МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
Кафедра інформаційних технологій

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

КОНСПЕКИ ЛЕКЦІЙ

014.09 Середня освіта (Інформатика)

Методичні рекомендації для самостійної роботи здобувачів

Одеса 2025

Електронне видання

ЗМІСТ

ВСТУП

Конспект лекцій містить матеріал для вивчення дисципліни «Якість програмного забезпечення та тестування». У ньому висвітлено питання, пов'язані із забезпеченням і контролем якості програмного забезпечення в рамках процесу його розробки. Особливу увагу приділено процесу тестування програмних продуктів.

В даний час існують різні значення терміна «якість». Філ Кросбі (Phil Crosby) в 1979 році дав визначення якості як «відповідність призначеним для користувача вимогам». Уотс Хемпфрі (Watts Hamphrey, оригінальний автор концепції моделі оцінки зрілості CMM, а також PSP і TSP − People Software
Process і Team Software Process, визначає якість як «досягнення відмінного рівня придатності до використання». Існують корпоративні стандарти управління якістю. Так, наприклад, компанія IBM ввела в обіг «якість, кероване ринковими потребами». Критерій Белдріджа (Baldrige) для організаційного якості (National Institute of Standards and Technology, "Baldrige National Quality Program", http://www.quality.nist.gov) використовує схожу фразу – «якість, що задається споживачем» (customer-driven quality), визначаючи задоволення споживача основним принципом щодо якості.

Найчастіше, поняття якості використовується відповідно до визначення системи менеджменту якості ISO 9001 як «ступінь відповідності властивих характеристик вимогам».

Сама «ступінь відповідності» також виступає в ролі обмеження проекту, а в додатку до індустрії програмного забезпечення представлена практично у всіх областях проектної діяльності − від управління вимогами («атрибути якості» як категорія нефункціональних вимог), до тестування (так званий наробіток на відмову, такі метрики як MTTF − Mean Time To Failure, тобто середній час між виявленими збоями системи, тощо).

В якійсь мірі, «прийнятну якість» можна порівнювати з рівнем обслуговування в рамках заданого SLA − Service Level Agreement, давно вже прийнятого на озброєння в телекомунікаційній індустрії. Таким чином, прийнятна якість може розглядатися як кількісно виражений компроміс між замовником і виконавцем щодо характеристик продукту, створюваного виконавцем в інтересах вирішення завдань замовника з урахуванням інших обмежень проекту (зокрема, вартістю, що часто іменується як «cost of quality» − «вартість якості»). Можна сказати, що такий погляд може в якійсь мірі розглядатися як розширення визначення в ISO 9001 з урахуванням досягнутого компромісу між замовником і виконавцем (постачальником) щодо характеристик якості.

Розглядаються питання якості програмного забезпечення, що виходять за рамки окремих процесів життєвого циклу. Так наприклад, якість програмного забезпечення є постійним об'єктом уваги програмної інженерії та обговорюється у багатьох областях знань в сфері інформаційних технологій, що цілком обгрунтовано, якщо врахувати воістину катастрофічний рівень «провалених»

проектів і незадоволеність користувачів програмних продуктів. У загальному випадку, описується ряд шляхів досягнення якості програмного забезпечення. Зокрема, ця область знань стосується «статичних технік», які не потребують виконання (створення) оцінюваних програмних систем, на відміну від «динамічних технік», розглянутих в галузі знань «Тестування».

SQA (Software Quality Assurance) концентрується на процесах. Роль SQA полягає в тому, щоб забезпечити відповідне планування процесів, подальше виконання процесів на основі заданого плану і проведення необхідних вимірювань процесів з передачею результатів вимірювань зацікавленим сторонам (організаційними структурам і особам).

Тестування, в загальному випадку, являє собою діяльність, виконувану для оцінки та вдосконалення програмного забезпечення. Ця діяльність, в загальному випадку, базується на виявленні дефектів і проблем в програмних системах.

Техніки управління якістю розділені на статичні (без виконання коду) і динамічні (з виконанням коду). Тестування входить до складу динамічних технік.

Тестування програмних систем складається з динамічної верифікації поведінки програм на кінцевому (обмеженому) наборі тестів, обраних відповідним чином з зазвичай виконуваних дій прикладної області та забезпечують перевірку відповідності необхідному поведінки системи.

Як вже зазначалося раніше, тестування програмного забезпечення тісно пов'язане з програмуванням. Більш того, модульне (unit-) та інтеграційне тестування все частіше розглядають як невід'ємний елемент діяльності з конструювання програмного забезпечення в SWEBOK.

Лекція 1. Введення. Предмет, ціль, задачі курсу. Основні поняття якості ПЗ

Стандарт ISO 9126:2001 регламентує зовнішні внутрішні характеристики якості. Перші відображають вимоги до функціонування програмного продукту. Для кількісного встановлення критеріїв якості, за якими буде здійснюватися перевірка і підтвердження відповідності ПЗ заданим вимогам, визначаються відповідні зовнішні вимірювані властивості (зовнішні атрибути) ПЗ, метрики (наприклад, час виконання окремих компонентів), діапазони зміни значень і моделі їх оцінки. Метрики використовуються на стадії тестування або функціонування і називаються зовнішніми метриками. Вони являють собою моделі оцінки атрибутів.

Внутрішні характеристики якості і внутрішні атрибути ПЗ використовуються для складання плану досягнення необхідних зовнішніх характеристик якості продукту. Для квантифікації внутрішніх характеристик якості застосовують внутрішні метрики, як інструмент перевірки відповідності проміжних продуктів внутрішнім вимогам до якості, які формулюються на процесах, що передують тестуванню.

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

Остаточна оцінка якості проводиться відповідно до стандарту ISO/IEC 14598. Якість може підвищуватися за рахунок постійного поліпшення використовуваного продукту виявленням, усуненням дефектів у ПЗ і їх запобіганням.

Область знань «Якість ПЗ (Software Quality)» складається з наступних розділів:

– концепції якості ПЗ (Software Quality Concepts);
– визначення і планування якості (Definition & Planning for Quality);
– техніки й види діяльності, що забезпечують гарантію якості, валідацію і верифікацію (Activities and Techniques for Software Quality Assurance, Validation & Verification –V&V);
– вимірювання при аналізі якості ПЗ (Measurement in Software Quality Analysis).

Концепції якості ПЗ – це зовнішні і внутрішні характеристики якості, їхні метрики, а також моделі якості, визначені на множині цих характеристик, що наведені в стандартах з якості – це шість характеристик і кожна з них має кілька атрибутів. До характеристик якості належать: – функціональність (functionality);
– надійність (realibility);
– зручність застосування (usability);
– ефективність (efficiency);
– супровід (maitainnability);
– переносність (portability).

Базова модель якості містить у собі ці характеристики і вони притаманні будь-якому типу програмних продуктів. При розробці вимог замовник формулює такі вимоги до якості, які найбільшою мірою підходять для програмного продукту, який замовляється.

Визначення і планування якості ПЗ ґрунтується на положеннях стандартів у цій області, складанні планів і графіків робіт, процедурах перевірки і ін. План забезпечення якості містить у собі набір дій для перевірки процесів забезпечення якості (верифікація, валідація і ін.) і формування документа з керування якістю.

Планування якості призначено для підтримки керування процесами досягнення якості продуктів проекту (зокрема проміжних робочих) і ресурсів – програмних, технічних, виконавських і ін. Воно також передбачає керування вимогами до процесів і продуктів і полягає в наступному:

– визначення продукту термінами заданими характеристиками якості;

– планування процесів для гарантії одержання необхідної якості; – вибір методів оцінки запланованих характеристик якості і встановлення відповідності продукту сформульованим вимогам.

У стандарті ISO/IEC 12207 визначені спеціальні процеси забезпечення якості – верифікація, валідація (атестація), спільний аналіз і аудит.

Види діяльності і техніки гарантії якості містять у собі, зокрема: інспекцію, верифікацію і валідацію ПЗ.

Інспекція ПЗ – аналіз і перевірка різних видів подання системи і ПЗ (специфікації, архітектурної схеми, діаграм, початкового коду тощо). Виконується на всіх процесах ЖЦ розробки ПЗ.

Верифікація ПЗ – процес забезпечення правильної реалізації ПЗ відповідно до специфікацій, виконується протягом усього життєвого циклу. Верифікація дає відповідь на питання, чи правильно створюється система.

Валідація – процес перевірки відповідності ПЗ функціональним і нефункціональним вимог і очікуваним потребам замовника.

Верифікація і валідація (V&V) можуть виконуватися, починаючи з ранніх стадій ЖЦ. Вони орієнтовані на отримання правильних функцій ПЗ, плануються і забезпечуються визначеними ресурсами з чітким розподілом ролей. Перевірка ґрунтується на використанні відповідних технік тестування для виявлення тих або інших дефектів і збирання статистики. Після зібрання даних оцінюється правильність реалізації вимог і роботи ПЗ у заданих умовах. Вимірювання при аналізі якості ПЗ ґрунтується на метриках продукту і даних, зібраних у процесі створення продукту при заданих ресурсах: оцінок процесів, ПЗ і його моделей, і передбачає документування вимірів.

Оцінювання якості продукту полягає у вимірюванні і оцінюванні якісних показників за допомогою даних про різні типи помилок і відмов під час тестування ПЗ і виконання коду на тестових даних. Ці дані аналізуються, перевіряються і використовуються при якісній і кількісній оцінки ПЗ. Для імітації роботи системи в режимі тестування розробляються тести з реальними вхідними даними для перевірки правильності роботи ПЗ на різних фрагментах програми і шляхах проходження в них операторів. У процесі тестування ПЗ виявляються різного роду помилки (відмови, дефекти, помилки тощо), кількість яких значною мірою може вплинути на одержання правильного і якісного результату.

З урахуванням типів виявлених помилок можна встановити наявність (або відсутність) відповідності реалізованих і нереалізованих функцій, заданих у вимогах до системи, а також оцінити способи реалізації нефункціональних вимог (продуктивності, надійності та ін.). Оцінюються процеси керування планами, інспекціями, прогонами і т.п. За цими оцінками приймаються рішення про завершення розробки продукту проекту і передачі його замовнику в експлуатацію, під час якої можуть бути внесені зміни щодо усунення помилок, визначення адекватності планів і вимог, оцінки ризиків перероблення ПЗ тощо.

Мета інспекцій – виявлення різних аномальних станів у ПЗ незалежними фахівцями команди експертів із залученням авторів проміжного або кінцевого

продукту. Експерти інспектують виконання вимог, інтерфейси, вхідні дані і т.п., а потім документують виявлені відхилення в проекті.

Призначення аудита – незалежна оцінка продуктів і процесів на відповідність регулюючим і регламентуючим документам (планам, стандартам і ін.), формулювання звіту про випадки невідповідності і пропозицій для їх коригування.

Таким чином, розгляд розділів SWEBOK свідчить по те, що це ядро містить весь необхідний набір знань з програмної інженерії, який повинні мати фахівці різних профілів (аналітики, інженери, програмісти, оцінювачі, контролери тощо), що виробляють програмний продукт.

Зазначимо, що ядро знань SWEBOK не позбавлено недоліків тактичного характеру. Так, між областями знань у цьому ядрі існують перетини за методами, концепціями і стратегіями, а деякі важливі напрями програмної інженерії взагалі не відбиті у ньому наявними областями знань. Це стосується, наприклад, методів доведення правильності програм, еволюції програм, розподілених і неоднорідних середовищ, взаємодії систем, таких методів програмування, як аспектне, агентне, сервісне й інші, а також аспектів захисту, безпеки тощо.

Основною проблемою в управлінні якістю є той факт, що визначення якості неясне і неоднозначне. Це викликано тим, що зазвичай термін «якість» розуміють неправильно. Причини: 1. Якість – це не окрема ідея або поняття, а багатомірна і різнопланова концепція. 2. Для будь-якого поняття і визначення існує декілька рівнів абстракції, наприклад, коли люди говорять про якість, одна частина розуміє під цим занадто широкий і розмитий смисл, а інша може вказувати на конкретне визначення. 3. Термін «якість» є невід'ємною частиною нашого повсякденного спілкування, але загальноприйняте і професійне використання може сильно відрізнятись. Популярний погляд на якість Загальноприйнята думка про якість – це дещо нематеріальне і «неосяжне», про нього можуть вестись суперечки та дискусії, його можна критикувати і вихваляти, але зважити і виміряти неможливо. Вирази типу «хороша якість» і «погана якість» служать наглядним прикладом. Люди так кажуть про щось невизначене для них, що вони не можуть чітко охарактеризувати і визначити. Отже, люди сприймають і інтерпретують якість по різному. Якість не може бути контрольованою і керованою. Якість не може бути кількісно виміряна. Такий погляд різко контрастує з професійним підходом до управління якістю – якість чітко визначена величина, яку можна виміряти і проконтролювати, вона піддається управлінню і покращенню. Інша популярна думка – якість нерозривно пов'язана з розкішшю, першим сортом і тонким смаком. Дорогий, досконально продуманий і технічно більш складний продукт розглядається як гарантія вищої якості ніж більш дешеві аналоги. За такою логікою Каділлак – якісна машина, а Шевроле – ні, незважаючи на надійність і кількість поломок. Якщо слідувати такому підходу, то якість обмежена визначеним класом дорогих продуктів. Недорогі продукти ж важко віднести до якісних. Професійний підхід до якості Існують чіткі та зручні для роботи визначення. В 1979 році Філ Кросбі визначив якість як «відповідність вимогам»

("conformance to requirements"). В 1970 році Юран і Гріна визначили якість як «придатність до використання» ("fitness for use"). Ці два визначення тісно пов’язані і добре узгоджуються. Ще декілька визначень якості: Уотс Хемпфрі – досягнення відмінного рівня придатності до використання; Компанія IBM – якість, яка управляється ринковими потребами (“market-driven quality”). Критерій Белдріджа для організаційної якості – “якість, яка задається споживачем” Визначення системи менеджменту якості ISO 9001 – ступінь відповідності наявних характеристик вимогам. «Відповідність вимогам» припускає, що вимоги мають бути настільки чітко визначені, що вони не можуть бути зрозумілі і інтерпретовані некоректно. Пізніше, на етапі розробки, проводяться регулярні вимірювання розробленого продукту для визначення відповідності вимогам. Будь-які невідповідності розглядаються як дефекти – невідповідність якості. Наприклад, специфікація на деяку модель радіостанції може вимагати можливість приймати визначену частоту радіохвиль на відстані більше ніж за 30 км від джерела віщання. У випадку, якщо радіостанція не може виконати дану вимогу, вона не відповідає вимогам до якості і має бути визнаною негідною і неякісною. Виходячи з таких принципів, якщо Каділлак відповідає усім вимогам до машин Каділлак, то це якісна машина. Якщо Шевроле відповідає усім вимогам до машин Шевроле, то це теж якісна машина. «Придатність до використання» приймає до уваги вимоги і очікування кінцевого користувача, який очікує, що продукт або надаваний сервіс буде зручним для нього. Однак різні користувачі можуть використовувати продукт по різному і це означає, що продукт має володіти максимально різноманітними варіантами використання. Згідно визначенню Юран кожен варіант використання – це характеристика якості і усі вони можуть бути класифіковані по категоріям в якості параметрів придатності до використання. Ці два визначення якості по суті однакові. Різниця в тому, що варіант «придатність до використання» вказує на важливу роль вимог і очікувань замовника.

Роль замовника, пов’язана з якістю, не може бути переоцінена. З точки зору замовника, якість продукту, який він придбав, складається з багатьох факторів: ціна, продуктивність, надійність тощо. Тільки замовник може розказати про якість, оскільки це єдине, що він дійсно купляє. Замовник не купляє продукт. Він купляє гарантії того, що усі його очікування до продукту будуть реалізовані.

Отже, визначення якості з точки зору замовника або користувача продукту: Якість – це придатність до використання. Чи робить даний продукт те, що мені потрібно? Чи спрощує роботу? Чи можу я його використати так, як мені потрібно?

Точка зору розробника: Якість – це відповідність специфікованим і забраним вимогам. Чи робить даний продукт усе те, що вказано у вимогах? Проблема в тому, що специфіковані і забрані вимоги – це зазвичай частина реальних вимог і очікувань замовника.

Отже, якість – це відповідність реальним вимогам, явним і неявним. Дуже часто неявні вимоги настільки очевидні для замовника або користувача, що він навіть не припускає, що вони невідомі розробникам. На прикладі автомобілів: замовник може детально описати вимоги до дизайну, параметрам двигуна, оформленню

салону, кольору кузова, але ніде не вказати, що шини мають бути круглими, а лобове скло – прозорим. Замовник буде повністю задоволеним тільки тоді, коли куплений продукт буде повністю відповідати його реальним і життєвим вимогам – специфікованим і інші.

Існує дві професії пов'язані з якістю програмного забезпечення: тестер (Quality Control) і QA manager. В чому різниця між Тестуванням и QA (Quality Assurance)? Контроль якості (QUALITY CONTROL) це вимірювання якості продукту. Забезпечення якості (QUALITY ASSURANCE) – це вимірювання і управління якістю процесу, який використовується для створення якості продукту (або якісного продукту).

Іншими словами. Quality Assurance – попередження дефектів шляхом перевірки і тестування процесу. Quality Control − виявлення дефектів шляхом перевірки і тестування продукту.

Лекція 2. Підходи до підвищення якості ПЗ. Стандарти якості. Узагальнена модель якості

ISO 9126 − «Інформаційна технологія. Оцінка програмного продукту. Характеристики якості та керівництво по їх застосуванню». ISO 9126 − це міжнародний стандарт, який визначає оцінні якості програмного забезпечення. Стандарт поділяється на 4 частини, які описують наступні питання: модель якості, зовнішні метрики якості, внутрішні метрики якості, метрики якості у використанні. Модель якості, встановлена у першій частині стандарту 9126-1, класифікує якість ПЗ в 6-ти структурних наборах характеристик, які в свою чергу деталізовані під-характеристиками (субхарактеристиками), такими як: Функціональність (functionality) – набір атрибутів, який характеризує відповідність функціональних можливостей ПЗ набору потрібної користувачу функціональності.

Деталізується субхарактеристиками:

− Придатність до застосування (suitability) – здатність ПС надавати належну множину функцій для розв'язання специфікованих задач і досягнення цілей користувача;

− Коректність (правильність, точність) (accuracy) – здатність ПС забезпечувати правильні або погоджені результати або впливи з необхідним ступенем точності;

− Здатність до взаємодії (в тому числі мережевої) (interoperability) – здатність взаємодіяти з однією чи більше специфікованими системами;

− Захищеність (security) – здатність забезпечувати захист інформації від несанкціонованого доступу осіб або систем і її доступність особам та системам з необхідними правами доступу.

Надійність (reliability) – набір атрибутів, які відносяться до здатності ПЗ зберігати свій рівень якості функціонування в встановлених умовах за визначений період часу.

Субхарактеристики:

− Рівень завершеності (відсутність помилок) (maturity) – здатність уникати відмови із-за внутрішніх дефектів;

− Стійкість до дефектів (fault tolerance) – здатність підтримувати встановлений рівень функціонування в умовах прояви дефектів в ПС, помилок даних або порушень специфікованого інтерфейсу;

− Відновлюваність (recoverability) – здатність відновлювати функціонування на заданому рівні і відновлювати пошкоджені програми та дані;
− Доступність; − Готовність.

Практичність (зручність застосування) (usability) – набір атрибутів, які відносяться до об'єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів.
Субхарактеристики:

− Зрозумілість (understandability) – властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв'язання визначених задач і які умови її використання;

− Простота використання (learnability) – властивості ПС, які забезпечують користувачу можливість засвоїти прийоми її застосування;

− Керованість (operability) властивості ПС, які забезпечують користувачу можливість керувати або контролювати її дії;

− Привабливість (attractiveness). Ефективність (efficiency)– набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах. Суб:

− Часова ефективність (time behavior) - властивості ПС, які спричиняють її здатність забезпечувати належний час відклику (відповіді) і обробки завдань, а також рівень пропускної здатності при виконанні функцій у встановлених умовах застосування;

− Використовуваність ресурсів (resource utilization) властивості ПС, які спричиняють її здатність використовувати ресурси в необхідні періоди часу при виконанні своїх функцій у встановлених умовах застосування. Супроводжуваність (maintainability) – набір атрибутів, які відносяться до об'єму робіт, які необхідні для проведення конкретних змін (модифікацій).
Субхарактеристики:

− Зручність для аналізу (analyzability) властивості ПС, які спричиняють можливість діагностування її недоліків або причин відмов, а також ідентифікації частин, які мають модифікуватись;

− Змінюваність (changeability) властивості ПС, які спричиняють можливість виконання встановлених видів модифікації;

− Стабільність (stability) − властивості ПС, які спричиняють її здатність мінімізувати неочікувані ефекти модифікацій;

− Зручність для тестування (testability) − властивості ПС, які спричиняють її здатність сприяти перевірці модифікованого ПЗ. Мобільність (portability) – набір

атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше.

Субхарактеристики:

− Здатність до адаптації (adaptability) − властивості ПС, які спричиняють її здатність адаптуватись для застосування в різноманітних специфікованих середовищах без використання дій або засобів відмінних від тих, що спеціально призначені для цих цілей;

− Простота встановлення (installability);

− Співвіснування (відповідність) (co-existence) властивості ПС, які спричиняють її здатність співіснувати з іншими незалежними ПС в спільному середовищі, розділяючи спільні ресурси;

− Здатність до заміщення (replaceability) − властивості ПС, які спричиняють її здатність використовуватись замість інших специфікованих ПС в середовищі їх застосування.

Підхарактеристика «Відповідність» не приведена і списку, але вона належить усім характеристикам. Ця характеристика має відображати відсутність протиріч з іншими стандартами або характеристиками. Наприклад, відповідність надійності і практичності. Кожна під характеристика якості далі розбивається на атрибути.

Атрибут – це сутність, яка може бути перевірена або виміряна в програмному продукті. Атрибути не визначені в стандарті із-за їх різноманітності в різних програмних продуктах. В стандарті виділена модель характеристик експлуатаційної якості.

Основні:

− Результативність – ступінь в якій користувачами досягаються задані цілі по точності і повноті розв’язування задач у встановленому контексті використання ПС;

− Продуктивність – продуктивність при розв’язанні основних задач програмної системи, яка досягається при реально обмежених ресурсах в конкретному зовнішньому середовищі застосування;

− Безпека – надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища;

− Задоволення потреб і витрат користувачів у відповідності з цілями застосування ПЗ.

Друга частина стандарту ISO 9126-2 присвячена формалізації зовнішніх метрик. Зовнішні метрики відносяться до атрибутів, які визначають поведінку ПЗ у зовнішньому середовищі і взаємодію з іншими програмами.

Третя частина стандарту ISO 9126-3 присвячена формалізації внутрішніх метрик. Такі метрики вимірюють атрибути внутрішніх характеристик ПЗ.

Метрика – це комбінація конкретного методу вимірювання (способу отримання значень) атрибуту сутності і шкали вимірювання (засобу, який використовується для структурування отриманих значень). Викладені загальні рекомендації по використанню відповідних метрик і взаємозв’язку між типами метрик.

Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, супроводжуючих, користувачів і менеджерів якості ПЗ. В ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПЗ.

Лекція 3. Концепція інженерії якості. Моделі та метрики якості. Класифікація моделей та їх призначення.

Що таке якість програмного забезпечення? Якщо спитати про це достатньо широку групу людей, які мають справу з розробкою, продажем або використанням ПЗ, можна отримати наступні відповіді:
− Легко використовувати;
− Висока продуктивність;
− Немаэє помилок;
− Не псує даних користувача при збоях;
− Можна використовувати на різних платформах;
− Може працювати 24 години на сутки і 7 днів на тиждень;
− Легко добавляти нові можливості; − Задовольняє потреби користувача; − Добре документовано.
Якість ПЗ по МакКолу Першою широко відомою моделлю якості ПЗ стала запропонована в 1977 МакКолом та іншими модель. В ній характеристики якості розділені на три групи: Фактори, які описують ПЗ з позицій користувача. Задаються вимогами. Критерії, які описують ПЗ з позицій розробника. Задаються як цілі. Метрики, які використовуються для кількісного описання і вимірювання якості. Фактори якості, яких було виділено 11, групуються в три групи по різним способам роботи людей з ПЗ. Отримана структура відображується у вигляді трикутника МакКола Критерії якості – це числові рівні факторів, поставлених як цілей при розробці. Об’єктивно оцінити або виміряти фактори якості безпосередньо важко. Тому МакКол ввів метрики якості, які з його точки зору легше виміряти і оцінити. Оцінки в його шкалі приймають значення від 0 до 10.

Метрики:

- Зручність перевірки на відповідність стандартам (auditability);
- Точність керування і обчислень (accuracy);
- Ступінь стандартності інтерфейсів (communication commonality);
- Функціональна повнота (completeness);
- Однорідність використовуваних правил проектування і документації (consistency);
- Ступінь стандартності форматів даних (data commonality);
- Стійкість до помилок (error tolerance);
- Ефективність роботи (execution efficiency);
- Розширюваність (expandability);
- Широта області потенційного використання (generality);

- Незалежність від апаратної платформи (hardware independence);
- Повнота протоколювання помилок та інших подій (instrumentation);
- Модульність (modularity);
- Зручність роботи (operability);
- Захищеність (security);
- Самодокументованість (selfdocumentation);
- Простота роботи (simplicity);
- Незалежність від програмної платформи (software system independence);
- Можливість співвідношення проекту з вимогами (traceability); − Зручність навчання (training).

Кожна метрика впливає на оцінку деяких факторів якості. Числовий вираз фактору представляє собою лінійну комбінацію значень впливаючих на нього метрик. Коефіцієнти цього виразу визначаються по різному для різних організацій, команд розробки, видів ПЗ та інші.

Модель якості МакКолла на найвищому рівні має три характеристики: функціональність, модифікованість і переносність, а на нижчих рівнях моделі – 11 підхарактеристик якості і 18 критеріїв (атрибутів) якості. Стандарт ISO 9126 пропонує використовувати для опису внутрішнього та зовнішнього якості ПЗ багаторівневу модель. На верхньому рівні виділено 6 основних характеристик якості ПЗ. Кожна характеристика описується за допомогою кількох вхідних у неї атрибутів. Для кожного атрибута визначається набір метрик, що дозволяють його оцінити. Множина характеристик і атрибутів якості згідно з ISO 9126. Визначення цих характеристик і атрибутів за стандартом ISO 9126:2001:

- Функціональність (functionality). Здатність ПЗ в певних умовах вирішувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно вирішує.
- Функціональна придатність (suitability). Здатність вирішувати потрібний набір задач.
- Точність (accuracy). Здатність видавати потрібні результати.
- Здатність до взаємодії (interoperability). Здатність взаємодіяти з потрібним набором інших систем.
- Відповідність стандартам і правилам (compliance). Відповідність ПЗ наявним індустріальним стандартам, нормативним і законодавчим актам, іншим регулюючим нормам.
- Захищеність (security). Здатність запобігати неавторизованому, тобто без вказівки особи, що намагається його здійснити, і недозволеному доступу до даних і програм.
- Надійність (reliability). Здатність ПЗ підтримувати визначену працездатність у заданих умовах.
- Зрілість, завершеність (maturity). Величина, зворотна частоті відмов ПЗ. Звичайно виміряється середнім часом роботи без збоїв і величиною, зворотною імовірності виникнення відмови за даний період часу.

- Стійкість до відмов (fault tolerance). Здатність підтримувати заданий рівень працездатності при відмовах і порушеннях правил взаємодії з середовищем.
- Здатність до відновлення (recoverability). Здатність відновлювати визначений рівень працездатності та цілісність даних після відмови, необхідні для цього час і ресурси.
- Відповідність стандартам надійності (reliability compliance). Цей атрибут доданий в 2001 році.
- Зручність використання (usability) або практичність. Здатність ПЗ бути зручним у навчанні та використанні, а також привабливим для користувачів.
- Зрозумілість (understandability). Показник, зворотний до зусиль, які затрачаються користувачами на сприйняття основних понять ПЗ та усвідомлення їх застосовності для розв'язання своїх задач.
- Зручність навчання (learnability). Показник, зворотний зусиллям, затрачуваним користувачами на навчання роботі з ПЗ.
- Зручність роботи (operability). Показник, зворотний зусиллям, що вживається користувачами для розв'язання своїх задач за допомогою ПЗ.
- Привабливість (attractiveness). Здатність ПЗ бути привабливим для користувачів. Цей атрибут доданий в 2001 році.
- Відповідність стандартам зручності використання (usability compliance). Цей атрибут доданий в 2001 році.
- Продуктивність (efficiency) або ефективність. Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділюваного для цього ресурсам. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до затрачуваних на це ресурсів усіх типів.
- Часова ефективність (time behaviour). Здатність ПЗ видавати очікувані результати, а також забезпечувати передачу необхідного об'єму даних за відведений час.
- Ефективність використання ресурсів (resource utilisation). Здатність вирішувати потрібні задачі з використанням певних об'ємів ресурсів певних видів. Маються на увазі такі ресурси, як оперативна й довгострокова пам'ять, мережні з'єднання, пристрої вводу та виводу та ін.
- Відповідність стандартам продуктивності (efficiency compliance). Цей атрибут доданий в 2001 році.
- Зручність супроводу (maintainability). Зручність проведення всіх видів діяльності, пов'язаних із супроводом програм.
- Аналізованість (analyzability) або зручність проведення аналізу. Зручність проведення аналізу помилок, дефектів і недоліків, а також зручність аналізу необхідності змін і їх можливих наслідків.
- Зручність внесення змін (changeability). Показник, зворотний трудозатратам на виконання необхідних змін.
- Стабільність (stability). Показник, зворотний ризику виникнення несподіваних ефектів при внесенні необхідних змін.

- Зручність перевірки (testability). Показник, зворотний трудозатратам на проведення тестування і інших видів перевірки того, що внесені зміни привели до потрібних результатів.
- Відповідність стандартам зручності супроводу (maintainability compliance). Цей атрибут доданий в 2001 році.
- Переносимість (portability). Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні й програмні аспекти оточення.
- Адаптованість (adaptability). Здатність ПЗ пристосовуватися різним оточенням без проведення для цього дій, крім заздалегідь передбачених.
- Зручність установки (installability). Здатність ПЗ бути встановленим або розгорнутим у певному оточенні.
- Здатність до співіснування (coexistence). Здатність ПЗ співіснувати з іншими програмами у загальному оточенні, ділячи з ними ресурси.
- Зручність заміни (replaceability) іншого ПЗ даним. Можливість застосування даного ПЗ замість інших програмних систем для вирішення тих же задач у певному оточенні.

Відповідність стандартам переносимості (portability compliance). Цей атрибут доданий в 2001 році.

Тема 4. Вимірювання в програмній інженерії. Модель процесу вимірювань. Артефакти. Парадигма «Ціль-питання-міра»

Процес вимірювання входить в групу організаційних процесів ЖЦ і непосредственно пов'язаний з процесом верхнього рівня − процесом управління. Він може виконуватися «в інтересах», перш за все процесів управління проектом, якістю, ризиком і процесу вдосконалення. Відповідальність за визначення, впровадження, супровід, оцінювання та покращення процесу вимірювання покладається на менеджерів (керівників) рівня організації або проекту.

Перспективи успіху в виконанні і вдосконаленні всіх видів діяльності в процесах ЖЦ істотно розширюються, коли рішення приймаються на основі аналізу фактичної, кількісної інформації. Ця інформація може бути отримана тільки за допомогою спостереження і вимірювання процесів, продуктів і ресурсів, залучених в процеси. Для того щоб вимір було економічно виправданим, воно має бути добре спроектовано і націлене на підтримку ділових цілей організації.

Існує чотири причини, що зумовлюють вимір процесів, продуктів і ресурсів в ЖЦ − це необхідність їх охарактеризувати, оцінити, передбачити або вдосконалювати.

Вимірювання характеристик процесів, продуктів і ресурсів забезпечує, перш за все, розуміння їх суті і створення базису для виконання порівняльно го аналізу.

Вимірювання з метою подальшого оцінювання допомагає визначати стан виконання планів, а заходи служать датчиками, використовуваними для контролю.

Крім того, оцінювання важливо для визначення можливості досягнення цілей якості та впливу удосконалень технології і процесу на продукти і процеси.

Вимірювання з метою передбачення спрямоване на досягнення розуміння взаємозв'язків між процесами і продуктами і побудова моделей цих взаємодії зв'язків, що дає можливість використовувати значення спостережуваних атрибутів для передбачення значень інших (спостережуваних) атрибутів. Це робиться для твердженню досяжних цілей за вартістю, планом-графіком і якості і раціонального розподілу ресурсів. Пророкують заходи служать також основою для визначення тенденцій в проектах і процесах, що дозволяє оновлювати оцінки вартості, часу і якості виходячи з поточних об'єктивних даних. проекції і оцінки, засновані на історичних даних, також допомагають аналізувати ризики та робити вибір альтернативних варіантів проекту.

Вимірювання з метою вдосконалення припускають збір кількісної інформації, яка допомагає розпізнавати проблеми та перешкоди на шляху поліпшення якості продукції та виконання процесу, а також планувати і контролювати витрати зусиль на здійснення вдосконалень. Вимірювання поточного стану процесу створюють порівняльний базис, завдяки якому можна судити про те, чи виконуються дії з удосконалення належним чином, і якими можуть бути побічні ефекти удосконалень.

Акцент в целеоріентірованной (інакше кажучи, керованому цілями) вимірюванні робиться на плануванні і здійсненні збору і аналізу інформації, яка допомагає досягти ділових цілей, а також на підтримку механізмів зворотного зв'язку (трасуванню) від заходів назад до діловим цілям.

Керований цілями процес вимірювання базується на 3 принципах і включає 10 етапів робіт.

Три принципу вимірювання такі:

− Цілі вимірювання витягуються з ділових цілей;

− Контекст вимірювання забезпечується побудовою ментальних моделей;

− Неформальні мети перетворюються в інформаційні структури вимірюваня.

Етапи процесу вимірювання такі:

1. Визначити ділові цілі.
2. Ідентифікувати об'єкти дослідження або досліджувані проблеми.
3. Визначити подцели.
4. Ідентифікувати сутності й атрибути, пов'язані з підцілі.
5. Формалізувати мети вимірювання.
6. Сформулювати питання, які потребують відповіді в кількісної формі, і визначити придатні для отримання відповідей на питання наочні засоби відображення залежностей (діаграми), які будуть використовуватися для того, щоб допомогти оцінити досягнення цілей вимірювання.
7. Визначити елементи даних, що збираються для конструювання діаграм.

8. Дати визначення використовуваних заходів і зробити ці визначення оперативниминими, заснованими на конкретних метриках (методах визначення значень і шкалах).
9. Ідентифікувати дії, які будуть зроблені для виконання вимірювань (визначення заходів).
10. Підготувати план реалізації вимірювань.

Основним механізмом перетворення ділових цілей в опис проблем, питань і заходів є ментальні моделі оперативних процесів інженерії або управління, що відображають взаємозв'язки, що існують між елементами, асоційованими з процесами.

Існує 4 типи елементів в уявленнях оперативних процесів:

− Те, що процеси отримують ззовні (входи і ресурси) і використовують або витрачають;

− Те, що процеси виробляють (результати) - основні продукти, проміжні продукти і ефекти;

− То, з чого процеси складаються (дії, граф-схеми виконання процесів і дійові особи) − структура процесів;

− Те, що процеси містять або зберігають (внутрішні артефакти, наприклад: напрацювання, незавершена робота, використовувані методи та інструменти).

Всі елементи в процесі є сутностями, що мають атрибути, прямо або побічно належать до ділових цілей. Ментальні моделі процесу повинні акцентувати увагу на тих елементах, які надають суттєвий вплив на процес або дають глибоке уявлення про нього.

Лекція 5. Забезпечення гарантії якості в життєвому циклі. Процес SQA. План якості

В архітектурі процесів ЖЦ ПС представлені два процеси, в яких міститься термін «якість» − це підтримує процес «забезпепечення гарантії якості »(SQA) і організаційний процес «управління якістю» (або «менеджмент якості» (від «quality management»)).

Перший з процесів пов'язаний з двома видами діяльності – впровадженням стандартів якості та відповідних процедур в розробку ПС і оцінкою прихильності цим стандартам і процедурам. Ці види діяльності здавна є супутні розробці ПС і вимоги до них включені практично в усі комкомплекси національних і галузевих стандартів з розробки програмного забезпепечення. Другий з процесів − «управління якістю», що включений до складу процесів ЖЦ з 1998 року (проекти стандартів ISO / IEC 12207 (1998 - 2001 роки), ISO / IEC TR 15504 (1998 - 2002 роки)). Включення цього процесу в архітектуру (поряд з процесами «управління проектом», «управління ризиком», «вимір» та ін.) свідчить про те, що багатьма організаціями-розробниками ПС накопичено достатній досвід виконання і управління процесами програмної інженерії (іншими словами, подолано бар'єр 2-го рівня зрілості по моделі CMM). Хоча процес SQA залишається

основоположним процесом гарантування якості ПС, для дотримання загального целеоріентірованного підходу до виконання дій в ЖЦ він інтегрується з процесом управління якістю, який і забезпечує моніторинг досягнення встановлених і передбачуваних вимог споживача до якості ПС.

Виходячи з положень стандарту ISO / IEC 12207, цілі процесу гарантування якості повинні бути такими, щоб їх досягнення давало впевненість в тому, що продукти ПС і процеси, що застосовуються для отримання цих продуктів, узгоджуються з пропонованими до них вимогами і відповідають затвердженим планам. «В результаті успішного здійснення процесу:

· Буде визначена стратегія виконання дій і рішення задач в рамках процесу SQA1. Ця стратегія повинна підтримуватися стандартами для кожного процесу і продукту, процедурами і інфраструктурою, які впроваджуються і супроводжуються на рівні організації або проектів;

· Відомості про дії та завдання щодо забезпечення якості будуть фіксуватися і супроводжуватися. Повинні бути визначені обліково-звітні документи за якістю, які будуть демонструвати відповідність процесу і робочих продуктів стандартам якості;

· Відповідність програмних продуктів, процесів і дій застосованим стандартам, процедурам, вимогам будуть об`єктивно веріфікуватись.

· Будуть ідентифікуватися проблеми або розбіжності з вимогами договору. Повинна бути організована звітність про виконання, відхилення і тенденції в діяльності по процесам ЖЦ перед відповідною аудиторією. Будь-які відхилення потрібно аналізувати, коригувати і запобігати в майбутньому їх появу.

Процес гарантування якості повинен координуватися з іншими процесами підтримки, такими як верифікація, валідація, спільний перегляд, аудит і управління рішенням проблем, і може використовувати їх результати». Цілі процесу гарантування якості повинні відповідати вищим цілям − задоволення вимог споживача до якості ПС. Контроль відповідності цілей процесу SQA вищим цілям якості ПС здійснює процес управління якістю.

Здійснення процесу SQA регламентується розробленим і документально оформленим, реалізованим і супроводжується (а при необхідності і актуалізуються) Планом виконання дій і рішення задач по процесу гарантування якості для ЖЦ проекту (скорочено, Планом якості від Software Quality Assurance Plan). План описує, яким чином організація-розробник гарантує забезпечення якості ПС. План повинен включати наступне:

· Стандарти, методології, процедури та інструменти для виконання дій з гарантування якості (або посилання на них в офіційній документації організації);

· Процедури для перевірки (огляду) договору та умови їх координації;

· Процедури для ідентифікації, збору, накопичення, супроводу і розміщення звітів про якість;

· Ресурси, план-графік робіт і відповідальності за проведення дій по гарантуванню якості;

· Опис окремих (обраних) дій і завдань з інших підтримуючих процесів, таких як Верифікація, Валідація, Спільний перегляд, Аудит і Управління рішенням проблем. Хоча багато аспектів якості ПС описуються в різних планах, наприклад, в Плані управління конфігурацією, Плані верифікації та валідації, Плані забезпечення безпеки функціонування ПС і ін., без єдиного Плану якості ці окремі плани можуть виявитися взаємно неузгодженими, а деякі аспекти якості ПС втраченими.

Масштаб, сфера застосування і зміст SQAP повинні відповідати розміру та складності програмної системи і ризику, який може виникнути в зв'язку з відмовами системи.

План може існувати у вигляді окремого документа або бути частиною плану забезпечення гарантії якості великої системи, що включає ПС в якості підсистеми. Він може посилатися на інші плани в проекті, наприклад, план управління ризиком.

Поряд з іншими робочими продуктами проекту План якості повинен знаходитися в сфері управління конфігурацією і піддаватися перевірці з боку керівництва проекту.

Лекція 6. Інструменти аналізу якості

Для ефективного виконання своїх функцій, група SQA повинна оцінити власні потреби в інструментах, які підтримують процеси вимірювання, забезпечення гарантії якості та управління якістю, а також верифікації та валідації.

Розшарування даних. Найпростіший з інструментів − таблиця розшарування даних. Призначена для класифікації даних за кількома критеріями. Приклад таблиці розшарування − форма збору даних про дефекти. Тут все безліч даних «розшаровується» виходячи з двох критеріїв − «тип дефекту» і «дата виявлення».

Для зіставлення даних по окремим верствам зручно використовувати діаграми: кругові, точкові, стовпчикові, стрічкові і інші.

Цей вид графічних інструментів підтримується засобами ведення електронних таблиць (наприклад, Excel). Для побудови діаграм використовується Майстер діаграм.

Діаграма виконання. Діаграма виконання (Run Chart) або тимчасової графік − зручний інструмент контролю ходу виконання процесу в масштабі часу і виявлення закономірностей в настанні будь-яких контрольованих подій (наприклад, зміни продуктивності праці в залежності від дня тижня або швидкості передачі інформації по каналах зв'язку в залежності від часу доби. По осі x на графіку − градація часу, а по осі y − градація події.

Діаграма розсіювання. Діаграма розсіювання (Scatter Diagram) використовується для перевірки гіпотез про наявність певного взаємозв'язку між двома вимірюваними величинами (наприклад, розміром модуля і щільністю дефектів, розміром ПС і витратами тощо). По осі x вказується шкала вимірювання для однієї величини (незалежної змінної), а по осі y − для іншої, залежної змінної.

Стовпчикова діаграма. На столбиковой діаграмі (Bar chart) послідовність значень представляється у вигляді стовпців (одному спостереженню відповідає один стовпець). Якщо вибрано кілька змінних, будується складова діаграма, де всі змінні показуються у вигляді груп стовпців (одна група для кожного спостереження), що дозволяє досліджувати розподіл даних по групам. Цей вид діаграм зручний, наприклад, для порівняння двох серій даних − виміряних і очікуваних.

Два спеціальних типу Столбикова діаграм − гістограми та діаграми Парето.

Гістограма. Гістограми (Histogram) створюються шляхом групування результатів вимірювань по секціях і подальшого підрахунку кількості «потрапляння» виміряних значень в кожну секцію. Секції представляють собою непересічні стовпці однакової ширини, в основі яких - рівні відрізки дійсної прямої, що покривають область можливих значень виміряних величин (безперервна шкала). Висота стовпчика пропорційна числу влучень значень в секцію.

Гістограми можуть використовуватися для характеристики спостережуваних значень практично будь-яких атрибутів продукту або процесу (наприклад, розміру модулів, часу усунення дефекту, часу між відмовами, кількості дефектів, знайдених при тестуванні тощо).

Діаграма Парето. Корисність застосування діаграм Парето (Парето Chart) в інженерії якості полягає в тому, що вони можуть використовуватися при аналізі процесів для залучення уваги розробників до тих факторів, які мають найбільший загальний вплив на систему, що розробляється, і відсіювання неістотних факторів, що дозволяє вчасно вибрати правильний напрямок роботи по поліпшенню процесів і продуктів. За допомогою цих діаграм можна, наприклад, знайти і графічно проілюструвати відповіді на такі питання, як «на попередженні яких помилок потрібно зосередити зусилля?» або «які дії в процесі вносять найбільший внесок в проблему?» та інші.

Контрольні карти. Контрольні карти (Control Chart) використовуються для аналізу стабільності процесів. Всі вони мають схожу структуру.

Карта містить центральну лінію (CL, Central Line), яка представляє середнє значення досліджуваної характеристики процесу, і дві лінії контрольних меж − верхній контрольний межа (UCL, Upper Control Limit) і нижній контрольний межа (LCL, Lower Control Limit), які обчислюються за однією з можливих заходів варіабельності процесу, наприклад, вибірковим середнім або діапазонами (розмахів) значень.

Карти для вибіркових середніх часто називають X-картами і будують для отримання відповіді на питання, «чи змінюється основна тенденція процесу в період спостереження?».

Причинно-наслідковий діаграма. Причинно-наслідковий діаграма (C & E, Cause and Effect diagram) використовується для того, щоб встановити безліч можливих причин появи однієї досліджуваної проблеми. Інші назви цього графічного інструменту − діаграма Ісікава, «риб'яча кістка» (fish-bone), діаграма аналізу

першопричин (root cause analysis). Найбільш ефективно застосовується в умовах колективної роботи над проблемою в режимі мозкового штурму.

Існує безліч інших графічних засобів, що застосовуються для представлення та аналізу інформації про процеси і програмних продуктах. Вони добре відомі і не потребують описі. Серед них:

- Діаграма "сутність-зв'язок» (entity-relations diagram);
- Діаграма потоків управління і даних (flow chart);
- Контрольний листок (check sheet) (приклад наведено в розділі 4);
- Матрична діаграма (приклад - сукупність матриць в методології QFD);
- Запитальник (checklist).

Лекція 7. Процес веріфікації та валідації. Види і методи перевірки. Вимоги до робочих продуктів. Життєвий цикл дефекту

Поряд з процесом забезпечення гарантії якості, в архітектурі процесів ЖЦ визначені ще чотири процесу підтримки, цілком спрямованих на підвищення якості ПС. Це процеси верифікації, Валідації, Спільного перегляду і аудиту, які призначені для перевірки відповідності робочих продуктів і процесів розробки своєму призначенню і підтвердження їх здатності забезпечити належну якість кінцевого програмного продукту.

Виходячи з положень стандарту ISO / IEC 12207 «призначення процесу верифікації полягає в підтвердженні того, що кожен робочий продукт ПС (Софтваре work product) і / або послуга процесу або проекту належним чином відображають встановлені вимоги. В результаті успішного здійснення процесу.

Призначення процесу валідації полягає в підтвердженні того, що вимоги, що стосуються конкретного застосування робочого продукту ПС за призначенням, задовольняються.

Методи, що застосовуються при виконанні процесів верифікації, валідації, спільного перегляду та аудиту, можна класифікувати по-різному, наприклад:

- Динамічні (пов'язані з виконанням программ8) і статичні (пов'язані з переглядом тексту);
- Пошукові (спрямовані на пошук помилок) і підтримують прийняття рішень (застосовуються для аналізу робочих продуктів з метою виявлення яких-небудь характерних особливостей, виконання оцінок і ін.);
- Для колективної роботи і для індивідуальної (аналітичної) роботи;
- Формальні (що вимагають чіткого дотримання методу) і неформальні;
- Колегіальні (виконувані колегами по роботі, рівними в правах, і не загрожують адміністративними наслідками) і ревізійні тощо.

Одні і ті ж методи можуть застосовуватися для виконання різних процесів перевірки, і, навпаки, при виконанні одного процесу перевірки може використовуватися кілька методів. Часто метод асоціюється з завданням процесу, для вирішення якої він застосовується, а сама задача має назву, однойменне

методу, наприклад, аналіз простежуваності, аналіз критичності і ін. Щоб уникнути плутанини з однойменними назвами методів, процесів і завдань далі при описі методів наводяться їх еквівалентні назви англійською мовою.

Для зручності викладу ми використовуємо класифікацію методів на колективні та індивідуальні аналітичні і попутно характеризуємо їх з точки зору інших класифікаційних ознак.

Аналітичні методи призначені для індивідуального дослідження робочих продуктів ПС з використанням (або без) інструментальних засобів підтримки. Вони можуть бути віднесені до категорії пошукових методів, методів підтримки прийняття рішень або мати подвійне призначення. Потрібно зауважити, що ряд аналітичних методів індивідуального застосування, які можуть використовуватися для цілей V & V, мають набагато більш широке поле докладання (наприклад, в процесах SQA, аналізу вимог, проектування, тестування або управління ризиком).

Колективні перевірки припускають участь в них декількох (по крайней мере, двох) осіб та прийняття узгодженого колективного рішення. Особи, які беруть участь в колективних перевірках, повинні, з одного боку, володіти методами індивідуальної аналітичної перевірки робочих продуктів, а з іншого − володіти навичками колективної роботи.

Колективні перевірки можуть проходити або у формі неформальних зібрань, або в формі формальних нарад, заздалегідь підготовлених і мають певну процедуру проведення. Формальні перевірки суворіше і вважаються більш ефективними, проте, їх складніше застосовувати на практиці. До того ж, в умовах браку знань процедур їх проведення і відсутності досвіду практичного застосування, формальні перевірки можуть не дати бажаної віддачі. Рішення про тип перевірки приймається керівництвом проекту на основі досвіду розробників і критичності проекту. Допускаються компромісні рішення про проведення напівформальних перевірок.

Для того щоб перевірки були ефективними, що пред'являються робочі продукти повинні відповідати певним вимогам.

При проведенні перевірок перевіряючі особи оперують терміном перевіряємий матеріал (ПМ) стосовно робочим продуктам будь-якого виду.

До структури ПМ ставляться вимоги (з якими можуть бути пов'язані показники якості ПМ).

Коректність. Кожне твердження ПМ має бути правильним.

Узгодженість. Кожен ПМ повинен містити узгоджені між собою твердження.

Повнота. Кожен ПМ повинен містити інформацію, необхідну для досягнення його цілей (цілей, для яких ПМ розроблявся). Кожен ПМ повинен відповідати вимогам до змісту, відбитим у всіх застосовуваних до нього стандартах.

Ясність. Сенс і призначення кожного твердження ПМ повинні бути зрозумілі.

Передбачає:

− визначеність. Кожне твердження ПМ має бути чітко ідентифікованим. Якщо це приклад − має використовуватися ключове слово «наприклад» тощо;

− несуперечливість. Кожне твердження ПМ повинна мати тільки один сенс (значення).

Лекція 8. Тестування програмних систем. Основні поняття та визначення. Види та рівні тестування

Тестування - невід'ємна складова процесу програмної інженерії, один з методів подальшого поліпшення якості розробленого програмного забезпечення системи за допомогою виявлення решти дефектів, не виявлених раніше всіма іншими видами перевірок.

Термін testing (тестування) широко використовується в літературі, але визначається по-різному.

Стандарт ANSI / IEEE Std. 610.12: 1990 визначає термін testing в самому його широкому сенсі як будь-яка дія з аналізу ПЗ (включаючи методи як динамічної, так і статичної перевірки).

Г. Майерс визначає цей термін в найвужчому сенсі: «тестування − процес виконання програми (або її частини) з метою виявлення помилок. Налагодження (debugging) − діагностування точної природи відомих помилок і їх виправлення».

Слово testing може бути переведено не тільки як тестування, а й як випробування (як це зроблено в стандартах ДСТУ 2844-94, ДСТУ 2853-94).

Однак, поняття «випробування» нам звичніше пов'язувати з завершальними стадіями ЖЦ, на яких виконується не пошук помилок, а підтвердження придатності розробленого програмного продукту.

У загальному випадку тестування розглядається як процес виконання програмної системи (або елементів ПС) з метою перевірки її відповідності встановленим вимогам та виявлення дефектів.

Рівні тестування програмного забезпечення:

− Unit Testing or Component Testing − модульне або компонентне тестування;

− Integration Testing − інтеграційне тестування;

− System Testing − системне тестування;

− Acceptance Testing − приймальне тестування.

Компонентне тестування перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо (модулі програми, об’єкти, функції тощо). Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити чи за підтримки середовищ розробки, таких як фреймовки (каркаси) для модульного тестування або інструменти для дебагу. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі багів (Bug Tracking System).

Один з найбільш ефективних підходів до компонентного (модульного) тестування – це підготовка автоматизованих тестів до початку основного

кодування ПЗ. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). При цьому підході створюються і інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір поки всі тести не будуть успішними.

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

- Компонентний інтеграційний рівень (Component Integration testing) перевіряє взаємодія між різними системами після проведення компонентного тестування;
- Системний інтеграційний рівень (System Integration testing) перевіряє взаємодію між різними системами після проведення системного тестування.

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

Існує два підходи до системного тестування:

- На базі вимог (requirements based) – для кожної вимоги пишуться тестові випадки (test cases), перевіряючі виконання даної вимоги.
- На базі випадків використання (use case based) – на основі уявлення про способи використання продукту створюються випадки використання системи (Use Cases). По конкретному випадку використання можна визначити один або більше сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які повинні бути протестовані.

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

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

Методи приймального тестування:

- Тестування замовником самостійно. Це ризиковано в тому плані що у замовника може не бути творчих ресурсів, а завантаження по поточним завданням може розтягти процес приймання;

– Тестування (Аудит) третьою стороною. Наймається спеціалізована компанія на тестуванні або підписується договір з конкурентом постачальника на надання послуг аудиту. Оптимально;

– Спільне тестування за сценаріями із замовником. Постачальник допомагає готувати пакет матеріалів для приймального тестування, готує команду замовника до методичного приймального тестування, контролює хід приймального тестування і терміни його виконання. Присутність інженера з тестування з боку виконавця допоможе краще зафіксувати розбіжності, зауваження та виявлені дефекти.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або реліз програми.

Незважаючи на те, що приймання знаходиться в кінці етапу (а в невеликих проектах і в кінці проекту) – готуватися до неї потрібно заздалегідь і перший прогін потрібно робити трохи раніше – щоб визначитися з повнотою і якістю робочого набору артефактів приймання, привчити до нього замовника, заздалегідь виявити можливі проблеми в приймальних тестах або в продукті. Всі види тестування програмного забезпечення, залежно від переслідуваних цілей, можна умовно розділити на наступні групи:

− Функціональні (Functional testing); − Нефункціональні (Non-functional testing);
− Пов’язані зі змінами (Regression testing).

Функціональні види тестування. Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів: Функціональне тестування (Functional testing), Тестування безпеки (Security and Access Control Testing), Тестування взаємодії (Interoperability Testing).

Нефункціональні види тестування. Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, «Як» система працює. Далі перераховані основні види нефункціональних тестів: тестування навантаження (Performance and Load Testing), стресове тестування
(Stress Testing), тестування стабільності або надійності (Stability / Reliability
Testing), об’ємне тестування (Volume Testing), Тестування установки (Installation testing), Тестування зручності користування (Usability Testing), Тестування на відмову і відновлення (Failover and Recovery Testing),
Конфігураційне тестування (Configuration Testing).

Тестування, пов’язане зі змінами. Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинне бути перетестоване для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки

програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту: Димове тестування (Smoke Testing), Регресійне тестування (Regression
Testing),Тестування збірки (Build Verification Test),Санітарне тестування або перевірка узгодженості / справності (Sanity Testing) Функціональне тестування (Functional testing). Тестування функціональності може проводиться у двох аспектах: – вимоги;

- бізнес-процеси.

Тестування в аспекті «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріоритезувати вимоги на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього пріоритезувати тестові сценарії (test cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал.

Тестування в сенсі «бізнес-процеси» використовує знання цих самих бізнес-процесів, які описують сценарії щоденного використання системи. У цьому випадку тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

Переваги функціонального тестування: імітує фактичне використання системи.

Недоліки функціонального тестування: можливість упущення логічних помилок у програмному забезпеченні, ймовірність надмірного тестування.

Досить поширеною є автоматизація функціонального тестування. Тестування безпеки (Security and Access Control Testing) – це стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту додатків, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Загальна стратегія безпеки ґрунтується на трьох основних принципах: конфіденційність, цілісність, доступність.

Конфіденційність – це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов користувач авторизований отримати доступ до даного ресурсу.

Цілісність – Існує два основних критерії при визначенні поняття цілісності:

− Довіра – очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів;

− Пошкодження і відновлення – у разі коли дані пошкоджуються або неправильно змінюються авторизованим або авторизованим користувачем, потрібновизначити наскільки важливою є процедура відновлення даних. Доступність являє собою вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об’єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності. Тестування взаємодії (Interoperability Testing) – це функціональне тестування, що перевіряє здатність

програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності
(compatibility testing) і інтеграційне тестування (integration testing).

Лекція 9. Методи тестування та побудови тестових наборів даних. Класифікація методів та їх огляд

Методи тестування розрізняються підходами до проектування тестів. Традиційно методи тестування поділяють на дві категорії − «чорний ящик» (без доступу до вихідного коду) і «білий» ящик (з доступом до вихідного коду), а також «сірий ящик», поєднуючий в собі два попередніх методи.

Також існує детальна класифікація методів тестування, заснована на підходах до проектування тестів.

1. Методи тестування, засновані на досвіді і інтуїції.

   Спеціалізоване тестування (Ad hoc). Це найбільш поширений (хоча і не зважає систематичним) підхід, при якому тести розробляються виходячи з інтуїції тестувальника і його досвіду в тестуванні подібних систем. Ефективність підходу повністю визначається майстерністю тестувальника. Підхід не вимагає докладної специфікації функцій ПО, але і не забезпечує оцінювання повноти тестового покриття. Розвитком підходу можна вважати «дослідне тестування» (exploratory testing), основні принципи якого − поєднання вивчення програмного продукту з проектуванням тестів і їх виконанням. Таке тестування зазвичай здійснюється незалежними тестувальниками стосовно завершеним програмним продуктам.
2. Методи, засновані на специфікації (або методи функціонального тестування).

   У традиційній класифікації їх відносять до методів «чорного ящика». Вони застосовуються для тестування зовнішніх і внутрішніх функцій програми і припускають наявність специфікації (формальної або не формальною), використовуваної в якості еталону. Відрізняються між собою підходами до вибору тестових даних з безлічі входів (вхідного простору) функцій.

   До основних методів функціонального тестування відносяться:

   - Таблиці рішень;
   - Функціональні діаграми;
   - Еквівалентну розбиття;
   - Аналіз граничних значень;
   - Розбиття вхідного простору на категорії;
   - Тестування переходів між станами;
   - Тестування з формальним специфікаціям.
3. Методи, засновані на аналізі коду (або структурні).

   У традиційній класифікації структурні методи відносять до методів «білого ящика». Згідно з цими методам структура програми представляється у вигляді спрямованого графа, в якому ідентифікуються потоки управління або потоки

даних. Відповідно, методи діляться на дві основні категорії: тестування потоку управління (шляхів) і тестування потоку даних.

У методах тестування потоку управління дані з вхідного простору вибираються таким чином, щоб забезпечити максимальне покриття коду. У методі тестування потоку даних тестові дані вибираються таким чином, щоб простежити шляхи кожної змінної в програмі від призначення значень до самого останнього використання (для всіх змінних). Цей метод вимагає великої кількості тестів, тому на практиці трасуються тільки найбільш критичні значення змінних. Особливий інтерес, наприклад, представляють значення змінних, що беруть участь в операціях поділу (необхідно запобігати можливості звернення подільника в 0), умови та цикли, виконання яких залежить від значень змінних. Метод тестування потоку даних може також застосовуватися для пошуку важко виявлених помилок часу виконання, пов'язаних з використанням пам'яті.

На закінчення короткого огляду структурних методів потрібно зазначити, що вони зазвичай застосовуються на рівні модульного тестування програми її розробником і чергуються з налагодженням, хоча можуть використовуватися і в процесах V&V для перевірки критичних програм.

Методи функціонального і структурного тестування повинні розглядатися не як альтернативні, а як взаємодоповнюючі, оскільки використовують різні джерела інформації для побудови тестів і призначені для виявлення різних типів помилок.

4. Методи спрямованого пошуку помилок.

Ці методи використовуються для проектування тестів, спеціально призначених для виявлення помилок певних типів. До основних методів цієї категорії відносяться:

− Припущення про помилки (error guessing); − Підсівши помилок (error seeding);
− Мутаційне тестування (mutation testing).

Згідно з методом висунення припущень про помилки на підставі «історичної» інформації про помилки, виявлені в подібних програмах, досвіду тестувальника, а також каталогів відомих помилок, складається список можливих помилок і помилкових ситуацій. Одним з «класичних» прикладів застосування методу є перевірка ділення на 0. У традиційній класифікації метод відноситься до категорії методів «чорного ящика».

Методи підсіву помилок і мутаційного тестування призначені для перевірки ретельності вже виконаного тестування. У традиційній класифікації вони відносяться до категорії методів «білого ящика».

У методі підсіву помилок в код, протестований на певному наборі тестів, спеціально вноситься невелика кількість помилок, а потім програма повторно тестується. Якщо при тестуванні виявляються не всі внесені помилки, набір тестів вважається недостатнім. Відношення кількості знайдених внесених помилок до загальної кількості внесених помилок передбачається приблизно рівним відношенню кількості знайдених реальних помилок до загальної кількості помилок, що містяться в програмі. Якщо виявлені всі внесені помилки, то

вважається, що, або набір тестів достатній, або, що ці помилки було занадто легко знайти.

При мутаційному тестуванні створюється багато копій основної програми, в кожну з яких вноситься невелика зміна, зване «мутацією», для імітації помилки в операторі або операнде, що не відбивається на синтаксичної коректності програми. Кожна копія тестується на одному і тому
ж наборі тестів. Якщо в процесі тестування всі внесені зміни виявлені, програма вважається протестованої адекватно і коректної (можуть бути також виявлені які раніше не виявлені помилки). Для того щоб метод був ефективним у виявленні помилок, потрібна велика кількість мутантів, що можливо тільки при автоматичній їх генерації.

5. Методи, засновані на аналізі очікуваного використання.

До цієї категорії віднесено два основні методи:

− Статистичне тестування;

− Тестування з операційного профілю.

При статистичному тестуванні вхідні дані для проектування тестів вибираються з вхідного простору відповідно до частоти їх появи в майбутніх сценаріях використання програми. При виконанні тестів фіксуються моменти відмов і обчислюються інтервали між відмовами МТБФ (Mean Time Between
Failure), зазвичай в одиницях процесорного часу (CPU) або часу виконання.
Отримана інформація використовується для оцінки надійності і передбачення моменту завершення тестування. Метод застосовується на рівні системного тестування. Для визначення частоти використання функцій програми, а також моментів відмов, в код вставляються команди-лічильники.

Тестування з операційного профілю застосовується в рамках методології інженерії надійності (SRE, від Software Reliability Engineering) і називається методом SRET (від Software Reliability Engineered Testing). Методологія SRE охоплює повний цикл розробки програмних систем з підвищеними вимогами до надійності, а тестування SRET (що є по суті статистичними) засновано на пріоритетності входів не тільки по частоті, але і з урахуванням критичності функцій, режимів і наслідків відмов систем при експлуатації.

До категорії методів, заснованих на аналізі очікуваного використання, можна віднести також метод тестування за сценаріями можливого використання, суть якого полягає в ідентифікації можливих сценаріїв роботи користувача і розробці по ним сценаріїв тестування (test-cases). Цей метод застосовується у всіх сучасних інструментах автоматизації функціонального тестування програм, що мають інтерфейс користувача (інструменти Capture- Replay). Він також використовується в популярній методології RUP (Rational Unify Process), згідно з якою сценарії тестування формуються на етапі аналізу і проектування по набору «бізнес-сценаріїв» (use-cases).

6. Методи, що враховують специфіку програмної системи.

Розглянуті вище методи універсальні і застосовні до будь-яких типів ПС. Однак вони не враховують характерних особливостей побудови систем різного типу, що вимагають застосування специфічних підходів до тестування.

Керуючись рекомендаціями SWEBOK, можна коротко охарактеризуємо такі методи:

- Тестування об'єктно-орієнтованих програм;
- Компонентне тестування;
- Тестування Web-додатків;
- Тестування графічного інтерфейсу користувача;
- Тестування протоколів на відповідність;
- Тестування систем реального часу;
- Тестування критичних систем.

Особливості типів ПС обумовлюють не тільки застосування спеціальних методів проектування тестів, скільки вибір методів і видів тестування, найбільш придатних для певних об'єктів тестування.

Лекція 10. Аналіз результатів тестування. Система відстеження дефектів. Вимірювання результатів. Критерії завершення тестування

На ефективність тестування і усунення дефектів дуже впливає ступінь чіткості процесу підготовки та аналізу звітів про проблеми, а також наявність хорошого інструменту для його підтримки. При відсутності промислового інструменту, група тестування може створити свою систему відстеження проблем, наприклад, за допомогою MS Access.

Цілі системи відстеження проблем:

− відстеження стану тестування і усунення дефектів;

− організація взаємодії між співробітниками і вирішення спірних питань щодо класифікації та пріоритетів усунення дефектів;

− визначення причин дефектів і виявлення «вузьких місць» в процесах розробки і тестування.

Зі звітами про проблеми в процесі тестування працюють тестувальники, розробники, керівник проекту і група якості. Тому для ефективного вирішення проблеми важливо визначити повноваження користувачів системи відстеження проблем (як виконавців процесу «Вирішення проблем»).

Процес рішення проблем, виявлених при тестуванні, може бути представлений наступною послідовністю кроків.

Крок 1. Відкриття проблеми. При виявленні проблеми тестувальник складає звіт про проблему («відкриває проблему») і передає його програмісту для аналізу. При автоматизованому веденні звітів тестувальник поміщає звіт в базу даних.

Крок 2. Вивчення. Програміст аналізує звіт і приймає рішення (згоден чи ні). Якщо проблема викликана випадковим відмовою, пов'язаних, наприклад, із збоєм устаткування, середовищем тестування, порушеннями технології тестування або

нерозумінням тестувальника − вона відхиляється. Якщо встановлено, що проблема пов'язана з дефектом в ПО, програміст встановлює пріоритет усунення дефекту. У спірних випадках рішення про пріоритет усунення приймає керівник проекту.

Крок 3. Дія. Програміст виконує усунення дефекту і повторне автономне тестування і повертає звіт з позначкою «Виправлена» тестувальників. При автоматизованому веденні звітів тестувальник знаходить звіти з цією відміткою в базі даних.

Крок 4. Закриття. Тестувальник виконує перевірку виправлень і закриває проблему. Закрити звіт про проблему може тільки тестувальник. При автоматизованому веденні звітів про проблеми вони зберігаються в базі даних для подальшого аналізу, формування звітів, а також для накопичення історичних даних. Звіти про відхилені дефектах можуть вилучатися з бази даних (але тільки тестувальником).

Кроки 1 − 4 стосуються відстеження кожного окремого дефекту, але для аналізу результатів і управління процесом тестування використовуються узагальнені і класифіковані дані про дефекти. Цей аналіз і узагальнення виконується менеджером групи тестування і менеджером проекту. Узагальнені дані використовуються також групою якості для аналізу поточного стану проекту.

Кількісний вимір результатів тестування грунтується на застосуванні метрик оцінювання поточного стану об'єктів тестування (метрики продукту) і процесу тестування (оцінка виконаних тестів).

Лекція 11. Автоматизація тестування

Автоматизоване тестування ПО (Automated Testing) − це процес верифікації програмного забезпечення, при якому основні функції та кроки тесту, такі як запуск, ініціалізація, виконання, аналіз і видача результату, виконуються автоматично за допомогою інструментів для автоматизованого тестування.

До основних переваг автоматизованого тестування можна додати: повторюваність − всі написані тести завжди будуть виконуватися одноманітно таким чином виключений людський фактор, тестувальник програмного забезпечення не пропустить тест, з необережності і не наплутає в результатах; швидке виконання − автоматизованому скрипту не потрібно звірятися з інструкціями та документаціями, це сильно економить час виконання; менші витрати на підтримку − коли автоматичні скрипти вже написані, на їх підтримку і аналіз результатів потрібно, як правило, менший час ніж на проведення того ж обсягу тестування вручну; звіти - автоматично розсилаються і зберігаються звіти про результати тестування; виконання без втручання − під час виконання тестів інженер-тестувальник може займатися іншими корисними справами, або тести можуть виконуватися в неробочий час.

До недоліків автоматизованого тестування можна віднести: повторюваність – ми відносили цю рису до переваг але це є одночасно і недоліком, так як тестувальник,

виконуючи тест вручну, може звернути увагу на деякі деталі і, провівши кілька додаткових операцій, знайти дефект, скрипт цього зробити не може; великі витрати на розробку − розробка автоматизованих тестів це складний процес, так як фактично йде розробка програми, яка тестує інший додаток. У складних автоматизованих тестах також є фреймворки, утиліти, бібліотеки та інше. Природно, все це потрібно тестувати і налагоджувати, а це вимагає часу; вартість інструменту для автоматизації − в разі якщо використовується ліцензійне ПО, його вартість може бути досить висока. вільно розповсюджувані інструменти як правило відрізняються більш скромним функціоналом і меншим зручністю роботи; пропуск дрібних помилок − автоматичний скрипт може пропускати дрібні помилки наперевірку яких він не запрограмований. Це можуть бути неточності в позиціонуванні вікон, помилки контролів і форм Для того щоб прийняти рішення про доцільність автоматизації додатки потрібно відповісти на питання «переважують чи в нашому випадку переваги?» − хоча б для деякої функціональності нашого застосування. При прийнятті рішення варто пам'ятати, що альтернатива − це ручне тестування, у якого є свої недоліки.

Процес автоматизації тестування ділиться на три етапи:

− Запис. Сценарій тестування може включати точки верифікації (verification points) для перевірки відповіді системи і зробити сценарії тестування залежними від даних, щоб виконувати один і той же сценарій з різними наборами вхідних даних;

− Поліпшення шляхом додавання коду з різноманітними функціями Типові зміни сценаріїв тестування – умовне галуження, рефакторинг і обробка виняткових ситуацій;

− Відтворення. Виконання сценаріїв, що емалюють дії, які виконував користувач ПС при записі тесту. Розбіжності реєструються, і тестірувальник може зробити висновок про те, чи добре функціонує система або регресійне тестування виявило проблеми.

Інструментальні засоби автоматизації записують взаємодію користувачів з ПС, а сформовані на цій основі сценарії використовуються для подальших тестів, тобто автоматизація тестування дозволяє оптимізувати якість складних ПС ефективним по вартості способом за прийнятний час. Це допомагає швидше випустити програмне забезпечення більш високої якості.

Лекція 12. Інструменти засоби тестування ПЗ.

При сучасних темпах зростання складності ПС, їх тестування має підтримуватися автоматизованими інструментами. Ринок інструментів тестування бурхливо розвивається. На зміну аналізатора коду і генераторів тестових даних для структурного і функціонального тестування, які працюють на певних програмних і апаратних платформах, а також інструментів, які підтримують окремі методи тестування, приходять промислові інтегровані інструменти підтримки різних

методів тестування, які працюють практично на всіх апаратних і програмних платформах.

Єдиних підходів до класифікації інструментів тестування немає, оскільки переважають інтегровані інструменти.

Інструменти управління тестуванням.

Менеджери конфігурації − відстежують і контролюють внесення змін до ПС на протязі її розробки і супроводу і підтримують цілісність розроблених і поставляються версій ПС. Основні функціональні характеристики цього типу інструментів: відстеження дефектів, управління запитами на внесення змін, відстеження змін в коді.

Менеджери проекту − підтримують функції планування і відстеження на етапах розробки і супроводу системи (оцінювання термінів, ресурсів, побудова мережевих графіків тощо). Інструменти даного типу можна застосовувати і для підтримки планування і відстеження процесу тестування, наприклад, для розподілу списку завдань за виконавцями і відстеження їх виконання.

Інструменти, що підтримують аналіз вимог і проекту.

Аналізатори планів, вимог і проектів − оцінюють специфікації на повноту, несуперечність і відповідність встановленим стандартам для специфікацій.

Будівники прототипів / емулятори системи − об'єднують дії з тестування з діями з аналізу та проектування і дозволяють швидко промоделювати вимоги і проектні рішення.

Інструменти трасування вимог − дозволяють встановити зв'язки вимог з проектом, кодом і тестами.

Планувальники тестів − підтримують процес планування тестування на всіх його рівнях.

Інструменти підтримки тестування на етапі реалізації і супроводу.

Статичні аналізатори вихідного коду. Досліджують вихідний код без його реального виконання. Можуть застосовуватися в якості допоміжних інструментів при плануванні та тестуванні, але в більшій мірі − це інструменти вимірювання та оцінювання якості коду.

Аудитори − аналізують код на відповідність стандартам кодування і встановленим правилам.

Вимірювачі складності − обчислюють метрики коду для визначення атрибутів складності шляхом оцінювання таких характеристик коду, як потік управління, операнди / оператори, дані, структура системи.

Інструменти генерації перехресних посилань - забезпечують посилання між різними сутностями (змінними).

Вимірювачі розміру − обчислюють число рядків вихідного коду (SLOC).

Інструменти підготовки тестів. Включають групу інструментів, підтримуючих підготовку тестових даних або розробку тестів.

Екстрактори даних − будують тести, витягуючи інформацію з існуючих баз даних або тестових наборів.

Генератори тестів по специфікації вимог − будують тести за вимогами, написаним на формальній мові специфікації.

Генератори тестових даних (за різними методами) − генерують вхідні дані для тестування відповідно до методу, реалізованим в інструменті.

Деякі генератори створюють дані з використанням датчиків випадкових чисел (для статистичного тестування).

Планувальники тестів − підтримують розробку та ведення планів тестування.

Інструменти виконання тестів. Найбільш представницька група інструментів.

Динамічні аналізатори покриття − оцінюють покриття коду тестами по відношенню до операторів, шляхах або модулів. Засновані на встановленні в код спеціальних операторів для відстеження шляхів виконання.

Інструменти Захоплення / програвання − автоматично записують дії тестувальника при виконанні програми у вигляді автоматичної тестової процедури (скрипта захоплення) і потім відтворюють (програють) ці скрипти.

Відладчики – безпосередньо підтримують автономне тестування, хоча їх призначення − пошук і усунення помилок (налагодження). Деякі отладчики мають можливості аналізу покриття.

Емулятори − можуть застосовуватися замість відсутніх компонентів системи (зазвичай апаратних, наприклад емулятори терміналів).

Аналізатори мережі − спеціальний клас інструментів тестування, призначених для аналізу трафіку мережі. Дозволяють моделювати роботу багатьох терміналів.

Аналізатори продуктивності − відстежують тимчасові характеристики системи або її компонентів.

Аналізатори помилок періоду виконання − відстежують роботу програми на наявність помилок, пов'язаних із замовленням-звільненням пам'яті, нульовими покажчиками тощо. Автоматично відстежують використання стеків і черг.

Інструменти управління виконанням тестів − загальне найменування категорії інструментів, які автоматизують різні функції настройки та виконання тестів, очищення системи після виконання тестів. До цієї категорії відносять також тестові драйвери і заглушки.

Тестові оцінювачі. До цього класу відносять інструменти, які виконують функції виявлення схильних до помилок компонентів.

Компаратори (утиліти порівняння файлів) − порівнюють результати тестування і виявляють відхилення (наприклад, порівнюють бітові образи екранів, файли). Інструменти Захоплення / програвання зазвичай мають у своєму складі такі компоненти.

Конвертори і аналізатори даних − конвертують дані в зручний для інтерпретації формат і виконують різні види статистичного аналізу цих даних.

Трасувальники дефектів / змін − зберігають інформацію про дефекти і генерують звіти. Зазвичай входять до складу систем управління конфігурацією і інструментів управління тестуванням.

ЛІТЕРАТУРА

1. Myers G.J. The Art Of Software Testing [Text] / G.J. Myers – New York: John Wiley & Sons, Inc., 2004. – 254 p. – ISBN 0- 471-46912-2.
2. Hutcheson M. L. Software Testing Fundamentals: Methods and Metrics. – John Wiley & Sons. – 2003. – 408 p.
3. Куликов С. С. Тестирование программного обеспечения. Базовый курс : практ. пособие. / С. С. Куликов. – Минск: Четыре четверти, 2015. – 294 с.
4. Kan H.N. Metrics and Models in Software Quality Engineering. – Second Edition, 2002. – 560 p.
5. Pandian, C. R. Software Metrics: A Guide to Planning, Analysis, and Application. – Auerbach Publications, 2004. – 312 p.
6. Schulmeyer, G. G. Verification and Validation of Modern SoftwareIntensive Systems / G. G. Schulmeyer, G. R. Mackenzie. – NJ: Prentice Hall, 2000. – 493 p.
7. Калбертсон Быстрое тестирование. – М.: Вильнюс, 2002. - 383с.
8. Катрани Т. Rational Rose 2000 и UML. Визуальное моделирование - ДМК Пресс, 2001.
9. Фаулер М., Скотт К. UML. Основы: Краткое руководство по унифицированному языку моделирования. 2-е издание - Символ – Плюс, 2002.
10. Ройс У. Управление проектами по созданию програмного обеспечения: Унифицированный подход - «Лори», 2002.
11. Брукшир Дж. Гленн. Введение в компьютерные науки. Общий обзор, 6-е издание.: Пер. с англ. – М. «Вильямс», 2001.
12. Г. Буч, Дж. Рамбо, А. Джекобсон. UML. Руководство пользователя. – М.:2005. – 257 с.
13. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль, Т.М. Коротун, Е.М.Лаврищева, В.Ю. Суслов - К.: Академпериодика-2007.-678 с.
14. Capability Maturity Model for Software, Version 1.1 / M.Paulk, B.Curtis et al. // CMU-SEI-TR-024, Soft. Engin. Institute, Pittsburg PA 15213, Feb. Pittsburg, 1993.-82 p.
15. Соммервил И. Инженерия программного обеспечения. 6-е издание. - М.-Спб.-Киев,- 2002. - 623 с.
16. Bevan, N. Specifying and Measuring Quality in Use. – Proceedings of the 22nd international conference on Software engineering, 2000. – PP. 322–331.
17. Halpern M. Addressing Quality Risk in Collaborative Automotive Design: Commentary. – Gartner Group, 2001. – P. 262–293.

18. IEEE Standard for a Software Quality Metrics Methodology Copyright ©, Institute of Electrical and Electronics Engineers, Inc. 2005.

ГЛОСАРІЙ ФАХОВИХ ТЕРМІНІВ

Аналіз вимог - виявлення і відображення обмежень на функції і систему у цілому.
Архітектура системи - структура (каркас) системи і підсистем або компонентів і інтерфейсів між ними.
Білої скриньки метод - забезпечення виявлення помилок на всіх зазначених її шляхах і потоках передач керування у внутрішній структурі програми. **Валідація** - перевірка відповідності розробленої ПС вимогам замовника. **Верифікація** - перевірка правильності реалізації функцій ПС з урахуванням вимог.
Взаємодія об'єктів - зв'язок між об'єктами через механізми повідомлень.
Відмовлення - перехід програми з працюючого стану в непрацюючий стан у зв'язку з виявленими помилками або дефектами в ній.
Вимога - угода або договір між замовником і виконавцем системи щодо властивостей її функцій, умов роботи в заданому середовищі.
Водоспадна (каскадна) модель - схема послідовності робіт, у якій кожна з них виконується один раз і в порядку, що зазначений у моделі життєвого циклу.
Гарантія якості програмного забезпечення - дії на кожнім етапі життєвого циклу з перевірки і підтвердження досягнення деяких показників якості відповідно до стандарту.
Дефект - це помилкова подія в роботі системи, що виникає внаслідок невірного опису специфікації вимог, проектних рішень, програм і т.п.
Діаграма - графічне подання сценаріїв роботи системи за допомогою класів, станів, подій і т.п.
Динамічне тестування - виконання програми для виявлення помилок у програмі, визначення їхньої причини й усунення.
Домен або предметна область - спектр задач, що подаються програмами і допускають схожі прийоми їхнього рішення.
Експлуатація - дії з виконання готової програмної системи.
Життєвий цикл - схема виконання робіт із проектування системи, починаючи з моменту прийняття рішення про необхідність її побудови і закінчуючи моментом її повного вилучення з експлуатації.
Задача системи - спосіб (технологія) досягнення мети системи з конкретними числовими характеристиками.
Менеджмент - професійне керування програмним проектом і колективом фахівців, утворюючих програмний продукт проекту.
Інженерія - планування і дисципліна керування програмуванням задач з метою одержання користі від властивостей та способів виконання продуктів.
Інженерія якості - процес керування наданням програмному забезпеченню властивостей якості (надійності, відмовостойкісті і т.п.).

Інженерія вимог - збирання, аналіз, оформлення умов і обмежень на розробку системи, погоджених як замовником, так і виконавцем.
Інтенсивність відмовлень - це частота появи відмовлень або дефектів у програмній системі при її тестуванні або експлуатації.
Інспекція коду - формальна перевірка опису програми, її типів і структур даних на їхню правильність відповідно до вимог.
Інформаційна модель - модель системи, у якій відображається структура даних і зв'язків з об'єктами, що їй користуються.
Інформаційна система - система, що виконує збирання, обробку, збереження і виробництво інформації з використанням автоматизованих процесорів і людей.
Інформаційне забезпечення - набір засобів для надання інформації користувачам про зміст і умови її застосування.
Каркас (фрейм) - різновид абстрактної архітектури для визначення виділених компонентів в системі.
Компонент - тип, клас, проектне рішення, програма, документація або інший продукт програмної інженерії, пристосований для практичного використання.
Компонентна розробка - конструювання програмної системи шляхом композиції знов створених і готових компонентів, що зберігаються в різних сховищах.
Кінцеві користувачі системи - професійні особи, що замовляють комп'ютерну систему і користуються нею.
Компонент повторного використання (КЦВ) - фрагмент знань про минулий досвід розроблення елементів системи, представлених так, що їх можливо використовувати не тільки розробником, а і користувачами після адаптації до нового середовища.
Конфігурація - варіант (версія) виготовленої програмної системи з ідентифікованих компонентів і підсистем.
Концептуальне моделювання - процес побудови моделі предметної області, орієнтованої на розуміння її людиною.
Критерій - кількісна або якісна характеристика системи, що дозволяє оцінити ступінь досягнення мети і сформулювати правила вибору необхідних засобів і технологій.
Критерій ефективності - критерій, що дозволяє оцінити ступінь досягнення мети з урахуванням зроблених витрат різних ресурсів.
Керування якістю - комплекс способів і системної діяльності з планування досягнення показників якості робочого і кінцевого програмного продукту.
Метрика - кількісна міра і шкала виміру характеристик програми або продукту.
Модель ЖЦ - типова схема послідовності робіт у процесах розроблення деякого типу програмного продукту.
Модель процесу - визначена послідовність дій, що супроводжує зміну стану програмного об'єкта.
Модель якості - чотирирівнева структура, що відображає характеристики, атрибути, метрики й оцінні елементи показників якості програмної системи. **Мова**

UML - діаграмний спосіб специфікації, візуалізації, "конструювання і документування продуктів у процесах ЖЦ.

Надійність програмної системи - це здатність системи зберігати свої властивості (безвідмовність, стійкість і ін.) у процесі перетворення вихідних даних у результати протягом визначеного проміжку часу за певних умов експлуатації.

Налагодження - перевірка програми на наявність у її описі помилок і їхнє усунення без внесення нових.

Нефункціональні вимоги - вимоги, що характеризують організаційні, виконавські, операційні аспекти роботи програмної системи в середовищі виконання.

Об'єкти керування - це функції перетворення об'єктів інтерфейсу в об'єкти сутності, аналогічно до відображення алгоритму обробки даних у системі.

Об'єктно-оріснтована модель - структура із сукупності об'єктів, що взаємодіють між собою, мають властивості і поведінку.

Оцінювання якості - дії, спрямовані на визначення ступеня задоволення програмного забезпечення якісним вимогам, що відповідають його призначенню.

Пакет - програмна структура з загальним механізмом організації елементів у групи підсистем різного рівня деталізації.

Переносність системи - можливість змінювати сервіс системи (ОС, зв'язки, мережні комунікації і т.п.) шляхом настроювання на нові умови середовища або платформи.

План тестування - опис стратегії, ресурсів і графіка тестування окремих компонентів і системи в цілому.

Помилка - недоліки в операторах програми або в технологічному процесі її розробки, що приводять до неправильної інтерпретації вхідної інформації і до неправильного рішення.

Програмна система - це комплекс інтегрованих прикладних програм і засобів, що реалізують функції предметної області в заданому середовищі.

Програмне забезпечення - це деяка конкретна функція системи (ОС, система керування БД тощо), що входить до складу ПС або сама ПС.

Принципи - базові концепції, що лежать в основі всієї області програмування.

Програмна інженерія - система методів, засобів і дисципліни планування, розробки, експлуатації і супроводу програмного забезпечення, здатного до масового відтворення.

Процес розробки - дії розробника з інженерії вимог, проектування, кодування і тестування програмного продукту.

Процес здачі - дії з передачі розробленого ІПП покупцеві.

Процес супроводу - дії за рішенням задач системи, підтримка системи в акгуальному стані для виконання функцій системи, керуванню модифікаціями або вилученню системи з уживання.

Проектування - перетворення вимог у послідовність проектних рішень і в архітектуру системи.

Реалізація програмної системи - перетворення проектних рішень у працюючу систему (синоніми: кодування, конструювання).
Сертифікація програмного продукту - процес для установлення відповідності програмної продукції (процесу або послуг) конкретному стандартові або технічним умовам зі спеціальним знаком або свідченням.
Специфікація - опис алгоритму, правил, обмежень дій об'єктів з урахуванням стандартів, критеріїв якості тощо.
Статичне тестування - аналіз і розгляд специфікацій компонентів на правильність подання без їхнього виконання на комп'ютері.
Супровід виконання реалізованих у системі задач і робот з внесення в неї змін після того, як вона передана користувачам для експлуатації.
Суб'єкт - хтось або щось поза системою, що взаємодіє із розробленою системою.
Тест - деяка програма, призначена для перевірки правильності роботи системи і виявлення в ній помилкових ситуацій.
Тестові дані - набір даних, що готуються на основі специфікації програм для перевірки роботи програмної системи.
Тестування - спосіб семантичного налагодження (перевірки) програми, що складається у виконанні послідовності різних наборів тестів і звірення отриманих результатів з відомими заздалегідь.
Функція - зміст дій, виконання яких покладається на відповідний елемент системи при заданих вимогах, умовах і обмеженнях.
Функціональні вимоги - це умови й обмеження на виконання мети і задач відповідно до заданих потреб замовника системи.
Характеристики якості (стандартні) - це функціональність (functionality), надійність (realibility), зручність (usability), ефективність (efficiency), супроводженість (maitainnability), переносність (portability).
Чорної скриньки метод - тестування реалізованих функцій шляхом перевірки відповідності реального поводження функцій з очікуваними результатами, виходячи зі специфікацій вимог.
Якість програмного забезпечення - сукупність властивостей продукту, що визначають його придатність задовольнити вимоги замовника щодо призначенню.