QA SUMMARY

Version 1.2

by Ochilnik

08.2022 – 02.2023

CONTENT

QA (Quality Assurance)

Quality Assurance забезпечує правильність та передбачуваність процесу певним стандартам якості ще до створення самого об'єкта.
Quality Control передбачає контроль дотримання вимог.
Тестування же, в свою чергу, забезпечує збір статистичних даних і внесення їх в документи, маючи на увазі, що тестований об'єкт вже існує.

QA (Quality Assurance) – Забезпечення якості продукту

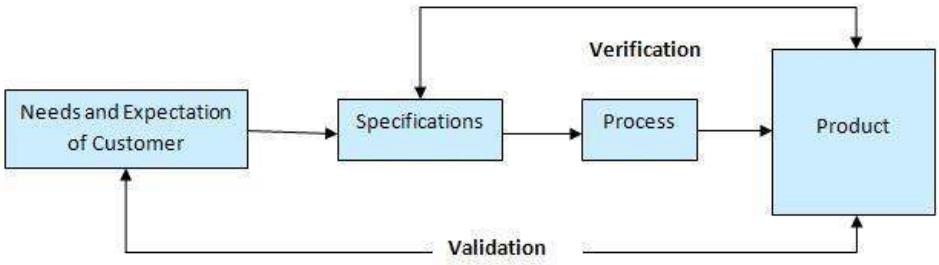
- перевірка технічних характеристик і вимог до ПЗ;
- оцінка ризиків;
- планування завдань для поліпшення якості продукції;
- підготовка документації, тестового оточення і даних;
- тестування;
- аналіз результатів тестування, а також складання звітів та інших документів.

QC (Quality Control) – Контроль якості продукту

- перевірка готовності ПЗ до релізу;
- перевірка відповідності вимог і якості даного проекту.

TESTING - Тестування

- планування;
- підготовка та оцінка програмного продукту і пов'язаних з цим результатів робіт;
- показати, що програмний продукт підходять для заявлених цілей;
- визначення дефектів.



VALIDATION: Підтвердження шляхом перевірки та надання об'єктивних доказів того, що вимоги щодо конкретного цільового використання чи застосування були виконані.

VERIFICATION: Підтвердження шляхом експертизи та надання об'єктивних доказів того, що визначені вимоги були виконані.

Характеристики хорошого тестування:

- Для кожної діяльності по розробці існує відповідна діяльність з тестування
- Кожен рівень тесту має мету тестування, характерні для цього рівня
- Аналіз і розробка тестів для даного рівня тестування повинні починатися під час відповідної діяльності з розробки.
- Тестувальники повинні брати участь в розгляді документів та коду, як тільки проекти стануть доступні в життєвому циклі розробки.

QUALITY

QUALITY (Якість) - ступінь, в якому компонент, система або процес відповідає визначеним вимогам та / або потребам та сподіванням споживача.

Характеристики якості ПЗ

- **RELIABILITY (Надійність)** - Здатність ПЗ підтримувати визначену працездатність у заданих умовах.
- **FUNCTIONALITY (Функціональність)** - Здатність ПЗ в певних умовах вирішувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно вирішує.
- **EFFICIENCY (Продуктивність)** – Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділеного для цього ресурсам. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до затрачуваних на це ресурсів усіх типів.
- **USABILITY (Зручність використання)** або практичність - Здатність ПЗ бути зручним у навчанні та використанні, а також привабливим для користувачів.
- **MAINTAINABILITY (Зручність супроводу)** - Зручність проведення всіх видів діяльності, пов'язаних із супроводом програм.
- **PORTABILITY (Переносимість) -** Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні й програмні аспекти оточення.

Способи вимірювання якості системи:

- Швидкість виявлення дефектів;
- Кількість відомих дефектів;
- Обсяг покриття тесту;
- Відсоток пройдених тестів.

DEFECT (BUG)

DEFECT (BUG) - це відхилення фактичного результату (actual result) від очікуваного результату (expected result).

INCIDENT (Інцидент) - будь-яка подія, знайдена в рамках тестування (або використання), яке потребує дослідження.

IMPROVEMENT / FEATURE / TASK / REQUEST / ISSUE (**Запит на зміну**) - це будь-яка пропозиція про вдосконалення продукту або його окремих властивостей.

ДЕФЕКТИ ВИНИКАЮТЬ ЧЕРЕЗ:
- Людський фактор;
- Брак часу;
- Складний код;
- Складність інфраструктури;
- Зміну технологій;
- Багато системних взаємодій.

ПРИЧИНИ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ:
- **MISTAKES OR ERRORS** - Помилки в коді. Людський фактор;
- **FAULTS, BUGS AND DEFECTS** - Несправності, помилки чи дефекти - недоліки в компоненті чи системі;
- **FAILURES** – Збій, це відхилення компонента або системи від його очікуваної дії.

ТРИ УМОВИ ЯКІ ЗАСВІДЧУЮТЬ, ЩО БАГ ІСНУЄ В ПРОГРАМІ
1. Відомий фактичний результат; (він же **ACTUAL RESULT**)
2. Відомий очікуваний результат;(він же **EXPECTED RESULT**)
3. Відомо, що результат з пункту 1 не дорівнює результату з пункту 2.

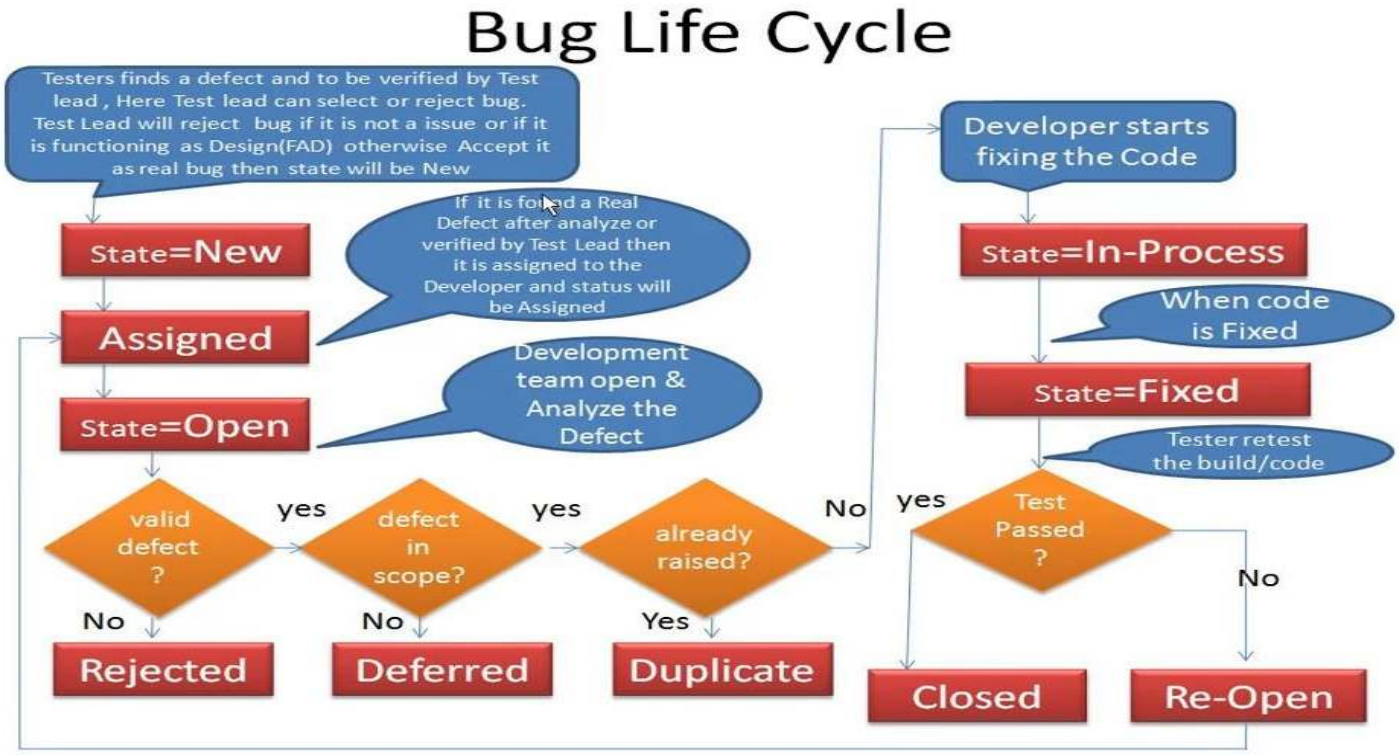
EXPECTED RESULT (Очікуваний результат) отримуємо зі Специфікації, спілкування, встановлених стандартів, статистичних даних, життєвого досвіду, здорового глузду, авторитетної думки та ін.

SPECIFICATION (Специфікація) – це детальний опис того, як має працювати Програмне Забезпечення. У більшості випадків баг – це відхилення від специфікації.

DEFECT LIFECYCLE

DEFECT LIFECYCLE (Життєвий цикл дефекту) - це послідовність етапів, які проходить дефект на своєму шляху з моменту його створення до остаточного закриття.

Назви етапів та життєвий цикл дефектів пов'язані з вибраною системою баг-трекінгу.



Основні етапи життєвого циклу дефекту

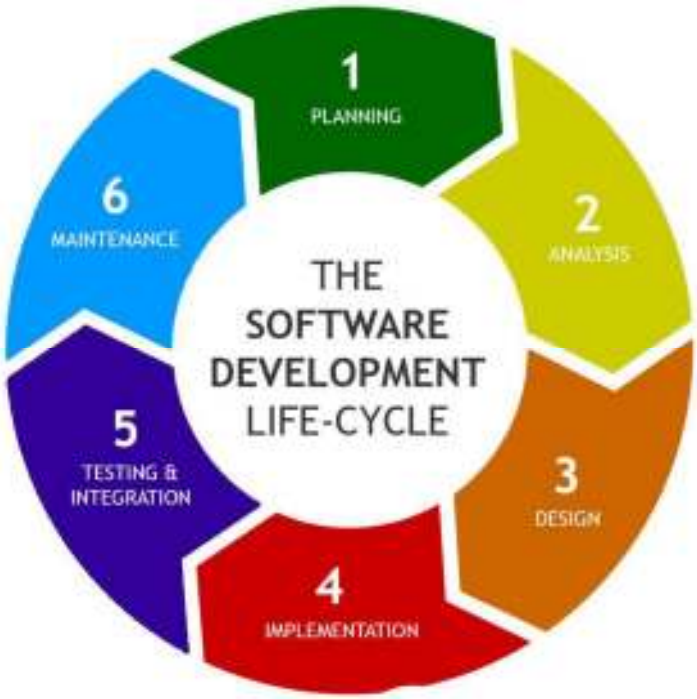
- **SUBMITTED (Виявлено)** – тестувальник знайшов баг.
- **NEW (Новий)** – дефект успішно занесений в систему.
- **DECLINED (Відхилено).** З різних причин дефект може і не вважатися дефектом або вважатися неактуальним дефектом, що змушує відхилити його.
- **DEFERRED (Відкладений).** Виправлення цього бага не несе цінності на даному етапі розробки або з інших причин, відстрочує час його виправлення.
- **OPENED (Відкрито).** Відповідальна особа визнала дефект дефектом, при чому таким, який потрібно виправити.
- **ASSIGNED (Призначено).** Виправлення поточного бага покладено на плечі певного розробника.
- **FIXED (Виправлено).** Відповідальний за виправлення бага розробник заявляє, що усунув дефект.
- **VERIFIED (Перевіреним).** Тестувальник перевіряє, чи дійсно відповідальний розробник виправив дефект, чи все-таки розробник безвідповідальний. Якщо бага більше немає, він отримує даний статус.
- **REOPENED (Повторно відкритим).** Якщо побоювання тестувальника виправдані і баг в новому білді не виправлений – він все так само потребує виправлення, тому заново відкривається.
- **CLOSED (Закритий).** У результаті певної кількості циклів баг все-таки остаточно усунений і більше не потребує уваги команди – він оголошується закритим.

SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

SYSTEMS DEVELOPMENT LIFE CYCLE (Життєвий цикл програмного забезпечення) – це сукупність окремих етапів робіт, що проводяться у заданому порядку протягом періоду часу. Використовується індустрією програмного забезпечення для проектування, розробки і тестування високоякісного програмного забезпечення.

PLANNING (Планування системи):

- точно вирішується що саме потрібно зробити, розробити, які потреби вирішити;
- визначаються проблеми, цілі і ресурси (такі, як персонал і витрати);
- вивчаються можливості альтернативних рішень шляхом зустрічей з клієнтами, постачальниками, консультантами та співробітниками;
- вивчається, як зробити продукт краще, ніж у конкурентів.



ANALYSIS (Аналіз системи):

- визначення документування вимог замовника;
- техніко-економічне обґрунтування;
- узгодження з замовником бачення кінцевого продукту і його функцій.

DESIGN (Дизайн системи):

- аналіз обсягу робіт;
- визначення і опис того, як система буде виглядати і як функціонувати;
- визначення елементів та компонентів системи;
- визначення рівня безпеки, архітектури, інтерфейсів і типів даних, якими оперує система.

IMPLEMENTATION (Розробка, впровадження і розгортання):

- власне процес розробки системи де пишеться код;
- фаза впровадження включає в себе конфігурацію і налаштування під певні вимоги і функції;
- тренінг кінцевого користувача.

TESTING & INTEGRATION (Дослідна експлуатація та інтеграція):

- об'єднання різних компонентів і підсистем в єдину цілісну систему;
- подаються різні вхідні дані і проводиться аналіз виходу, поведінки і функціонування;
- відновлення після збійних тестів;
- перевірка результатів;
- дослідження несподіваних результатів;
- запис помилок.

MAINTENANCE (Підтримка системи):

- періодична технічна підтримка системи;
- заміна старого обладнання;
- оцінка продуктивності;
- апдейти певних компонентів, щоб система відповідала потрібним стандартам і новітнім технологіям.

TESTING

ТЕСТУВАННЯ ПЗ - процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності.

Тестування - це планова і упорядкована діяльність.

ЦІЛІ ТЕСТУВАННЯ

- Виявлення дефектів (Перевірка на відповідність вимогам);
- Підвищення впевненості щодо рівня якості;
- Надання інформації для прийняття рішень;
- Запобігання дефектів.

СТРАТЕГІЯ В ТЕСТУВАННІ:

- Підвищити ймовірність виявлення багів (вибрати ті модулі та функції, ймовірність появи багів в яких велика);
- Створення тестів, які можуть бути використані повторно;
- Ефективно використовувати ресурси;
- Вкластися в рамки, передбачені графіком і бюджетом проекту;
- Отримати на виході якісний продукт.

СІМ ПРИНЦИПІВ ТЕСТУВАННЯ

1. ТЕСТУВАННЯ ВИЯВЛЯЄ ПРИСУТНІСТЬ ДЕФЕКТІВ.
2. ВИЧЕРПНЕ ТЕСТУВАННЯ НЕМОЖЛИВЕ.
3. РАННЄ ТЕСТУВАННЯ.
4. ГРУПУВАННЯ ДЕФЕКТІВ.
5. ПАРАДОКС ПЕСТИЦИДІВ.
6. ТЕСТУВАННЯ ЗАЛЕЖИТЬ ВІД КОНТЕКСТУ.
7. ВІДСУТНІСТЬ ДЕФЕКТІВ ОМАНЛИВА.

ТЕСТУВАННЯ ТА ЯКІСТЬ

- Тестування знижує загальний рівень ризику в системі;
- Тестування дає впевненість у якості ПЗ, якщо не виявлено дефектів;
- Тестування допомагає покращити якість, коли виявлені дефекти;
- Тестування є одним із заходів із забезпечення якості;
- Коли аналізуються засвоєні уроки та першопричини, можна покращити процеси і як наслідок покращити якість.

ПРИЧИНА ПРИПИНЕННЯ ТЕСТУВАННЯ - Зацікавлені сторони отримали достатньо інформації для прийняття рішень:

- Залишковий рівень ризику, включаючи технічні та бізнес ризики;
- Обмежений час та бюджет.

НЕЗАЛЕЖНІСТЬ ТЕСТУВАННЯ від низького до високого рівня:

- Тести, розроблені особою, яка написала програмне забезпечення;
- Тести, розроблені іншою людиною;
- Тести, розроблені особою з іншої організаційної групи;
- Тести, розроблені людиною з іншої компанії.

FUNDAMENTAL TEST PROCESS

ФУНДАМЕНТАЛЬНИЙ ПРОЦЕС ТЕСТУВАННЯ

- Планування і управління;
- Аналіз і проектування;
- Впровадження і реалізація;
- Оцінка критеріїв виходу і написання звітів;
- Дії по завершенню тестування.

TEST PLANNING (Планування тесту) – визначення цілей та специфікацій для досягнення цілей та місії проекту.

Test control (Тестовий контроль) - це поточна діяльність щодо порівняння фактичного прогресу з планом та звітування про стан, включаючи відхилення від плану.

TEST ANALYSIS AND DESIGN (**Аналіз і проектування**) Проаналізувати основу тестування (test basis) та цілі тесту, визначені на етапі планування, щоб бути перевіреними та однозначними. Оновити їх, якщо потрібно.

Основні завдання:

- Створити і визначити пріоритетні умови випробувань та очікувані результати;
- Створити пріоритетні тест кейси високого рівня;
- Перевірити двонаправлену простежуваність між тестовою базою та тестовими кейсами;
- Підготувати необхідні дані тесту та середовище.

TEST IMPLEMENTATION AND EXECUTION (Впровадження і реалізація)

- Завершення виконання та визначення пріоритетних тестових випадків;
- Розробка та визначення пріоритетності тестових процедур;
- Створення тестових даних;
- Створення тестових наборів для ефективного виконання тесту;
- Перевірка правильності налаштування тестових середовищ;
- Виконання процедур тестування;
- Реєстрація результатів виконання тесту;
- Порівнювання фактичних результатів із очікуваними результатами;
- Повідомлення та аналіз розбіжностей;
- Повторне виконання тесту, який раніше не вдався.

EVALUATING EXIT CRITERIA AND REPORTING (Критерії виходу та звітність)

Виконання тесту оцінюється відповідно до визначених цілей. Якщо вони будуть досягнуті, ми починаємо оцінювати критерії виходу.

Основні завдання:

- Складання підсумкового звіту про тести для зацікавлених сторін;
- Перевірка тестових журналів відповідно до критеріїв виходу, визначених у плануванні тесту;
- Оцінка того, чи потрібно більше тестів або зазначені критерії виходу слід змінити.

TEST CLOSURE ACTIVITES (Дії по завершенню тестування)

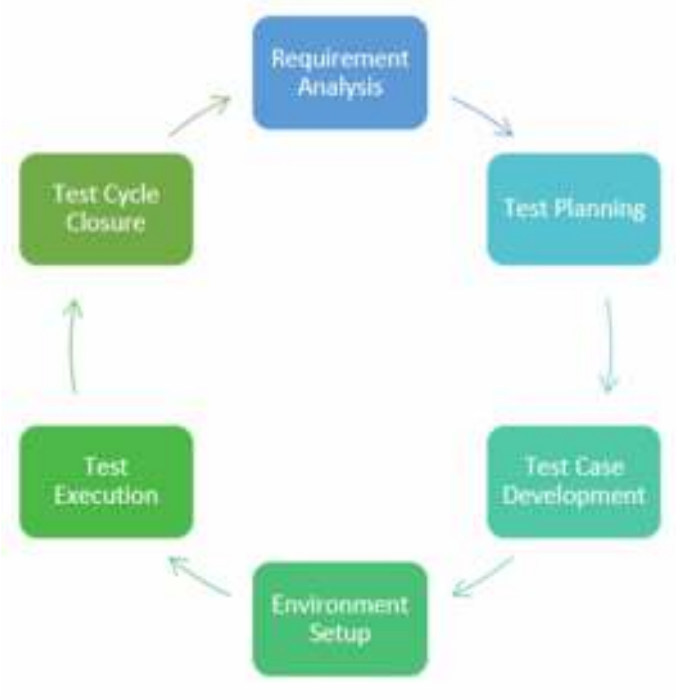
Основні завдання:

- Заплановані результати були доставлені;
- Закриття звітів про випадки або збір записів змін для тих, хто залишається відкритим;
- Аналіз уроків, отриманих для майбутніх випусків;
- Доопрацювання та архівування тестового програмного забезпечення, тестового середовища, тестової інфраструктури;
- Документування прийняття системи.

SOFTWARE TESTING LIFE CYCLE (STLC)

SOFTWARE TESTING LIFE CYCLE (Життєвий цикл тестування програмного забезпечення) - це послідовність конкретних заходів, що проводяться в процесі тестування для забезпечення досягнення цілей щодо якості програмного забезпечення.

REQUIREMENT ANALYSIS (Аналіз вимог):
- вивчення вимог з точки зору тестування;
- деталі про пріоритети в тестуванні;
- визначення типів тестів, які потрібно виконати;
- визначення деталей середовища тестування;
- аналіз доцільності автоматизації (якщо потрібно).



TEST PLANNING (Планування тестів):
- визначення вимог до тестів;
- оцінка ризиків;
- вибір стратегії тестування;
- визначення ресурсів;
- створення розкладу / послідовностей;
- розробка Плану тестування;

TEST CASE DEVELOPMENT (Дизайн тестів):
- аналіз обсягу робіт;
- визначення і опис тестових випадків;
- визначення і структурування тестових процедур;
- огляд і оцінка тестового покриття;

ENVIRONMENT SETUP (Розробка тестів):
- запис або програмування тестових скриптів;
- розробка тест кейсів;
- створення / підготовка зовнішніх наборів даних;

TEST EXECUTION (Виконання тестів):
- - виконання тестових процедур;
- - оцінка виконання тестів;
- - відновлення після збійних тестів;
- - перевірка результатів;
- - дослідження несподіваних результатів;
- - запис помилок;

TEST CYCLE CLOSURE (Оцінка тестів):
- оцінка покриття тестовими випадками;
- оцінка покриття коду;
- аналіз дефектів;
- доробка тест-кейсів;
- збір метрик;
- визначення критеріїв завершення і успішності тестування.

TYPES OF TESTING

Вибір типів та методів тестування, який слід використовувати, залежить від ряду факторів, включаючи тип системи, регуляторні норми, вимоги клієнта чи спільні дії, рівень ризику, тип ризику, мета випробування, наявний документ, знання про тестери, час та бюджет, життєвий цикл розвитку, використання моделей справ та попередній досвід виявлених типів дефектів.

CLASSIFICATION DIFFERENT TYPES OF TESTING

ISTQB

guru_99

CIS

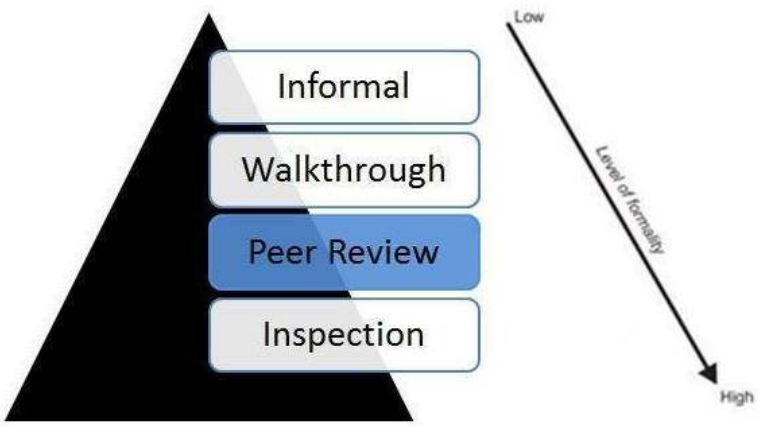
TYPES OF TESTING

REQUIREMENTS TESTING

Методи тестування вимог	Критерії оцінки якості вимог
▪ Перевірка документації; ▪ Аналіз поведінки системи; ▪ Прототипування.	▪ Правильність; ▪ Повнота вимог; ▪ Зрозумілість; ▪ Несуперечливість; ▪ Вимога не повинна визначати технічне рішення; ▪ Вимірюваність; ▪ Досяжність; ▪ Реалізовуваність; ▪ Важливість (пріоритетність) Парето; ▪ Відслідковуваність.

STATIC TESTING

Це тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері.



INFORMAL REVIEW - рецензування, що ґрунтується на неформальній (недокументованій) процедурі. Основна мета недорогий спосіб знайти потенційні дефекти.

WALKTHROUGH REVIEW - покроковий розбір, проведений автором документа збору інформації та забезпечення однакового розуміння змісту документа.

TECHNICAL / PEER REVIEW - рецензування програмного продукту, що розробляється, що проводиться співробітниками компанії-розробника з метою знаходження дефектів та внесення поліпшень.

INSPECTION REVIEW - тип рівноправного аналізу, що ґрунтується на візуальній перевірці документів для пошуку помилок. Наприклад, порушення стандартів розробки та невідповідність документації вищого рівня. Найбільш формальна методика рецензування і тому завжди ґрунтується на документованій процедурі.

DYNAMIC TESTING

це тестова діяльність, що передбачає експлуатацію програмного продукту.

FUNCTIONAL TESTING системи включає тести, які оцінюють функції, які виконуються системою. Функціональне тестування розглядає зазначену поведінку і часто також називається black box testing (тестування чорного ящика, тестування на основі специфікації).

NON-FUNCTIONAL TESTING оцінює характеристики систем і програмного забезпечення, які відповідають за “наскільки добре” веде система. Відповідає на питання **ЯК?**

Нефункціональне тестування, як і функціональне тестування, може і часто повинно виконуватися на всіх рівнях тестування, до того ж, як можна раніше. Методи чорного ящика можуть використовуватися для отримання умов тестування та тестових прикладів для нефункціонального тестування.

CHANGES RELATED TESTING підтверджувальне і регресійне тестування після корегування продукту.

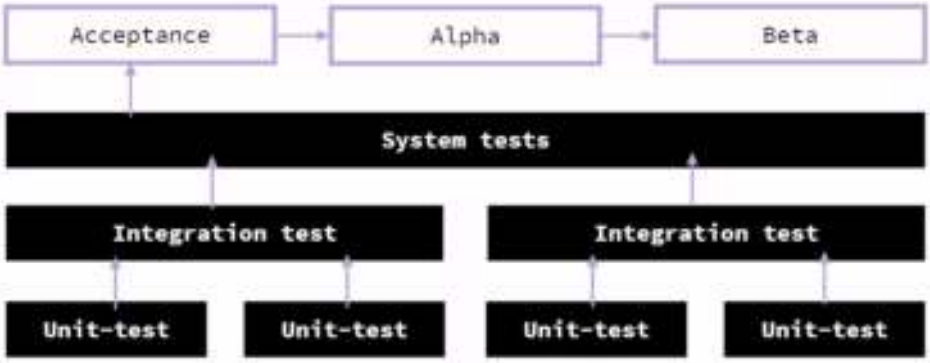
FUNCTIONAL TYPES OF TESTING

UNIT TESTING - це тип тестування програмного забезпечення, у якому тестуються окремі модулі чи компоненти програмного забезпечення. Ціль полягає в тому, щоб перевірити, що кожна одиниця програмного коду працює належним чином. Модульне тестування виконується розробниками під час розробки (фаза кодування) програми.

INTEGRATION TESTING визначається тип тестування, у якому програмні модулі інтегруються логічно і тестуються як група. Метою даного рівня тестування є виявлення дефектів взаємодії між програмними модулями за її інтеграції.

SYSTEM TESTING – це рівень тестування, який перевіряє закінчений та повністю інтегрований програмний продукт. Метою системного тесту є оцінка наскрізних технічних характеристик всієї комп'ютерної системи.

ACCEPTANCE TESTING - це перевірка на відповідність **Acceptance criteria**. Може бути **Alpha, Beta, Gamma тестування**.



Альфа тестування (alfa testing) - імітація реальної роботи із системою штатними розробниками або реальна робота із системою потенційними користувачами\замовником. Найчастіше альфа тестування проводиться на ранній стадії розробки продукту, але в деяких випадках може проводитися для готового продукту як внутрішнього приймального тестування.

Бета тестування (beta testing) - інтенсивне використання майже готової версії продукту (як правило, програмного або апаратного забезпечення) з метою виявлення максимальної кількості помилок у його роботі для їх подальшого усунення перед остаточним виходом (релізом) товару ринку, до масовому споживачеві.

Гамма тестування (gamma testing) – тестуються виправлені помилки знайдені на рівні **Beta** тестування та перевіряється на відсутність нових помилок.

NON-FUNCTIONAL TYPES OF TESTING

PERFORMANCE TESTING це тестування, яке проводиться з метою визначення, як швидко працює програма або її частина під деякою навантаженням. Тестування продуктивності намагається підрахувати продуктивність.

LOAD TESTING це тестування продуктивності. Воно зазвичай проводиться для того, щоб оцінити поведінку програми (додатку) з заданим очікуваним навантаженням. Цим навантаженням може бути, наприклад, кількість користувачів, які одночасно працюватимуть з програмою. Такий вигляд Тестування дозволяє отримати час відгуку всіх найважливіших бізнес-транзакцій.

STRESS TESTING тип тестування продуктивності, який проводиться для оцінки системи або компонента рівнів очікуваних робочих навантажень або за предметами, або з обмежувальною доступністю ресурсів, таких як доступ до пам'яті або серверам.

RECOVERY TESTING перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.

VOLUME TESTING це тип тестування програмного забезпечення, коли програмне забезпечення піддається величезному обсягу даних. Це також згадується як тестування повені. Об'ємне тестування проводиться для аналізу продуктивності системи шляхом збільшення обсягу даних у базі даних.

SCALABILITY TESTING - це нефункціональне тестування, яке вимірює продуктивність мережі або системи, коли кількість запитів користувача збільшується або зменшується. Метою тестування масштабованості є забезпечення того, щоб програма могла обробляти прогнозоване збільшення користувальницького трафіку, обсягу даних, частоти підрахунку транзакцій і т. д. Воно перевіряє здатність системи, процесів і баз даних задовольнити зростаючу потребу.

ENDURANCE TESTING - це тип тестування програмного забезпечення, при якому система тестується з навантаженням, розтягнутим протягом значного часу, для оцінки поведінки системи при тривалому використанні. Мета тестування полягає в тому, щоб переконатися, що програма здатна витримати розширене навантаження без будь-якого погіршення часу відгуку.

USER INTERFACE (UI) TESTING - це діяльність, спрямована на перевірку якості інтерфейсу користувача, а також його відповідності всім нормам і вимогам.

USER EXPERIENCE-TESTING / USABILITY TESTING це тестування зручності - процес взаємодії з тестованим продуктом, в якому вивчається рівень зручності, а також задоволеності при використанні.

CONFIGURATION / COMPATIBILITY TESTING дозволяє перевірити працездатність програмної системи в умовах різних операційних систем, апаратних та програмних конфігурацій. Тестується ефект впливу на продуктивність при змінах у конфігурації.

PORTABILITY TESTING перевіряє доступність до переносу додатка на різні платформи.

LOCALIZATION TESTING - перевірка відповідності вмісту продукту, програми чи документа до культурних, мовних та інших вимог конкретного регіону.

INTERNATIONALIZATION TESTING - перевірка продукту на можливість та відповідність вимогам локалізувати програму до будь-якої культури, регіону та мови (кодування Unicode, направлення тексту та ін.).

Локалізація та інтернаціоналізація разом називаються **глобалізацією**.

CHANGES RELATED

RE-TESTING /CONFIRMATION TESTING - проводиться для підтвердження виправлення помилки та роботи даного функціоналу.

SMOKE TESTING це перевірка найважливішого функціоналу, виявлення явних помилок. Проводиться покриттям тестами якомога більшого функціоналу в найкоротші терміни.

REGRESSION TESTING проводиться з метою перевірки працездатності функціоналу, що існує, та перевірки на відсутність сторонніх помилок після оновлення білда (внесення правок або доповнень в систему).

SANITY TESTING це вид тестування, що використовується з метою доказу працездатності конкретної функції або модуля відповідно до заявлених технічних вимог.

OTHERS TYPES

INSTALLATION TESTING – перевірка успішності установки програми, її налаштування та видалення. Знижує ризики втрати даних користувача, втрати працездатності програми та ін.

SECURITY TESTING — це вид тестування програмного забезпечення, який має на меті оцінити та перевірити цілісність системи від стороннього вторгнення. А саме автентифікацію, авторизацію, доступність, конфіденційність і безвідмовність аплікації під тестами.

ACCESSIBILITY TESTING — це область знання, яка займається вивченням питань доступності сайтів, мобільних додатків і програмного забезпечення для людей з обмеженими можливостями.

AD-HOC / RANDOM / MONKEY TESTING – це вид тестування, який виконується без підготовки до тестування продукту, без визначення очікуваних результатів, проєктування тестових сценаріїв. Це неформальне, імпровізаційне тестування. Воно не вимагає ніякої документації, планування, процесів, яких слід дотримуватися при тестуванні.

END TO END TESTING - це методологія тестування програмного забезпечення для перевірки потоку додатків від початку до кінця. Метою наскрізного тестування є імітація реального сценарію користувача та перевірка системи, що перевіряється, та її компонентів для інтеграції та цілісності даних.

TEST-DESIGN TECHNIQUES

BLACK-BOX / BEHAVIOR-BASED / SPECIFICATION-BASED TECHNIQUES

Техніки чорного ящика виходять з базису тестування (тестові умови, тестові сценарії та тестові дані), який може включати вимоги, специфікації, сценарії використання та user stories.

Тестові сценарії можуть використовуватися для визначення невідповідностей та відхилень між вимогами та реалізацією.

Техніки тест дизайну чорного ящика:
- **EQUIVALENCE PARTIONING** (еквівалентне розбиття);
- **BOUNDARY VALUE ANALYSIS** (аналіз граничних значень);
- **PAIRWISE TESTING** (метод всіх пар);
- **DECISION TABLE** (таблиці прийняття рішень);
- **STATE TRANSITION TABLE** (перехід станів);
- **USE-CASES** (сценарій використання).

EQUIVALENCE PARTIONING

Клас еквівалентності - це набір значень змінної, який вважається еквівалентним. Поведінка системи однакова для всіх значень, які належать одному класу еквівалентності.

Тестові сценарії еквівалентні, якщо вони тестують одне й те саме (якщо при використанні одного з значень (не) виявлено помилку, то і для інших значень діапазону - вважаються, що помилки (не) виявлені.

Алгоритм використання техніки:
- визначити класи еквівалентності;
- вибрати одного представника з кожного класу;
- виконати тести з обраними значеннями.

BOUNDARY VALUE ANALYSIS

Аналіз граничних значень перевіряє межі числового діапазону. Значення з валідного діапазону називаються валідні граничні значення, та якщо з невалідного діапазону - невалідні значення.
Межі еквівалентності бувають закриті (100≤N≤1000) та відкриті (1000≤N).

Даний метод має два підходи:
- коли межа сусідніх еквівалентних класів проходить між ними і не належать жодному з класів, в такому випадку ми маємо два граничних значення (по боках точки);
- коли межа належить одному з класів, в цьому випадку необхідно перевірити три значення - точку кордону і значення по боках.

Мета техніки - знайти помилки пов'язані з граничними значеннями.

Алгоритм техніки:
- виділити класи еквівалентності;
- визначити граничні значення цих класів та визначити до якого діапазону відноситься гранична точка і чи відноситься;
- для кожного кордону провести тести по перевірці значення до кордону, на кордоні і після кордону.

PAIRWISE TESTING

Ця методика систематичної комбінаторики тестів, котра забезпечує ефективне зниження кількості тестів для перевірки реакції системи на можливі комбінації значень її вхідних параметрів.

Методика заснована на статистичному припущенні, що достатньо перевірити усі можливі значення пар вхідних параметрів для того, щоб виявити більшість дефектів системи залежних від вхідних параметрів.
Цей метод гарантує, що всі комбінації з пар значень будь-яких змінних будуть протестовані

Алгоритм:
- визначити змінні;
- визначити діапазон значень змінних;
- побудувати таблицю, де
  кількість стовпців = кількості змінних;
  кількість рядків = добуток кількості значень двох найбільш вагомих змінних;
  тест-кейс=рядок таблиці.

DECISION TABLE

Застосовують в разі залежності вихідних даних або поведінки програми від комбінації значень вхідних параметрів, при перевірці бізнес-правил.

Кроки побудови таблиці:
- визначити/записати всі умови;
- порахувати кількість можливих комбінацій умов;
- заповнити комбінації;
- прибрати зайві комбінації;
- записати дії.

STATE TRANSITION TABLE

Кінцевий автомат (finite state machine) обчислювальна модель, що складається з обмеженого числа станів і переходів між цими станами.
Будь-яка система, що дає різний висновок на один і той же введення в залежності від того, що сталося раніше - кінцевий автомат.
Алгоритм роботи системи відображається як правило у виді діаграми.

Діаграма переходу станів показує початковий і кінцевий стан системи, а також описує переходи між станами. При цьому:
- діаграма переходу станів показує лише валідні переходи;
- діаграма складається з пар переходів між двома станами;
- якщо переходу між двома станами немає, то перехід вважається невалідним.

USE-CASES

Сценарій використання - сценарій, який описує використання системи користувачем для досягнення певної мети. Користувачами (actors) можуть бути як люди так і інші системи.

Переваги:
- охоплюють функціональність з точки зору користувача, а не з технічної точки зору;
- не залежить від парадигми програмування;
- дозволить активно залучати користувачів в процесі збору вимог;
- легко дозволяють визначити базові компоненти системи, у зв'язку між ними;
- служить основою для розробки тест-кейсів для системного та приймального тестування.

WHITE BOX TECHNIQUES / GLASS BOX / STRUCTURE-BASED

Техніки білого ящика виходять з базису тестування (тестові умови, тестові сценарії, тестові дані), який може включати, код, архітектуру або будь-яке інше джерело інформації про структуру програмного забезпечення.

Вимірювання покриття засноване на елементах структури (код, інтерфейси і т.д.);

Специфікації використовуються як джерело додаткової інформації для визначення очікуваних результатів тестових сценаріїв.

Техніки тест дизайну білого ящика:
- **STATEMENT TESTING AND COVERAGE** (тестування покриття операторів);
- **DECISION TESTING AND COVERAGE / BRANCH TESTING** (тестування умов).

STATEMENT COVERAGE TESTING

Передбачає створення тестів, які покривають як можна більше дій в коді програми.

Перевіряється мінімальна кількість шляхів потрібна для покриття всіх станів.

DECISION TESTING AND COVERAGE

Передбачає створення тестів, які покривають як можна більше переходів/рішень між діями.

EXPERIENCE BASED TESTING

Техніки, що ґрунтуються на досвіді виходять з базису тестування (тестові умови, тестові сценарії та тестові дані), який може включати знання та досвід тестувальників, розробників, користувачів та інших зацікавлених осіб.

Техніки тест дизайну, що ґрунтуються на досвіді:
- **ERROR GUESSING** (вгадування помилок);
- **EXPLORATORY TESTING** (дослідницьке тестування);
- **CHECK-LISTS BASED TESTING** (тестування на основі чек-лістів).

ERROR GUESSING

Це метод тестування на основі досвіду, коли аналітик тестів використовує свій досвід, щоб вгадати проблемні області програми. Для цієї техніки обов’язково потрібні кваліфіковані та досвідчені тестувальники.

EXPLORATORY TESTING

Дослідницьке тестування – це одночасне навчання, дизайн тесту та процес виконання тесту. Можна сказати, що в цьому тестуванні планування тестів, аналіз, проектування та виконання тестів виконуються разом і миттєво.

CHECK-LISTS BASED TESTING

Метод створення тестів, заснований на досвіді, у якому досвідчений тестувальник використовує високорівневі списки. Список містить пункти, які потрібно відзначити або запам'ятати, або складається з набору правил або критеріїв, згідно з якими верифікується програмний продукт.

PROJECT DOCUMENTATION

TEST STRATEGY

Стратегія тестування - це план проведення робіт з тестування системи або її модуля, що враховує специфіку функціональності і залежності з іншими компонентами системи і платформи.

TEST PLAN

Тест План - це документ, що описує весь обсяг робіт з тестування, починаючи з опису тестованих об'єктів, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх дозволу.
Що треба тестувати? Опис об'єкта тестування системи, додатки, обладнання, список функцій та опис тестованої системи та її компонентів окремо.
Як тестуватимете? стратегія тестування, а саме види тестування порядок їх застосування стосовно об'єкту тестування.
Коли тестуватимете? послідовність проведення робіт підготовка (Test Preparation), тестування (аналіз результатів (Test Result Analysis) у розрізі запланованих фаз розробки.
Критерії початку тестування (готовність тестової платформи (тестового стенду), закінченість розробки необхідного функціоналу, наявність всієї необхідної документації.
Критерії закінчення тестування (результати тестування, які відповідають критеріям якості продукту.

Основні атрибути:
Test plan identifier - Унікальний номер.
Introduction - Введення.
Test items - Елементи, функції тестування.
Features to be tested - Що будемо тестувати.
Features not to be tested - Що Не будемо тестувати.
Approach - Якою буде підхід до тестування.
Item pass / fail criteria - Критерії проходження тестів.
Suspension criteria and resumption requirements - Критерії виходу з тестування.
Test deliverables - Перелік документів, які повинні бути розроблені.
Testing tasks - Розписуємо завдання, які будуть виконувати тестувальники.
Environmental needs - Вимоги до оточення для початку тестування (Наприклад, яка повинна бути відеокарта при тестуванні ігрових проектів).
Responsibilities - Відповідальність (зазвичай це одна особа - TeamLead).
Staffing and training needs - Чому потрібно навчити тестувальників до початку тестування (наприклад, для тестування Casino проектів потрібно мати досвід від 6 міс).
Schedule - Графік активностей, контрольні точки, календарні дати по кожній.
Risks - Які загальні ризики для проекту з акцентом на процесі тестування?
- Відсутність ресурсів на момент початку тестування;
- Відсутність необхідного програмного забезпечення, даних або інструментів, або доступу до даних субпідрядника;
- Затримки в тренінгу;
- Зміни у вимогах.

Approvals – Перелік осіб,що погоджують.
Testware - це артефакти, отримані під час тестування.

TEST CASE

Тестовий Випадок **-** це документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції або її частини.

Основні атрибути:
№ (Номер) унікальний ідентифікатор тест кейсу.
Description (Опис) - короткий опис суті перевірки.
Preconditions (Попередні кроки) - опис дій, які потрібно виконати, але прямого відношення до перевірки вони не мають.
Steps to reproduce (Кроки) - опис дій, необхідні для перевірки.
Expected result (Очікуваний результат) - що ми очікуємо отримати після виконання кроків.
Priority (Пріоритет) - пріоритет перевірки.
Status (Статус) - результат перевірки.
Designer (Дизайнер) - автор Тест Кейса;
Created (Дата створення) - дата створення Тест Кейса.

TEST SUITE

Тестовий Набір – це набір тест кейсів які об'єднані тим, що відносяться до одного тестованого модуля, функціональності, пріоритету або одного типу тестування.

CHECK LIST

Контрольний Список - це документ, який описує, що має бути протестовано.

Він дозволяє не забувати про важливі моменти, фіксувати результати своєї роботи і відслідковувати статистику про статус програмного продукту.

BUG REPORT

Звіт про знаходження помилки **–** докладна покрокова інструкція для відтворення помилки.

Основні атрибути:
ID - номер звіту у системі.
Summary - короткий опис проблеми, що явно вказує на причину помилки.
Preconditions - умови, які мають бути дотримані для відтворення помилки.
Project - назва тестованого проекту.
Component - назва частини або функції продукту, що тестується.
Version - версія продукту, в якій було знайдено баг.
Severity - показник, що відображає вплив дефекту на працездатність програми.
Status - поточний стан бага.
Author (reporter) - творець баг репорту.
Assigned to - співробітник, який займається багом зараз.
Environment - оточення, на якому було знайдено баг, ОС, ім'я та версія браузера тощо.
Steps to reproduce - кроки відтворення.
Actual result - фактичний результат, отриманий після кроків відтворення.
Expected result - очікуваний результат.
Attachments - вкладення, що допомагає відтворити баг та прояснити ситуацію.

TEST SUMMARY REPORT

Підсумковий звіт про тестування **–** документ, що підбиває підсумки завдань та результатів тестування.