналітиком проведіть огляд загроз і розробіть вимоги до безпеки.

Створення огляду застосування	Визначте сценарії, що відносяться до завдань забезпечення безпеки. Уважно вивчите кожен сценарій на предмет можливості відступу від бізнес-правил. Постарайтеся знайти уразливості в сценаріях, забезпечення безпеки, що відносяться до завдань
Розбиття застосування	Визначте межі довіри у рамках вашого застосування. На логічній діаграмі центру обробки даних створіть нову зону, яка відбиватиме ці межі. Для кожної підсистеми визначите, чи є надійними потоки даних, що входять, і призначене для користувача введення. Якщо ні, то вирішите, як вони повинні аутентифікуватися і авторизуватися
Визначення загроз	Для кожного ресурсу або функції визначите реалізовані сценарії, що зачіпають їх. На їх прикладі обговорите, як система повинна використовуватися, а як — ні
Визначення вразливих місць	Переконайтеся, що усі частини системи були розглянуті і що усі відомі типи загроз були проаналізовані

5. Операція: Розробка моделі продуктивності

Моделювання продуктивності — це процес, що допомагає визначити потенційні проблеми з продуктивністю в застосуванні і знайти рішення для цих проблем. Моделювання продуктивності проводиться на основі вимоги до якості, яка, у свою чергу, розбита на завдання. Кожне завдання має свій бюджет, призначений для вирішення питань продуктивності під час реалізації.

Огляд вимог до якості	Визначте сценарії, пов'язані з вимогами до продуктивності
Визначення об'єму роботи	За списком вимог до продуктивності визначите об'єм робіт для застосування
Визначення необхідного рівня продуктивності	Використовуючи оцінки об'єму робіт і список вимог до якості, визначите необхідний рівень продуктивності для кожного ключового сценарію. Він включає такі характеристики, як час відгуку, пропускну спроможність і використовувані ресурси
Визначення бюджету для вирішення питань продуктивності	Визначте об'єм ресурсів, що впливають на продуктивність, який дозволить добитися необхідної продуктивності. Прикладами таких ресурсів є час виконання і пропускна спроможність мережі
Розподіл бюджету	Розподіліть ресурси між технологічними операціями для кожного сценарію
Оцінка бюджету	Знайдіть бюджетні асигнування, для яких існує ризик нестачі для досягнення необхідної продуктивності
Перевірка моделі	Виявіть сценарії, на які не виділені бюджетні асигнування

6. Операція: Створення архітектурної моделі

Створення архітектурної моделі може значно понизити наявні ризики. Важливо якомога раніше звернути увагу на ризики в проекті і, тим самим, врахувати їх при ухваленні стратегічних і архітектурних рішень, поки зміни основних компонентів архітектури ще не вимагають великих витрат. Створення ранніх прототипів знижує загальні ризики і невизначеності в проекті. Це, у свою чергу, дозволяє точніше виконувати планування і оцінки в подальших ітераціях. Моделі можуть бути тимчасовими і відкидатися, як тільки вони відповідають на поставлені питання, або можуть бути основою для архітектури застосування.

Вивчення ризиків	Визначте елементи, які можуть допомогти у визначенні ризику або ухваленні архітектурного рішення
Планування	Визначте необхідний вид моделі
Побудова і робота моделі	Побудуйте модель. Зконцентруйтесь на вирішуваній проблемі. Переконайтеся, що модель належним чином описує досліджуване питання

7. Операція: Створення архітектури інфраструктури

Архітектура інфраструктури визначає оточення, в якому працюватиме застосування. Добре розроблена архітектура інфраструктури гарантує установку

застосування відповідно до вимог до якості. Починайте розробку архітектури інфраструктури, як тільки з'являється уявлення про майбутнє застосування. При необхідності змінюйте розробку у міру того, як покращується розуміння підсистем у рамках загальної архітектури. Для побудови архітектури інфраструктури використайте логічну діаграму центру обробки даних.

Створення архітектури інфраструктури	Створіть логічну діаграму центру обробки даних. Працюйте з операціями, якщо вони застосовні, для кращого розуміння робочого оточення. У логічному центрі обробки даних створіть логічні сервери, що відбивають фізичне оточення. Перевірте діаграму, що вийшла
Зіставлення архітектури рішення	Відобразіть ділянки діаграми застосування на логічну діаграму центру обробки даних. Перевірте правильність цього відображення. Перевірте діаграму застосування за допомогою логічної діаграми центру обробки даних
Коригування архітектури рішення	У міру визначення нових підсистем виявляйте необхідні зміни в архітектурі інфраструктури

Лекція 6. Формулювання концепції проекту

1. Основні операції формування концепції проекту.
2. Розробка вимог до якості.
3. Створення сценарію.

1. Основні операції формування концепції проекту

Перед запуском проекту необхідно чітко сформулювати його концепцію. Формулювання і донесення центральної концепції є найважливішим елементом в підтримці спрямованості проекту. Впродовж життя проекту його концепція може мінятися під впливом зовнішніх або внутрішніх чинників. Якщо подібні зміни сталися, то важливо відразу змінити проект згідно нової концепції. Концепція дає уявлення про майбутніх користувачів. Способи використання і завдання цих користувачів мають бути виявлені за допомогою *збірних образів* (personas). Концепція дає зрозуміти, чи прив'язаний проект до часу або до функціонала, що реалізовується. Сама концепція і пов'язана з нею діяльність створюють надійний фундамент, на якому будуватиметься увесь проект.

Операції:

Написання концепції	Узагальніть усю відому про проект інформацію
Визначення збірних образів	Визначте групи користувачів
Уточнення збірних образів	Визначте характер відмінностей між збірним образом і користувачем

Операція: Написання концепції

Концепція — це коротке і точне формулювання мети створення нової системи або поліпшення існуючої. Вона дає команді уявлення про проект і його рушійні чинники, а також містить обгрунтування розробки системи. У концепції має бути присутнім високорівневий опис користувачів системи. Концепція повинна також містити опис потреб користувачів і способів їх задоволення застосуванням, що розробляється. Нарешті, концепція визначає, чи прив'язаний випуск системи до термінів або до функціональних можливостей. Концепція має бути написана мовою, зрозумілим майбутнім користувачам. Написання стислої і предметної концепції є складним завданням.

Узагальнення початкових даних проекту	Напишіть один-два абзацу про контекст проекту. У них має бути описана передумова створення нової системи або поліпшення існуючої
Пояснення рушійних чинників	Опишіть основні вимоги або виділений інтервал часу, які управляють випуском продукту. Будьте короткі
Визначення користувачів	Визначте тих користувачів, які отримають найбільшу користь від системи
Визначення основних достоїнств	Опишіть передбачувані достоїнства нової або поліпшеної системи

Операція: Визначення збірних образів

Збірний образ — це вигаданий персонаж, що представляє групу користувачів. Збірні образи використовуються для вивчення потреб цілого

сегменту користувачів шляхом розгляду характеристик одного вигаданого персонажа. Для визначення збірного образу вивчіть сегмент користувачів, що взаємодіють з системою, зберіть їх досвід, уміння і цілі, які вони розділяють. На основі цих даних створіть вигаданий персонаж, що представляє цей сегмент. Збірний образ є також інструментом, за допомогою якого можна отримати інформацію про користувачів, згаданих в концепції. Збірні образи використовуються при створенні сценаріїв і при проведенні дослідницьких тестувань.

Визначення ролей	Серед користувачів, визначених в концепції, виберіть групу користувачів, що взаємодіють з системою
Створення збірного образу	Для створення збірного образу зберіть реальні дані. Використайте дані, отримані в дослідженнях зручності використання і при відвідуваннях клієнтів, в роботі з фокус-групами і при маркетингових дослідженнях, щоб переконатися, що збірний образ представляє реальних користувачів

Операція: Уточнення збірних образів

Збори і огляди часто допомагають виявити нові деталі про способи використання і рівень знань. Використайте такі збори для перевірки збірних образів. Періодично уточнюйте їх, щоб вони відповідали поточній цільовій аудиторії системи. Якщо у збірних образах відбуваються зміни, повідомляйте про це замовника.

Визначення відмінностей	Виявляйте схожість і відмінності в рівні знань, способах використання, взаємодії і завданнях, наявні між збірним образом і реальним користувачем. Якщо
Оновлення і передача змін	Додайте нові збірні образи і змініть існуючі

2. Розробка вимог до якості

Вимоги до якості використовуються для опису нефункціональних вимог або обмежень на функціональні можливості системи. Вимоги до якості мають бути точними і не бути суб'єктивними. Вони виявляються, їм привласнюються пріоритети і — якщо вони заплановані на поточну ітерацію — документуються.

Визначення вимог до якості методом мозкового штурму	Визначте цілі вимог до якості
Створення моментальних знімків діяльності збірного образу	Визначте характерний проміжок часу дій збірного образу
Визначення пріоритетів в списку вимог до якості	Визначте важливість кожної вимоги
Написання вимог до якості	Задокументуйте вимоги до якості

Визначення вимог безпеки	Задокументуйте вимоги до системи безпеки

Операція: Визначення вимог до якості методом мозкового штурму

Проведіть колективне обговорення вимог до якості, проаналізуйте кожен сценарій і визначте, які взаємодії в них повинні супроводжуватися вимогами до якості. Під час обговорення виявите аспекти системи, для яких в сценаріях були пропущені необхідні вимоги до якості. Список вимог до якості містить нефункціональні вимоги до застосування, він же є списком обмежень функціональних можливостей системи. Бізнес-аналітик кожного разу переоцінює і коригує список вимог до якості, коли в процесі тестування виявляються нові вимоги або коли з'являються зміни в проекті.

Визначення необхідних рівнів якості	У теці вимог *Провідника команди* (Team Explorer) необхідно відкрити список вимог до якості. Він пов'язаний з проектом. Якщо необхідно, імпортуйте описувачі вимог до якості, які були створені безпосередньо в Провіднику команди
Визначення вимог до якості	Проаналізуйте сценарії на предмет відповідності бажаним стандартам якості усіх моментів, пов'язаних зі взаємодією користувача і системи

Операція: Створення моментальних знімків діяльності

Моментальні знімки діяльності — це необов'язкові засоби, які використовуються спільно з моделлю збірного образу. При написанні сценаріїв моментальні знімки допомагають конкретизувати деякі деталі. Моментальний знімок діяльності — це опис одного дня або короткого періоду в існуванні збірного образу. При цьому завжди описується поточний стан, без застосування пропонованої технології, що допомагає продемонструвати, чим нова система або продукт будуть корисні для користувача. Працюючи з конкретними прикладами діяльності, простіше оцінити користь від кожного сценарію і на підставі цього призначити їм пріоритети.

Створення прототипу робочого дня збірного образу	Для кожного збірного образу потрібно виділити один або декілька періодів часу в їх існуванні
Пошук технологічних можливостей	Потрібно вивчити щодня збірного образу і виявите місця, де застосування продукту виявляється корисним користувачеві за рахунок рішення яких-небудь його завдань або розширення можливостей

Операція: Визначення пріоритетів в списку вимог до якості

Пріоритети в списку вимог до якості визначаються, виходячи з їх важливості і впливу, що робиться, на користувача. Вимоги, які найбільш важливі користувачеві, повинні реалізовуватися в першу чергу. Список вимог до якості, впорядкованих по пріоритетах, включається в план ітерації. Той, у свою чергу, використовується при плануванні розробки для підвищення ефективності використання ресурсів. При додаванні або видаленні вимог, а також при зміні потреб користувачів слід наново визначити пріоритети списку вимог до якості.

Визначення загального пріоритету	Кожному сценарію в списку призначте загальний пріоритет. У полі пріоритету списку сценаріїв впишіть відповідне числове значення
Опис пріоритетних сценаріїв	Використовуючи шаблон опису сценарію, контурно змалюйте найбільш пріоритетні сценарії, забезпечивши їх коротким описом, але, в той же час, досить детальним, щоб розробники на його основі змогли зробити оцінки
Передача запитів для виконання оцінок	Пошліть розробникам запит на виконання загальних оцінок для найпріоритетніших вимог
Розбиття вимог чи зміна пріоритетів	Якщо ціна якої-небудь вимоги до якості перевищує бюджет ітерації, спробуйте розбити цю вимогу

Операція: Написання вимог до якості

Вимоги до якості використовуються для опису нефункціональних вимог або обмежень на функціональні можливості системи. Вимоги до якості мають бути точними і не бути суб'єктивними. Їх опис повинен надавати досить інформації розробникам, щоб ті могли спроектувати і запрограмувати функціональні можливості, що задовольняють цим вимогам.

Документування вимоги до якості	Опишіть вимогу до якості в полі опису його описувача. При розробці вимог до якості використайте наступний мовний шаблон: «контекст», «дія», «відповідь». Вимога до якості має бути таким, щоб його можна було протестувати. Проектування залиште архітекторам і розробникам
Прикріплення допоміжної інформації	До вимоги до якості прикріпіть пов'язані з ним сценарії, якщо такі є. Якщо вимога до якості зачіпає усі сценарії або більшість з них, просто вкажіть це замість того, щоб прикріплювати їх усі

Операція: Визначення вимог безпеки

Вимоги до системи безпеки визначають рівень, до якого система захищатиме себе і ресурси. Це можуть бути конфіденційні дані, вимоги закону або нематеріальні активи, такі як репутація компанії, торгові секрети або інтелектуальна власність. Вимоги до системи безпеки мають бути конкретні, а для ресурсів, що захищаються, має бути можливість перевірки захисту. Спосіб захисту описувати не треба. Вимоги до системи безпеки зазвичай формулюються пропозиціями, що розпочинаються з дієслова, наприклад: «Запобігти доступу неавторизованих користувачів до облікових записів клієнтів». Ресурси, що захищаються, мають бути чітко визначені.

Документування вимог до системи безпеки	Опишіть вимоги до системи безпеки в полі опису описувача вимоги до якості. Переконайтеся, що вимоги до системи безпеки обернені на конкретні ресурси і можуть бути протестовані. Проектування залиште архітекторам і розробникам

Прикріплення допоміжної інформації	Прикріпіть посилання на законодавчі акти, якщо вони є. Прикріпіть усі сценарії, що відносяться до завдань забезпечення безпеки. Якщо вимога до системи безпеки зачіпає усі сценарії або більшість з них, просто вкажіть це замість того, щоб прикріплювати їх усе

3. Створення сценарію

У сценаріях формулюються функціональні завдання системи. Для визначення цих завдань уважно вивчите усі потреби збірних образів системи. Ці завдання можуть бути спочатку просто виписані у вигляді списку, а пізніше описані у вигляді сценаріїв. Намагайтеся не допускати сценаріїв, що описують невдалі або неоптимальні шляхи поставлених завдань. Сценарії можуть створюватися під час «мозкового штурму», за допомогою моментальних знімків або пробних тестів. Усі вони додаються в список сценаріїв. Коли сценарії призначаються на наступну ітерацію, вони документуються і їм привласнюються пріоритети. Створення сценаріїв закінчується, коли складений графік реалізації усіх сценаріїв в ітераціях, або після створення архітектурного прототипу.

Операції:

Робота над сценаріями методом «мозкового штурму»	Визначте завдання системи
Створення моментальних знімків діяльності	З'ясуйте типові способи використання системи
Визначення пріоритетів в списку сценаріїв	Визначте пріоритети для кожного сценарію
Створення опису сценарію	Виберіть відповідний узагальнений образ
Розкадровування сценарію	Підберіть інструмент для розкадровування

Операція: Робота над сценаріями методом мозкового штурму

Спочатку список сценаріїв складається з елементів сценаріїв. Їх використовують для визначення цілей застосування. Елемент сценарію складається з імені і короткого опису, що визначає унікальність елементу. Елементи стають описувачами після синхронізації сценаріїв. Проведіть колективне обговорення і додайте сценарії в список сценаріїв, проаналізуйте завдання застосування і те, як воно використовуватиметься кожним збірним образом. Початковий список сценаріїв створюється в першій ітерації, а потім кожного разу, коли міняються вимоги або виявляються нові, список сценаріїв слід переглядати і коригувати.

Визначення завдань системи	У теці вимог Провідника команди відкрийте список сценаріїв. Список сценаріїв пов'язаний з проектом. Імпортуйте усі сценарії, створені за допомогою Провідника команди

Формулювання сценаріїв	Виберіть мету і розгляньте різні способи, якими збірний образ може досягти її або потерпіти в цьому невдачу. Для способу, яким збірний образ намагається досягти мети, виберіть описове ім'я. Потім додайте елемент сценарію з цим ім'ям в список сценаріїв. При визначенні концепції проекту слід переконатися, що уявлення про майбутнє застосування охоплює усі сценарії. При зміні представлення слід коригувати концепцію

Операція: Визначення пріоритетів в списку сценаріїв

При визначенні пріоритету сценарію в списку враховуються як його значущість для користувачів, так і для застосування в цілому. Пріоритети визначають черговість реалізації сценаріїв. Сам процес визначення пріоритетів допомагає виявити найбільш важливі і цінні сценарії, які будуть реалізовані в найближчій ітерації.

Визначення загального пріоритету	Кожному сценарію в списку призначте загальний пріоритет. У полі пріоритету списку сценаріїв впишіть відповідне числове значення
Опис пріоритетних сценаріїв	Використовуючи шаблон опису сценарію, контурно змалюйте найбільш пріоритетні сценарії, забезпечивши їх коротким описом, але, в той же час, досить детальним, щоб розробники на його основі змогли зробити оцінки
Передача запитів для виконання оцінок	Пошліть менеджерові проекту і розробникам запит на виконання загальних оцінок для найпріоритетніших вимог
Розбиття вимог або зміна пріоритетів	Якщо ціна якого-небудь сценарію перевищує бюджет ітерації, спробуйте розбити цю вимогу

Операція: Формулювання опису сценарію

Опис сценарію зі списку пишеться в процесі створення початкового плану ітерації, т. е. коли вона стає кандидатом на реалізацію в найближчому сценарії. Рівень деталізації має бути достатнім для передачі потреб збірного образу розробникам і для створення контрольних прикладів. Великі сценарії можуть бути розбиті на дрібніші, такі, що укладаються в рамки однієї ітерації. Після створення опису воно публікується на порталі проекту, а майбутнім розробникам і тестувальникам розсилаються повідомлення, щоб вони проглянули їх.

Вибір відповідного збірного образу	Зі списку сценаріїв виберіть сценарій, що увійшов до найближчої ітерації або важливий з точки зору архітектури. Відкрийте шаблон опису сценарію в Microsoft Word і збережіть документ, давши йому ім'я сценарію, щоб відрізняти його від інших, вже написаних сценаріїв

Формулювання опису сценарію	Пишіть сценарій в розділі опису відповідного документу. Із самого початку описуйте кожну дію, що виконується збірним образом в процесі досягнення поставленої мети. Дії, вже описані в інших сценаріях, записуйте схемний, а в тих місцях, де сценарій відрізняється від інших, будьте найбільш детальні
Розбиття сценаріїв	Якщо детальна оцінка (складена як сума завдань розробки) перевищує довжину ітерації, то такий сценарій є кандидатом на розбиття. Розбиття, що вийшло в результаті, підсценарії в сумі повинні відповідати початковому сценарію. Для кожного нового сценарію слід визначити характерні відмінності від інших. На основі шаблону сценарію створіть хоч би два документи, що описують сценарії

Операція: Розкадровування сценарію

Розкадровування покращує сценарій і забезпечує його інформацією про призначений для користувача інтерфейс. Вона виключно корисна, коли призначений для користувача інтерфейс відіграє важливу роль в задоволенні потреб зовнішніх користувачів. Розкадровування також допомагає підігнати проект під вимоги різних організацій. Вона може включати різні графічні елементи, такі як знімки екранів, макети застосувань, диа-граммы послідовності екранів. Вона також може включати короткий опис поведінки в системі збірних образів. Графічне представлення сценаріїв значно покращує взаєморозуміння між кінцевими користувачами і розробниками. Якщо розкадровування створені для кожного окремого сценарію, то їх можна об'єднати в загальний огляд застосування.

Вибір сценаріїв	Відберіть сценарії, що вимагають розкадровування
Створення знімків екрану	Для розкадровування добре підходять такі інструменти, як Microsoft Visio і Microsoft PowerPoint. Створіть екрани, які повинен бачити збиральний персонаж. Схемний намалюйте призначений для користувача інтерфейс, потрібний сценарієм. Спільно з командою розробників переконаєтеся, що отриманий призначений для користувача інтерфейс не суперечить вибраній метафорі
Створення діаграми послідовності екранів	Прямокутниками зображуйте екрани, а стрілками — їх послідовність

Лекція 7. Ведення проекту згідно MSF.

1. Основні операції ведення проекту.
2. Зборка продукту.
3. Випуск продукту.
4. Усунення дефекту.
5. Закриття дефекту.

1. Основні операції ведення проекту.

Ведення проекту — це постійний моніторинг і документування його стану. Щоб забезпечити постійний контроль над проектом, допускається внесення необхідних змін. Необхідно відстежувати як хід проекту в цілому, так і виконання окремих ітерацій. Потрібно своєчасно виявити дефекти, визначити ризики і присвоїти їм пріоритети.

Операції:

Огляд цілей	Встановіть мінімальний прийнятний рівень
Оцінка ходу виконання	Підтримуйте графік проекту в актуальному стані
Оцінка порогових значень показників тестів	Проаналізуйте звіти про виконані тести
Класифікація дефектів	Створіть список дефектів, що підлягають класифікації
Визначення ризику	Визначте вимоги до якості, пов'язані з великими ризиками

Операція: Оцінка ходу виконання

Для успішної реалізації проекту важливо відстежувати і оцінювати хід його виконання. Правильне планування і управління проектом залежить від можливості оцінювати його поточний стан. Крім того, потрібно відстежувати організаційні, технічні і проектні ризики. Проект можна прив'язати до часу або до функціонала, що реалізовується. У прив'язаних до часу проектах крайні терміни частенько визначаються подіями на ринку, положенням конкурентів, існуючими нормами, фінансовими чинниками і іншими причинами, пов'язаними з бізнесом. Навіть проекти з фіксованим функціоналом можуть мати бажану дату реалізації. У проектах, не фіксованих за часом, правильний прогноз дати завершення благотворно позначиться на рівні продажів, маркетингових показниках, на впровадженні, навчанні і експлуатаційних чинниках.

Оцінка списку обов'язкових робіт	Якщо є передбачувана дата реалізації проекту, визначите число необхідних ітерацій на підставі тривалості початкової ітерації. Використайте дані із *звіту про загальний темп проекту (*velocity report) для обчислення кількості людино-дня для ітерації. Твір двох цих значень дасть об'єм трудовитрат, що залишилися, виражений в людино-дні
Перевірка залежностей	Перевірте залежності в графіці і переконайтеся, що вони дотримуються

Аналіз здійсненності	При необхідності оцініте число ітерацій, необхідних для реалізації планованих функцій, і визначите, чи відповідає хід робіт цілям, визначеним в концепції проекту

Операція: Оцінка порогових значень показників тестів

У документі з описом підходів до тестування вказані порогові значення для показників тестів. Необхідно періодично аналізувати просування проекту відповідно до цих показників і при необхідності їх коригувати. Ефективність тестів, число несправностей, охоплення коду, детальні звіти про виконані тести — усе це допоможе контролювати виконання проекту. Потім, при необхідності, можна скоректувати курс.

Аналіз звітів про виконані тести	Звіти про виконані тести допоможуть зрозуміти поточний стан справ, якщо порівняти результати з пороговими значеннями
Визначення завдань і ризиків тестування	Спільно з фахівцями з тестування створіть і розподіліть завдання тестування

Операція: Визначення ризику

Своєчасно невиявлений і неконтрольований ризик може негативно позначитися на виконанні проекту. Ризики мають місце в різних областях. Наприклад, в одному проекті ризик може бути пов'язаний з ергономікою, оскільки ця область принципово важлива для замовника, а в іншому проекті ризики можуть бути пов'язані з продуктивністю. Важливо чітко визначити і врахувати найбільш небезпечні ризики в перших же ітераціях проекту. У міру появи додаткових відомостей про сценарії і вимоги до якості в кожній подальшій ітерації приділите час пошуку нових можливих ризиків.

Визначення вимог до якості, пов'язаних з ризиком	Проаналізуйте вимоги до ергономіки. Визначте, чи являється інтерфейс користувача ключовим чинником оцінки застосування замовником
Перевірка сценаріїв	Проаналізуйте сценарії на предмет потенційних ризиків
Питання інтеграції	Які бібліотеки сторонніх постачальників можуть використовуватися застосуванням? Чи перевірені вони? Якщо ні, виявите невідомі характеристики бібліотек, які необхідно перевірити
Формулювання ризиків і розставляння їх пріоритетів	Якщо в розглянутих областях виявлені загрози для успішної реалізації проекту, створіть описувач ризику. Сформулюйте потенційні проблеми, пов'язані з цим ризиком, і задокументуйте їх в описувачі

Операція: Огляд цілей

Після усіх ітерацій, окрім нульової, продукт має бути в стабільному стані і готовим до постачання. Встановіть показник *мінімального прийнятного рівня* (minimum acceptance level), щоб надалі порівнювати реалізовані сценарії і вимоги до якості з цим рівнем. Мінімальний прийнятний рівень повинен постійно переоцінюватися з урахуванням змін потреб замовника, стану ринку і т. д. Мінімальний прийнятний рівень оновлюється після кожної ітерації.

Установка мінімального прийнятного рівня	Оцініть очікування, які зв'язують з продуктом, що розробляється, замовники, і ринок в цілому. Врахуйте конкурентів, існуючі аналоги і потреби бізнесу
Звіт про поточний стан	Сформуйте звіт про роботи, що залишилися, для оцінки ходу виконання проекту

Операція: Класифікація дефектів

Класифікація (triage) — процес аналізу знову виявлених або наново оброблюваних дефектів, призначення ним пріоритетів і прив'язки до певних ітерацій. Класифікацію проводить менеджер проекту за участю інших учасників проектної групи.

Створення звіту про дефекти	Видайте запит системі відстежування дефектів про знову виявлені і наново оброблювані дефекти
Аналіз дефектів	Ознайомтеся з кожним виявленим дефектом, притягаючи розробників, тестувальників і менеджера проекту. Визначте важливість виправлення кожного з них до кінця проекту і поточної ітерації
Привласнення пріоритету і прив'язка до ітерації	Кожному дефекту присвойте пріоритет і визначте, в якій ітерації його необхідно виправити, щоб його обробку можна було врахувати в графіці проекту
Відтворення незрозумілих дефектів	Якщо звіт про дефект незрозумілий або ні упевненості в істинній дії дефекту, група управління програмою повинна спробувати відтворити дефект, щоб краще зрозуміти його реальний вплив і можливості повторної появи

2. Зборка продукту

В процесі зборки створюється єдиний продукт, що включає існуючий код і усі нові пакети змін. Проте зміни, що вносяться, мають бути коректні. Пакет змін має бути скомпільований і мають бути виконані компонувальні тести. Пакет змін, що додається, не повинен провокувати проблеми — перевірка цього факту є частиною процесу приймання зборки. До кожної зборки додається набір коментарів, що пояснюють зміни, що вносяться.

Операції:

Запуск зборки	Мінімізуйте залежності
Перевірка зборки	Перевірте основні функції
Виправлення зборки	Виділіть помилки компіляції
Приймання зборки	Протестуйте зборку

Операція: Запуск зборки

Щоб гарантувати цілісність, запускайте зборку продукту всякий раз, коли вимагається включити в нього зміни. Це допомагає синхронізувати зміни і надає усім розробникам робочу платформу. Щоб прискорити занадто повільну зборку, переглянете залежності.

Мінімізуйте залежності	Чим менше в застосуванні залежностей, тим швидше відбувається його зборка
Початок зборки	Запускайте зборку всякий раз, коли можна інтегрувати зміни. Створюйте «чисту» зборку тільки у разі абсолютної необхідності. «Чисті» складки робляться ночами. Якщо процес зборки не виконується без попередньої «чистої» зборки, перевірте залежності

Операція: Перевірка зборки

Необхідно підтримувати якість застосування при внесенні будь-якої зміни. Проводьте перевірочні випробування, що демонструють, що базова функціональність зборки не торкнулася. Перевірочні випробування також проводяться для перевірки змін, зроблених в процесі виправлення дефекту і для контролю працездатності знову доданих функцій.

Перевірка основних функцій	Виконайте мінімальну підмножину тестів, зокрема, перевірочний тест зборки, що іноді називається «димовим», щоб переконатися в стабільності базової функціональності системи
Перевірка змінених функцій	Запустіть додаткові тести і переконайтеся, що усі заплановані пакети змін додані і зміни коректні
Інформування учасників про готовність зборки	Якщо усі тести пройшли, проінформуйте усі зацікавлені сторони, що зборка допущена до роботи

Операція: Виправлення зборки

В процесі перевірочних випробувань може виникнути ситуація, коли початковий код не компілюється на складальній машині або зовнішні компоненти не працюють коректно. Щоб виправити зборку, виявіть помилки періоду компіляції і виконання і проінформуйте розробників, відповідальних за відповідні компоненти. Затримки в рішенні проблем зі зборкою гальмують виконання інших робіт, наприклад тестування.

Виділення помилок компіляції	Перед компіляцією запустите сценарій, що очищає використовувані при зборці каталоги від непотрібних файлів. Компіляція початкового коду і побудова компонентів на складальній машині може не проходити, хоча та ж процедура нормально проходила на машині розробника
Обробка помилок періоду виконання	Фіксуйте усі помилки періоду виконання і інформуйте про них розробника — безпосередньо і за допомогою описувача дефекту
Перевірка вмісту	Порівняйте вміст зборки з вказаним в специфікації: чи немає зайвих файлів і чи не пропущені потрібні
Перевірка настановних сценаріїв і виконуваних файлів	Прогляньте настановні сценарії і перевірте, чи розташовані необхідні файли в каталогах, вказаних в специфікації і сценаріях

Перевірка версій пов'язаних модулів	Перевірте версії використовуваних при компонуванні зовнішніх модулів, наприклад бібліотек сторонніх постачальників. Використання некоректних версій може привести до непередбачуваної поведінки застосування
Виправлення і перекомпонування зборки	Після вирішення проблеми треба відразу наново створити зборку, щоб не блокувати роботу інших учасників проекту

Операція: Приймання зборки

Коли зборка допускається до роботи, це говорить про те, що реалізовані деякі мінімальні вимоги і зборка готова до подальшої експлуатації. Для приймання зборки переконаєтеся в тому, що компіляція і компонування проходять без збоїв, а перевірочні тести покривають усю базову функціональність. Додайте до складальних тестів перевірки, зроблені в останній ітерації.

Підтримка складальних тестів	Додайте або видаляйте розділи складальних тестів, щоб вони забезпечували адекватну перевірку складок
Спостереження за зборкою	Регулярно переглядайте звіти про деталі зборки

3. Випуск продукту

У певний момент набір реалізованих функцій стає достатнім для випуску продукту. Існує декілька типів випусків : альфа-, бета-версія і остаточна версія або допрацьований випуск. Цей випуск зробить більшу дію, ніж сама команда, яка його створювала, про нього судитимуть користувачі. Для програмних продуктів, створених для зовнішніх споживачів, потрібні маркетингові і торгові заходи. Системи, створені для внутрішнього застосування, вимагають організації навчання і підтримки експлуатації. Щоб заздалегідь почати готувати відповідних людей, складається план випуску. Процес випуску і пов'язаний з ним план багато в чому залежить від типу продукту і способу його застосування.

Операції:

Виконання плану випуску	Перевірте правильність матеріалу
Перевірка випуску	Створіть свій розділ для випуску
Нотатки про випуск	Задокументуйте виявлені обмеження
Розгортання продукту	Створіть настановний комплект

Операція: Виконання плану випуску

У плані випуску згадуються усі зацікавлені в отриманні нового випуску. Цей план дозволяє підготуватися до випуску продукту виробників носіїв, замовників і групу експлуатації. План дозволяє також проінформувати осіб, відповідальних за подальшу роботу з продуктом, про те, коли вони зможуть його отримати і на яку технічну підтримку має право розраховувати.

Перевірка матеріалів, пов'язаних з випуском	Перевірте, що матеріали, що відносяться до маркетингу, торгівлі, навчання і приймання замовником, відповідають складу продукту

Координація постачання продукту	Забезпечте постачання або розгортання продукту

Операція: Перевірка випуску

Коли виконані викладені в концепції і в описі підходу до тестування вимоги до функцій, якості і термінів, можна розглядати можливість випуску продукту (у тому числі альфа- або бета-версий). Щоб зборка стала офіційним випуском, потрібно піддати її остаточному регресійному тестуванню і проаналізувати його результати. Після завершення тестування потрібно вирішити, чи виправляти виявлені дефекти або розглядати зборку як кандидата на випуск.

Створення свого розділу для випуску	Виділіть в ієрархії проекту свій розділ для кандидата на випуск. Це захистить код і дозволить вносити тільки вибрані зміни
Виконання регресійних тестів	Виконайте для кандидата на випуск повний регресійний тест
Документування дефекту	Якщо тест не проходить, необхідно створити новий звіт про дефект. Оцініть вплив дефекту і навчіться його відтворювати

Операція: Створення нотаток про випуск

Нотатки про випуск — це короткий набір відомостей, супроводжуючий кожен випуск продукту. У цих замітках зібрані вимоги до середовища, описи установки і запуску застосування, нових можливостей, виправлених помилок, відомих дефектів і способів обходу проблем. З нотаток про випуск користувач дізнається подробиці про нову версію.

Документування виявлених обмежень	Опишіть вимоги середовища

Операція: Розгортання продукту

Завершуючий етап в створенні продукту — його розгортання. Розгортання служить для збору усіх компонентів, необхідних для кінцевого постачання. В деяких випадках сюди входить запис продукту на майстер-носій. У інших — установка продукту на технологічному сервері або на веб-вузлі, з якого він завантажуватиметься користувачами. Який би механізм не застосовувався, мета одна — передача системи користувачам.

Створення установочного комплекту	Зробіть для продукту установочний комплект, який спростить установку або розгортання застосування
Поширення випуску	Забезпечте постачання системи споживачам

4. Усунення дефекту

Наявність дефекту вказує на потенційну необхідність зміни вже працюючої програми. Усунення дефекту не повинне робити побічного ефекту. Щоб виправлення дефекту не порушувало працюючий код, процес корекції має бути методичним і керованим. Код, змінений при усуненні дефекту, необхідно перевірити на відповідність правилам кодування, автономно протестувати, переглянути, інтегрувати в застосування і зареєструвати в системі управління версіями. Усе це робить відповідальний за обробку дефекту учасник проектної

групи. Якщо усі перераховані дії не будуть виконані, «виправлення» буде гірше за початкову проблему.

Операції:

Відтворення дефекту	Керуйтеся описом дефекту
Створення або зміна тесту модуля	Визначте область охоплення тесту модуля
Визначення причини виникнення дефекту	Виділіть функціональну область
Перепризначення дефекту	Змініть опис дефекту
Вибір стратегії усунення дефекту	Проаналізуйте виявлений дефект
Зміна програми	Знайдіть потрібний код
Виконання тесту модуля	Виберіть тест модуля з колекції
Рефакторинг коду	Визначте складність
Огляд коду	Перевірте правильність імен
Інтеграція змін	Перевірте залежності

Операція: Відтворення дефекту

Передусім, потрібно спробувати відтворити дефект. Якщо зробити це не вдається, означає опис дефекту не містить достатню і правильну інформацію або несправність є такою, що перемежається. Необхідно з'ясувати дійсні умови прояву дефекту, виявити його і виправити.

Використання опису	Якщо в звіті про дефект міститься опис послідовності його відтворення, наслідуйте цю інструкцію
Отримання додаткових відомостей	Якщо відтворити дефект не вдається, зберіть більше відомостей. Використайте дані тестів і попередні звіти про дефекти, пов'язані з рішенням подібних проблем. Прогляньте звіти про помилки, з якими працювали інші розробники або фахівці з тестування
Відмова від обробки дефекту	Якщо прояв дефекту так і не вдалося відтворити, може бути прийняте рішення припинити обробку дефекту (перевести його в стан «Closed») на тій основі, що він не відтворюється («Unable to Reproduce»)

Операція: Визначення причини виникнення дефекту

Причину виникнення виявленого дефекту повинен визначити розробник. Для пошуку кореня проблеми розробник може застосовувати різні тактики і засоби. Стратегія усунення дефекту, як правило, виявляється досить очевидної після виявлення причини. Для пошуку хороші такі інструменти, як журнали, відлагоджувачі, лістинги і ін.

Виділення функціональної області	Виключіть з пошуку джерел проблеми області коду, не зухвалі підозри
Трасування підозрілого коду	Використайте візуальні можливості відладчика при пошуку дефекту

Аналіз системи усіма доступними засобами	Окрім трасування застосовуйте усі доступні засоби пошуку несправностей, у тому числі від сторонніх виробників
Локалізуйте проблему	Максимально звузьте діапазон пошуку дефекту
Аналіз коду	Якщо застосування відладчика неможливе з міркувань продуктивності або обмеженості ресурсів, проаналізуйте початковий текст рядок за рядком

Операція: Перепризначення дефекту

Причиною перепризначення дефекту може служити недолік відомостей про нього, малий досвід розробника по усуненню подібних проблем або балансування завантаженості виконавців. При перепризначенні дефекту додайте в описувач зведення про причину виконання цієї операції.

Зміна дескриптора дефекту	У бланку документування дефекту вкажіть причину перепризначення в полі діалогу
Зміна власника	Змініть відомості про особу, відповідальну за обробку дефекту

Операція: Вибір стратегії усунення дефекту

Розробник повинен вибрати оптимальний метод вирішення проблеми, що виключає додавання нових дефектів. При виборі способу виправлення дефекту враховуються питання архітектури, продуктивності і ресурсів. Рішення має бути націлене на усунення проблеми. Проте для відповідності вимогам і забезпечення можливостей користувачів, рішення може бути прийняте на шкоду якимсь функціям системи.

Аналіз виявленого дефекту	Визначте тип дефекту : проблеми з архітектурою, ресурсами або продуктивністю
Проблема з архітектурою	Визначте, на які архітектурні рішення вплине усунення дефекту
Проблема з ресурсами	Визначте фрагменти коду, на які зроблена дія
Проблема з продуктивністю	Визначте, на які фрагменти коду вплине усунення дефекту

Операція: Зміна програми

Після вибору підходу до усунення дефекту розробник повинен внести зміни в код. При цьому зборка повинна залишитися працездатною, не повинно бути внесено нових помилок, а в описувач дефекту мають бути внесені відповідні зміни.

Отримання потрібного коду	Знайдіть усі файли, які треба змінити для виправлення дефекту
Створення нового коду	Для виправлення дефекту, можливо, доведеться написати код в нових файлах
Зміна існуючих файлів	Для виправлення дефекту може знадобитися зміна існуючих файлів : додавання або видалення коду

Операція: Створення або зміна тесту модуля

За допомогою тесту модуля перевіряється коректність реалізації певного програмного компонента. Виконання тестів модулів при розробці зменшує число дефектів, що виявляються на етапі тестування, і дозволяє боротися з регресом —

появою нових дефектів при виправленні вже виявлених або додаванні нових функцій. Тести модулів не є тестами функцій або взаємодії; їх єдине призначення — перевірити автономну роботу фрагмента коду. Розробники повинні переконатися в тому, що існуючі тести модулів проходять після написання нового коду. Тестування повинне проводитися упродовж усього проекту — від реалізації архітектурних основ на початку до внесення доповнень у кінці проекту. Є різні типи тестових завдань, наприклад для перевірки сценаріїв і вимог до якості. При цьому усі різновиди тестів однаково виконуються і виявляють дефекти.

Визначення типу тесту модуля	Визначте типи тестів модулів, що розробляються. *Тести правильної роботи* (positive unit tests) перевіряють код в нормальному режимі і контролюють коректність результатів. У *тестах неправильної роботи* (negative unit tests) умисне некоректно використовується код і перевіряється його стійкість і адекватність обробки помилок. Тести з внесенням несправностей дозволяють виявити аномалії в обробці помилок
Створення або зміна тесту модуля	Для кожного завдання по розробці визначите необхідні тести модулів, якими можна перевірити якомога більше функцій. Напишіть тест модуля, переконайтеся, що він не проходить, напишіть або виділіть фрагмент коду шляхом рефакторинга і проженете тест модуля. Повторіть процедуру для усіх вибраних тестів модулів
Перевірка тесту модуля	Проженіть тест і переконайтеся, що він не проходить для незавершених фрагментів і проходить для тих елементів, які працюють як вимагається

Операція: Виконання тесту модуля

Тест модуля покриває певну область коду. Використовуючи комбінацію таких тестів, розробники і тестувальники визначають якість певного розділу програми. Виконуйте тести модулів після зміни початкового коду. Виконання тесту модуля підтверджує працездатність тієї частини програми, яка покривається цим тестом.

Визначення відповідних тестів модулів	Знайдіть тест або тести, що дозволяють найбільш адекватно перевірити відповідний елемент програми
Виконання тесту модуля	Проженіть тест для коду, який він покриває
Аналіз результатів тесту	Після закінчення кожного етапу відмітьте його відповідним чином: Pass (Пройшов), Fail (Не пройшов), Skip (Пропущений), Warning (Вимагає уваги) або Blocked (Заблокований)
Відладка коду	Виправіть помилки в програмі, що відноситься до завдання

Операція: Рефакторинг коду

В процесі рефакторинга коду проганяйте тести модуля після кожної зміни. При такому підході ризик ушкодження програми мінімальний. По можливості виконуйте автоматизований рефакторинг, оскільки автоматизовані процеси мінімізують вірогідність помилок. Рефакторинг потрібно проводити постійно.

Визначення складності	При додаванні нових функцій, зверніть увагу на ті частини програми, структура яких стала складнішою
Застосування рефакторинга	При рефакторингу вносите за один раз одну зміну. Змініть код і усі посилання на змінену область
Виконання тестів модуля	Виконайте тести модуля, щоб переконатися в семантичній цілісності коду після рефакторинга. Виправіть усі непрацездатні тести модулів

Операція: Огляд коду

Огляд коду застосовується для виявлення областей, з якими можуть виникнути проблеми в процесі подальшої розробки або тестування. Він також дає можливість дізнатися думки інших розробників про даний код. Огляди коду дозволяють учасникам, працюючим в тій же області, наслідувати прецеденти, створені попередніми розробниками. У таких партнерських зустрічах потрібно присутність другого обізнаного колеги, з яким розробник повинен пройти по усіх змінах, перш ніж зареєструвати код в системі управління версіями. Мають бути виконані тести модулів і проведений аналіз коду. У оглядах треба зосередитися на областях, які не можна перевірити за допомогою компілятора або за допомогою аналізу коду, — на продуктивності, читабельності, безпеці.

Перевірка правильності імен	Імена класів і методів повинні відбивати призначення відповідних фрагментів коду
Перевірка адекватності коду	Даний код повинен відповідати завданню, для якого він написаний. Допустимі тільки такі зміни коду, які додають або змінюють функції системи
Перевірка розширюваності	Написаний код повинен допускати розширюваність (якщо ставилося таке завдання) або можливість повторного використання в інших частинах системи
Перевірка допустимої	Код, що повторюється, має бути зібраний в загальних функціях
Перевірка складності алгоритму	Число можливих гілок коду має бути мінімальним. Право на існування мають тільки явно необхідні гілки
Перевірка безпеки коду	Перевірте захист об'єктів, рівні привілеїв і використання даних в точках входу. При перевірці використайте контрольний список
Внесення змін за результатами огляду	Внесіть зміни, намічені в результаті огляду коду, скомпілюйте програму, виконайте тести модулів і виконайте аналіз коду. Якщо які-небудь тести не пройшли, виявіть помилки і виправіть код

Операція: Інтеграція змін

Щоб застосування було стійким і погодженим, необхідно організувати інтеграцію змін в код. Коли код складається з невеликих змін, кожне з яких відповідає завданню по розробці або виправленню дефекту, його простіше підтримувати в стабільному стані. Легше знайти і ізолювати потенційну проблему. Після інтеграції останнього завдання по розробці, відповідальний за реалізацію відповідного сценарію або вимоги до якості, розробник перевіряє усю

функціональність від початку до кінця. Для стану сценарію або вимоги до якості встановлюється значення Resolved (Оброблений) і описувач призначається тестувальникові.

Перевірка залежностей	Якщо завдання залежить від інших завдань, які ще не завершені, дочекайтеся, коли вони будуть інтегровані в систему
Тестування і інтеграція інших завдань по розробці	Перевірте, що зміни, що вносяться вами, пов'язані з виправленням дефекту, реалізацією частини сценарію або вимоги до якості, нормально працюють спільно із вже інтегрованими змінами
Реєстрація пакету змін	Збільште номер в полі «Resolved in Build» (Вирішено у зборці) для завдання виправлення дефекту або в полі «Integration Build» (Зборка з реалізацією) для завдання по розробці
Завершення завдання	Якщо описувач, з яким пов'язані зроблені зміни, представляє сценарій або вимогу до якості, а ви не є його власником, повідомите власника про те, що завершили внесення змін

5. Закриття дефекту

Підстав для закриття дефекту може бути декілька. Із закритим дефектом в поточній ітерації більше не виконуються ніякі дії. Дефект може бути закритий тому, що він виправлений, його усунення відкладене до наступного випуску продукту, він перестав бути актуальним, його неможливо відтворити або він відповідає раніше зареєстрованому дефекту. Закриття дефекту зазвичай відбувається при класифікації або після усунення і перевірки цього факту.

Перевірка виправлення	Спробуйте відтворити дефект
Закриття дефекту	Підтвердіть, що дефект повторює існуючий

Операція: Перевірка виправлення

Необхідно переконатися, що дефект виправлений коректно і не порушені інші функції. Після того, як розробник виправив дефект, в справу вступає тестувальник, який відповідає за прогін тестів. Якщо тести проходять успішно, дефект вважається закритим. Інакше він наново призначається розробникові.

Спроба відтворити дефект	Спробуйте відтворити дефект, виконавши представлену в описувачі дефекту послідовність дій
Несподівана поведінка	Виконайте суміжні функції і спробуйте знайти несподівані моменти в поведінці системи, які можуть бути пов'язані з дефектом. Особливо важливо це у разі перевірки завдання по розробці, нереалізованій повністю
Отримання додаткових відомостей про дефект	Якщо в описувачі дефекту недостатньо даних або вони незрозумілі, звернетеся до потрібного розробника або тому, хто виявив і задокументував дефект

Перепризначення дефекту	Якщо тестове завдання не проходить, дефект знову призначається розробникові

Операція: Закриття дефекту

Підстав для закриття дефекту є декілька. Частенько дефекти виправляються розробниками, потім позначаються як вирішені і у кінці закриваються менеджерами проектів. Дефект закривається, якщо його виправлення переноситься на інший випуск; вважається неактуальним, якщо доведено, що його не можна відтворити або підтверджено, що він дублює існуючий.

Оновлення описувача повторного дефекту	Якщо дефект аналогічний існуючому, в його описувачі потрібно послатися на аналог
Дефекти, що не усуваються	Якщо дефект не буде виправлений, детально опишіть причину. Це може бути обмеження системи або відповідність дизайну
Оновлення описувача закритого дефекту	Якщо більше не вдається добитися появи дефекту, додайте відповідні відомості в описувач і закрийте дефект

Лекція 8. Реалізація завдань по розробці згідно MSF.

1. Основні операції по розробці.
2. Реалізація завдання по розробці бази даних

1. Основні операції по розробці.

Завдання по розробці — невелика частина діяльності розробника, пов'язана з вимогою до якості або сценарієм. В результаті реалізації завдання по розробці до системи додається нова функція. Після завершення завдання по розробці необхідно провести тестування модулів, огляд коду і його аналіз, інтеграцію коду і реєстрацію його у базі коду. Потім сценарій або вимога до якості передається на тестування.

Операції:

Оцінка завдання по розробці	Оцініть тривалість завдання по розробці
Створення або зміна тесту модуля	Визначте тип тесту модуля
Написання програми	Виберіть компонент
Аналіз коду	Визначте правила застосування
Виконання тесту модуля	Виберіть тест модуля з набору
Рефакторинг коду	Визначте складність
Огляд коду	Перевірте правильність імен
Інтеграція змін	Перевірте залежності

Операція: Оцінка вартості завдання по розробці

Оцінка вартості завдання по розробці допомагає обмежити набір функцій, що реалізовуються, визначити розклад і розподілити пріоритети. Оцінку усіх завдань по розробці і рішення усіх пов'язаних з цим проблем необхідно виконати до наради по плануванню ітерації. Якщо загальна вартість завдань по розробці перевищує витрати на ітерацію, завдання має бути відкладене або перепризначене. Після затвердження завдання за його оцінку відповідає розробник.

Оцінка на основі досвіду	Проводьте оцінку з урахуванням часу виконання аналогічних завдань
Балансування завантаження	Якщо в результаті оцінки з'ясовується, що об'єм робіт перевищує можливий рівень для ітерації, спільно з менеджером проекту спробуйте перерозподілити завантаження або перенести завдання на іншу ітерацію
Деталізація завдання	Розгляньте завдання по розробці з урахуванням інших подібних завдань, а також вимог до якості і сценаріїв, для яких ще не призначені завдання по розробці. Створіть для них завдання

Визначте способи інтеграції	Спільно з іншими учасниками групи розробки виробіть чітку картину інтеграції цієї функціональності з іншими функціями

Операція: Написання програми

Створіть код, визначте змінені області, напишіть тести модулів, проведіть огляд коду і внесіть в нього необхідні зміни, на завершення зареєструйте код на сервері управління версіями. Вам також може потрібно виконати відладку, рефакторинг і використати проміжні версії початкового коду. Після того, як код написаний, необхідно виконати його огляд і переконатися, що його якість відповідає вимогам і прийнятим стандартам. Створений код не повинен робити шкідливих побічних ефектів на інші частини застосування.

Вибір компонента для коду	Зіставте завдання з компонентом, в якому реалізується відповідна функціональність
Створення нових класів або методів	За допомогою діаграми класів створіть класи, необхідні для реалізації цієї функціональності
Реалізація алгоритму методу	Витягніть існуючі класи, які треба змінити, з системи управління версіями

Операція: Аналіз коду

Аналіз коду — це процес перевірки звичайного або керованого коду .NET на предмет відповідності керівним принципам по розробці. Для керованого коду .NET в процесі аналізу перевіряється відповідність генерованих складок рекомендаціям Microsoft .NET Framework Design Guidelines. Пропонується автоматична перевірка складок на наявність більш ніж 200 видів дефектів, таких як порушення угод по іменуванню, помилки в конструкції бібліотек, а також проблеми локалізації, безпеки і продуктивності. Завдання аналізу коду при роботі з новими базами коду — забезпечити відсутність дефектів. Для існуючих баз з великим числом правил формування попереджень мета полягає в мінімізації числа попереджень в кожній категорії.

Визначення прийнятних правил	Виберіть прийнятні правила для цієї категорії застосувань
Забезпечення відсутності дефектів	Перевірте код на відповідність правилам
Зменшення числа дефектів по категоріях	Для існуючих баз коду з великим числом проблем з відповідністю правилам визначите базові показники рівня проблем. При використанні автоматизації, визначите число попереджень в існуючій базі коду

2. Реалізація завдання по розробці бази даних

Завдання по розробці бази даних — це діяльність розробника, пов'язана з вимогою до якості або сценарієм. В результаті реалізації завдання по розробці бази даних до системи додається нова функція. Після завершення цього завдання необхідно провести тестування модулів, огляд коду і його аналіз, інтеграцію коду і реєстрацію його в існуючій базі коду. Потім сценарій або вимога до якості передається на тестування. Описаний тут ізольований інтерактивний процес розробки баз даних забезпечує гнучкість тієї частини застосування, яка пов'язана з

даними. Це досягається завдяки використанню окремого проекту розробки бази даних, який забезпечує автономні зміни. Найважливіше, що ці зміни можуть бути перевірені автономно за допомогою інструментальних засобів автоматизованого тестування, що гарантує відсутність регресу. Таким чином, при кожній зміні у базі даних проводиться її тестування. Усі зміни негайно потрапляють у вашу ізольовану базу даних («пісочницю») і відразу ж тут тестуються. Тільки після цієї зміни можна розгортати на робочому сервері — ризик некерованих змін при цьому мінімальний. З іншого боку, такий підхід забезпечує високу гнучкість, завдяки тісній співпраці з групою розробки основного застосування. Використання завдань по розробці дозволяє учасникам проектної групи ефективно взаємодіяти і досягати спільних цілей.

Операції:

Оцінка завдання по розробці бази даних	Проаналізуйте поставлене завдання
Поновлення локального середовища проекту	Витягніть з системи управління версіями необхідну версію проекту бази даних
Кодування	Створіть нові або відновіть існуючі об'єкти схеми
Виконання тесту модуля	Визначте тип тесту модуля бази даних
Рефакторинг коду	Визначте складність
Огляд коду	Перевірте правильність імен
Інтеграція змін	Перевірте залежності

Операція: Оцінка завдання по розробці бази даних

Оцінка вартості завдань по розробці бази даних допомагає обмежити набір функцій, що реалізовуються, визначити розклад і розподілити пріоритети. Оцінка усіх завдань по розробці бази даних виконується на нараді по плануванню ітерації. Якщо загальна вартість завдань по розробці перевищує витрати на ітерацію, завдання має бути відкладене або перепризначувала. Менеджер проекту і бізнес-аналітик визначають пріоритети завдань і відкладають виконання найменш пріоритетних з них. Після затвердження завдання за її оцінку відповідає розробник. Перш ніж приступати до оцінки завдання по розробці бази даних, розробник баз даних повинен переглянути призначене йому завдання, розібратися у вимогах і обговорити оцінку з іншими розробниками баз даних. В результаті усі розробники матимуть єдине уявлення про завдання.

Аналіз поставленого завдання	Знайдіть пов'язаний з поставленим завданням описувач
Деталізація завдання	Розгляньте завдання по розробці з урахуванням інших подібних завдань, а також вимог до якості і сценаріїв, для яких ще не призначені завдання по розробці. Створіть для них завдання
Оцінка на основі досвіду	Проводьте оцінку з урахуванням часу виконання аналогічних завдань

Балансування завантаження	Менеджер проекту і бізнес-аналітик визначають пріоритети завдань і відкладають виконання найменш пріоритетних з них. Якщо в результаті оцінки з'ясовується, що об'єм робіт перевищує можливий рівень для ітерації, спільно з менеджером проекту спробуйте змінити пріоритети і перерозподілити завантаження
Визначте способи інтеграції	Спільно з іншими учасниками групи розробки виробіть чітку картину інтеграції цієї функціональності з іншими функціями

Операція: Поновлення локального середовища проекту

Перш ніж почати розробку, треба настроїти середовище розробки локальної бази даних. Спочатку необхідно витягнути з системи управління початковим кодом потрібну версію проекту. Цей проект потрібно зібрати і розгорнути на ізольованому сервері (sandbox). Це і буде локальне ізольоване середовище, в якому розробник перевірятиме кожну зміну, що вноситься в проект бази даних, перш ніж повертати початковий текст в систему управління версіями і розгортати систему.

Витягніть з системи управління версіями необхідну версію проекту бази даних	Визначте версію проекту бази даних, з яким ви хочете синхронізуватися
Визначення ізольованого сервера	Виберіть сервер баз даних, який ви використовуватимете як «пісочниці»
Компонування проекту бази даних	Переконайтеся, що параметри проекту баз даних налаштовані на ізольований сервер
Розгортання проекту баз даних на ізольованому сервері	Розгорніть проект баз даних на ізольованому сервері

Операція: Кодування

Наступний крок — власне написання коду для завдання по розробці бази даних. Це означає оновлення існуючих об'єктів схеми бази даних або створення нових. Після реалізації чергового пакету змін його треба розгорнути на ізольованому сервері для тестування.

Створення нових або оновлення існуючих об'єктів схеми	При необхідності створіть нові об'єкти схеми
Зборка проекту бази даних	Зберіть проект бази даних
Розгортання проекту баз даних на ізольованому сервері	Розгорніть проект баз даних на ізольованому сервері

Операція: Виконання тесту модуля бази даних

За допомогою тесту модуля перевіряється коректність реалізації певного програмного компонента. Виконання тестів модулів при розробці знижує

кількість дефектів, що виявляються на етапі тестування, і дозволяє боротися з регресом — появою нових дефектів при виправленні вже виявлених дефектів або при додаванні нових функцій. Тести модулів не є тестами функцій або взаємодії; їх єдине призначення — перевірка автономної роботи фрагмента коду. Розробники повинні переконатися, що існуючі тести модулів проходять після написання нового коду. Тестування повинне проводитися на протязі усього проекту — від реалізації архітектурних основ на початку до внесення доповнень у кінці проекту. При цьому усі різновиди тестів однаково виконуються і виявляють дефекти. При тестуванні модулів баз даних необхідно перед написанням тестів модулів створити або відновити план генерування даних, щоб при тестуванні були необхідні тестові дані.

Визначення типу тесту модуля	Визначте типи тестів модулів бази даних, що розробляються. *Тести правильної роботи* (positive unit tests) перевіряють код в нормальному режимі і контролюють коректність результатів. *Тести неправильної роботи* (negative unit tests) умисне некоректно використовують код і перевіряють його стійкість і адекватність обробки помилок
Створення або зміна плану генерації тестових даних	Якщо плану генерації тестових даних не існує, створіть його
Створення або зміна тесту модуля бази даних	Напишіть нові тести модулів для перевірки об'єктів схеми
Виконання тесту модуля бази даних	Переконайтеся в правильному налаштуванні параметрів виконання тесту модуля бази даних
Аналіз результатів тесту	Якщо тест модуля не пройшов з тієї причини, що був невірним, виправіть його і знову виконайте
Відладка коду	Виправіть помилки в програмі, що відноситься до завдання
Виконання тестів модуля застосування	При ітеративній розробці бази даних іноді доцільно виконувати тести модулів застосування спільно з тестами модулів бази даних. Оскільки застосування частенько дуже інтенсивно звертається до даних, зміни у базі даних можуть вплинути на прикладний шар, якщо їх не брати до уваги. Спільне виконання тестів застосування і бази даних дасть упевненість в тому, що проблеми будуть помічені завчасно

Операція: Рефакторинг бази даних

В процесі рефакторинга коду проганяйте тести модуля після кожної зміни. При такому підході ризик ушкодження програми мінімальний. По можливості виконуйте автоматизований рефакторинг, оскільки автоматизовані процеси мінімізують вірогідність помилок. Відмітимо, що рефакторинг потрібно проводити постійно.

Визначення складності	При додаванні нових функцій звернете увагу на ті частини програми, структура яких стала складнішою

Застосування рефакторинга	При рефакторингу вносите за раз одну зміну. Змініть код і усі посилання на змінену область
Зборка проекту бази даних	Зберіть проект бази даних
Розгортання проекту баз даних на ізольованому сервері	Розгорніть проект баз даних на ізольованому сервері
Виконання тестів модуля бази даних	Виконайте тести модуля, щоб переконатися в семантичній цілісності коду після рефакторинга

Операція: Огляд коду

Огляд коду застосовується для виявлення областей, з якими можуть виникнути проблеми в процесі подальшої розробки або тестування. Огляд коду також дає можливість дізнатися думки інших розробників про даний код. Огляди коду дозволяють учасникам, працюючим в тій же області, наслідувати прецеденти, створені попередніми розробниками. У таких партнерських зустрічах потрібна присутність другого обізнаного колеги, з яким розробник повинен розглянути усі зміни, перш ніж зареєструвати код в системі управління версіями.

Перевірка правильності імен	Імена об'єктів схеми бази даних повинні відбивати призначення відповідних сутностей
Перевірка адекватності коду	Даний код повинен відповідати завданню, для якого він написаний. Допустимі тільки такі зміни коду, які додають або змінюють функції системи
Перевірка допустимої складності коду	Код, що повторюється, має бути зібраний в загальних функціях
Внесення змін за результатами огляду	Внесіть усі зміни, заплановані в результаті огляду
Зборка проекту бази даних	Зберіть проект бази даних
Розгортання проекту баз даних на ізольованому сервері	Розгорніть проект баз даних на ізольованому сервері
Виконання тестів модуля бази даних	Виконайте тести модуля і переконайтеся у відсутності регресу

Операція: Інтеграція змін

Щоб застосування було стійким і погодженим, необхідно організувати інтеграцію змін в код. Коли код складається з невеликих змін, кожне з яких відповідає завданню по розробці або виправленню дефекту, його простіше підтримувати в стабільному стані. Легше знайти і ізолювати потенційну проблему. Для стану сценарію або вимоги до якості встановлюється значення Resolved (Оброблений) і описувач призначається тестувальникові.

Перевірка залежностей	Якщо завдання залежить від інших завдань, які ще не завершені, дочекайтеся, коли вони будуть інтегровані в систему

Зборка проекту бази даних	Зберіть проект бази даних
Розгортання проекту баз даних на ізольованому сервері	Розгорніть проект баз даних на ізольованому сервері
Тестування і інтеграція інших завдань по розробці баз даних	Перевірте, що зміни, що вносяться вами, пов'язані з виправленням дефекту, реалізацією частини сценарію або вимоги до якості, нормально працюють спільно із вже інтегрованими змінами
Реєстрація пакету змін	Зареєструйте змінений код в системі управління версіями початкового коду. Якщо тест реалізований або виконаний не повністю, збережете увесь код як проміжну версію (shelveset). Інакше встановите ознаку вирішеного завдання. В результаті описувачу буде зіставлений новий пакет змін
Завершення завдання	Встановіть для описувача ознаку Resolved (Оброблений) і призначте сценарій одному з тих тестувальників, які писали тестові завдання для цього сценарію. Призначте цього тестувальника новим власником описувача завдання

Лекція 9. Тестування вимог до якості Agile ПЗ.

1. Основні операції тестування
2. Перевірка сценарію

1. Основні операції тестування

Завдання реалізації вимоги до якості передається на тестування, коли в поточній зборці відбиті обмеження, що накладаються цією вимогою. Для перевірки згаданих обмежень треба розуміти, з чим вони пов'язані. Частенько з вимогою пов'язані сценарії, які демонструють ці обмеження. Тестування вимоги до якості має на увазі успішне завершення тестів на продуктивність, безпеку, стрес і навантаження. На підставі результатів тестування створюються описувачі дефектів, в яких документуються виявлені проблеми.

Операції:

Визначення підходу до тестування	Визначте контекст проекту
Створення тесту продуктивності	Визначте призначення тесту
Створення тесту безпеки	Вивчіть точки входу
Створення стресового теста	Спроектуйте тест
Створення теста навантаження	Визначте призначення теста навантаження
Вибір і запуск тестового завдання	Виберіть, який тест запускати
Документування дефекту	Ідентифікуйте дефект
Виконання дослідницьких тестів	Встановіть тривалість сеансу

Операція: Визначення підходу до тестування

Підхід до тестування — це документ, що визначає стратегію створення і виконання тестів. У нім також визначені стандарти якості для продукту, що поставляється. Підхід до тестування служить відправною точкою для планування тестів на початку проекту, але продовжує існувати і мінятися по ходу проекту. Підхід повинен описувати різні методики, наприклад тести, як ручні, так і автоматичні. Формулювання підходу до тестування включає визначення контексту тесту і його кінцевої мети (чи цілей), а також вибір відповідних методик, з урахуванням цілей і в контексті проекту. Перед початком чергової ітерації цей документ треба злегка коригувати, відбиваючи цілі тестування в майбутній ітерації і додаючи опис використовуваних тестових даних.

Визначте контекст проекту	Визначте ризики проекту, а також користувачів, на яких може позначитися їх дія. Знайдіть особливі ситуації, які можуть вплинути на рівень необхідного тестування. Наприклад, діючі в предметній області норми можуть впливати на напрями тестування
Визначення мети тесту	Визначте цілі проекту, досягненню яких сприятиме тест
Оцінка можливих методів тестування	Розгляньте різні інструментальні засоби для тестування
Визначення показників тестування	Спільно з розробниками визначите реалістичні порогові значення, які будуть критеріями проходження тесту. Це обов'язковий показник

Операція: Створення тесту продуктивності

За допомогою тесту продуктивності визначається час реакції застосування і перевіряється відповідність вимогам до якості. Тестування продуктивності необхідно проводити в кожній ітерації і резервувати час для усунення основних проблем і для проведення повторного тестування.

Визначення призначення тесту	Необхідно чітко сформулювати призначення тесту. Наприклад, за допомогою тесту перевіряється відсутність істотних відмінностей в часі відгуку системи для користувачів з різними способами підключення до Інтернету
Опис конфігурації тесту	Визначте конфігурацію тесту
Проектування тесту	Систематизуйте тестовані події і сценарії, що відповідають їм. Розділіть сценарії на області, критичні і не критичні до продуктивності. Спільно з архітектором побудуйте модель продуктивності, яка буде основою для кодування
Документування етапів тестування	Виділіть і опишіть етапи тестування, щоб тестувальники могли однаково виконувати і інтерпретувати тести. Помилки на цьому кроці можуть привести до невірної оцінки продуктивності. Скрізь, де це можливо, автоматизуйте створення тестових завдань

Операція: Створення тесту безпеки

Тести безпеки або випробування на проникнення (penetration tests) мають справу із загрозами, виявленими при проектуванні. Тестування цього типу ділиться на три частини: вивчення, визначення порушень і розшук. В результаті випробування на проникнення можуть бути виявлені нові уразливості, що спричинить формулювання нових вимог до якості або ідентифікацію дефекту. В результаті реалізації такої вимоги або усунення дефекту мають бути заблоковані відповідні точки входу і доступ до ресурсів. З цього виходить, що тестувальники мають бути обізнані про елементи моделі загроз не гірше архітекторів. Цей вид тестування вимагає спеціальних навичок: треба уміти імітувати дії потенційного

зловмисника. Доступ до моделі загроз дає тестувальникам можливість систематично атакувати систему в цілях перевірки, не порушуючи її роботу.

Вивчення точок входу	Визначте точки входу системи і її функціональних блоків для захисту внутрішніх об'єктів. Використайте зведення з моделі загроз для визначення можливих напрямів атак
Ідентифікація проблем	Напишіть тестові завдання із запуском тестів прямого або псевдовипадкового доступу до об'єктів. Для перевірки спеціальних захисних функцій потрібно пряме втручання. Наприклад, подивіться, чи не можна з URL отримати ідентифікатор сеансу і змінити номер рахунку
Використання слабкостей захисту	Додайте тестові завдання з використанням будь-яких слабкостей захисту. Деякі з цих тестів будуть дослідницькими, а не націленими на виправлення проблем. Враховуйте час, необхідний для опису способів обходу захисту. Несанкціонований доступ до системи є дефектом і для його виправлення потрібно знати спосіб доступу до захищеного ресурсу. Сценарії запуску тестів безпеки повинні відбивати загальну тактику захисту секретних даних, оберігання від несанкціонованого доступу і заборони доступу для нелегітимних користувачів

Операція: Створення стресового теста

Стрес-тести дозволяють визначити граничні можливості застосування. З їх допомогою з'ясовують верхню межу, перейшовши яку застосування не забезпечує прийнятного рівня реактивності і повністю припиняє роботу. Наприклад, можна дати застосуванню високе навантаження і подивитися, чи немає витоків пам'яті і чи нормально звільняються об'єкти збирачем сміття.

Стрес-тести повинні проводитися в кожній ітерації. Вони є різновидом тестів продуктивності і дозволяють перевіряти екстремальні умови роботи застосування, наприклад, коли навантаження перевищує типову або триває довше за звичайний. Стрес-тести дозволяють спрогнозувати поведінку застосування при перевищенні максимально можливого навантаження. Також вони служать для перевірки стійкості і надійності застосування.

Визначення призначення тесту	Призначення стрес-теста має бути чітко сформульоване, разом з описом конкретних параметрів і умов, таких як граничні значення
Проектування автоматизованого теста	Систематизуйте особливості тестового середовища, умови тестування, включаючи передумови, число віртуальних користувачів, сценарій і ресурси. Розберіться з розподілом дій і сценаріїв

Створення тесту	Задокументуйте виконувані користувачем дії. Додайте перевірку коректності отримуваних результатів в стресових умовах. Щоб тести були динамічними, використайте прив'язку до даних — звернення до таких джерел даних, як SQL Server, Microsoft Excel або Microsoft Access

Операція: Створення теста навантаження

Тест навантаження є різновидом тесту продуктивності. Тести навантажень дозволяють перевірити відповідність застосування вимогам до якості в умовах навантаження. З їх допомогою вимірюється продуктивність застосування в типових сценаріях роботи користувачів. Тести навантажень дозволяють спрогнозувати поведінку застосування досягши максимального навантаження. Тестами навантажень доцільно перевіряти ті 20% кода застосування, на які доводиться 80% його роботи.

Визначення призначення тесту	Призначення теста навантаження має бути чітко сформульоване, разом з описом конкретних параметрів і умов, таких як допустимий діапазон значень
Проектування автоматизованого теста	Систематизуйте особливості тестового середовища, умови тестування, включаючи передумови, число віртуальних користувачів, сценарій і ресурси. Розберіться з розподілом дій і сценаріїв
Створення тесту	Задокументуйте виконувані користувачем дії. Додайте перевірку коректності отримуваних результатів в стресових умовах. Щоб тести були динамічними, використайте прив'язку до даних — звернення до таких джерел даних, як SQL Server, Microsoft Excel або Microsoft Access

Операція: Вибір і запуск тестового завдання

Найважливіший момент в тестуванні — це власне виконання тестів відповідної зборки. Результати тестів можуть бути абсолютно різними: від успішного завершення до повного виходу системи з ладу. Не можна забувати фіксувати результати тестів в описувачі відповідного сценарію або вимоги до якості.

Визначення теста, що запускається	Визначте тести, які треба виконати, і присвойте їм пріоритети на основі відомостей в описувачі завдання тестування, загального плану тестування і положень документу, що визначає підходи до тестування
Визначення конфігурації тесту	Встановіть і настройте тестову конфігурацію, включаючи апаратні засоби, програми і тестові дані
Отримання зборки	Витягніть зборку з системи контролю версій або отримайте її з іншого авторизованого джерела
Виконання тесту	Виберіть теку з тестом сценарію або вимоги до якості

Аналіз результатів тесту	Зв'яжіть з результатами тесту перевірений сценарій або вимогу до якості
Закриття описувача	Якщо усі тести із завдання тестування виконані, закрийте описувач цього завдання

Операція: Документування дефекту

Перш ніж створювати описувач нового дефекту, перевірте, чи не реєструвалася раніше подібна проблема. Інакше, зафіксуйте усі подробиці в системі відстежування дефектів. Необхідно описати послідовність відтворення проблеми, після чого, в процесі класифікації, їй буде призначений пріоритет. Опишіть дефект як можна детальніше — це допоможе фахівцям визначити оптимальний шлях його усунення. Кожному відкритому дефекту має бути призначений власник.

Перевірка повторення дефекту	Перш ніж створювати описувач нового дефекту, перевірте, чи немає опису аналогічного дефекту у базі даних
Опис дефекту	Якщо аналогів у дефекту немає і він може бути відтворений, виконаєте послідовність дій, що призводить до його появи і детально змалюйте те, що відбувається в описувачі
Призначення дефекту	Визначте основну область проблеми. Якщо зачіпається декілька областей, виберіть одну

Операція: Виконання дослідницьких тестів

Дослідницьке тестування — це спосіб перевірки продукту без певного набору тестів. Для дослідницького тестування підходять багато евристичних методів. Серед них — використання збірних образів, прикметники і говір, аналіз змінних, пошук діапазону і тестування різних станів.

Представлений в цьому керівництві евристичний метод описує, як продукт тестується з точки зору збірного образу з метою збору нових вимог. Для проведення подібного тестування представте збірний образ і виконуйте функції системи, намагаючись досягти певної мети. Якщо ви виявите нові потенційні цілі або з'ясуєте, що існуючі функції не задовольняють вимогам збірного образу, додайте нові або змініть існуючі сценарії. Обмежуйте сеанси дослідницького тестування двома годинами.

Установка тривалості сеансу	Тривалість сеансу дослідницького тестування не має бути менше півгодини і більше двох годин
Задоволення потреб збірного образу	Виберіть збірний образ з опублікованого списку персонажів
Виявлення нових цілей	Проаналізуйте труднощі, з якими стикається збірний образ. Якщо здолати їх дуже складно, відкрийте новий дефект або додайте в список сценаріїв новий сценарій або вимогу до якості

2. Перевірка сценарію

Типовий сценарій включає як завдання тестування, так і завдання по розробці. Ці завдання призначаються тестувальникам, які розробляють і проганяють тестові завдання. Завдання реалізації сценарію передається на тестування, коли реалізовані функції інтегровані у зборку і вона готова до тестування. Щоб перевірити, що у зборці реалізовані необхідні функції, що виходять з сценарію, треба розбиратися в нім і в граничних умовах. Для перевірки усіх функцій зборки і граничних умов необхідно написати перевірочні тести. Усі виявлені дефекти мають бути задокументовані за допомогою описувачів.

Операції:

Визначення підходу до тестування	Визначте контекст проекту
Створення перевірочного теста	Визначте область тестування і середовище
Вибір і запуск тестового завдання	Виберіть, який тест запускати
Документування дефекту	Ідентифікуйте дефект
Виконання дослідницьких тестів	Встановіть тривалість сеансу

Операція: Створення перевірочного теста

Перевірочними тестами досліджують відповідність реалізованих функцій вимогам сценаріїв. При перевірочному тестуванні застосування розглядається як «чорний ящик» і особлива увага приділяється кінцевим користувачам. Перевірочні тести не мають бути зав'язані на проектні рішення. Для написання перевірочних тестів досліджуйте труднощі, які виникнуть в різних ситуаціях. Не усі труднощі вказані в сценаріях. При проектуванні і написанні тестових сценаріїв для перевірочних тестів фахівці розглядають ситуації, які можуть мати місце в реальному житті.

Визначення області тестування і середовища	Виділіть область, в якій повинен виконуватися тест. Тести ітерації — це набір автоматичних тестових завдань, що запускаються після перевірочних тестів зборки. При цьому проходження цих тестів не означає працездатність зборки. Помітимо, що тільки автоматичні тести можуть бути тестами ітерації
З'ясування подробиць виконання тесту	Визначте тестові дані, необхідні для тестового завдання. Витягніть з системи управління версіями опис підходу до тестування. У розділ, що відноситься до відповідної ітерації, додайте опис тестових даних
Створення тестових завдань	Створіть для сценарію теку з тестами, якщо її ще немає

Лекція 10. Розробка проекту бази даних.

1. Операції по створенню проекту бази даних.
2. Розгортання проекту бази даних.

1. Операції по створенню проекту бази даних

Операції по створенню проекту бази даних включають розробку початкового проекту бази даних для використання командою розробників. У рамках основного процесу створюється автономне представлення бази даних, що розробляється, яке доступне усій команді за допомогою системи управління початковим кодом. Створення такого проекту дозволяє вести незалежну ітеративну розробку бази даних одночасно декількома розробниками в стилі, який зменшує ризики при внесенні змін. Ця операція складається з таких дій, як створення, конфігурацію і тестування проекту бази даних.

Частенько буває треба створити кероване середовище розробки для вже існуючої бази даних. В цьому випадку потрібна додаткова дія — імпорт робочої бази даних в новий проект. Виконати це можна, тільки маючи певні права в робочій базі даних, а тому це завдання варто доручити адміністраторові бази даних, який має необхідні права доступу. Майстер проекту бази даних (Database Project Wizard) корисний при виконанні дій, описаних нижче, він спрощує створення проекту бази даних. Після його завершення, проте, може знадобитися додаткова конфігурація.

Операції:

Створення проекту бази даних	Вивчіть поставлене завдання
Імпорт існуючої бази даних	Імпортуйте схему існуючої бази даних в проект бази даних
Установка параметрів зборки і розгортання	Вкажіть сервер, де проводитиметься розробка
Зміна згенерованих сценаріїв	Відкоригуйте сценарії, що виконуються перед розгортанням
Перевірка проекту бази даних	Виконайте зборку проекту бази даних
Розміщення проекту бази даних в службі управління початковим кодом	Помістіть проект бази даних в систему управління початковим кодом

Операція: Створення проекту бази даних

Першим кроком в цій операції є створення проекту бази даних на основі відповідного шаблону. Крім того, мають бути задані різні параметри проекту бази даних, оскільки від цього залежить подальше життя проекту.

Вивчення поставленого завдання	Прогляньте описувачі завдань, поставлених перед вами

Створення проекту нової бази даних	Вирішіть, який тип проекту вам потрібний — SQL Server 2000 або SQL Server 2005
Конфігурація властивостей проекту	Вирішіть, як буде організований проект бази даних — по типах об'єктів або схемою
Установка необов'язкових параметрів	Конфігуруйте необов'язкові параметри бази даних, включаючи SET -параметри

Операція: Імпорт існуючої бази даних

Частенько буває треба створити проект бази даних на основі вже існуючої бази шляхом імпорту її схеми. Це дозволяє імпортувати схему визначення об'єктів і додати їх до автономного проекту бази даних, розміщеного у файловій системі, тим самим надаючи можливість міняти їх локально усередині проекту.

Імпорт існуючої бази даних в новий проект	Вкажіть з'єднання з базою даних, з якої проводитиметься імпорт схеми даних, а також вкажіть обліковий запис з необхідними повноваженнями в ній

Операція: Установка параметрів зборки і розгортання

Впродовж усього часу розробки бази даних, розробники будуть постійно, на ітеративній основі збирати і розгортати проект бази даних на своїх локальних серверах. Саме тому так важливо під час первинної конфігурації проекту настроїти параметри для зборки і установки, що задаються за умовчанням, які б якнайкраще підходили для середовища розробки.

Вибір сервера, на якому проводитиметься тестування	Це має бути локальний сервер баз даних, призначений для тестування ітеративних змін
Налаштування параметрів установки	Вирішіть, чи буде база даних при кожній установці автоматично створюватися з нуля, або на ній розгортатимуться тільки виконані зміни

Операція: Зміна згенерованих сценаріїв

Проект бази даних обмежений її рівнем, а тому не підтримує безпосередньо об'єкти рівня сервера, та і деякі об'єкти схеми даних також не підтримуються проектом безпосередньо. Проте усі ці об'єкти легко можуть бути додані в сценарії, що запускаються до і після розгортання бази даних. Тому слід при створенні проекту внести необхідні зміни в сценарії, додавши туди потрібні об'єкти і вирази SQL.

Зміна сценарію, що запускається перед установкою	У цей сценарій додайте облікові записи сервера, тим самим задавши відповідність між користувачами і обліковими записами
Зміна сценарію, що запускається після установки	У цьому сценарії виконаєте налаштування прав доступу до об'єктів схеми, так щоб вони створювалися відразу з необхідними дозволами: заборона, дозвіл або відгук прав

Операція: Перевірка проекту бази даних

Після виконання усіх необхідних налаштувань проекту слід зібрати і розгорнути його на локальному сервері, щоб переконатися в його готовності. Це дуже важливий крок, що допомагає виявити в сценаріях пропущені або невірно задані конфігураційні параметри.

Зборка проекту бази даних	Виконайте зборку проекту бази даних
Установка проекту бази даних на локальному сервері для перевірки	Встановіть проект бази даних на локальному сервері

Операція: Розміщення проекту бази даних в службі управління початковим кодом

Після перевірки проект бази даних можна розмістити в службі управління початковим кодом, щоб він був доступний усій команді.

Реєстрація проекту в службі управління початковим кодом	Переконайтеся, що служба управління початковим кодом конфігурована правильно
Публікація проекту в службі управління початковим кодом	Помістіть проект в службу управління початковим кодом і присвойте цій дії описувач
Описувач завершення створення проекту	Завершіть завдання по створенню проекту бази даних і призначте воно менеджерові проекту

2. Розгортання проекту бази даних

Кожен проект бази даних врешті-решт має бути розгорнутий в робочому середовищі. У розгортанні проекту бази даних описується, як адміністратор баз даних може перетворити це на керований і передбачуваний процес. Він полягає в отриманні правильної версії проекту бази даних і перевірці його шляхом побудови зборки, установки і виконання набору тестів модулів. Адміністратор баз даних повинен зрозуміти, які зміни треба внести в згенеровані сценарії. І, нарешті, сценарій створення бази даних має бути запущений на робочому сервері з подальшою перевіркою бази даних.

Операції:

Синхронізація проекту бази даних	Вивчіть призначене вам завдання по розгортанню бази даних
Перевірка зборки	Виконайте зборку проекту
Виконання тестування модулів бази даних	Виконайте тестування модулів бази даних
Аналіз змін	У службі управління початковим кодом проглянете історію змін, виконаних після останньої установки
Розробка сценарію створення бази даних	Визначте з'єднання до бази даних, де виконуватиметься установка, а також необхідні властивості створення

Резервне копіювання робочої бази даних	Створіть резервну копію робочої бази даних
Розгортання на тестовому сервері	Виконайте сценарій створення бази даних на тестовому сервері, конфігурованому як робочий сервер
Установка бази даних	Виконайте сценарій створення бази даних на робочому сервері

Операція: Синхронізація проекту бази даних

Перед розгортанням адміністраторові баз даних слід синхронізувати локальну базу даних з необхідною версією.

Вивчення поставленого завдання у рамках робіт по розгортанню	Прогляньте описувач завдання, поставленого перед вами у рамках робіт по розгортанню
Синхронізація проекту бази даних з необхідною версією	Визначте версію, з якою буде синхронізований проект бази даних

Операція: Перевірка зборки

Перед установкою проекту бази даних на робочий сервер його важливо спочатку зібрати і встановити на локальному сервері, щоб перевірити якість зборки.

Зборка проекту бази даних	Виконайте зборку проекту
Установка проекту бази даних на тестовий сервер	Розгорніть проект бази даних на локальному тестовому сервері

Операція: Виконання тестування модулів бази даних

Перед установкою необхідно виконати набір тестів модулів проекту бази даних, щоб переконатися у відсутності внесених в нього дефектів.

Виконання тестування модулів бази даних	Переконайтеся в правильній конфігурації виконуваних тестів модулів
Аналіз результатів тестування	Якщо тести завершилися невдало з причини наявних в них помилок, створіть опис дефекту і передайте його власникові тесту модуля для вивчення і виправлення

Операція: Аналіз змін

Адміністраторові баз даних важливо точно розуміти, які зміни передбачається передавати на сервер баз даних. Визначити їх він може або переглядом на сервісі управління початковим кодом історії змін з моменту попередньої установки, або порівнянням схем даних між робочою і розгорнутою базою даних.

Перегляд історії змін з моменту попередньої установки за допомогою служби управління початковим кодом	Скористайтеся історією служби управління початковим кодом для перегляду усіх змін, виконаних після останньої установки проекту бази даних
Порівняння схеми даних для аналізу відмінностей між робочим сервером і проектом	Створіть сеанс порівняння схем баз даних для виявлення відмінностей між базою даних на робочому сервері і проектом
Перегляд файлу журналу змін	Прогляньте файл журналу змін проекту бази даних для розуміння змін, вироблених в ній з моменту попередньої установки

Операція: Розробка сценарію створення бази даних

Сценарій створення бази даних має бути згенерований і при необхідності змінений. В якості місця установки в проекті вказується робочий сервер. Потім генерується сценарій. Для обліку специфічних особливостей оточення в отриманий сценарій можуть бути внесені зміни. Ці зміни можуть включати команди перенесення даних.

Завдання з'єднання до сервера, на якому виконуватиметься створення, а також завдання додаткових параметрів	Конфігуруйте параметри проекту бази даних, задавши з'єднання до сервера, де створюватиметься база даних, а також пов'язані з цим властивості
Генерація сценарію створення бази даних	Згенеруйте сценарій створення бази даних
Зміна сценарію створення бази даних	Прогляньте отриманий сценарій створення бази даних і переконайтеся, що він виконує очікувані дії, необхідні для синхронізації початкового проекту з робочою базою даних

Операція: Створення резервної копії робочої бази даних

Перед розгортанням бази даних необхідно зробити резервну копію існуючої бази даних. Це надзвичайно важливо, оскільки дає можливість повернути базу даних в початковий стан у разі яких-небудь помилок.

Створення резервної копії робочої бази даних	Безпосередньо перед розгортанням створіть повну резервну копію робочої бази даних, використовуючи стандартні технології резервування, вживані у вашій організації

Операція: Установка бази даних на тестовому сервері

Перед установкою на робочий сервер можна виконати необов'язковий крок — протестувати розгортання на тестовому сервері, що має аналогічну

конфігурацію. При цьому виконуються усі дії, які виконуватимуться при реальному розгортанні, — запуск сценаріїв і перевірка їх результатів.

Виконання на тестовому сервері сценаріїв розгортання	Перед установкою на робочий сервер запустите сценарії розгортання в редакторові T - SQL на тестовому сервері, щоб переконатися в їх успішності
Порівняння схем даних для перевірки розгортання	Створіть сеанс порівняння схем даних для перегляду відмінностей між базою даних проекту і базою даних на тестовому сервері

Операція: Установка бази даних

Тепер усе готово для установки бази даних на робочому сервері. Установка виконується запуском на сервері сценарію створення бази даних з подальшою перевіркою результатів.

Запуск сценарію створення бази даних на робочому сервері	Використовуючи редактор T - SQL, виконаєте сценарій створення бази даних на робочому сервері
Порівняння схем даних для перевірки розгортання	Створіть сеанс порівняння схем даних для перегляду відмінностей між базою даних проекту і робочою базою даних
Закриття описувача установки	Відмітьте завдання установки як виконану

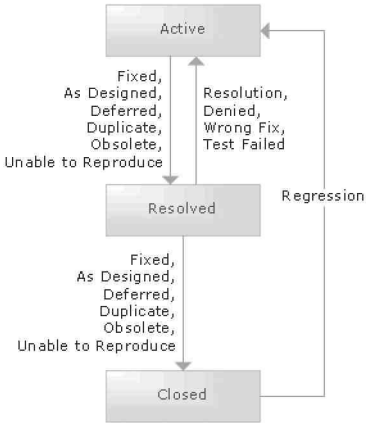
Лекція 11. Описувачі в MSF.

1. Описувачі дефектів.
2. Описувачі-сценарії.
3. Ризики.
4. Завдання.

1. Описувачі дефектів

В описувачі дефекту зберігаються дані про ту, що мала місце або існуючу проблему з системою. Призначення описувачів дефектів — надання користувачам відомостей, необхідних для повного розуміння впливу проблеми на систему. Інформація, що міститься в звіті про помилку, повинна полегшувати її відтворення. Проблеми повинні чітко визначатися з результатів тестів. Чіткість і дохідливість опису дефекту, як правило, підвищують вірогідність його усунення.

Стани і переходи



Стан: Новий: Щоб розробники могли усунути дефект, він має бути зареєстрований відразу після виявлення. Перш ніж зареєструвати дефект, треба перевірити існуючі дані про помилки і переконатися, що такий дефект ще не зареєстрований.

■ **Перехід: Від нового до активного:** Новий описувач дефекту створюється у вузлі Work Items (Описувачі) Провідника команди.

□ **Основа: Новий:** Дефект вважається новим, коли створюється уперше. Вказуйте основу для створення усіх описувачів дефектів як «New» (Новий), якщо вони не є збоями при зборці продукту.

□ **Основа: Збій при зборці:** Основою для створення описувача дефекту вважається збій при зборці, якщо виявлена проблема є прямим наслідком безуспішної зборки продукту.

Стан: Активний: Коли ви виявляєте дефект і вносите дані про нього за допомогою Team Explorer, описувач дефекту автоматично встановлюється в активний стан. Активний стан вказує на те, що проблема існує і її потрібно вирішувати.

■ **Перехід: Від активного до вирішеного:** Щоб перевести проблему стояння вирішеної, її потрібно призначити або особі, що створила описувач дефекту, або тестувальникові щоб вони змогли перевірити, чи усунений дефект.

□ **Основа: Виправлений:** Дефект вважається виправленим («Fixed») після внесення змін до відповідного коду і його реєстрацій в системі управління

версіями. Зв'яжіть дефект з набором змін (changeset) після реєстрації виправленого коду.

□ **Основа: Так задумано:** Основа вирішення проблеми «As Designed» (Так задумано) вказується, якщо уявний дефект відповідає очікуваному стану або поведінці системи.

□ **Основа: Відкладений:** Основа «Deferred» (Відкладений) встановлюється для дефекту, який не виправлятиметься в цій ітерації. Його виправлення відкладається до наступних ітерацій або версій.

□ **Основа: Повтор:** Основа вирішення проблеми «Duplicate» (Повтор) встановлюється для дефектів, які вже мали місце. Додайте посилання на дефект, що повторюється, щоб авторові запису про дефект було простіше підтвердити повторення помилки, перш ніж закрити запис.

□ **Основа: Неактуальний:** Основа «Obsolete» (Неактуальний) встановлюється для дефектів, які вже непридатні до продукту. Наприклад, якщо йдеться про проблему, пов'язану з функцією, від якої відмовилися.

□ **Основа: Неможливо відтворити:** Основа «Unable to Reproduce» (Неможливо відтворити) встановлюється для помилок, які розробник не може наново відтворити на своєму комп'ютері.

Стан: Вирішений: Дефект знаходиться в стані вирішеного, коли він відпрацьований розробником або під час класифікації. Основою для переходу в цей стан може бути «Fixed» (Виправлений) або «As Designed» (Так задумано).

■ **Перехід: Від вирішеного до закритого:** Дефект вважається закритим, коли його рішення перевірене ініціатором створення описувача дефекту або тестувальником.

□ **Основа: Виправлений:** Дефект закривається з основою «Fixed» (Виправлений), коли автор описувача дефекту переконується, що зборка продукту містить виправлення.

□ **Основа: Так задумано:** Дефект закривається з основою «As Designed» (Так задумано), коли автор описувача дефекту погоджується з тим, що дефект відповідає запланованій поведінці системи.

□ **Основа: Відкладений:** Дефект закривається з основою «Deferred» (Відкладений), якщо автор описувача дефекту згоден, що виправлення дефекту треба відкласти.

□ **Основа: Повтор:** Дефект закривається з основою «Duplicate» (Повтор), якщо автор описувача дефекту підтверджує, що цей описувач відповідає вже зафіксованій проблемі.

□ **Основа: Неактуальний:** Дефект закривається з основою «Obsolete» (Неактуальний), якщо автор описувача дефекту згоден, що описана проблема більше не може мати відношення до продукту.

□ **Основа: Неможливо відтворити:** Дефект закривається з основою «Unable to Reproduce» (Неможливо відтворити), якщо автор описувача дефекту не може відтворити ситуацію виникнення зафіксованої проблеми або дати детальніші інструкції для її повторної ініціації.

■ **Перехід: Від вирішеного до активного:** Якщо рішення не може бути провірено, стан описувача дефекту повертається до активного («Active»). Наприклад, якщо при роботі виправленого коду знову повторюється та ж

проблема, тестувальник може повернути дефект в стан активного. При поверненні дефекту в активний стан не забудьте перепризначувати дефект відповідному фахівцеві для класифікації або виправлення.

□ **Основа: Рішення заборонене:** Дефект повертається в стан активного, якщо вирішення проблеми неприпустимо. Щоб спростити роботу іншим людям, які займатимуться цією проблемою, надайте конкретну інформацію про причини заборони.

□ **Основа: невірне виправлення:** Дефект повертається в стан активного, якщо виправлення було некоректним. Детально опишіть, як і чому виправлена версія працювала невірно.

□ **Основа: Тест не проходить:** Дефект повертається в стан активного, якщо тести показують, що помилка не виправлена. Детально опишіть, який тест і в якій зборці (build) не пройшов.

Стан: Закритий: Дефект в змозі «закритий» не вимагає подальших дій в поточній версії продукту. Дефект закривається після перевірки його рішення.

■ **Перехід: Від закритого до активного:** Закритий дефект може бути переведений в стан активного, якщо регресійне тестування показує повторне виникнення проблеми.

□ **Основа: Рецидив:** Якщо при регресійному тестуванні дефект виявляється знову, переведіть його в активний стан і класифікуйте. У полі «Reason» (Основа) встановите значення «Regression» (Рецидив).

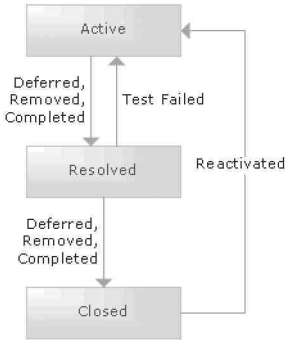
Поля	
Назва	Обов'язкове. У полі «Title» (Назва) коротко описується вирішувана проблема. Назва має бути досить інформативною, щоб фахівці, що здійснюють класифікацію, могли зрозуміти, на яку частину системи впливає проблема і яким чином
Область	Полі «Area» (Область) застосовується для угрупування дефектів по функціях або проектних групах в ієрархії проекту. Область має бути допустимим вузлом в ієрархії проекту
Ітерація	У полі «Iteration» (Ітерація) вказується ітерація, в якій дефект виправлений
Кому призначений	У цьому полі («Assigned To») вказується фахівець, який в даний момент відповідає за обробку дефекту. Якщо для усунення дефекту потрібно декілька різних виправлень, він може розглядатися як сценарій і може бути призначений наступній в ієрархічному ланцюжку особі. Після інтеграції усіх частин виправлення звіт про дефект призначається тестувальникові
Пріоритет	Обов'язкове. Пріоритет («Priority») — це суб'єктивна оцінка важливості. Пріоритет 1 вказує, що продукт не готовий до постачання і має бути виправлений якнайскоріше. Пріоритет 2 вказує на важливий дефект, який немає необхідності виправляти негайно, але необхідно усунути до постачання продукту. Пріоритет 3 вказує на дефект, який можна виправляти або ні, залежно від ресурсів, часу і ризиків

Стан	Обов'язкове. У полі «State» (Стан) вказує один з можливих станів дефекту : «Active» (Активний), «Resolved» (Вирішений) або «Closed» (Закритий)
Основа	Обов'язкове. У полі «Reason» (Основа) вказується, на якій підставі дефект переведений в його поточний стан. Наприклад, «Fixed» (Виправлений) — одна з можливих підстав перекладу дефекту в стан вирішеного
Опис	Полі «Description» (Опис) служить для опису проблеми і кроків для її відтворення
Архів	У міру обробки помилки в полі «History» (Архів) накопичуються записи. При кожній зміні, пов'язаній з дефектом, в це поле додається запис з відомостями про те, які зроблені зміни, чому, а також іншими подробицями
Перешкода	У полі «Issue» значення «Yes» (Так) і «No» (Ні) вказують на наявність або відсутність проблем, що якимсь чином перешкоджають виправленню дефекту. Якщо в полі вказане «Yes» (Так), в звіті про проблеми менеджера проекту міститимуться відомості про відповідний дефект
Знайдено у зборці	У цьому полі («Found in Build») вказується номер зборки, в якій був виявлений дефект
Вирішено у зборці	У цьому полі («Resolved in Build») зберігається номер зборки, в якій проблема була розв'язана
Назва тесту	У цьому полі («Test Name») вказується назва тесту, пов'язаного з цим дефектом
Ідентифікатор тесту	У цьому полі («Test ID») вказується ідентифікатор тесту, пов'язаного з цим дефектом
Розташування тесту	У цьому полі («Test Path») вказується шлях, по якому розташований тест, пов'язаний з цим дефектом
Посилання	Посилання («Links») на пов'язані з цим описувачем інші описувачі, гіперпосилання, набори змін або файли з початковим кодом
Вкладення	У цьому полі («File Attachments») вказуються файли, що містять додаткові відомості про дефект
Ранг	Відносний пріоритет з урахуванням інших описувачів («Rank»)
Класифікація	Містить результати наради по класифікації («Triage»). Якщо поле не заповнене, означає дефект не класифікований

Вимоги до якості

У вимогах до якості фіксуються такі характеристики системи, як продуктивність, завантаження, доступність, спеціальні можливості, зручність обслуговування і супроводу. Ці вимоги зазвичай мають форму обмежень на функціонування системи.

Стани і переходи



Стан: Нове: Вимоги до якості додаються в список, який знаходиться в теці вимог бібліотеки документів, або за допомогою описувача в Team Explorer (Провідник команди).

■ **Перехід: Від нового до активного**

□ **Основа: Нове:** Вимога до якості активується як нове при першому створенні.

Стан: Активне: Робота з вимогою до якості починається в активному стані. Бізнес-аналітик фіксує вимогу, вказуючи для нього інформативну назву, і заповнює поле опису як можна детальнішими відомостями про вимогу. Після того, як вимога повністю сформульована, бізнес-аналітик призначає його провідному розробникові. У полі «Specified» (Сформульовано) встановлюється значення «Yes» (Так) і вимога залишається в активному стані на час його реалізації. Провідний розробник координує з іншими розробниками дії, необхідні для реалізації вимог.

■ **Перехід: Від активного до обробленого**

□ **Основа: Завершено:** Вимога до якості переводиться в стан обробленого на підставі «Completed» (Завершено), якщо команда розробників закінчила написання коду, що реалізовує цю вимогу. Провідний розробник призначає вимогу тестувальникові.

□ **Основа: Відкладено:** Вимога до якості вважається обробленою з основою «Deferred» (Відкладено), якщо в цій ітерації реалізувати його неможливо. Реалізація вимоги може бути відкладена із-за нестачі часу у розробників або виявлення проблем, блокуючих роботу. У полі «Iteration» (Ітерація) треба вказати правильний номер ітерації, в якій вимога буде реалізована. Якщо здійснення вимоги відкладається до наступної версії продукту, поле «Iteration» треба залишити порожнім. Не забудьте детально описати причини, по яких відкладена реалізація вимоги, і вкажіть плановані терміни роботи над ним.

□ **Основа: Видалено:** Вимога до якості віддаляється («Removed»), якщо більше не вважається доцільним. При видаленні вимоги перевірте поля «Issue» (Перешкода) і «Exit Criteria» (Умови завер шения). Зазвичай для видалених вимог в цих полях міститься «No» (Ні).

Стан: Оброблене: Після реалізації вимоги до якості провідний розробник встановлює його стан в «Resolved» (Оброблене). Провідний розробник також призначає цю вимогу тестувальникові після чого може початися перевірка вимоги.

■ **Перехід: Від обробленого до закритого**

□ **Основа: Завершено:** Вимога до якості закривається як «Completed» (Завершено), коли тестувальник повідомляє про проходження тестів. При завершенні роботи з вимогою перевірте поля «Issue» (Перешкода) і «Exit Criteria» (Умови завершення). Зазвичай для реалізованих вимог в цих полях міститься «No» (Ні).

□ **Основа: Відкладено:** Вимога до якості вважається закритою з основою «Deferred» (Відкладено), якщо в цій ітерації реалізувати його неможливо.

□ **Основа: Видалено:** Вимога до якості закривається як видалене («Removed»), якщо воно більше не вважається доцільним.

■ **Перехід: Від обробленого до активного**

□ **Основа: Тест не проходить:** Вимога до якості повертається в активний стан, якщо не проходить один або декілька тестів. Шалівка тировщик повинна знову призначити цю вимогу ведучому разра ботчику, який його зафіксував. Тестувальник також повинен створити відповідний описувач дефекту для збою тесту.

Стан: Закрите: Тестувальник переводить роботу з вимогою в стан «Closed» (Закрито) при проходженні усіх тестів. Вимога також закривається, якщо воно відкладається, віддаляється або розбивається на дрібніші.

■ **Перехід: Від закритого до активного**

□ **Основа: Повторна активація:** Відкладена вимога до каче ству активується наново, коли починається ітерація, на яку на значена його реалізація. Якщо вимога все ще потрібна задокументи ровать, призначте його бізнес-аналітику Якщо ж вимога готова до реалізації, призначте його провідному розробникові. При повторній активації видалених вимог, наслідуйте ту ж процедуру, що і для відкладених вимог.

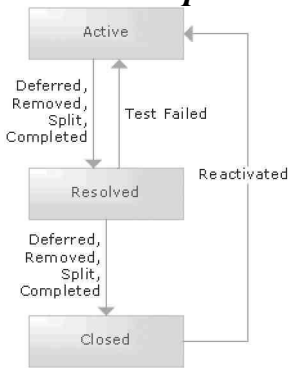
Назва	Обов'язкове. Назва має бути максимально інформативною
Область	Полі «Area» (Область) застосовується для угрупування вимог до якості по функціях або проектних групах. Область має бути допустимим вузлом в ієрархії проекту
Ітерація	У полі «Iteration» (Ітерація) вказується ітерація, в якій вимога до якості реалізована в коді
Тип	Є п'ять типів вимог до якості: «Load» (Навантаження), «Stress» (Стрес), «Performance» (Продуктивність), «Platform» (Платформа), «Other» (Інше) і «Security» (Безпека)
Кому призначено	У цьому полі («Assigned To») вказується фахівець, який в даний момент відповідає за реалізацію вимоги
Стан	Обов'язкове. У полі «State» (Стан) вказується один з можливих станів вимоги : «Active» (Активне), «Resolved» (Оброблене) або «Closed» (Закрите)

Основа	У полі «Reason» (Основа) вказується, на якій підставі вимога до якості переведена в його поточний стан. Наприклад, «Completed» (Завершено) — одна з можливих підстав перекладу вимоги в стан закритого
Опис	Полі «Description» (Опис) служить для опису вимоги до якості. Опишіть вимогу як можна детальніше, щоб розробникові було простіше його реалізувати, а тестувальникові — перевірити
Архів	У міру обробки помилки в полі «History» (Архів) накопичуються записи. При кожній зміні вимоги в це поле додається запис з відомостями про те, які зроблені зміни, чому, а також інші подробиці
Перешкода	У полі «Issue» значення «Yes» (Так) або «No» (Ні) вказують на наявність або відсутність проблем, що якимсь чином перешкоджають реалізації вимоги. Якщо в полі вказане «Yes» (Так) в звіті про проблеми, який отримує менеджер проекту, будуть відомості про цю вимогу
Умови завершення	У полі «Exit Criteria» значення «Yes» (Так) або «No» (Ні) вказують, чи входить реалізація цієї вимоги в список обов'язкових робіт (backlog) для поточної ітерації. Це поле застосовується для синхронізації різних представлень списку обов'язкових робіт (список сценаріїв, набір вимог до якості і план ітерації). Якщо умови завершення встановлені в «Yes» (Так), ця вимога до якості відображається в контрольному списку проекту і одному з кінцевих продуктів. Це поле в наступній версії буде перейменовано в «Iteration Backlog» (Список обов'язкових робіт в ітерації)
Ранг	Значення в цьому полі («Rank») вказує важливість цієї вимоги до якості відносно усіх інших вимог до програмного продукту
Зборка з реалізацією	У полі «Integration Build» зберігається номер зборки (build), в якій міститься реалізація цієї вимоги
Ідентифікатор	У полі «ID» зберігається унікальний ідентифікаційний номер вимоги до якості
Приблизний порядок величини	Приблизний порядок величини («Rough order of magnitude») — це спосіб оцінки трудовитрат на реалізацію сценарію або вимоги до якості. Якщо вимагається виконати не більше шести завдань, що вимагають 1-2 дні, в цьому полі вказується значення 1. Якщо вимагається виконати від 6 до 12 завдань, що вимагають 1-2 дні, в цьому полі вказується значення 2. Якщо трудовитрати більші, в цьому полі вказується значення 3 і розглядається можливість розбиття на частини завдання реалізації сценарію або вимоги до якості

2. Описувачі-сценарії

Сценарій (scenario) — це різновид описувача, в якому записується один з можливих варіантів взаємодії користувача з системою. Сценарій містить запис послідовності дій користувача, які він виконує для досягнення деякої мети. У одних сценаріях фіксуються успішні спроби, в інших — безуспішні. При записі сценарію треба вказувати, який конкретний шлях досягнення мети описується, оскільки таких шляхів множина.

Стани і переходи



Стан: Новий: Сценарії додаються в список, який знаходиться в теці сценаріїв бібліотеки документів.

■ **Перехід: Від нового до активного**

□ **Основа: Новий:** Сценарій вважається новим, коли створюється уперше.

Стан: Активний: Початковий стан сценарію — активний. Бізнес-аналітик створює сценарій, вказуючи для нього інформативну назву, і заповнює поле опису як можна детальнішими відомостями про нього. Після того, як сценарій повністю описаний, бізнес-аналітик призначає його провідному розробникові. У полі «Specified» (Описаний) встановлюється значення «Yes» (Так) і сценарій залишається в активному стані на час його реалізації. Провідний розробник координує з іншими розробниками дії, необхідні для реалізації цього сценарію.

■ **Перехід: Від активного до обробленого**

□ **Основа: Завершений:** Сценарій переводиться в стан обробленого на підставі «Completed» (Завершений), якщо команда розробників закінчила написання коду, що реалізовує цей сценарій. Провідний розробник призначає сценарій тестувальникові

□ **Основа: Розділений:** Сценарій переводиться в стан обробленого на підставі «Split» (Розділений), якщо з'ясувалося, що він занадто об'ємний або що потрібно опис додаткових деталей. При розбитті сценарію створіть нові сценарії і зв'яжіть їх з початковим сценарієм.

□ **Основа: Відкладений:** Основа «Deferred» (Відкладений) встановлюється для сценарію, який не буде реалізований в цій ітерації. Реалізація сценарію може бути відкладена із-за нестачі часу у розробників або виявлення проблем, блокуючих роботу. У полі «Iteration» (Ітерація) треба вказати правильний номер ітерації, в якій сценарій буде реалізований. Якщо реалізація сценарію відкладається до наступної версії продукту, поле «Iteration» (Ітерація) треба залишити порожнім. Не забудьте надати детальний опис причин, по яких відкладена реалізація сценарію, і вкажіть плановані терміни роботи над ним.

□ **Основа: Видалений:** Сценарій віддаляється («Removed»), якщо його реалізація більше не вважається доцільною. При видаленні сценарію перевірте поля «Issue» (Перешкода) і «Exit Criteria» (Умови завершення). Зазвичай для видалених сценаріїв в цих полях міститься «No» (Ні).

Стан: Оброблений: Після реалізації сценарію провідний розробник встановлює його стан в «Resolved» (Оброблений). Провідний розробник також призначає цей сценарій тестувальникові після чого можна починати його перевірку.

■ **Перехід: Від обробленого до закритого**

□ **Основа: Завершений:** Сценарій закривається як «Completed» (Завершений), коли тестувальник повідомляє про проходження тестів. При завершенні роботи з сценарієм перевірте поля «Issue» (Перешкода) і «Exit Criteria» (Умови завершення). Зазвичай для завершених сценаріїв в цих полях міститься «No» (Ні).

□ **Основа: Розділений:** Сценарій закривається на підставі «Split» (Розділений), якщо з'ясувалося, що він занадто об'ємний або потрібно опис додаткових деталей.

□ **Основа: Відкладений:** Основа «Deferred» (Відкладений) встановлюється для сценарію, який не буде реалізований в цій ітерації.

□ **Основа: Видалений:** Сценарій віддаляється («Removed»), якщо його реалізація більше не вважається доцільною.

■ **Перехід: Від обробленого до активного**

□ **Основа: Тест не проходить:** Якщо які-небудь тести сценарію не проходять, тестувальник повинен повернути сценарій в активне положення і знову призначити його провідному розробникові, що створив цей сценарій. Тестувальник також повинен створити той, що відповідає опису дефекту для збою тесту.

Стан: Закритий: Тестувальник закриває сценарій при проходженні усіх тестів. Сценарій також закривається, якщо він відкладається, віддаляється або розбивається на дрібніші.

■ **Перехід: Від закритого до активного**

□ **Основа: Повторна активація:** Сценарій може бути наново переведений в активний стан при зміні функціональності.

Назва **Область**	Обов'язкове. Назва повинна відбивати мету виконання сценарію. Назва має бути максимально інформативною Полі «Area» (Область) застосовується для угрупування сценаріїв по функціях або проектних групах. Область має бути допустимим вузлом в ієрархії проекту
Ітерація	Ітерація, в якій реалізований сценарій
Кому	Відповідальний за роботу з сценарієм
Стан	Обов'язкове. У полі «State» (Стан) вказується один з можливих станів сценарію : «Active» (Активний), «Resolved» (Оброблений) або «Closed» (Закритий)

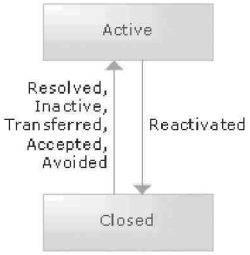
Основа	У полі «Reason» (Основа) вказується, на якій підставі сценарій переведений в його поточний стан. Наприклад, «Completed» (Завершений) — одна з можливих підстав перекладу сценарію в стан закритого
Опис	Опис сценарію на досить загальному рівні. Детальний опис сценарію має бути в одному з кінцевих продуктів
Архів	У міру обробки сценарію в полі «History» (Архів) накопичуються записи. При кожній зміні, пов'язаній з сценарієм, в це поле додається запис з відомостями про те, які зроблені зміни, чому, а також інші подробиці
Перешкода	У полі «Issue» значення «Yes» (Так) або «No» (Ні) вказують на наявність або відсутність проблем, що якимсь чином перешкоджають реалізації сценарію. Якщо в полі вказане «Yes» (Так), в звіті про проблеми менеджера проекту буде присутнім відповідний сценарій
Умови завершення	У полі «Exit Criteria» значення «Yes» (Так) або «No» (Ні) вказують, чи входить цей сценарій в список обов'язкових робіт (backlog) для поточної ітерації. Це поле застосовується для синхронізації різних представлень списку обов'язкових робіт (список сценаріїв, набір вимог до якості і план ітерації). Якщо умови завершення встановлені в «Yes» (Так), цей сценарій відображається в контрольному списку проекту і одному з кінцевих продуктів. Це поле в наступній версії буде перейменовано в «Iteration Backlog» (Список обов'язкових робіт в ітерації)
Ранг	Значення в цьому полі («Rank») вказує важливість цього сценарію відносно усіх інших сценаріїв
Зборка з реалізацією	У полі «Integration Build» зберігається номер зборки («Build»), в якій міститься реалізація цього сценарію
Ідентифікатор	У полі «ID» зберігається унікальний ідентифікаційний номер сценарію
Приблизний порядок величини	Якщо вимагається виконати від 6 до 12 завдань, що вимагають 1-2 дні, в цьому полі вказується значення 2. Приблизний порядок величини («Rough order of magnitude») — це спосіб оцінки трудовитрат на реалізацію сценарію або вимоги до якості. Якщо трудовитрати більші, в цьому полі вказується значення 3 і розглядається можливість розбиття на частини завдання реалізації сценарію або вимоги до якості. Якщо вимагається

3. Ризик

Важливою складовою управління проектом є ідентифікація ризиків і контроль над ними. Ризик — це вірогідна подія або чинник, який може зробити негативний вплив на проект в майбутньому. Описувачі ризиків дозволяють документувати і відстежувати технічні і організаційні ризики проекту. Рискам можна зіставити завдання, які треба виконати для їх зменшення. Наприклад,

створення деякого архітектурного прототипу може понизити технічний ризик. Учасники проекту повинні відноситися до виявлення ризиків як до позитивного явища, оскільки це дає максимум даних про можливі проблеми. Обстановка в колективі має бути такою, щоб особи, що ідентифікують ризики, не боялися висловлювати свою — новаторську або спірну — точку зору. Проектні команди з позитивним відношенням до ризиків мають більше шансів своєчасно виявити і врахувати ризики, чим команди з негативним до них відношенням.

Стани і переходи



Стан: Новий: Описувач нового ризику створюється у будь-який момент, коли визначається ризик. Цей новий описувач повинен містити назву і опис, які чітко визначають потенційні наслідки ризику. Також має бути вказана можлива дія ризику. Новий описувач ризику може бути створений будь-яким учасником команди і має бути призначений фахівцеві, який відстежуватиме цей ризик до його закриття або перепризначення.

■ **Перехід: Від нового до активного:** Описувач нового ризику створюється у будь-який момент, коли визначається ризик. Для його створення використовується меню в Team Explorer.

□ **Основа: Новий:** Описувач нового ризику створюється у будь-який момент, коли потенційний ризик може вплинути на проект. Для його створення використовується меню в Team Explorer.

Стан: Активний: Коли за допомогою Team Explorer створюється новий описувач ризику, його стан автоматично встановлюється в «Active» (Активний). Активний стан ризику вказує, що може статися деяка подія, здатна вплинути на проект. Кожен ризик має бути призначений деякому власникові.

■ **Перехід: Від активного до закритого**

□ **Основа: Понижений:** Ризик може бути закритий з вказівкою основи «Mitigated» (Понижений), якщо були зроблені деякі дії для запобігання виникненню ризикованої події і/або зменшення дії ризику до прийнятного рівня.

□ **Основа: Неактивний:** Ризик може бути закритий на підставі «Inactive» (Неактивний), коли вірогідність його виникнення можна ігнорувати. Ніяких дій робити не вимагається.

□ **Основа: Переведений:** Ризик може бути закритий на підставі «Transferred», якщо його можна винести за рамки проекту. При цьому сам ризик не зникає. Він лише не впливає на проект в його поточному стані. Приклади перекладу ризиків : переміщення ризику на наступну версію, залучення сторонніх консультантів або купівля готового програмного компонента замість його розробки.

□ **Основа: Прийнятий:** Деякі події неможливо запобігти або понизити їх дію. У таких випадках члени команди приймають ризик, усвідомлюючи його

ефект. При цьому вказується основа закриття «Accepted» (Прийнятий).

□ **Основа: Анулюваний:** Ризику може більше не бути із-за змін в проекті або змін в самому ризику. Відстежувати його в проекті більше немає підстав і він закривається як «Avoided» (Анулюваний).

Стан: Закритий: Закриваються ризики, що більше не представляють загрози для проекту. При ретроспективному аналізі ітерації, в якій був закритий ризик, ризик може детально обговорюватися.

■ **Перехід: Від закритого до активного**

□ **Основа: Повторна активація:** Ризик може виникнути знову і при цьому повторно активується. Його стан при цьому міняється з «Closed» (Закритий) на «Active» (Активний).

Назва	Обов'язкове. У полі «Title» (Назва) коротко описується потенційний ризик. Назва має бути досить інформативною, щоб учасникам команди було зрозуміло, з чим пов'язаний ризик
Область	Полі «Area» (Область) застосовується для угрупування ризиків по функціях або проектних групах. Область має бути допустимим вузлом в ієрархії проекту
Кому призначено	Відповідальний за відстежування ризику. Звичайно це менеджер проекту або архітектор, але відповідальним може бути і будь-який інший член проектної групи
Серйозність	Серйозність («Severity») містить оцінку впливу несприятливого ефекту, рівень втрат або можливих витрат на усунення ризику («Low» — низька, «Medium» — середня, «High» — висока або «Critical» — критична)
Основа	У полі «Reason» (Основа) вказується, на якій підставі ризик переведений в його поточний стан. Наприклад, «Avoided» (Анулюваний) — одна з можливих підстав перекладу ризику в стан закритого
Стан	Обов'язкове. У полі «State» (Стан) вказується один з можливих станів ризику : «Active» (Активний) або «Closed» (Закритий)
Ітерація	Ітерація, в якій може статися ризикована подія
Пріоритет	Пріоритет описує важливість ризику по відношенню до інших ризиків; зручний для визначення послідовності, в якій слід знижувати ризики
Посилання	Посилання («Links») на пов'язані з цим описувачем інші описувачі, гіперпосилання, набори змін або файли з початковим кодом
Вкладення	У цьому полі («File Attachments») вказуються файли, що містять додаткові відомості про риску

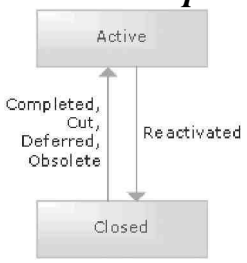
Перешкода. В полі «Issue» значення «Yes» (Так) або «No» (Ні) вказують на наявність або відсутність проблем, що якимсь чином перешкоджають обробці ризику. Якщо в полі вказане «Yes» (Так), в звіті про проблеми менеджера проекту буде присутнім відповідний сценарій

Умови завершення	У полі «Exit Criteria» значення «Yes» (Так) або «No» (Ні) вказують, чи входить обробка цього ризику в список обов'язкових робіт (backlog) для поточної ітерації. Це поле застосовується для синхронізації різних представлень списку обов'язкових робіт (список сценаріїв, набір вимог до якості і план ітерації). Якщо умови завершення встановлені в «Yes» (Так), цей ризик відображається в контрольному списку проекту і одному з кінцевих продуктів. Це поле в наступній версії буде перейменовано в «Iteration Backlog» (Список обов'язкових робіт в ітерації)
Опис	У цьому полі міститься опис ризику, його дія на проект і можливі варіанти його зменшення
Архів	У цьому полі фіксуються усі зміни, що відбуваються з описувачем

4. Завдання

Описувач завдання служить для представлення деякої роботи. Представники кожної ролі використовують описувачі завдань по-своєму. Наприклад, розробникам компонентів з їх допомогою призначаються роботи, необхідні для реалізації сценаріїв і вимог до якості. Тестувальники використовують описувачі завдань для розподілу робіт по написанню і прогону тестових завдань. Описувачі завдань також можуть застосовуватися для попередження про повернення в попередній стан або для пропозиції виконати деякі дослідження. Взагалі, описувач завдання є універсальним засобом розподілу робіт у рамках проекту. Відмітимо, що деякі поля описувачів завдань застосовуються тільки для певних ролей.

Стани і переходи



Стан: Новий: Новий описувач завдання може бути створений у будь-який момент, коли виявляється потреба у виконанні тієї або іншої роботи. Існують три типи завдань. Завдання по розробці пов'язані з виробленнями архітектурних рішень, а також з реалізацією сценаріїв або вимог до якості. Тестові завдання відповідають необхідним тестам. Третій тип відноситься до робіт, що виходять за рамки цих двох областей. Для створення описувача завдання використовується Team Explorer (Провідник команди).

- **Перехід: Від нової до активної**

□ **Основа: Нова:** Завдання вважається новим, якщо створюється уперше.

Стан: Активна: Коли за допомогою Team Explorer створюється новий описувач завдання, його стан автоматично встановлюється в «Active» (Активний). Завдання активне, коли виконується деяка пов'язана з нею робота. Кожному

завданню по розробці або тестуванню має бути призначений власник і дисципліна.

■ **Перехід: Від активного до закритого**

□ **Основа: Завершена:** Завдання закривається як завершена («Completed»), якщо виконані усі складові її роботи.

□ **Основа: Відкладена:** Основа «Deferred» (Відкладена) встановлюється для завдання, яке не буде виконане в цій ітерації. Виконання завдання може бути відкладене із-за нестачі часу у розробників або виявлення проблем, блокуючих роботу. У полі «Iteration» (Ітерація) треба вказати правильний номер ітерації, в якій завдання буде виконано.

□ **Основа: Неактуальна:** Завдання закривається з основою «Obsolete» (Неактуальна), якщо її описувач представляє роботу, яка більше не потрібна для реалізації продукту.

□ **Основа: Вирізана:** Завдання закривається з основою «Cut» (Вирізана), якщо функції, що реалізовуються нею, видалені з продукту.

Стан: Закрита: Завдання в змозі «закрита» не вимагає подальших дій в поточній версії продукту. Завдання по розробці закривається після інтеграції створеного коду в систему. Тестове завдання закривається, коли проходять усі тести для області, до якої відноситься це завдання.

■ **Перехід: Від закритого до активного**

□ **Основа: Повторна активація:** Завдання по розробці або шалівка тированию може бути наново переведене в активний стан при зміні функціональності.

Назва	Обов'язкове. У полі «Title» (Назва) коротко описується вирішувана задача. Назва має бути досить інформативною, щоб фахівці, що здійснюють класифікацію, могли зрозуміти, на яку частину системи впливає вирішувана задача і яким чином
Дисципліна	Вказує тип завдання : розробка, тестування або інше. Завдання по розробці і тестуванню припускають певний стан робіт при закритті завдання
Область	Полі «Area» (Область) застосовується для угрупування завдань по функціях або проектних групах. Область має бути допустимим вузлом в ієрархії проекту
Ітерація	Запланована ітерація, в якій сталося виправлення дефекту
Кому	Відповідальний за виконання завдання
Стан	Обов'язкове. У полі «State» (Стан) вказується один з можливих станів завдання : «Active» (Активна) або «Closed» (Закрита)
Основа	Обов'язкове. У полі «Reason» (Основа) вказується, на якій підставі завдання переведене в її поточний стан. Завдання може бути закрите, тому що вона завершена, відкладена, вирізана або неактуальна

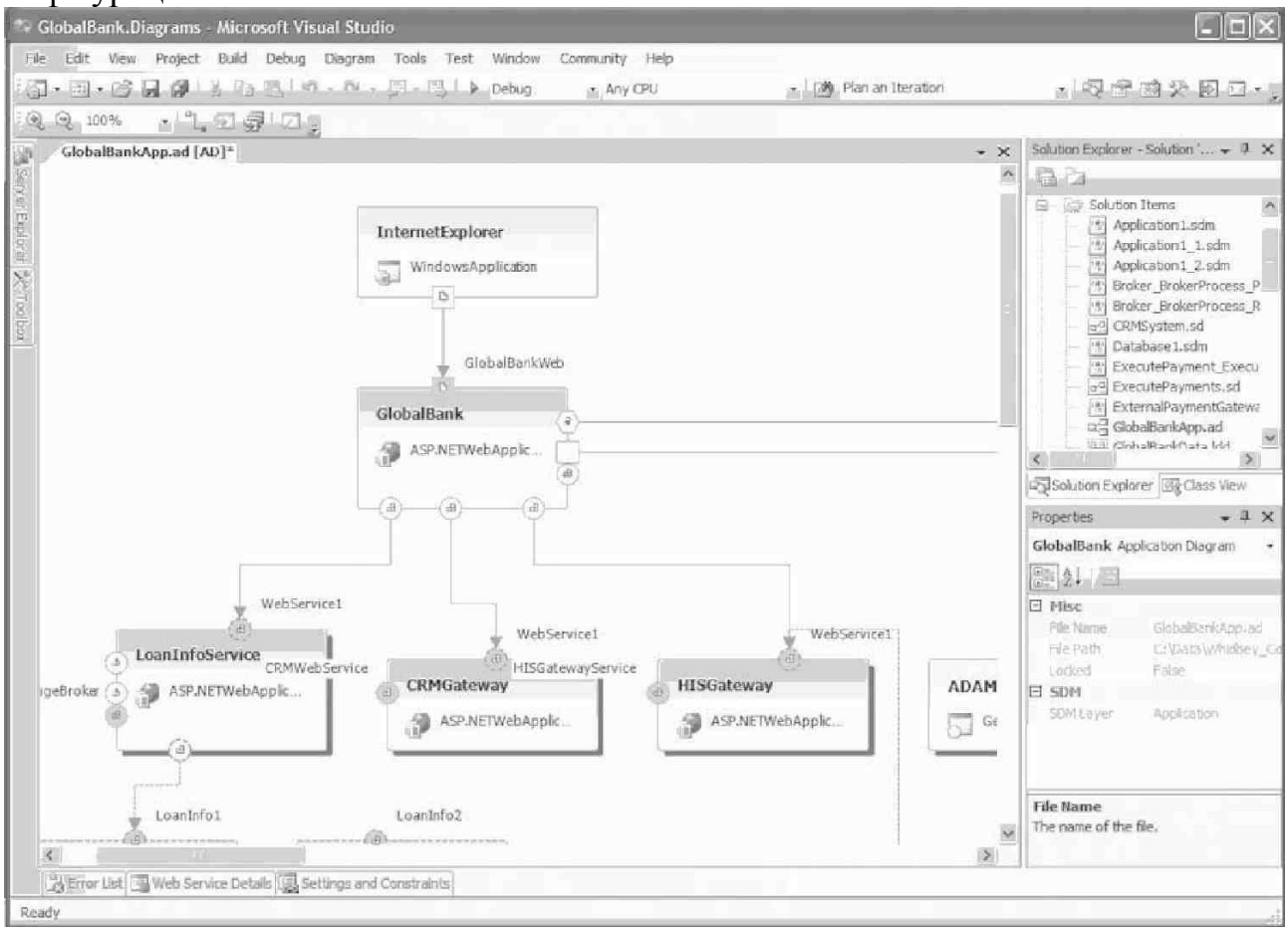
Ранг	Обов'язкове. Ранг (поле «Rank») — це суб'єктивна оцінка важливості. Значення «1» вказує на високу важливість завдання — її необхідно виконати якомога раніше. Значення «2» привласнюється досить важливим завданням, які мають бути виконані після завдань з рангом 1. Ранг 3 мають завдання, що виконуються після завдань з рангом 1 і 2
Короткий	Короткий опис завдання
Детальний опис і архів	Детальний опис завдання і набір усіх попередніх записів про завдання
Вкладення	Посилання на файли або інші описувачі. Для завдань по розробці вказуються посилання на відповідні сценарії або вимоги до якості. Також вказуються файли з пояснюючими текстами і інші допоміжні дані
Перешкода	У полі «Issue» значення «Yes» (Так) або «No» (Ні) вказують на наявність або відсутність проблем, що якимсь чином перешкоджають рішенню задачі. Якщо в полі вказане «Yes» (Так), завдання буде відображено в звіті про проблеми менеджера проекту
Умови завершення	У полі «Exit Criteria» значення «Yes» (Так) або «No» (Ні) вказують, чи являється це завдання однієї з обов'язкових робіт («Backlog») для поточної ітерації. Це поле застосовується для синхронізації різних представлень списку обов'язкових робіт (список сценаріїв, набір вимог до якості і план ітерації). Якщо умови завершення встановлені в «Yes» (Так), це завдання відображається в контрольному списку проекту і одному з кінцевих продуктів. Це поле в наступній версії буде перейменовано в «Iteration Backlog» (Список обов'язкових робіт в ітерації)
Зборка з реалізацією	Номер зборки, що містить реалізовані функції
Робота (у год.), що залишилася	Об'єм робіт, що залишилися до завершення завдання. Це поле синхронізується з Microsoft Project і може використовуватися при застосуванні останнього
Виконана робота (у год.)	Об'єм виконаної для вирішення завдання роботи. Це поле синхронізується з Microsoft Project і може використовуватися при застосуванні останнього

Лекція 12. Результати робіт згідно методології MSF.

1. **Діаграма застосування**
2. **Діаграма класів**
3. **План ітерації**
4. **Логічна діаграма центру обробки даних**
5. **Ручний тест**
6. **Список сценаріїв**
7. **Діаграма системи**
8. **Концепція**

1. Діаграма застосування

На діаграмі застосування відображається програмний комплекс в цілому і містяться такі компоненти, як веб-служби, веб- і Windows-додатки, а також пов'язані ресурси, серед яких зовнішні бази даних, веб-служби і служби Biz, - Talk. На цій діаграмі показані взаємозв'язки між цими застосуваннями і їх поточна конфігурація.

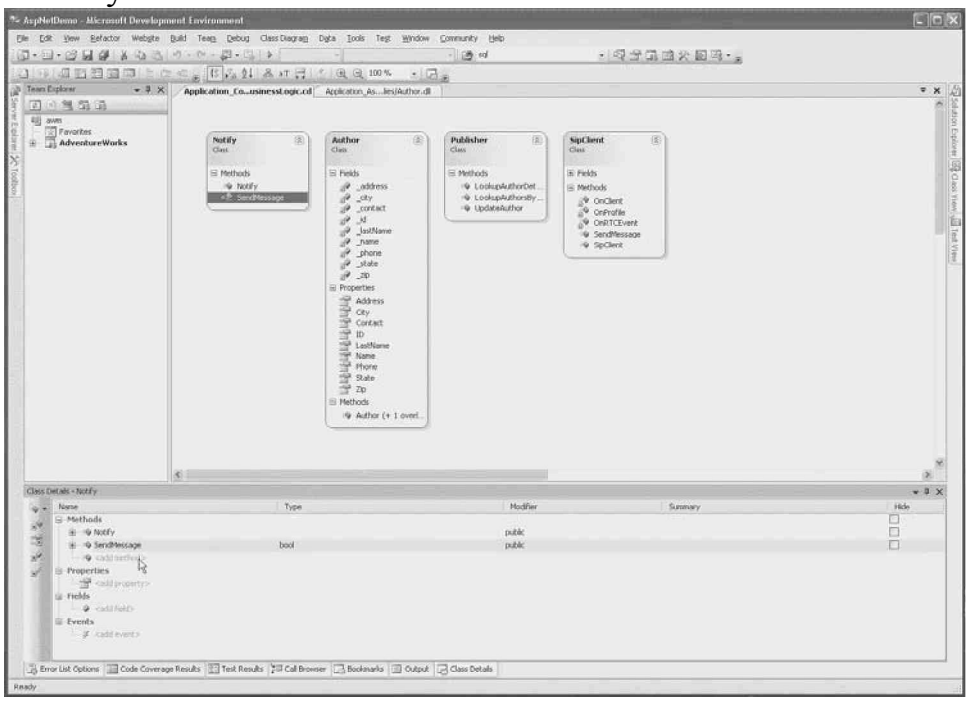


Пакет змін

Пакет змін — це логічне об'єднання зроблених змін. Зміни бувають затриманими, відкладеними і зафіксованими. Затримані зміни — це група змін, виконаних у рамках однієї акції, але ще не переданих у базу даних на публікацію і постійне зберігання. Відкладені зміни — це зміни, які ще не передані у базу даних, але увесь їх вміст, включаючи змінені файли, знаходиться на сервері. Зафіксовані зміни — це набір змін, які збережені в архіві бази даних і доступні усім для перегляду.

2. Діаграма класів

На діаграмі класів відображається логічне або фізичне об'єднання класів і їх взаємозв'язку.

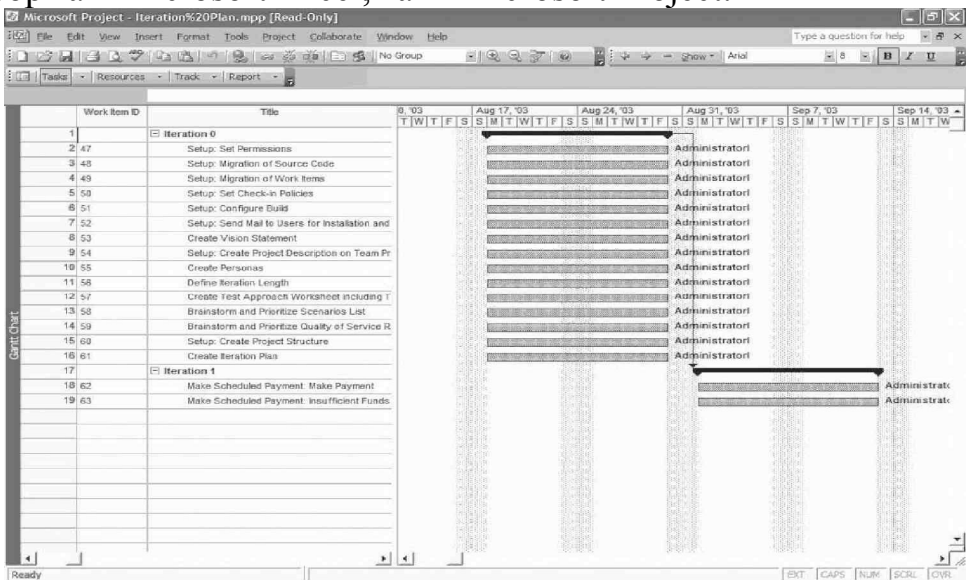


Початковий код

У початковий код входять усі види початкового коду : написані на мовах програмування, сценаріїв, розмітки і запитів.

3. План ітерації

План ітерації — це набір сценаріїв, вимог до якості і завдань, що відноситься до однієї ітерації. Остаточно план затверджується на зборах по плануванню ітерації, безпосередньо перед її початком. Він може бути виконаний як у форматі Microsoft Excel, так і Microsoft Project.

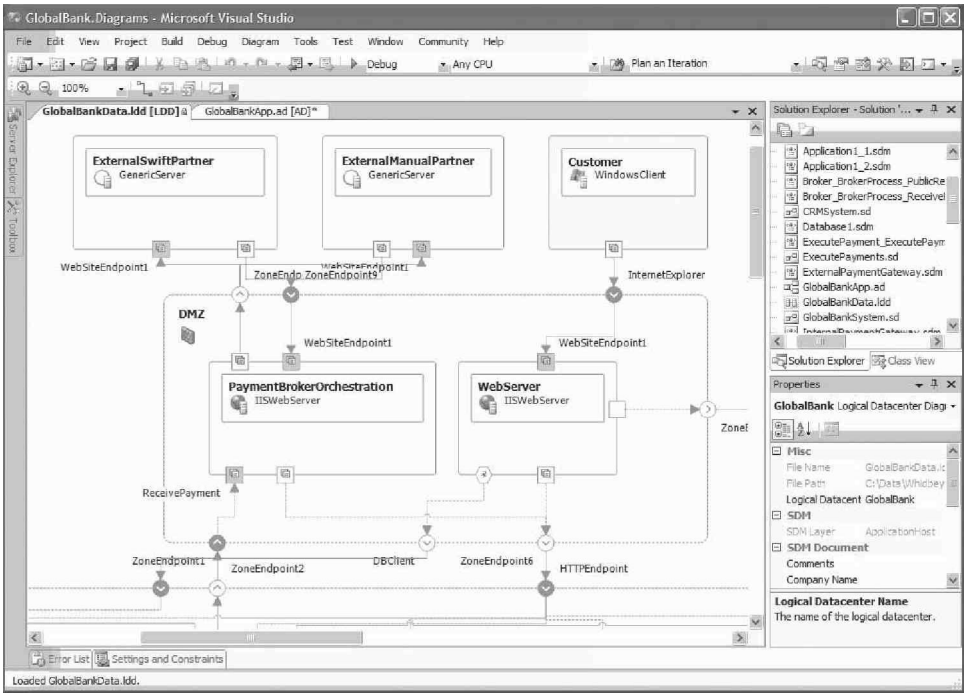


Тест навантаження

Сценарій теста навантаження включає набір тестів, описи навантаження і різних інструментів, що допомагають перевірити застосування в режимі тестування навантаження і стресового.

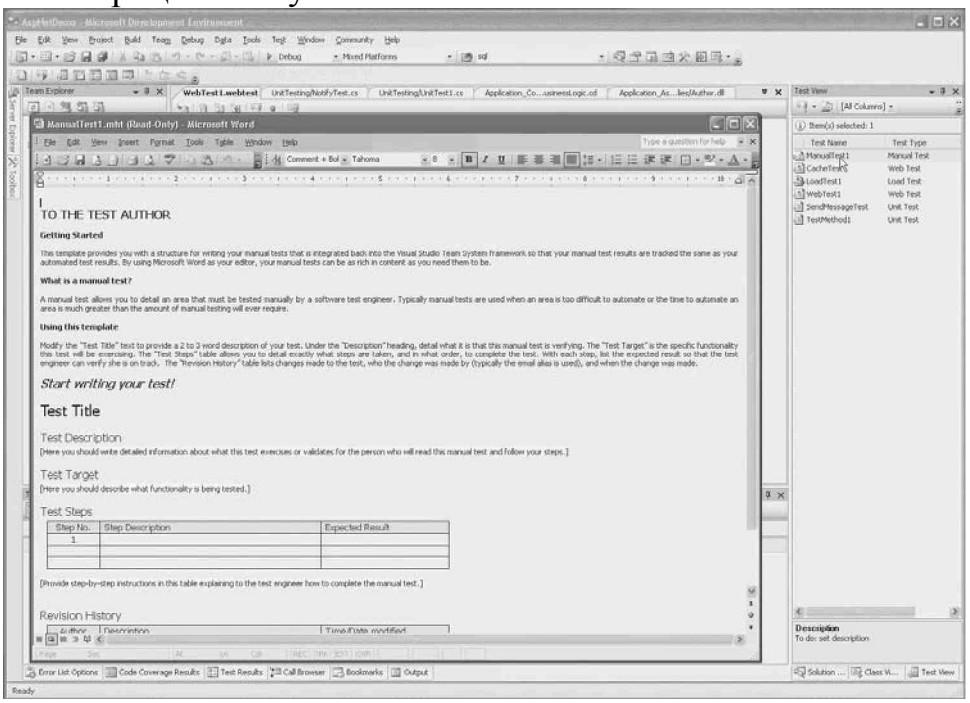
4. Логічна діаграма центру обробки даних

У логічній діаграмі центру обробки даних визначені (задокументовані) конкретні конфігурації серверів застосувань, таких як Internet Information Server, SQL Server або BizTalk Server, кожен з яких грає свою роль, наприклад захищеного публічного веб-сервера. На цій діаграмі показуються взаємозв'язки між серверами. Логічні сервери можуть об'єднуватися в зони, що визначають логічні межі зв'язків. Зони можуть мати обмеження на типи логічних серверів, які в них можуть бути включені, а також на напрям і види зв'язків, що ведуть за межі зони.



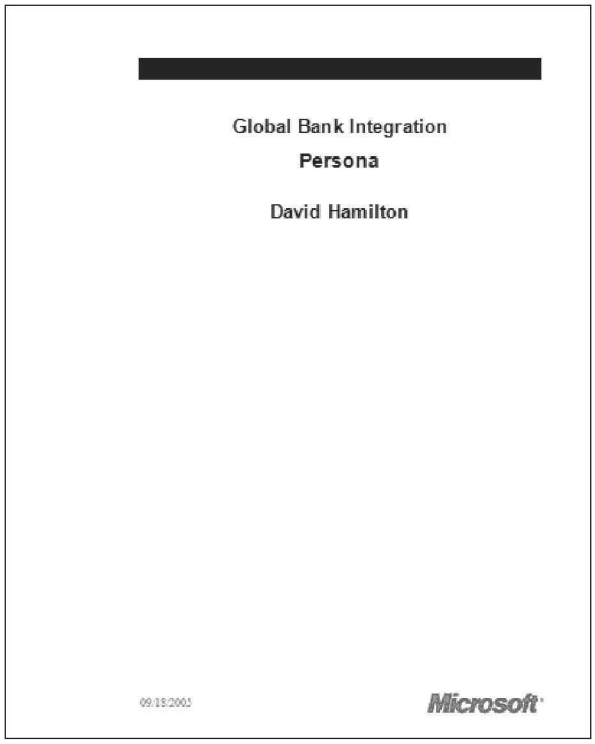
5. Ручний тест

Тести, що виконуються в ручному режимі, описуються в документах формату Word або в простих текстових файлах, що містять перелік дій, які треба виконати в процесі тестування.



Збірний образ

Збірний образ (persona) описує типові уміння, можливості, потреби, бажання, робочі звички, завдання і дані про утворення певної групи користувачів. Збірний образ — це вигаданий персонаж, що об'єднує в собі найважливіші характеристики реальної групи користувачів. Роздумувати про цілу групу користувачів простіше, уявляючи собі одну людину, оскільки одного легше зрозуміти, ніж групу. Кожного разу, маючи справу зі збірним образом, ми як би працюємо з окремою людиною, а насправді звертаємося до цілої групи, яку він втілює.



Контрольний список проекту

Контрольний список проекту — це перелік завдань, який має бути складений до початку проекту або ітерації.

Список вимог до якості

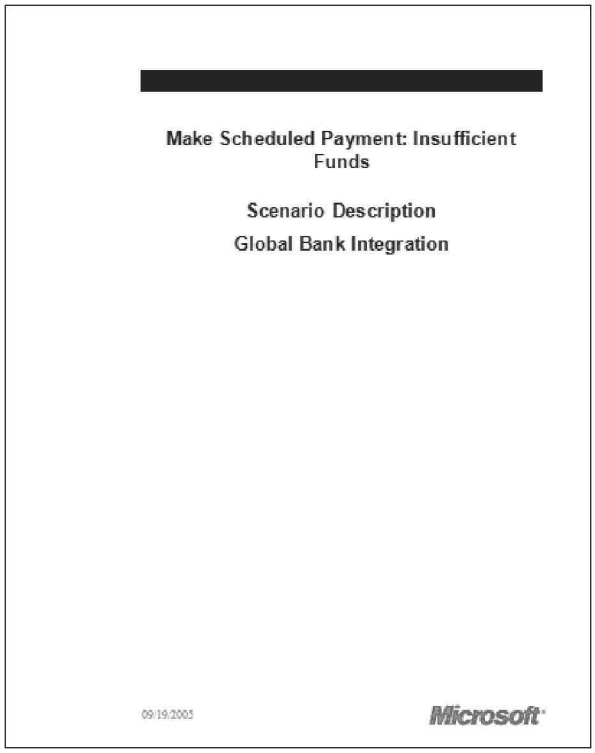
Цей список є переліком вимог до якості, які необхідно реалізувати. Після проведення огляду і призначення пріоритетів, список вимог до якості і список сценаріїв визначають мінімально прийнятний рівень проекту.

План випуску продукту

У план випуску продукту включаються усі заходи, пов'язані з випуском застосування. Він містить матеріали для усіх учасників, залучених у випуск. План випуску продукту може бути складений в Microsoft Office Project.

Опис сценарію

У сценарії описується певна частина взаємодії користувача з системою. У сценарії міститься запис послідовності дій користувача, які він виконує для досягнення деякої мети. У одних сценаріях фіксуються успішні спроби, в інших — безуспішні. При написанні сценаріїв слід бути конкретним. Оскільки можлива нескінченна кількість можливих сценаріїв, важливо записати тільки ті з них, які мають характерні відмінності.



6. Список сценаріїв

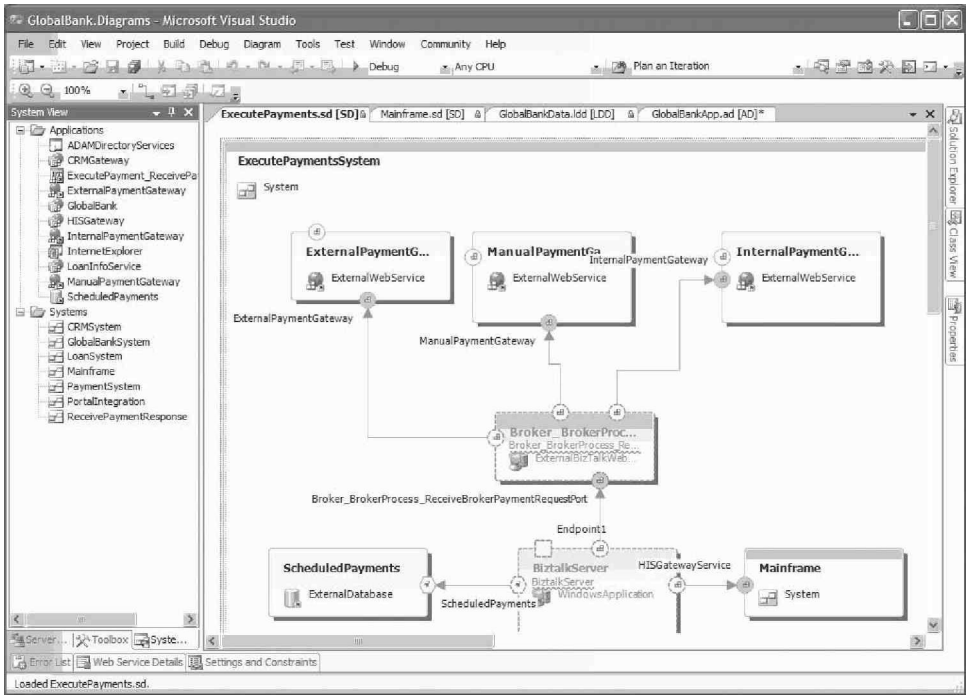
У списку сценаріїв міститься перелік обов'язкових робіт по реалізації сценаріїв проекту. Після проведення огляду і призначення пріоритетів, список вимог до якості і список сценаріїв визначають мінімально прийнятний рівень проекту.

Розкадровування

Опис деталей призначеного для користувача інтерфейсу в сценаріях може бути як текстовим, так і візуальним. Розкадровування використовується для ілюстрації деяких деталей в сценаріях до написання коду. Вона може бути створена в Microsoft Office PowerPoint або у будь-кому другом графічному редакторові.

7. Діаграма системи

Діаграма системи — це представлення структури застосувань і/або інших систем у вигляді сукупності компонентів. Зв'язки між застосуваннями і системами показують, як вони конфігуруються при розгортанні системи. Для ілюстрації зовнішніх інтерфейсів компонентів можуть використовуватися кінцеві точки. Системи можуть бути вкладені один в одного на будь-яку глибину. Системи можуть розроблятися як від низу до верху шляхом об'єднання існуючих застосувань або систем, так і зверху вниз шляхом опису характеристик системи із застосуванням так званих застосувань-прототипів (shadow application).



Групова зборка

Групова зборка — це компіляція усіх файлів, бібліотек або компонентів в новий набір виконуваних файлів, бібліотек або компонентів. Групові зборки, що пройшли контрольне тестування, називаються такими, що самотестуються (self — test), а ті, що не пройшли — самозаперечуваними (self — toast).

Підхід до тестування

Опис мети тестування, тестового покриття, методів тестування і тестових даних міститься у відповідній електронній таблиці. Для кожної ітерації є свій розділ з описом цілей тестування цієї ітерації і вживаних методів.

Результат тестування

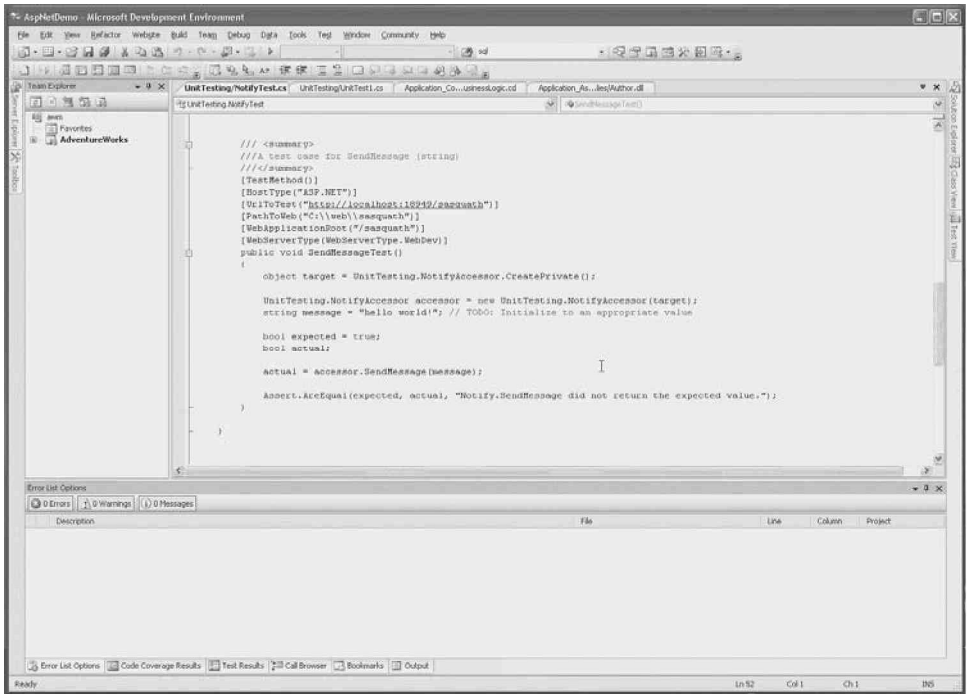
Результат тестування — цей висновок про результати виконання тестів. Можливі висновки: тест пройдений, не пройдений або не завершений.

Модель загроз

Модель загроз дозволяє відстежувати використання точок входу, через які можна отримати доступ до внутрішніх ресурсів. Якщо загроза можлива, то вона стає уразливістю в захисті.

Тест модуля

Тести модулів зазвичай пишуться розробником для перевірки поведінки окремих методів або їх наборів. Крім того, вони можуть використовуватися для тестування методом «прозорої скриньки» або у поєднанні з тестуванням навантаження і стресового.

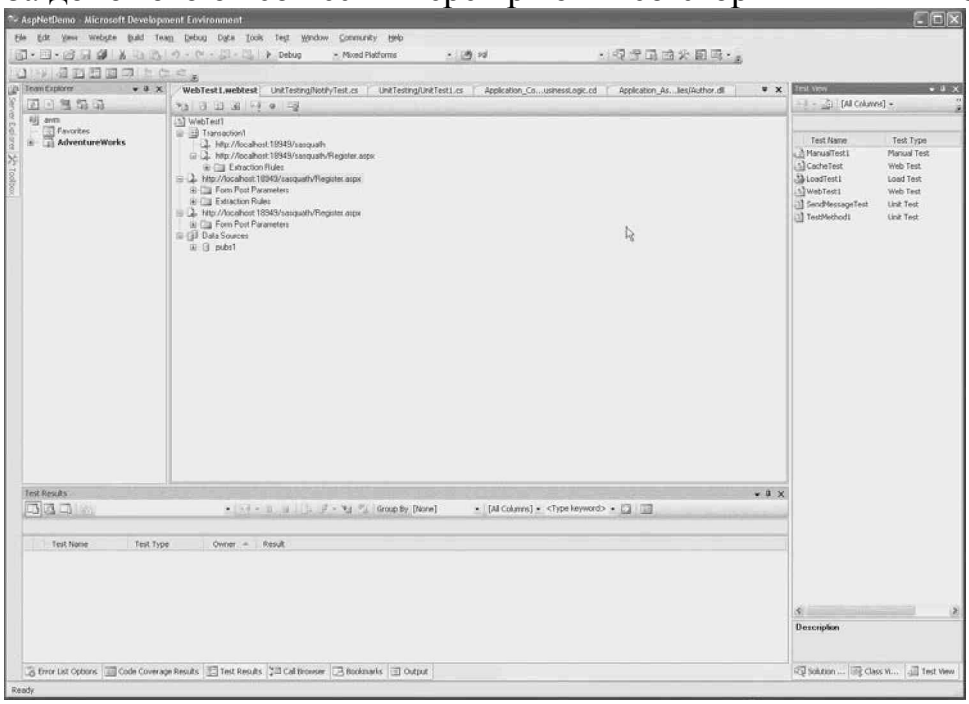


8. Концепція

Концепція проекту — цей опис закінченого продукту, його достоїнств і користі, яку зацікавлені сторони зможуть отримати від його використання.

Веб-тест

За допомогою веб-тестів перевіряють веб-сторінки і HTTP -запити.



Прототип

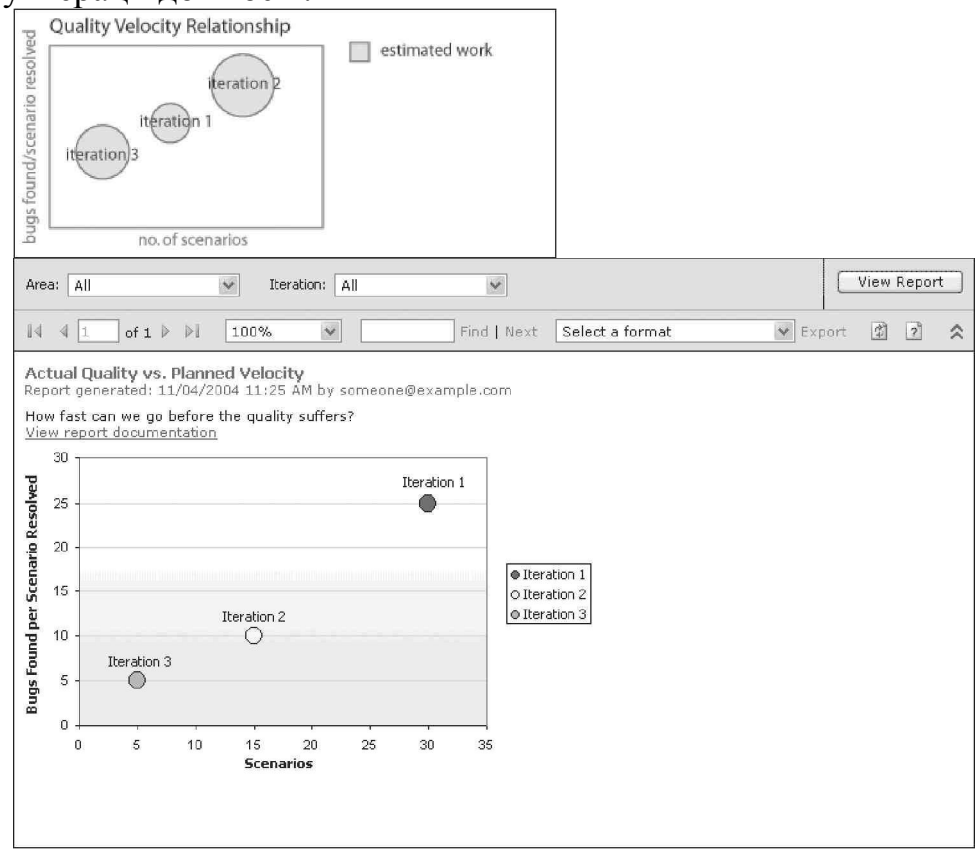
Прототип — це невеликий фрагмент коду, частенько дослідницький за своєю природою, призначений для вирішення певної проблеми або для зниження ризику. Прототип може бути виконаний на будь-якій мові програмування.

Лекція 13. Звіти про розробку Agile ПЗ в MSF.

1. Якість або швидкість
2. Пріоритетні дефекти
3. Інтенсивність дефектів
4. Індикатори якості
5. Робота, що залишилася
6. Позапланова робота
7. Темп
8. Відновлені роботи

1. Якість або швидкість

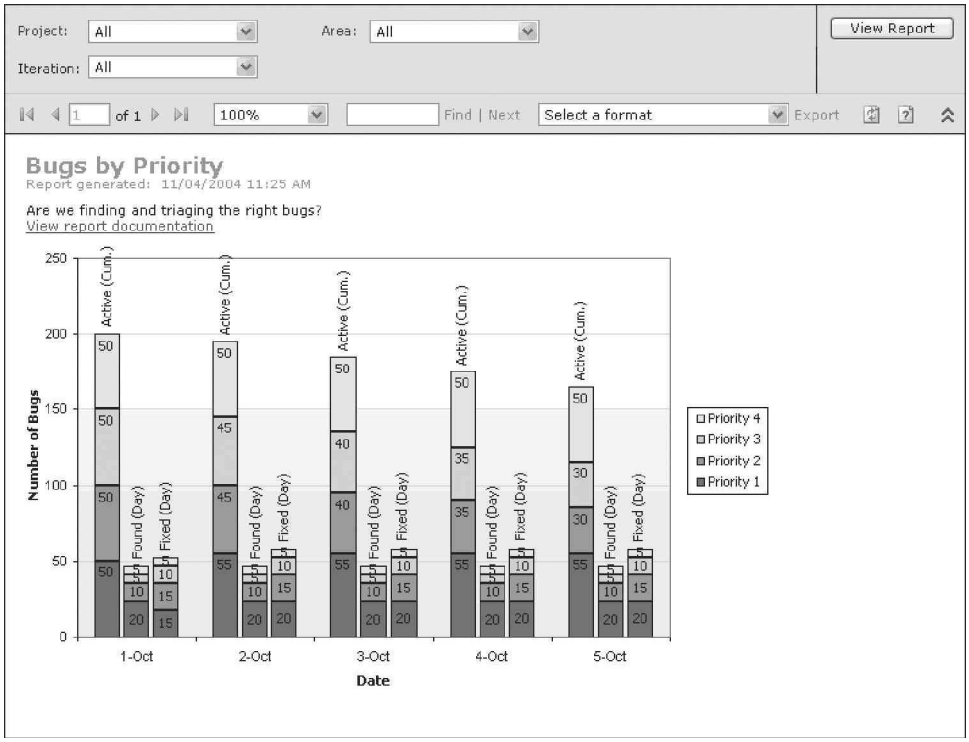
Як багато сценаріїв може бути виконано без втрати якості? Поки команди в роботі використовують приказку «поспішиш — людей насмішиш», в проекті є резерви для прискорення. Завдання менеджера проекту полягає в тому, щоб знайти такий баланс, при якому швидкість розробки буде максимальною при належній якості. На наступній діаграмі показано відношення передбачуваного об'єму ітерації до якості.



2. Пріоритетні дефекти

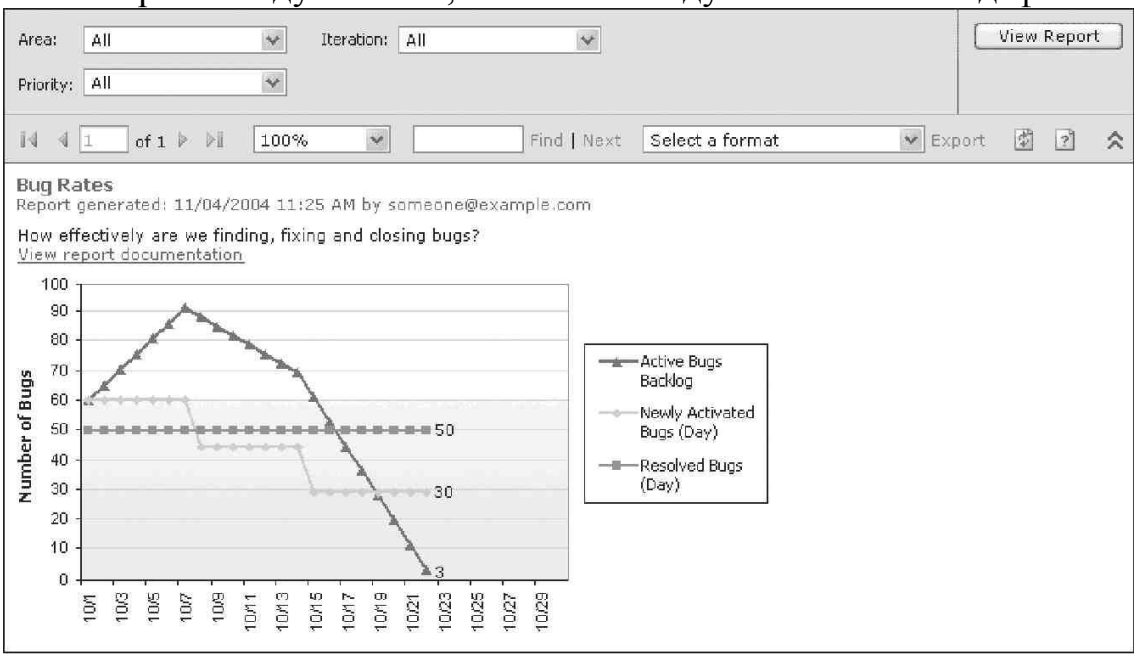
Чи правильно був виконаний пошук дефектів і призначення ним пріоритетів? Звіт, на якому показані пріоритетні дефекти, допомагає зрозуміти, наскільки ефективним був їх пошук і визначення важливості. Виявлення дефектів є невід'ємною частиною процесу розробки продукту. Проте частенько користувачеві докучають саме ті дефекти, які непросто виявити. Якщо дефекти, що мають високу важливість, не були виявлені, а є надмірно велика кількість дрібних недоліків, то слід направити зусилля тестування на пошук серйозних

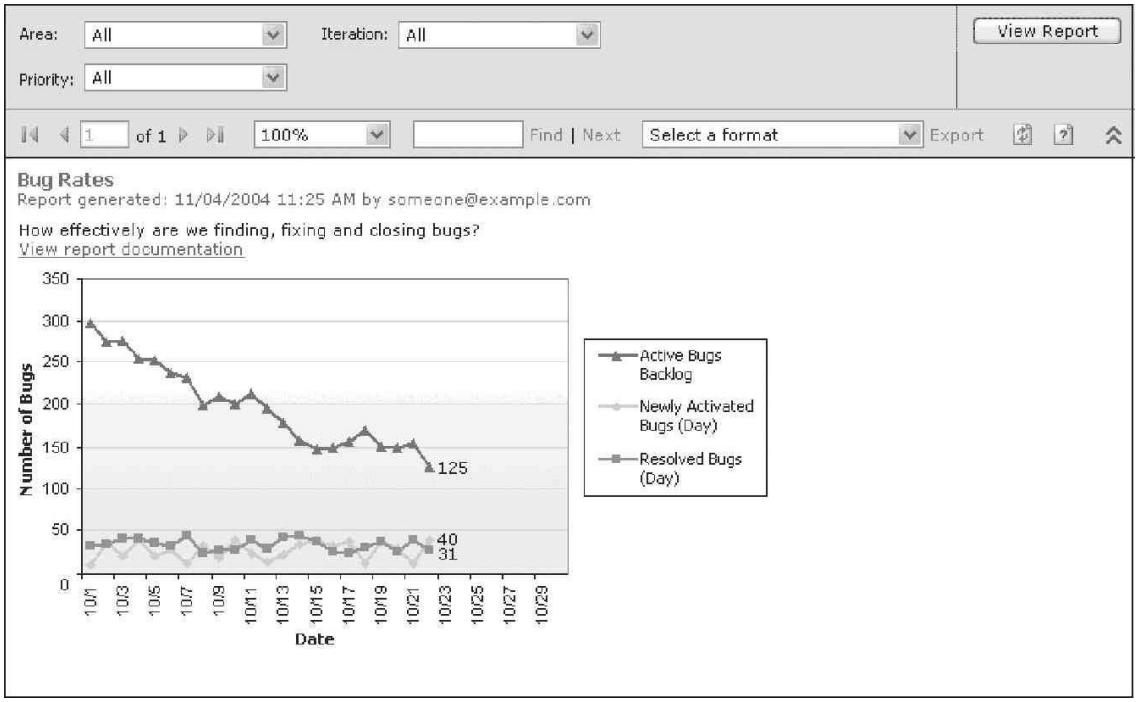
дефектів. Процес визначення пріоритетів дефектів таїть в собі небезпека переоцінити їх настільки, що їх кількість перевищить можливості їх усунення, або, навпаки, недооцінити їх, через що користувачі будуть украй невдоволені продуктом.



3. Інтенсивність дефектів

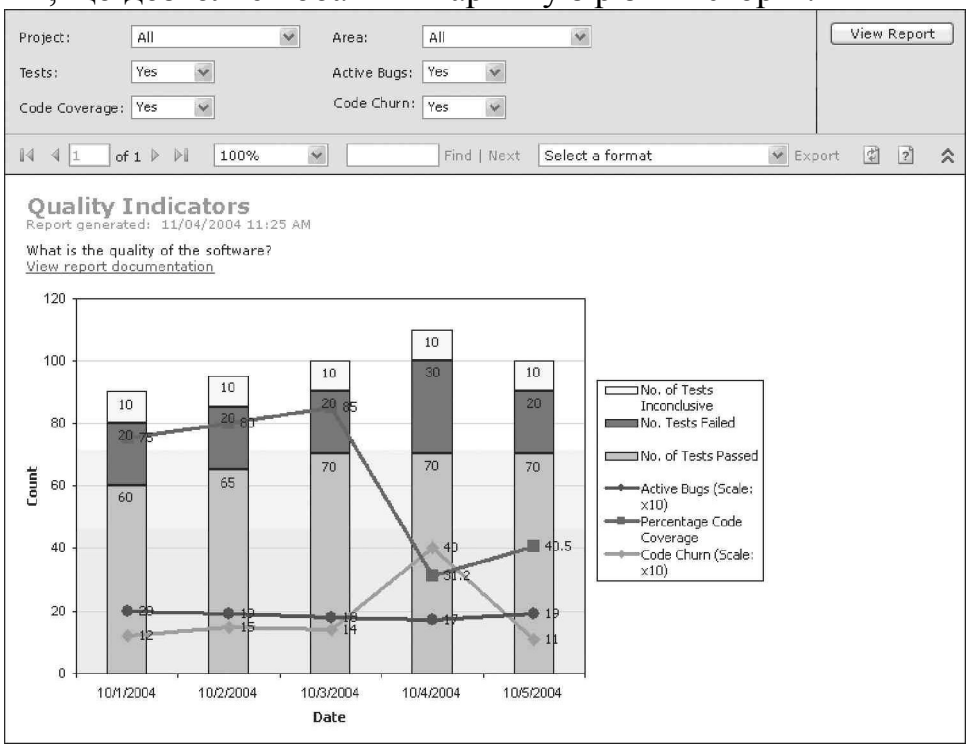
Наскільки ефективно виявляються, виправляються і закриваються дефекти? Оцінювати їх інтенсивність краще всього в зіставленні з усією поточною діяльністю команди і іншим метрикам на діаграмі індикаторів якості (Quality Indicators). Наприклад, високий коефіцієнт виявлення дефектів може свідчити як про погано написане, не повністю інтегрованому коді, так і про ефективне тестування. З іншого боку, низький коефіцієнт виявлення може означати високу якість продукту, або неефективне тестування. Для правильної оцінки корисні показники покриття коду тестами, мінливості коду і інтенсивності дефектів.





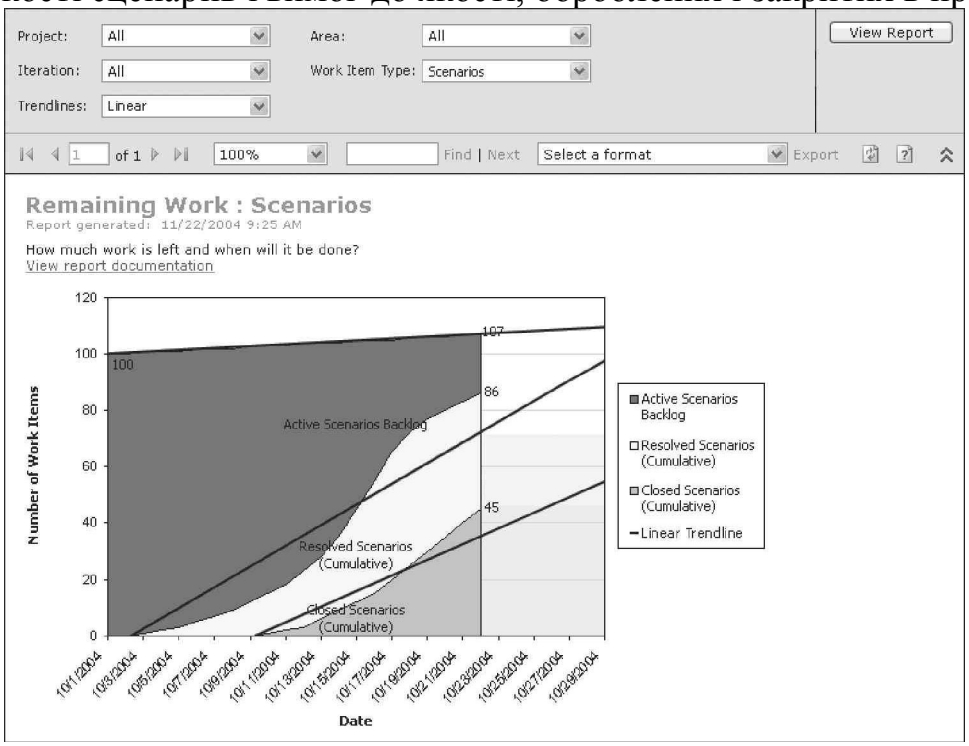
4. Індикатори якості

Яка якість продукту? У ідеальному випадку на діаграмах коефіцієнтів проходження тестів, кількості дефектів і мінливості коду можна побачити схожі картини, але так буває нечасто. У разі виявлення невідповідностей слід детальніше вивчити відповідні серії складок і даних. На наступній діаграмі об'єднані результати тестів, покриття ними коду, мінливість коду і кількість дефектів, що дозволяє побачити картину з різних сторін.



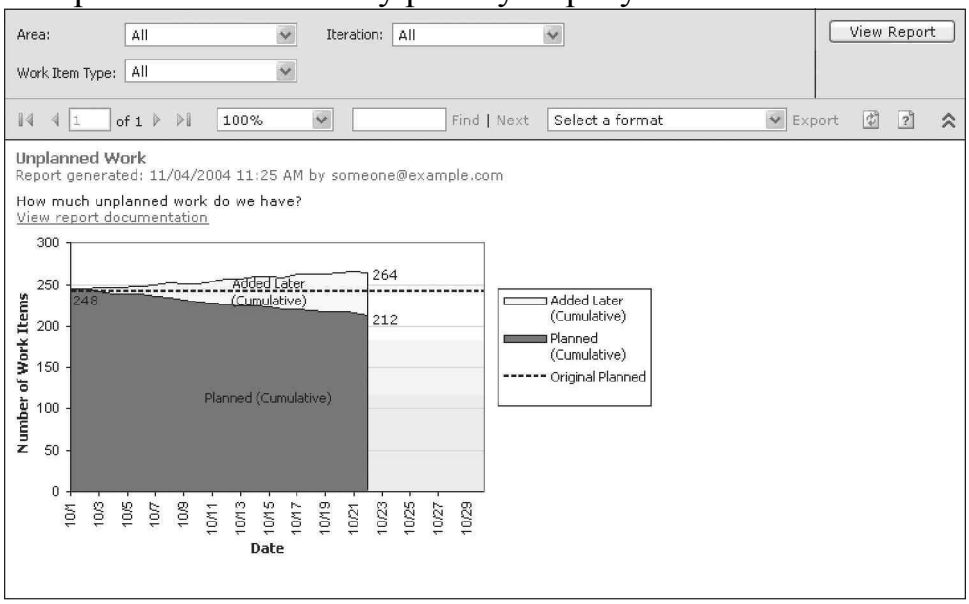
5. Робота, що залишилася

Скільки ще роботи належить виконати і коли вона має бути закінчена? На звідній діаграмі процесів показана кількість роботи, що залишилася, вимірюваної у кількості сценаріїв і вимог до якості, оброблених і закритих в процесі ітерації.



6. Позапланова робота

Який об'єм позапланових робіт? На наступній діаграмі відображений об'єм робіт, що не залишився, а загальний, розділений, у свою чергу, на заплановану і позапланову роботу. Украй рідко трапляється виконання усієї роботи за проектом раніше терміну, навіть у рамках окремої ітерації. Дуже корисно резервувати значний запас часу для непередбачених робіт (таких як виправлення дефектів). До того ж, якщо раптом не виявиться достатнього резерву часу, ви можете бути вимушені робити позапланову роботу за рахунок планової.



7. Темп

Наскільки швидко працює команда? Темп є одним з ключових параметрів такої оцінки. Він показує, наскільки швидко виконуються планові роботи, щоденні зміни плану, а також зміни від ітерації до ітерації. Ці дані, доповнені оцінкою якості, корисні при плануванні наступної ітерації. Ця діаграма, так само як і діаграма роботи, що залишилася, важлива для аналізу щодобового темпу у рамках ітерації або середнього темпу за ітерацію у рамках проекту.

8. Відновлені роботи

Як багато завдань доводиться поновлювати? Відновлені роботи — ті, які передчасно були відмічені як завершені або закриті. Невеликий рівень таких робіт цілком прийнятний (скажімо, менше 5%). Проте високий рівень або його підвищення повинні насторожити менеджера проекту. В цьому випадку слід виявити причину і усунути її.