МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ОЦІНЮВАННЯ ЯКОСТІ

КОНСПЕКТ ЛЕКЦІЙ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інформаційні вимірювальні технології»
спеціальності 152 Метрологія та інформаційно-вимірювальна техніка

Укладач: М. В. Добролюбова

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензент *Помазун, О. М.*, к.е.н., доцент,
доцент, кафедра інформаційних систем в економіці, ІІТЕ,
КНЕУ імені Вадима Гетьмана

Відповідальний
редактор *Богомазов, С. А.*, к.т.н., доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 8 від 02.06.2023 р.)
за поданням Вченої ради приладобудівного факультету
(протокол № 5/23 від 29.05.2023 р.)

Навчальний посібник забезпечує лекційний курс з дисципліни «Інформаційні технології оцінювання якості».
Навчальний посібник знайомить студентів з фундаментальними поняттями в області оцінювання якості програмного забезпечення, які необхідні при проєктуванні та експлуатації програмної складової інформаційно-вимірювальних систем і приладів. Чітка структурованість лекцій допоможе студентові швидше і легше оволодіти теоретичними положеннями для кращого розуміння підходів при формуванні необхідних властивостей та характеристик програмного забезпечення в процесі його розробки, що дозволить використовувати набуті знання на практиці при вирішенні конкретних задач різного ступеня складності.

Реєстр. № НП 22/23-775. Обсяг 18,91 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ПЕРЕДМОВА

Курс лекцій з дисципліни «Інформаційні технології оцінювання якості» відповідає освітньо-професійній програмі «Інформаційні вимірювальні технології» за спеціальністю 152 «Метрологія та інформаційно-вимірювальна техніка».

Курс лекцій допомагає організувати самостійну роботу студента для вивчення фундаментальних понять в області оцінювання якості програмного забезпечення, які необхідні при проєктуванні та експлуатації програмної складової інформаційно-вимірювальних систем і приладів, а також сприяє кращому розумінню підходів при формуванні необхідних властивостей та характеристик програмного забезпечення в процесі його розробки, що дозволяє використовувати набуті знання при вирішенні конкретних задач різного ступеня складності.

Курс лекцій базується на чіткому викладі теоретичних матеріалів, містить презентації до кожного лекційного заняття та контрольні питання з відповідних тем, що необхідні для закріплення здобутих знань і навичок.

ЗМІСТ

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.1 МЕТА, ПОНЯТТЯ ТА ТЕРМІНИ КЛАСИФІКАЦІЯ

Лекція 1

ОСНОВНІ ПОНЯТТЯ ТА ВИЗНАЧЕННЯ В ОБЛАСТІ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета лекції: Розгляд основних понять, пов'язаних з тестуванням програмного забезпечення.

Зміст лекції

1. Поняття та мета тестування, передумови його появи.
2. Філософія тестування, завдання та концепції.
3. Тестувальник та його обов'язки.
4. Оманливі твердження щодо тестування.

Поняття та мета тестування, передумови його появи

- **Тестування програмного забезпечення** – процес аналізу програмного засобу та супутньої документації з метою виявлення дефектів та підвищення якості продукту.
- **Testing** – процес, що складається з усіх дій життєвого циклу, як статичних, так і динамічних, пов'язаних з плануванням, підготовкою та оцінкою програмних продуктів і пов'язаних з ними робочих продуктів, щоб визначити, що вони задовольняють визначеним вимогам, з метою демонстрації їх відповідності призначенню та виявлення дефектів [1].
- **Тестування** – процес, що підтверджує правильність програми та демонструє, що помилок у програмі немає.
- **Тестування** – процес виконання програми (або частини програми) з наміром (або метою) знайти помилки.

- Мета тестування – знайти помилки у програмі і, тим самим, підвищити її надійність та цінність.
- **Баг** у програмуванні – жаргонне слово, що зазвичай означає помилку в програмі або системі, через яку програма видає несподівану поведінку і, як наслідок, результат.
- **Реліз** – передача Програмного забезпечення користувачеві.
- **QA** – це процес або результат формування необхідних властивостей та характеристик продукції в міру її створення, а також підтримка цих характеристик при зберіганні, транспортування та експлуатації продукції [1, 2].
- **QC** – це процес знаходження помилок у продукті, з метою їхнього подальшого виправлення [1, 2].
- **Функціональне тестування** – це тестування ПЗ з метою перевірки реалізованості функціональних вимог, тобто можливості ПЗ у певних умовах вирішувати завдання, потрібні користувачам [1, 2].
- **Нефункціональне тестування** – тестування властивостей, які не належать до функціональності системи [1, 2].
- 1950-60-і роки. Процес тестування формалізований, багато уваги приділялося exhaustive «вичерпному» тестуванню [1, 3, 4].
- 1970-і роки. Тестування програмного забезпечення позначалося як «процес, спрямований на демонстрацію коректності товару».
- 1980-і роки Тестування розширилося таким поняттям, як попередження дефектів. Проектування тестів – найбільш ефективний із відомих методів попередження помилок.
- 1990-і роки. До поняття «тестування» стали включати планування, проектування, створення, підтримку та виконання тестів та тестових оточень.
- 2000-і роки. З розвитком Інтернету та розробкою великої кількості веб-додатків особливу популярність стало набувати «гнучке тестування» та такі підходи, як «розробка під управлінням тестуванням (test-driven

development, TDD)». З'явилося ще ширше визначення тестування, коли до нього було додано поняття «оптимізація бізнес-технологій» (BTO).

Філософія тестування, завдання та концепції

- Види діяльності, що охоплює тестування програмного забезпечення:
  - постановка задачі для тестування;
  - проектування, написання тестів;
  - тестування тестів;
  - виконання тестів;
  - вивчення результатів тестування.
- Завдання тестування:
  - відтворити якомога більше багів;
  - пропустити якнайменше пріоритетних для користувача багів.
- Концепції тестування:
  - мета тестування – це 100% перевірка ПЗ;
  - критерій ефективності тестування – це кількість багів, знайдених до релізу.

Тестувальник та його обов'язки

- Стартові технічні навички:
  - знання іноземних мов;
  - впевнене володіння комп'ютером на рівні по-справжнього просунутого користувача та бажання постійно розвиватися у цій галузі;
  - програмування;
  - бази даних та мова SQL;
  - розуміння принципів роботи мереж та операційних систем;
  - розуміння принципів роботи веб-додатків та мобільних додатків;
  - розуміння принципів самого тестування.
- Особисті якості:
  - підвищена відповідальність та старанність;

- добрі комунікативні навички, здатність ясно, швидко, чітко висловлювати свої думки;
- терпіння, усидливість, уважність до деталей, спостережливість;
- хороше абстрактне та аналітичне мислення;
- здатність ставити нестандартні експерименти, схильність до дослідницької діяльності.

Оманливі твердження щодо тестування

- Міфи [5]:
  - не треба розумітися на комп'ютерах;
  - обов'язково треба добре знати програмування;
  - в тестуванні все просто;
  - в тестуванні купа рутини та нудьги;
  - тестувальника повинні всьому навчити;
  - у тестувальники йдуть ті, хто не зміг стати програмістом;
  - в тестуванні складно збудувати кар'єру;
  - тестувальник «винний у всьому»;
  - тестувальники скоро будуть не потрібні, оскільки все буде автоматизовано.

Закріплення матеріалу

1. Роз’яснити значення наступних термінів: «тестування», «баг», «реліз», «QA», «QC», «функціональне тестування» та «нефункціональне тестування».
2. В чому полягає мета тестування?
3. Описати основні завдання та концепції тестування.
4. Які види діяльності охоплює тестування програмного забезпечення?
5. Які стартові навички необхідні тестувальнику?
6. Які міфи про тестування вам відомі? Спробуйте їх розвінчати.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.2 КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ

Лекція 2

ПРОЦЕСИ ТЕСТУВАННЯ І РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета лекції: Розгляд основних моделей програмного забезпечення та ознайомлення із стадіями життєвого циклу тестування.

Зміст лекції

1. Моделі розробки програмного забезпечення.
2. Життєвий цикл тестування.

Моделі розробки програмного забезпечення

- **Модель розробки програмного забезпечення** (Software Development Model, SDM) – це структура, яка систематизує різні види проєктної діяльності, їх взаємодію та послідовність у процесі розробки програмного забезпечення (ПЗ).
- Вибір моделі розробки ПЗ безпосередньо впливає на процес тестування та визначає вибір стратегії, розклад, потрібні ресурси тощо.
- Класичні моделі розробки [6]:
  - водоспадна;
  - v-подібна;
  - ітераційна інкрементальна;
  - спіральна;
  - гнучка.
- **Водоспадна модель** (waterfall model) передбачає одноразове виконання кожної з фаз проєкту, які, в свою чергу, суворо слідують одна за одною [1].

- Недоліки водоспадної моделі:
  - не передбачена участь користувачів програмного забезпечення;
  - поява тестування лише в середині проекту з максимумом наприкінці;
  - не передбачена можливість адаптувати проєкт до змін у вимогах;
  - пізнє створення працюючого продукту.
- Переваги водоспадної моделі:
  - кожна стадія має чіткий результат, який перевіряється;
  - команда у кожен момент часу виконує один вид роботи;
  - добре працює для відносно простих завдань або ж у великих проєктах з стабільними, сформульованими на початку вимогами.
- Фази водоспадної моделі:
  - загальне планування;
  - вимоги користувача;
  - системні вимоги;
  - технічна архітектура;
  - деталізований дизайн;
  - розробка та відладка;
  - інтеграція і модульні тести;
  - інсталяційне тестування;
  - системне тестування;
  - приймальне тестування;
  - ітогова звітність.
- **V-подібна модель** (V-model) – це різновид водоспадної моделі, який передбачає планування робіт з тестування на ранніх стадіях життєвого циклу [2].
- **Прототип** – це діючий програмний компонент, за допомогою якого реалізуються окремі функції та зовнішні інтерфейси програмного продукту, що розробляється.

- Недоліки v-подібної моделі:
  - відсутність раннього прототипування;
  - недостатня гнучкість та адаптованість;
  - складність усунення помилок, що виникли на ранніх стадіях життєвого циклу проекту.
- Переваги v-подібної моделі:
  - кожна стадія має чіткий результат, який перевіряється;
  - поява тестування з першої стадії;
  - добре працює для проєктів зі стабільними вимогами.
- Фази v-подібної моделі співпадають з фазами водоспадної моделі.
- **Ітераційна інкрементальна модель** (iterative model, incremental model) – модель, що передбачає розбиття проєкту на відносно невеликі проміжки (ітерації), кожен із яких у загальному випадку може включати всі класичні стадії, властиві водоспадній і v-подібній моделям [1].
- **Ітераційна модель розробки** – життєвий цикл розробки, коли проєкт розбивається на зазвичай велику кількість ітерацій [1].
- **Ітерація** – це повний цикл розробки, що призводить до випуску виконуваного продукту, підмножини кінцевого продукту у розробці, яка зростає від ітерації до ітерації, щоб стати кінцевим продуктом.
- **Інкрементальна модель** (модель поступового розвитку) – це життєвий цикл розробки, коли проєкт розбивається на серію кроків, кожен з яких забезпечує частину функціональності в загальних вимогах проєкту [1].
- **Білд** (build)– діяльність з розробки, за допомогою якої повна система компілюється та зв’язується, з метою отримання доступної узгодженої системи, включаючи всі останні зміни).
- Недоліки ітераційної інкрементальної моделі:
  - недостатня гнучкість всередині ітерацій;
  - складність усунення помилок, що виникли на ранніх стадіях життєвого циклу проекту;

- високі накладні витрати.

- Переваги ітераційної інкрементальної моделі:
  - раннє прототипування;
  - просте керування ітераціями;
  - можливість декомпозиції проекту на ітерації, якими можна керувати;
  - добре працює для об'ємних і складних проектів, що виконуються великими командами протягом тривалого часу.
- **Спіральна модель** (spiral model) – це модель, яка відображає базову концепцію розробки, згідно з якою набору операцій у кожному циклі відповідає така ж кількість фаз, як і моделі водоспадного процесу, починаючи з формулювання вимог і закінчуючи кодуванням кожної окремої програми [1].
- Основні принципи спіральної моделі:
  - розробка варіантів продукту з опрацюванням цілей, альтернатив та обмежень, що відповідають різним варіантам вимог, з можливістю повернення до попередніх версій;
  - аналіз ризиків та створення прототипів програмного продукту як засобу спілкування із замовником для уточнення та виявлення вимог;
  - планування наступних варіантів з оцінкою альтернатив та аналізом ризиків, пов'язаних із переходом до наступного варіанту; перехід до розробки наступного варіанту до завершення попереднього у разі, коли ризик завершення чергового варіанта (прототипу) стає невиправдано високий;
  - активне залучення замовника до роботи над проєктом. Замовник бере участь в оцінці чергового прототипу програмного продукту, уточненні вимог при переході до наступного, оцінці запропонованих альтернатив чергового варіанту.

- Недоліки спіральної моделі:
  - висока залежність успіху від якості аналізу ризиків;
  - складність застосування для невеликих проектів;
  - високі накладні витрати.
- Переваги спіральної моделі:
  - глибокий аналіз ризиків;
  - добре працює для великих проектів;
  - можливість раннього прототипування.
- **Гнучка модель** (agile model) – це модель, яка є сукупністю різних підходів до розробки програмного забезпечення та базується на «agile-маніфесті» [1, 7].
- Постулати «agile-маніфесту»:
  - люди та взаємодія важливіші за процеси та інструменти;
  - працюючий продукт важливіший за вичерпну документацію;
  - співпраця із замовником важливіша за погодження умов контракту;
  - готовність до змін важливіша за дотримання початкового плану.
- Недоліки гнучкої моделі:
  - складність реалізації для великих проектів;
  - складність побудови стабільних процесів.
- Переваги гнучкої моделі:
  - максимальне залучення замовника;
  - багато роботи з вимогами;
  - тісна інтеграція тестування та розробки;
  - мінімізація документації.

Життєвий цикл тестування [8]

- **Стадія 1.** Загальне планування та аналіз вимог.
- **Стадія 2.** Уточнення критеріїв приймання: формулювання або уточнення метрик та ознак можливості / необхідності початку, призупинення, поновлення, завершення або припинення тестування.

- **Стадія 3.** Уточнення стратегії тестування.
- **Стадія 4.** Розробка тест-кейсів.
- **Стадія 5.** Виконання тест-кейсів.
- **Стадія 6.** Фіксація знайдених дефектів.
- **Стадія 7.** Аналіз результатів тестування.
- **Стадія 8.** Звітність.

Закріплення матеріалу

1. Розкрити суть водоспадної моделі життєвого циклу розробки програмного продукту.
2. Прокоментувати переваги та недоліки водоспадної моделі життєвого циклу розробки програмного продукту.
3. Розкрити суть v-подібної моделі життєвого циклу розробки програмного продукту.
4. Прокоментувати переваги та недоліки v-подібної моделі життєвого циклу розробки програмного продукту.
5. Розкрити суть ітераційної інкрементальної моделі життєвого циклу розробки програмного продукту.
6. Прокоментувати переваги та недоліки ітераційної інкрементальної моделі життєвого циклу розробки програмного продукту.
7. Розкрити суть спіральної моделі життєвого циклу розробки програмного продукту.
8. Прокоментувати переваги та недоліки спіральної моделі життєвого циклу розробки програмного продукту.
9. Розкрити суть гнучкої моделі життєвого циклу розробки програмного продукту.
10. Перелічити постулати «agile-маніфесту».
11. Прокоментувати переваги та недоліки гнучкої моделі життєвого циклу розробки програмного продукту.
12. Перелічити та прокоментувати стадії життєвого циклу тестування.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.2 КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ

Лекція 3

ТЕСТУВАННЯ ДОКУМЕНТАЦІЇ І ВИМОГ

Мета лекції: Розгляд поняття вимог, їх класифікації та технік тестування.

Зміст лекції

1. Поняття «вимоги».
2. Важливість вимог.
3. Джерела та шляхи виявлення вимог.
4. Рівні та типи вимог.
5. Властивості якісних вимог.
6. Техніки тестування вимог.

Поняття «вимоги»

- **Вимога** (requirement) – опис того, які функції та з дотриманням яких умов має виконувати програма в процесі розв'язання необхідної для користувача задачі [1].
- **Вимога** (requirement) – умова або здатність, необхідна користувачеві для вирішення проблеми чи досягнення мети, якою має володіти система або компонент системи, щоб задовольнити контракт, стандарт, специфікацію або інший офіційно встановлений документ.

Важливість вимог

- Важливість вимог полягає в тому, що вони:
  - дозволяють зрозуміти, що та з дотриманням яких умов система повинна робити;
  - надають можливість оцінити масштаб змін та керувати змінами;

  - є основою формування плану проєкту (зокрема плану тестування);
  - допомагають запобігати або вирішувати конфліктні ситуації;
  - спрощують розміщення пріоритетів у наборі завдань;
  - дозволяють об'єктивно оцінити рівень прогресу у розробці проєкту.
- Види документації:
  - продуктна документація;
  - проєктна документація.
- Складові продуктної документації:
  - план проєкту (project management plan) та тестовий план (test plan);
  - вимоги до програмного продукту (product requirements document, PRD) та функціональні специфікації (FSD, SRS);
  - архітектура та дизайн (architecture and design);
  - тест-кейси та набори тест-кейсів (test cases, test suites);
  - технічні специфікації (technical specifications).
- Складові проєктної документації:
  - супровідна документація та документація користувача (user and accompanying documentation);
  - маркетингова документація (market requirements document, MRD).
- **Документація по розробці** включає ті документи, які пропонують, уточнюють, планують, переглядають, тестують і впроваджують продукти команд розробників в індустрії програмного забезпечення [9].
- **План управління проєктом** – це офіційний затверджений документ, який визначає, як проєкт виконується, контролюється та управляється.
- **Тест-план** – це документ, що описує обсяг, підхід, ресурси та графік запланованих тестових заходів [1].
- **Документ з вимогами до продукту**, PRD – це документ, що описує продукт, який буде створювати компанія. Метою документа з вимогами до продукту (PRD) або специфікації продукту є чітке й недвозначне

визначення мети продукту, його функцій, функціональності та поведінки.

- **Специфікація** (технічні характеристики) – це документ, який визначає, в ідеалі, в повній, точній і перевіряємій формі, вимоги, дизайн, поведінку, або інші характеристики компонента або системи, а також, часто, процедури для визначення того, чи були виконані ці положення.
- **Документ з функціональними специфікаціями**, FSD – це документ, який використовується для детального опису передбачуваних можливостей продукту, зовнішнього вигляду та взаємодії з користувачами і розробникам програмного забезпечення.
- **Специфікація вимог до програмного забезпечення**, SRS – це документ, який максимально повно описує очікувану поведінку програмної системи [10].
- **Архітектура**. **Дизайн**. Архітектура програмного забезпечення для системи – це структура або структури системи, які містять елементи, їх зовні видиму поведінку та відношення між ними. Архітектура – це дизайн, але не весь дизайн є архітектурою [11].
- **Тестовий приклад** – це набір вхідних значень, передумов виконання, очікуваних результатів і пост-умов виконання, розроблених для конкретної мети або умови тестування.
- **Набір тестів** – це набір із кількох тестових випадків для компонента або системи, що тестується, де пост-умова одного тесту часто використовується як умова для наступного [1].
- **Технічні характеристики** – це скрипти, вихідний код, мова визначення даних тощо.
- **Проєктна документація** – це інші очікування та результати, які не є частиною програмного забезпечення, що реалізує команда, але є необхідними для успішного завершення проєкту в цілому [10].

- **Документація користувача** – це документація, яка належить до документації по продукту або послузі, що надаються кінцевим користувачам [10].
- **Документ з ринкових вимог**, MRD. MRD детально описує цільові сегменти ринку та питання, пов'язані з комерційним успіхом [10].

Джерела та шляхи виявлення вимог

- Основні техніки для збору та виявлення вимог [12]:
  - інтерв'ю;
  - робота із фокусними групами;
  - анкетування;
  - семінари та мозковий штурм;
  - спостереження;
  - прототипування;
  - аналіз документів;
  - моделювання процесів та взаємодій;
  - самостійний опис.

Рівні та типи вимог [13]

- **Бізнес-вимога** – це все, що описує фінансову, ринкову чи іншу бізнес-вигоду, яку клієнти чи організація, що розвивається, бажають отримати від продукту.
- **Бачення та масштаби**. У документі про бачення та обсяги бізнес-вимоги об'єднані в єдиний продукт, який створює основу для подальших робіт з розробки.
- **Вимоги користувачів –** це загальні формулювання цілей користувачів або бізнес-завдань, які користувачі повинні виконувати.
- **Варіант використання –** це послідовність транзакцій у діалозі між дійовою особою та компонентом або системою з відчутним результатом, де дійова особа може бути користувачем або будь-що, що може обмінюватися інформацією з системою.

- **Історія користувача –** це вимога користувача або бізнес-вимога високого рівня, яка зазвичай використовується в гнучкій розробці програмного забезпечення, та зазвичай складається з одного або більше речень повсякденною чи діловою мовою, що відображає, яка функціональність потрібна користувачеві, будь-які нефункціональні критерії, а також містить критерії прийнятності.
- **Сценарій –** це гіпотетична історія, яка використовується, щоб допомогти людині зрозуміти складну проблему або систему.
- **Бізнес-правило –** це твердження, яке визначає або обмежує певний аспект бізнесу. Воно призначене для затвердження бізнес-структури або для контролю або впливу на поведінку бізнесу. Бізнес-правило виражає конкретні обмеження для створення, оновлення та видалення постійних даних в інформаційній системі.
- **Атрибут якості** – це особливість або характеристика, яка впливає на якість продукту.
- **Функціональна вимога** – це вимога, що визначає функцію, яку повинен виконувати компонент або система. Функціональні вимоги описують спостережувану поведінку системи за певних умов, і дії, які система дозволить виконувати користувачам.
- **Нефункціональна вимога** – це вимога, яка стосується не функціональності, а таких атрибутів, як надійність, ефективність, зручність використання, ремонтопридатність та портативність.
- **Обмеження, стиснення** – це обмеження дизайну та реалізації, що законно мінімізують можливості, доступні розробнику.
- **Вимоги до зовнішнього інтерфейсу** – це вимоги, які описують зв’язки між системою та рештою всесвіту. Вони включають інтерфейси для користувачів, обладнання та інші програмні системи.
- **Вимоги до даних** – це вимоги, які описують формат, тип даних, дозволені значення або значення за замовчуванням для елемента даних; склад складної структури бізнес-даних або звіт, який буде створено.

- **Інструмент управління вимогами** – це інструмент, який підтримує запис вимог, атрибутів вимог (наприклад, пріоритет, відповідальний за знання) та анотації, а також полегшує відстеження через рівні вимог та управління змінами вимог. Деякі інструменти керування вимогами також надають засоби для статичного аналізу, наприклад, перевірку узгодженості та порушення попередньо визначених правил вимог.

Властивості якісних вимог

- Властивості якісної вимоги [10]:
  - завершеність (completeness);
  - атомарність, поодинокість (atomicity);
  - несуперечність, послідовність (consistency);
  - недвозначність (unambiguousness, clearness);
  - виконуваність (feasibility);
  - обов'язковість, потрібність (obligatoriness) та актуальність (up-to-date);
  - простежуваність (traceability);
  - модифікованість (modifiability);
  - проранжованість за важливістю, стабільністю, терміновістю (ranked for importance, stability, priority);
  - коректність (correctness) і перевіряємість (verifiability).
- **Простежуваність** – це можливість ідентифікувати пов’язані елементи в документації та програмному забезпеченні.
- **Вертикальна простежуваність** – це простежуваність вимог по рівнях документації розробки до компонентів.
- **Горизонтальна простежуваність** – це простежуваність вимог до рівня тестування по рівнях тестової документації.
- **Інструмент управління вимогами** – це інструмент, який підтримує запис вимог, атрибутів вимог та анотації, а також полегшує відстеження через рівні вимог і зміни вимог управління.

- **Матриця простежуваності** – це двовимірна таблиця, яка співвідносить дві сутності, тобто дозволяє простежити зв'язки одного об'єкта з іншим, що дає змогу визначити досягнуте охоплення та оцінити вплив запропонованих змін.

Техніки тестування вимог

Основні техніки тестування вимог [1]:

- взаємний перегляд (peer review): побіжний перегляд (walkthrough), технічний перегляд (technical review), формальна інспекція (inspection);
- запитання;
- тест-кейси та чек-листи;
- вивчення поведінки системи;
- рисунки (графічне надання);
- прототипування.

Закріплення матеріалу

1. Розкрити суть поняття «вимога» та пояснити в чому полягає їх важливість.
2. Як залежить вартість виправлення помилки від моменту її виправлення?
3. Назвати види документації та їх складові.
4. Назвати основні техніки для збору та виявлення вимог та пояснити їх суть.
5. Назвати рівні та типи вимог.
6. Назвати властивості якісної вимоги.
7. Назвати способи виявлення та усунення проблем із завершеністю, атомарністю, несуперечністю, недвозначністю, виконуваністю, обов’язковістю, простежуваністю, кодифікованістю, проранжованістю та коректністю.
8. Назвати та пояснити суть технік тестування вимог.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.2 КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ

Лекція 4

РІЗНОВИДИ ТА НАПРЯМИ ТЕСТУВАННЯ

Мета лекції: Розгляд спрощеної та розгорнутої класифікації тестування.

Зміст лекції

1. Спрощена класифікація тестування.
2. Детальна класифікація тестування.
    2.1. Схема класифікації тестування.
    2.2. Класифікація за запуском коду на виконання.
    2.3. Класифікація за доступом до коду та архітектури додатку.
    2.4. Класифікація за ступенем автоматизації.
    2.5. Класифікація за рівнем деталізації додатку (за рівнем тестування).
    2.6. Класифікація за (спаданням) ступенем важливості функцій, що тестуються (за рівнем функціонального тестування).
    2.7. Класифікація за принципами роботи з додатком.
    2.8. Класифікація за природою додатку.
    2.9. Класифікація за фокусуванням на рівні архітектури додатку.
    2.10. Класифікація за залученням кінцевих користувачів.
    2.11. Класифікація за ступенем формалізації.
    2.12. Класифікація за цілями та задачами.
    2.13. Класифікація за техніками та підходами.
    2.14. Класифікація за моментом виконання (хронології).
3. Альтернативні і додаткові класифікації тестування.
4. Класифікація за приналежністю до тестування за методом білої та чорної скринь.

Спрощена класифікація тестування

- Спрощена класифікація тестування [1, 14]:
  - за запуском коду на виконання: статичне тестування, динамічне тестування;
  - за доступом до коду та архітектури додатку: метод білої скриньки, метод чорної скриньки, метод сірої скриньки;
  - за ступенем автоматизації: ручне тестування, автоматизоване тестування;
  - за рівнем деталізації додатку (за рівнем тестування): модульне (компонентне) тестування, інтеграційне тестування, системне тестування;
  - за (спаданням) ступеня важливості функцій, що тестуються (за рівнем функціонального тестування): димове тестування, тестування критичного шляху, розширене тестування;
  - за принципами роботи з додатком: позитивне тестування, негативне тестування.

Детальна класифікація тестування

Схема класифікації тестування

- Еталонної класифікації тестування не існує.
- В матеріалах ISTQB наведено найбільш узагальнений та загальноприйнятий погляд на питання класифікації тестування, але й там немає єдиної схеми, яка б поєднувала всі варіанти класифікації.
- Детальна класифікація тестування [1, 14]:
  - за запуском коду на виконання: статичне, динамічне;
  - за доступом до коду та архітектури додатку: метод білої скриньки, метод чорної скриньки, метод сірої скриньки;
  - за ступенем автоматизації: ручне, автоматизоване;
  - за рівнем деталізації додатку (за рівнем тестування): модульне (компонентне), інтеграційне, системне;

- за (спаданням) ступеня важливості функцій, що тестуються (за рівнем функціонального тестування): димове, критичного шляху, розширене;
- за природою додатку: веб, мобільне, настільне;
- за фокусуванням на рівні архітектури додатку: рівень представлення, рівень бізнес-логіки, рівень даних;
- за залученням кінцевих користувачів: альфа, бета, гама;
- за ступенем формалізації: на основі тест-кейсів, дослідницьке, вільне (інтуїтивне);
- за цілями та задачами: за принципами роботи з додатком (позитивне, негативне), функційне, нефункційне, інсталяційне, регресійне, повторне, приймальне, за зручністю використання, за доступністю, за інтерфейсом, за безпекою, за інтернаціоналізацією, за локалізацією, за сумісністю (конфігураційне, кросбраузерне), порівняльне, демонстраційне, вичерпне, за надійністю, за відновлюваністю, за даними і базами даних, за використанням ресурсів, за продуктивністю (навантажувальне, масштабованості, об’ємне, стресове, конкурентне), за техніками автоматизації (під управлінням даних, ключовими словами, поведінкою), на основі структур коду (виразів, гілок, умов, комбінацій умов, рішень, рішаючи умов, шляхів);
- за техніками та підходами: позитивне, негативне, на основі досвіду тестувальника (дослідницьке, вільне), по ступеню втручання в роботу додатку (інвазивне, неінвазивне), на основі вибору вхідних даних (класів еквівалентності, граничних умов, ортогональних масивів, доменне, попарне), на основі коду (за потоком управління, за потоком даних, за діаграмою або таблицею станів, інспекція коду), на основі джерел помилок (передбачення помилок, евристична оцінка, додавання помилок, мутаційне), на основі

середовища виконання (тестування в процесі розробки, операційне), на основі поведінки додатку (за таблицею прийняття рішень, за діаграмою або таблицею станів, за специфікаціями, за моделями поведінки додатку, на основі варіантів використання, паралельне, на основі випадкових даних, A/B);

– за моментом виконання: загальна універсальна логіка послідовності тестування (позитивне просте, негативне просте, позитивне складне, негативне складне), за ієрархією компонентів (висхідне, низхідне, гібридне), за концентрацією уваги на вимогах і їх складових (тестування вимог, тестування функцій них складових, тестування нефункційних складових);
– типові загальні сценарії: типовий загальний сценарій 1 (димове, критичного шляху, розширене), типовий загальний сценарій 1 (модульне, інтеграційне, системне), типовий загальний сценарій 3 (альфа, бета, гама).

Класифікація із запуску коду на виконання

- **Статичне тестування** (static testing) – це тестування артефакту розробки програмного забезпечення, наприклад, вимог, дизайну або коду, без виконання цих артефактів, наприклад, огляди чи статичний аналіз.
- **Динамічне тестування** (dynamic testing) –це тестування, що включає програмне забезпечення компонента або системи.

Класифікація з доступу до коду та архітектури програми

- **Метод білої скриньки** (white box testing, open box testing, clear box testing, glass box testing) – це тестування на основі аналізу внутрішньої структури компонента чи системи.
- **Метод чорної скриньки** (black box testing, closed box testing, specificationbased testing) – це тестування, функціональне або нефункціональне, без посилання на внутрішню структуру компонента або системи.

- **Метод сірої скриньки** (gray box testing) – це метод тестування програмного забезпечення, який є комбінацією методу тестування чорної скриньки та методу тестування білої скриньки. У Grey Box Testing внутрішня структура частково відома. Це передбачає доступ до внутрішніх структур даних і алгоритмів для цілей проєктування тестових випадків, але тестування проводиться на рівні користувача або на рівні чорної скриньки.
- **Проєктне тестування** – це підхід до тестування, при якому тестові сценарії розробляються на основі архітектури та/або детального проєкту компонента або системи.
- **Тестування на основі вимог** – це підхід до тестування, у якому тестові випадки розробляються на основі цілей тестування та умов тестування, що випливають із вимог.
- Переваги методу білої скриньки:
  - показує приховані проблеми та спрощує їх діагностику;
  - допускає досить просту автоматизацію тест-кейсів та їх виконання на ранніх стадіях розвитку проєкту;
  - має розвинену систему метрик, збір та аналіз яких легко автоматизується;
  - стимулює розробників до написання якісного коду;
  - багато технік цього методу є перевіреними, добре себе зарекомендували рішеннями, що базуються на строгому технічному підході.
- Недоліки методу білої скриньки:
  - не може виконуватися тестувальниками, які не мають достатніх знань у галузі програмування;
  - тестування сфокусовано на реалізованій функціональності, що підвищує ймовірність пропуску нереалізованих вимог;

  - поведінка додатку досліджується у відриві від реального середовища виконання та не враховує його вплив;
  - поведінка додатку досліджується у відриві від реальних сценаріїв користувача.
- Переваги методу чорної скриньки:
  - тестувальник не зобов'язаний мати (глибокі) знання в області програмування;
  - поведінка додатку досліджується в контексті реального середовища виконання та враховує його вплив;
  - поведінка програми досліджується в контексті реальних сценаріїв користувача;
  - тест-кейси можна створювати вже на стадії появи стабільних вимог;
  - процес створення тест-кейсів дозволяє виявити дефекти вимог;
  - допускає створення тест-кейсів, які можна багаторазово використати на різних проєктах.
- Недоліки методу чорної скриньки:
  - можливе повторення частини тест-кейсів, які вже виконані розробниками;
  - висока ймовірність того, що частина можливих варіантів поведінки програми залишиться непротестованою;
  - для розробки високоефективних тест-кейсів потрібна якісна документація;
  - діагностика виявлених дефектів складніша порівняно з техніками методу білої скриньки;
  - у зв'язку з широким вибором технік та підходів ускладнюється планування та оцінка трудовитрат;
  - у разі автоматизації можуть знадобитися складні дорогі інструментальні засоби.

- Метод сірої скриньки поєднує переваги та недоліки методів білої та чорної скриньки.

Класифікація за рівнем автоматизації

- **Ручне тестування** (manual testing) – це тестування, що здійснюється тестувальником, який виконує всі дії з додатком, що тестується вручну, крок за кроком, і вказує, чи був конкретний крок виконаний успішно чи ні. Ручне тестування завжди є частиною будь-якого тестування. Це особливо корисно на початковому етапі розробки програмного забезпечення, коли програмне забезпечення та його інтерфейс користувача недостатньо стабільні, і починати автоматизацію не має сенсу.
- **Автоматизоване тестування** (automated testing, test automation) – це використання програмного забезпечення для керування виконанням тестів, порівняння фактичних результатів із прогнозованими результатами, налаштування попередніх умов тестування та інших функцій управління тестуванням та звітів про тестування. Зазвичай автоматизація тестування включає автоматизацію вже існуючого ручного процесу, в якому використовується формалізований процес тестування.
- Переваги автоматизованого тестування:
  - швидкість виконання тест-кейсів може в рази та на порядки перевищувати можливості людини;
  - відсутність впливу людського фактора у процесі виконання тест-кейсів;
  - мінімізація витрат при багаторазовому виконанні тест-кейсів;
  - здатність засобів автоматизації виконати тест-кейси, у принципі непосильні для людини через свою складність, швидкість або інші фактори;

  - здатність засобів автоматизації збирати, зберігати, аналізувати, агрегувати та представляти у зручній для сприйняття людиною формі колосальні обсяги даних;
  - здатність засобів автоматизації виконувати низькорівневі дії з програмою, операційною системою, каналами передачі даних тощо.
- Недоліки автоматизованого тестування:
  - необхідний висококваліфікований персонал через те, що автоматизація – це «проєкт усередині проєкту»;
  - високі витрати на складні засоби автоматизації, розробку та супровід коду тест-кейсів;
  - автоматизація вимагає більш ретельного планування та управління ризиками, оскільки в іншому випадку проєкту може бути завдано серйозної шкоди;
  - засобів автоматизації дуже багато, що ускладнює проблему вибору того чи іншого засобу і може спричинити фінансові витрати (і ризики), необхідність навчання персоналу (або пошуку фахівців);
  - у разі відчутної зміни вимог, зміни технологічного домену, переробки інтерфейсів (як користувача, так і програмних) багато тест-кейси стають безнадійно застарілими і вимагають створення наново.
- **Покриття, тестове покриття** – це ступінь, виражена у відсотках, до якої зазначений елемент покриття був випробуваний набором тестів.

Класифікація за рівнем деталізації програми (за рівнем тестування)

- **Модульне (компонентне) тестування** (unit testing, module testing, component testing) – це тестування окремих програмних компонентів.
- **Інтеграційне тестування** (integration testing) – це тестування, яке проводиться для виявлення дефектів інтерфейсів і взаємодій між інтегрованими компонентами чи системами.

- **Компонентне інтеграційне тестування** (component integration testing) – це тестування, яке проводиться для виявлення дефектів в інтерфейсах і взаємодії інтегрованих компонентів.
- **Парне інтеграційне тестування** (pairwise integration testing) – це форма інтеграційного тестування, призначена для пар компонентів, які працюють разом.
- **Системне інтеграційне тестування** (system integration testing) – це тестування інтеграції систем та пакетів; тестування інтерфейсів із зовнішніми організаціями.
- **Інкрементне тестування** (incremental testing) – це тестування, при якому компоненти або системи інтегруються і тестуються по одному або кілька за раз, поки всі компоненти або системи не будуть проінтегровані і протестовані.
- **Тестування інтерфейсу** (interface testing) – це тип інтеграційного тесту, пов'язаний із тестуванням інтерфейсів між компонентами або системами.
- **Потокове тестування** (thread testing) – це підхід до компонентного інтеграційного тестування, у якому послідовна інтеграція компонентів слідує за реалізацією підмножин вимог, на відміну від інтеграції компонентів за рівнями ієрархії.
- **Тестування системи** (system testing) – це процес тестування інтегрованої системи для перевірки відповідності заданим вимогам.
- **Рівень тестування** (test level) – це група тестових дій, які організовуються та управляються разом. Рівень тестування пов'язаний з обов'язками у проєкті. Прикладами рівнів тестування є компонентне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

Класифікація за (спаданням) ступенем важливості функцій, що тестуються (за рівнем функціонального тестування)

- **Димове тестування** (smoke test) – це підмножина всіх певних/запланованих тестових випадків, які охоплюють основні функціональні можливості компонента або системи, щоб переконатися, що найважливіші функції програми працюють, але не торкаючись дрібніших деталей.
- **Вступне тестування** (intake test) – це особливий випадок димового тестування, щоб вирішити, чи готовий компонент або система до детального і подальшого тестування. Вступний тест зазвичай проводиться на початку етапу виконання тесту.
- **Тестування з перевірки збірки** (build verification test) – це набір автоматичних тестів, які підтверджують цілісність кожної нової збірки та перевіряють її ключову/основну функціональність, стабільність та тестованість. Це галузева практика, коли збірки випускаються дуже часто (наприклад, agile-проєкти), і таке тестування запускається при кожній новій збірці до того, як збірку буде випущено для подальшого тестування.
- **Критичний шлях** (critical path) – це найдовша послідовність заходів у плані проєкту, яка має бути виконана вчасно, щоб проєкт завершився у встановлений термін. Дія на критичному шляху не може бути розпочата, поки попередня дія не буде завершена; якщо вона затримується на один день, весь проєкт буде відкладено на день, якщо тільки дія, що слідує за відкладеною дією, не буде завершена на день раніше.
- **Розширений тест** (extended test) – ідея полягає в тому, щоб розробити комплексний набір тестів прикладної системи шляхом моделювання основних можливостей у вигляді розширених варіантів використання.

Класифікація за принципами роботи із додатком

- **Позитивне тестування** (positive testing) – це процес тестування, при якому система перевіряється на відповідність припустимим вхідним даним. У цьому тестуванні тестувальник завжди перевіряє лише припустимий набір значень і перевіряє, чи очікувано веде себе додаток при очікуваних вхідних даних. Основна мета цього тестування полягає в тому, щоб перевірити, чи програмне забезпечення не показує помилку, коли цього бути не повинно, і показує помилку, коли це має бути. Таке тестування має проводитися, з позитивної точки зору та виконувати лише позитивний сценарій. Позитивне тестування завжди намагається довести, що даний продукт і проєкт завжди відповідають вимогам і специфікаціям.
- **Негативне тестування** (negative testing) – це тести, спрямовані на те, щоб показати, що компонент або система не працює. Негативне тестування пов’язане зі ставленням тестувальників, а не з конкретним підходом до тестування або технікою розробки тестів, наприклад, тестування з неприпустимими вхідними значеннями або виключеннями.
- **Недійсне тестування** (invalid testing) – це тестування з використанням вхідних значень, які повинні бути відхилені компонентом або системою.

Класифікація за природою програми

- **Тестування веб-додатків** (web-applications testing) – це тестування, яке пов'язане з інтенсивною діяльністю в області тестування сумісності, тестування продуктивності, автоматизації тестування з використанням широкого спектру інструментальних засобів.
- **Тестування мобільних додатків** (mobile applications testing) – це тестування, яке вимагає підвищеної уваги до тестування сумісності, оптимізації продуктивності, автоматизації тестування із застосуванням емуляторів мобільних пристроїв.

- **Тестування настільних додатків** (desktop applications testing) – це тестування з особливостями, які залежать від предметної області додатку, нюансів архітектури, ключових показників якості тощо.

Класифікація за фокусуванням на рівні архітектури програми

- **Тестування рівня представлення** (presentation tier testing) – це тестування, яке сконцентроване на тій частині програми, що відповідає за взаємодію із «зовнішнім світом». Тут досліджуються питання зручності використання, швидкості відгуку інтерфейсу, сумісності із браузерами, коректності роботи інтерфейсів.
- **Тестування рівня бізнес-логіки** (business logic tier testing) – це тестування, яке відповідає за перевірку основного набору функцій програми та будується на базі ключових вимог до додатку, бізнес-правил та загальної перевірки функціональності.
- **Тестування рівня даних** (data tier testing) – це тестування, яке сконцентровано на тій частини програми, що відповідає за збереження та деяку обробку даних. Тут особливий інтерес має тестування даних, перевірка дотримання бізнес-правил, тестування продуктивності.
- **Багаторівнева архітектура додатків** – це клієнт-серверна архітектура, в якій поділяються функції представлення, обробки та зберігання даних.

Класифікація із залученням кінцевих користувачів

- **Альфа-тестування** (alpha testing) – це імітаційне або фактичне експлуатаційне тестування потенційними користувачами/замовниками або незалежною тестовою групою на сайті розробників, але за межами організації розробників. Альфа-тестування часто використовується для готового програмного забезпечення як форма внутрішнього приймального тестування.
- **Бета-тестування** (beta testing) – це операційне тестування потенційними та/або існуючими користувачами/замовниками на зовнішньому сайті, не пов’язаному з розробниками, для визначення того, чи задовольняє

компонент або система потребам користувача/замовника та чи вписується в бізнес-процеси. Бета-тестування часто використовується як форма зовнішнього приймального тестування готового програмного забезпечення для отримання зворотного зв'язку від ринку.

- **Гамма-тестування** (gamma testing) – це тестування, яке виконується, коли програмне забезпечення готове до випуску із зазначеними вимогами, це тестування здійснюється безпосередньо, пропускаючи всі внутрішні дії з тестування. Програмне забезпечення майже готове до фінальної версії. Ніякої розробки функцій або вдосконалення програмного забезпечення не проводиться, а виправлення помилок є єдиним кодом. Гамма-перевірка виконується, коли додаток готовий до випуску відповідно до зазначених вимог, і ця перевірка виконується безпосередньо.

Класифікація за ступенем формалізації

- **Тестування на основі тест-кейсів** (скриптове тестування, scripted testing, test case based testing) – тестування, при якому виконання тесту здійснюється шляхом дотримання попередньо задокументованої послідовності тестів.
- **Дослідницьке тестування** (exploratory testing) – неформальна техніка розробки тестів, коли тестувальник активно контролює дизайн тестів під час виконання цих тестів і використовує інформацію, отриману під час тестування, для розробки нових і кращих тестів.
- **Сесійне тестування** (session-based testing) – підхід до тестування, при якому тестові дії плануються як безперервні сеанси розробки та виконання тестів; часто використовується разом із дослідницьким тестуванням.
- **Тестування на основі чек-листів** (checklist-based testing) – метод розробки тестів, заснований на досвіді, за допомогою якого досвідчений тестувальник використовує високорівневий список елементів, які

потрібно визначити, перевірити чи запам'ятати, або набір правил чи критеріїв, за якими продукт має бути перевірений.

- **Вільне (інтуїтивне) тестування** (ad hoc testing) – тестування, яке проводиться неофіційно; формальна підготовка до тесту не проводиться, не використовується визнана техніка розробки тестів, немає очікувань на результати та діяльність з виконання тестів визначається довільно.
- **Тестування на основі досвіду** (experience-based testing) – це тестування на основі досвіду, знань та інтуїції тестувальника.

Класифікація за цілями та завданнями

- **Функціональне тестування** (functional testing) – тестування на основі аналізу специфікації функціональності компонента або системи.
- **Тестування функціональності** (functionality testing) – процес тестування з метою визначення функціональності програмного продукту (здатність програмного продукту забезпечувати функції, які задовольняють заявленим і передбачуваним потребам, коли програмне забезпечення використовується за певних умов).
- **Нефункціональне тестування** (non-functional testing) – тестування атрибутів компонента або системи, які не стосуються функціональності, наприклад надійність, ефективність, зручність використання, ремонтопридатність і портативність.
- **Інсталяційне тестування** (installation testing, installability testing) – процес перевірки можливості встановлення програмного продукту. Можливість встановлення – це здатність програмного продукту встановлюватись у певному середовищі.
- Ситуації, в яких інсталяційне тестування перевіряє сценарії та аспекти роботи інсталятора:
  - нове середовище виконання;
  - оновлення існуючої версії;
  - зміна поточної версії на більш стару;

- повторна установка додатку з метою усунення проблем, що виникли;
- повторний запуск інсталяції після помилки, що призвела до неможливості продовження інсталяції;
- видалення додатку;
- встановлення нового додатка із родини додатків;
- автоматична інсталяція без участі користувача.

- **Регресійне тестування** (regression testing) – тестування попередньо протестованої програми після модифікації, щоб переконатися, що дефекти не були внесені або виявлені в незмінних областях програмного забезпечення в результаті внесених змін. Виконується при зміні програмного забезпечення або його середовища.
- **Повторне тестування, тестування на підтвердження** (re-testing, confirmation testing) – тестування, що запускає тестові випадки, які не вдалося виконати під час останнього запуску, щоб перевірити успішність коригувальних дій.
- **Приймальне тестування** (acceptance testing) – формальне тестування з врахуванням потреб користувачів, вимог та бізнес-процесів, яке проводиться, щоб визначити, чи задовольняє система критеріям прийняття, і щоб користувач, клієнти чи інший уповноважений орган могли визначити, чи приймати систему чи ні.
- **Виробниче приймальне тестування** (factory aceptance testing) – приймальні випробування, що проводяться на місці, де продукт розробляється, та виконуються працівниками організації-постачальника, щоб визначити, чи відповідає компонент або система вимогам, як правило, включаючи апаратне та програмне забезпечення.
- **Операційне приймальне тестування** (operational aceptance testing, production acceptance testing) – операційне тестування на етапі приймального випробування, яке зазвичай виконується в

(симульованому) операційному середовищі персоналом операційного та/або системного адміністрування, з особливою увагою до експлуатаційних аспектів, наприклад можливості відновлення, поведінки ресурсів, можливост встановлення та технічної відповідності.

- **Підсумкове приймальне тестування** (site aceptance testing) – приймальне тестування користувачами/замовниками на їх сайті, щоб визначити, чи відповідає компонент або система потребам користувача/замовника та чи вписується в бізнес-процеси, як правило, включаючи апаратне та програмне забезпечення.
- **Операційне тестування** (operational testing) – тестування, проведене для оцінки компонента або системи в робочому середовищі.
- **Операційне середовище** (operational environment) – апаратні та програмні продукти, які встановлюються на сайтах користувачів або клієнтів, де будуть використовуватися компонент або система, що тестується. Програмне забезпечення може включати операційні системи, системи управління базами даних та інші програми.
- З**ручність використання** (usability testing) – здатність програмного забезпечення бути зрозумілим, засвоєним, використаним і привабливим для користувача при використанні за певних умов.
- **Зрозумілість** (understandability) – здатність програмного продукту дозволити користувачеві зрозуміти, чи підходить програмне забезпечення, і як його можна використовувати для певних завдань і умов застосування.
- **Навчання** (learnability) – здатність програмного продукту дозволити користувачеві вивчити його застосування.
- **Працездатність** (operability) – здатність програмного продукту дозволити користувачеві керувати ним.
- **Привабливість** (attractiveness) – здатність програмного продукту бути привабливим для користувача.

- **Тестування доступності** (accessibility testing) – тестування для визначення простоти, з якою користувачі з обмеженими можливостями можуть використовувати компонент або систему.
- **Тестування інтерфейсу** (interface testing) – тип інтеграційного тесту, який стосується тестування інтерфейсів між компонентами або системами.
- **Тестування інтерфейсу прикладного програмування, тестування API** (API testing) – тестування, яке здійснюється шляхом подання команд програмному забезпеченню, що тестується, безпосередньо за допомогою програмних інтерфейсів додатку.
- **Тестування інтерфейсу командного рядка, тестування CLI** (CLI testing) – тестування, яке виконується шляхом подання команд програмному забезпеченню, яке тестується, за допомогою спеціального інтерфейсу командного рядка.
- **Тестування графічного інтерфейсу** (GUI testing) – тестування, яке здійснюється шляхом взаємодії з тестованим програмним забезпеченням через графічний інтерфейс користувача.
- **Тестування безпеки** (security testing) – тестування для визначення безпеки програмного продукту.
- **Глобалізація** (globalization) – процес розробки ядра програми, чиї функції та дизайн коду не ґрунтуються лише на одній мові чи локалі. Натомість їх конструкція розроблена для введення, відображення та виведення визначеного набору мовних скриптів з підтримкою Unicode, і даних, що належать певним локалям.
- **Можливість локалізації** (localizability) – розробка базового програмного коду та ресурсів, що дозволяє локалізувати програму на різні мовні версії без будь-яких змін у вихідному коді.
- **Тестування локалізації** (localization testing) перевіряє якість локалізації продукту для певної цільової культури/локалу. Цей тест базується на

результатах тестування глобалізації, яке перевіряє функціональну підтримку цієї конкретної культури/мови. Тестування локалізації можна виконати лише на локалізованій версії продукту

- **Тестування на сумісність, тестування функціональної сумісності** (compatibility testing, interoperability testing) – процес тестування для визначення функціональної сумісності програмного продукту (здатності взаємодіяти з одним або кількома вказаними компонентами або системами).
- **Конфігураційне тестування, портативне тестування** (configuration testing) – процес тестування для визначення портативності програмного продукту (легкість, з якою програмний продукт може бути перенесений з одного апаратного чи програмного середовища в інше).
- **Міжбраузерне тестування** (cross-browser testing) – тестування, яке допомагає вам переконатися, що ваш веб-сайт або веб-програма правильно функціонують у різних веб-браузерах. Як правило, QA інженери створюють окремі тести для кожного браузера або створюють тести, які використовують багато умовних операторів, що перевіряють тип використовуваного браузера та виконують команди, які його стосуються.
- **Мобільне тестування** (mobile testing) – це тестування з кількома операційними системами (і різними версіями кожної ОС, особливо з Android), декількома пристроями (різні марки та моделі телефонів, планшетів, фаблетів), кількома операторами зв'язку (у тому числі міжнародними), різними швидкостями передачі даних (3-5G, LTE, Wi-Fi), кількома розмірами екрана (а також роздільними здатностями і співвідношеннями сторін), кількома елементами керування вводом (включаючи вічні фізичні клавіатури BlackBerry) і безліч технологій – GPS, акселерометри – які веб- та настільні програми майже ніколи не використовують.

- **Тестування відповідності, тестування на відповідність, тестування нормативів** (compliance testing, conformance testing, regulation testing) – процес тестування для визначення відповідності компонента або системи (здатність дотримуватися стандартів, конвенцій або правил у законах та подібних приписах).
- **Якість даних** (data quality) – атрибут даних, який вказує на правильність деяких попередньо визначених критеріїв, наприклад, бізнес-очікувань, вимог до цілісності даних, узгодженості даних.
- **Тестування цілісності бази даних** (database integrity testing) – тестування методів і процесів, які використовуються для доступу до даних (бази) та керування ними, щоб переконатися, що методи доступу, процеси та правила даних функціонують належним чином і що під час доступу до бази даних дані не пошкоджені, несподівано не видалені, оновлені чи створені.
- **Тестування використання ресурсів, тестування зберігання** (resource utilization testing) – процес тестування для визначення використання ресурсів програмного продукту.
- **Тестування ефективності** (efficiency testing) – процес тестування для визначення ефективності програмного забезпечення прпродукт (здатність процесу надавати передбачуваний результат відносно кількості використаних ресурсів).
- **Тестування зберігання** (storage testing) – це визначення того, чи використовують певні умови обробки більше пам’яті, ніж передбачалося.
- **Порівняльне тестування** (comparison testing) – тестування, під час якого порівнюються слабкі та сильні сторони програмного забезпечення з продуктами конкурентів.
- **Демонстраційне тестування** (qualification testing) – формальне (офіційне) тестування, яке зазвичай проводить розробник для

споживача, щоб продемонструвати, що програмне забезпечення відповідає встановленим вимогам.

- **Вичерпне тестування** (exhaustive testing) – тестовий підхід, у якому набір тестів містить усі комбінації вхідних значень та передумов.
- **Тестування надійності** (reliability testing) – процес тестування для визначення надійності програмного продукту (здатності програмного продукту виконувати свої необхідні функції за встановлених умов протягом певного періоду часу, або за задану кількість операцій).
- **Тестування відновлюваності** (recoverability testing) – процес тестування для визначення можливості відновлення програмного продукту (здатність програмного продукту відновити заданий рівень продуктивності та відновити дані, які безпосередньо постраждали в разі збою).
- **Тестування відмовостійкості** (failover testing) – тестування шляхом імітування режимів відмови або фактичного виникнення відмов у контрольованому середовищі. Після збою механізм аварійного перемикання тестується, щоб гарантувати, що дані не будуть втрачені або пошкоджені, а також будуть підтримуватись всі узгоджені рівні обслуговування (наприклад, доступність функцій або час відгуку).
- **Тестування продуктивності** (performance testing) – процес тестування для визначення продуктивності програмного продукту.
- **Навантажувальне тестування** (load testing) – тип тестування продуктивності, яке проводиться для оцінки поведінки компонента або системи зі збільшенням навантаження, наприклад кількості паралельних користувачів і/або кількості транзакцій, щоб визначити, яке навантаження може обробляти компонент або система.
- **Ємнісне тестування** (capacity testing) – тестування, щоб визначити, скільки користувачів та/або транзакцій дана система підтримуватиме та при цьому відповідатиме вимогам продуктивності.

- **Тестування масштабованості** (scalability testing) – тестування для визначення масштабованості програмного продукту (можливість оновлення програмного продукту задля роботи з підвищеними навантаженнями).
- **Об'ємне тестування** (volume testing) – тестування, коли система піддається впливу великих об’ємів даних.
- **Стресове тестування** (stress testing) – тип тестування продуктивності, що проводиться для оцінки системи або компонента на рівні або за межами очікуваних чи визначених робочих навантажень, або з обмеженою доступністю ресурсів, таких як доступ до пам’яті чи серверів.
- **Тестуванням на руйнування програмного забезпечення** (destructive testing) –тестування, що забезпечує правильну або передбачувану поведінку програмного забезпечення, коли програмне забезпечення піддається неналежному використанню або неправильному вводу, спробам призвести до збою програмного продукту, спробам зламати програмний продукт, перевірити надійність програмного продукту.
- **Конкурентне тестування** (concurrency testing) – тестування для визначення того, як компонент або система обробляє появу двох або більше дій протягом одного і того ж проміжку часу, яке досягається шляхом чергування дій або одночасного виконання.

Класифікація за техніками та підходами

- **Тестування на основі досвіду тестувальника, сценаріїв, чек-листів:** дослідницьке, вільне (інтуїтивне) тестування.
- **Класифікація за ступенем втручання у роботу додатку:** інвазивне, неінвазивне тестування.
- **Класифікація з технік автоматизації:** тестування під управлінням даними, тестування під управлінням ключовими словами, тестування під управлінням поведінкою.

- **Класифікація на основі (знання) джерел помилок:** тестування передбаченням помилок, евристична оцінка, мутаційне тестування, тестування додаванням помилок.
- **Класифікація на основі вибору вхідних даних:** тестування на основі класів еквівалентності, тестування на основі граничних умов, доменне тестування, попарне тестування, тестування на основі ортогональних масивів.
- **Класифікація на основі середовища виконання:** тестування в процесі розробки, операційне тестування, тестування на основі коду (тестування по потоку управління, тестування по потоку даних, тестування за діаграмою або таблицею станів, інспекція коду), тестування на основі структур коду (тестування на основі виразів, тестування на основі гілок, тестування на основі умов, тестування на основі комбінацій умов, тестування на основі окремих умов, що породжують розгалуження, тестування на основі рішень, тестування на основі шляхів), тестування на основі (моделей) поведінки програми (тестування за таблицею прийняття рішень, тестування за діаграмою або таблицею станів, тестування за специфікаціями, тестування за моделями поведінки програми, тестування на основі варіантів використання, паралельне тестування, тестування на основі випадкових даних, A/B-тестування).
- **Інвазивне тестування (нав'язливе)** (intrusive testing) – тестування, яке збирає інформацію про час та обробку під час виконання програми, що може змінити поведінку програмного забезпечення у порівнянні з його поведінкою в реальному середовищі. Інвазивне тестування зазвичай включає додатковий код, вбудований у програмне забезпечення, що тестується, або додаткові процеси, що виконуються одночасно з програмним забезпеченням, яке тестується, на тому ж процесорі. Інвазивне тестування можна вважати різновидом тестування переривань, яке використовується для перевірки того, наскільки добре система

реагує на вторгнення та переривання свого звичайного робочого процесу.

- **Рівень проникнення** (level of intrusion) – рівень, до якого модифікується тестовий об’єкт шляхом налаштування його для тестування.
- **Неінвазивне тестування (ненав'язливе)** (nonintrusive testing) – тестування, яке є прозорим для тестованого програмного забезпечення, тобто не змінює його часові характеристики чи характеристики обробки. Ненав’язливе тестування зазвичай включає додаткове обладнання, яке збирає інформацію про час або обробці та обробляє цю інформацію на іншій платформі.
- **Тестування під управлінням даними** (DDT) (data-driven testing) – метод створення сценаріїв, за якого вхідні дані тесту та очікувані результати зберігаються в таблиці або електронній таблиці, щоб один керуючий сценарій міг виконувати всі тести в таблиці. Тестування на основі даних часто використовується для підтримки застосування інструментів виконання тестів, таких як інструменти захоплення/відтворення.
- **Тестування під управлінням ключовими словами** (KDT) (keyworddriven testing) – метод створення сценаріїв, за якого файли даних використовуються для збереження не лише текстових даних та очікуваних результатів, але й ключових слів, пов’язаних з додатком, що тестується. Ключові слова інтерпретуються спеціальними допоміжними сценаріями, які викликаються контрольним сценарієм або тестом.
- **Тестування під управлінням поведінкою** (поведінкове тестування, BDT) (behavior-driven testing). Поведінкові тести – тести, що фокусуються на поведінці, а не на технічному виконанні програмного забезпечення.

- **Тестування передбаченням помилок** (error guessing) – метод проєктування тестів, при якому досвід тестувальника використовується для прогнозування дефектів, які можуть бути присутніми в компоненті або системі, що тестуються, в результаті допущених помилок, і для розроблення тестів спеціально для їх виявлення.
- **Помилково-орієнтоване тестування** (failure-directed testing) – тестування програмного забезпечення, засноване на знанні типів помилок, допущених у минулому, які є ймовірними для системи, що тестується.
- **Евристична оцінка** (heuristic evaluation) – метод перевірки зручності використання, спрямований на усунення проблем зручності використання в інтерфейсі користувача або дизайні інтерфейсу користувача. За допомогою цього методу рецензенти досліджують інтерфейс і оцінюють його відповідність визнаним принципам юзабіліті (зручності використання) («евристика»).
- **Мутаційне тестування** (mutation testing) – тестування, при якому два або більше варіантів компонента або системи виконуються з однаковими вхідними даними, вихідні дані порівнюються та аналізуються у випадках розбіжностей.
- **Тестування додаванням помилок** (error seeding) – процес навмисного додавання відомих помилок до тих, які вже є в комп’ютерній програмі, з метою моніторингу швидкості виявлення та видалення, а також оцінки кількості помилок, що залишилися в програмі.
- **Тестування на основі класів еквівалентності** (equivalence partitioning) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання представників із розділів еквівалентності. У принципі тестові випадки розроблені таким чином, щоб охопити кожен розділ принаймні один раз.

- **Тестування на основі граничних умов** (boundary value analysis) – метод розробки тесту чорної скриньки, в якому тестові приклади розробляються на основі граничних значень (вхідних значень або вихідних значень, які знаходяться на границі розділу еквівалентності або на найменшій відстані по обидва боки від границі, наприклад, мінімальне або максимальне значення діапазону).
- **Доменне тестування** (domain analysis, domain testing) – метод розробки тесту чорної скриньки, який використовується для визначення ефективних та дієвих тестових випадків, коли кілька змінних можна або потрібно тестувати разом. Він спирається на розподіл еквівалентності та аналіз граничних значень і узагальнює його.
- **Попарне тестування** (pairwise testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання всіх можливих дискретних комбінацій кожної пари вхідних параметрів.
- **N-комбінаційне тестування** (n-wise testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання всіх можливих дискретних комбінацій будь-якого набору з n вхідних параметрів.
- **Тестування на основі ортогональних масивів** (orthogonal array testing) – систематичний спосіб перевірки комбінацій усіх пар змінних за допомогою ортогональних масивів. Це значно зменшує кількість усіх комбінацій змінних для перевірки всіх парних комбінацій.
- **Тестування в процесі розробки** (development testing) – формальне або неформальне тестування, що проводиться під час реалізації компонента або системи, зазвичай у середовищі розробки розробниками.
- **Тестування по потоку управління** (control flow testing) – підхід до структурного тестування, в якому тестові випадки призначені для виконання певних послідовностей подій. Існують різні методи для тестування потоку управління, наприклад, тестування рішень,

тестування умов та тестування шляху, кожен з яких має свій специфічний підхід та рівень охоплення потоку управління.

- **Тестування по потоку даних** (data-flow testing) – техніка розробки тесту білої скриньки, в якій тестові випадки розроблені для виконання пар визначення-використання змінних.
- **Тестування за діаграмою або таблицею станів** (state transition testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання припустимих і неприпустимих переходів між станами.
- **Діаграма станів** (state diagram) – діаграма, яка зображує стани, які може приймати компонент або система, а також події або обставини, які викликають і/або є результатом переходу одного стану в інший.
- **Таблиця станів** (state table) – сітка, що показує результуючі переходи для кожного стану в поєднанні з кожною можливою подією, показуючи як припустимі, так і неприпустимі переходи.
- **Кінцевий автомат** (finite state machine) – обчислювальна модель, що складається з кінцевої кількості станів і переходів між цими станами, можливо, із супутніми діями.
- **Інспекція (аудит) коду** (code review, code inspection) – тип експертної перевірки, яка заснована на візуальному огляді документів для виявлення дефектів, тобто порушення стандартів розробки та невідповідність документації вищого рівня. Найбільш формальний метод перевірки, тому завжди заснований на задокументованій процедурі.
- **Тестування на основі виразів** (statement testing) – метод проєктування тестів білої скриньки, в якому тестові випадки розроблені для виконання операторів (оператор – це сутність на мові програмування, яка зазвичай є найменшою неподільною одиницею виконання).

- **Тестування на основі гілок** (branch testing) – метод проєктування тестів білої скриньки, в якому тестові приклади розроблені для виконання розгалужень (гілка – це базовий блок, який можна вибрати для виконання на основі конструкції програми, в якій доступний один з двох або більше альтернативних шляхів програми, наприклад case, jump, перейти до, якщо-то-інакше.).
- **Тестування на основі умов** (condition testing) – метод проєктування тестів білої скриньки, в якому тестові набори призначені для виконання умов (умова – це логічний вираз, який можна оцінити як True або False, наприклад, A > B).
- **Тестування на основі комбінацій умов** (multiple condition testing) – метод проєктування тестів білої скриньки, в якому тестові набори призначені для виконання комбінацій результатів однієї умови (в межах одного оператора).
- **Тестування на основі окремих умов, що породжують розгалуження («вирішальних умов»)** (modified condition decision coverage testing) – техніка розробки тестових випадків для виконання результатів умов розгалуження, які незалежно впливають на результат рішення, і відкидання умов, які не впливають на кінцевий результат.
- **Тестування на основі рішень** (decision testing) – метод проєктування тестів білої скриньки, в якому тестові випадки призначені для виконання результатів прийняття рішень (рішення – це програмна точка, в якій потік керування має два або більше альтернативних маршрути, наприклад, вузол з двома або більше посиланнями на окремі гілки).
- **Тестування на основі шляхів** (path testing) – метод проєктування тестів білої скриньки, в якому тестові випадки призначені для виконання шляхів.
- **Тестування за таблицею прийняття рішень** (decision table testing) – метод проєктування тестів чорної скриньки, у якому тестові набори призначені для виконання комбінацій вхідних даних та/або стимулів

(причин), показаних у таблиці рішень (таблиця, що показує комбінації вхідних даних та/або стимулів (причин) з пов'язаними з ними результатами та/ або дії (ефекти), які можна використовувати для розробки тестових випадків).

- **Тестування за моделями поведінки програми** (model-based testing) – тестування на основі моделі компонента або системи, що тестується, наприклад, моделі зростання надійності, використання моделей, таких як робочі профілі або поведінкові моделі, такі як таблиця рішень або діаграма переходу станів.
- **Тестування на основі варіантів використання** (use case testing) – метод проєктування тестів чорної скриньки, у якому тестові варіанти розроблені для виконання сценаріїв варіантів використання.
- **Тестування на основі історії користувача** (user story testing) – метод проєктування тестів чорної скриньки, у якому тестові випадки розробляються на основі історій користувачів для перевірки їх правильної реалізації.
- **Паралельне тестування** (parallel testing) – тестування нової або зміненої системи обробки даних з тими ж вхідними даними, які використовуються в іншій системі. Інша система розглядається як еталон порівняння.
- **Тестування на основі випадкових даних** (random testing) – метод проєктування тестів чорної скриньки, у якому тестові випадки вибираються, можливо, з використанням псевдовипадкового алгоритму генерації, щоб відповідати робочому профілю. Цей метод можна використовувати для тестування нефункціональних атрибутів, таких як надійність і продуктивність.
- **Операційний профіль** (operational profile) – представлення окремого набору завдань, що виконується компонентом або системою, можливо, на основі поведінки користувача під час взаємодії з компонентом або системою та ймовірності їх виникнення. Завдання скоріше логічне, ніж

фізичне, і може виконуватися на кількох машинах або виконуватися в несуміжних часових сегментах.

- **«Мавпяче тестування»** (monkey testing) – тестування за допомогою випадкового вибору з великого діапазону вхідних даних і шляхом випадкового натискання кнопок, не знаючи, як використовується продукт.
- **A/B-тестування, спліт-тестування** (A/B testing, split testing) – це дизайн для встановлення причинно-наслідкового зв'язку між змінами та їх впливом на поведінку, яку спостерігає користувач.

Класифікація за моментом виконання (хронології)

- **Висхідне тестування** (bottom-up testing) – поступовий підхід до інтеграційного тестування, коли спочатку тестуються компоненти нижчого рівня, а потім використовуються для полегшення тестування компонентів вищого рівня. Цей процес повторюється до тих пір, поки компонент у верхній частині ієрархії не буде перевірено.
- **Низхідне тестування** (top-down testing) – інкрементальний підхід до інтеграційного тестування, коли компонент у верхній частині ієрархії компонентів тестується першим, а компоненти нижнього рівня моделюються заглушками. Перевірені компоненти потім використовуються для тестування компонентів нижнього рівня. Процес повторюється до тих пір, поки компоненти найнижчого рівня не будуть перевірені.
- **Гібридне тестування, сендвіч-тестування** (hybrid testing) – це процес, в якому, по-перше, вхідні дані для функцій інтегруються у висхідному шаблоні. Потім вихідні дані для кожної функції інтегруються зверху вниз. Основною перевагою цього підходу є ступінь підтримки раннього випуску обмеженої функціональності.

Альтернативні та додаткові класифікації тестування

- **Дерево класифікації** (classification tree) – це дерево, що показує ієрархічно впорядковані розділи еквівалентності, яке використовується

для розробки тестових випадків у методі тестування на основі дерева класифікацій.

- **Тестування на основі дерева класифікацій** (classification tree method) – метод проєктування тестів чорної скриньки, в якому тестові випадки, описані за допомогою дерева класифікації і призначені для виконання комбінацій представників вхідних та/або вихідних доменів.
- **Тестування на основі синтаксису** (syntax testing) – метод проєктування тестів чорної скриньки, в якому тестові випадки розробляються на основі визначення вхідної та/або вихідної області.
- **Комбінаторне тестування** (combinatorial testing) – засіб для визначення відповідної підмножини тестових комбінацій для досягнення заздалегідь визначеного рівня охоплення під час тестування об’єкта з кількома параметрами, де кожен із цих параметрів має кілька значень, що призводить до більшої кількості комбінацій, ніж можливо перевірити протягом дозволеного часу.
- **Тестування всіх комбінацій** (all combinations testing) – тестування всіх можливих комбінацій всіх значень для всіх параметрів.
- **Тестування з вибором значень-представників** (each choice testing) – одне значення з кожного блоку для кожного розділу має використовуватися принаймні в одному тестовому випадку.
- **Тестування з вибором базового набору значень** (base choice testing) – для кожного розділу вибирається блок базового вибору і базовий тест формується з використанням базового вибору для кожного розділу. Подальші тести вибираються шляхом збереження всіх, крім одного базового варіанту, постійними та використанням кожного небазового варіанту в кожному іншому параметрі.
- **Тестування за графом причинно-наслідкових зв'язків** (cause-effect graphing) – метод проєктування тестів чорної скриньки, в якому тестові сценарії розробляються на основі причинно-наслідкових графів (графічне представлення вхідних даних та/або стимулів (причин) з

відповідними вихідними даними (ефектами), які можна використовувати для розробки тестових сценаріїв).

- **Тестування по потоку даних** (data-flow testing) – метод проєктування тестів білої скриньки, в якому тестові випадки розроблені для виконання пар визначення-використання змінних.
- **Стратегія з усіма визначеннями** (all-definitions testing) – набір тестів вимагає, щоб кожне визначення кожної змінної охоплювалося принаймні одним використанням цієї змінної (c-use або p-use).
- **Стратегія використання всіх обчислень** (all-c-uses testing) – для кожної змінної та кожного визначення цієї змінної включіть принаймні один шлях без визначення від визначення до кожного використання обчислень.
- **Стратегія використання всіх предикатів** (all-p-uses testing) – для кожної змінної та кожного визначення цієї змінної включіть принаймні один шлях без визначення від визначення до кожного використання предиката.
- **Стратегія для всіх видів використання** (all-uses testing) – тестовий набір включає принаймні один сегмент шляху від кожного визначення до кожного використання, яке може бути досягнуто цим визначенням.
- **Стратегія All-DU-path** (all-du-paths testing) – тестовий набір включає кожен шлях від кожного визначення кожної змінної до кожного використання цього визначення.

Класифікація за приналежністю до тестування за методом білої та чорної скриньок

- **Метод білої скриньки**: статичне тестування, динамічне тестування (рідко), ручне тестування (мало), автоматизоване тестування, модульне (компонентне) тестування, інтеграційне тестування, системне тестування (мало), димове тестування (мало), тестування критичного шляху (мало), розширене тестування (мало), позитивне тестування, негативне тестування, тестування веб-додатків, тестування мобільних

додатків, тестування настільних додатків, тестування рівня представлення (мало), тестування рівня бізнес-логіки, тестування рівня даних, альфа-тестування (мало), бета-тестування (майже ніколи), гама-тестування (майже ніколи), тестування на рівні тест-кейсів, функціональне тестування, інсталяційне тестування (рідко), регресійне тестування, повторне тестування, приймальне тестування (дуже рідко), операційне тестування (дуже рідко), тестування зручності використання (дуже рідко), тестування доступності (дуже рідко), тестування інтерфейсу, тестування безпеки, тестування інтернаціоналізації (мало), тестування локалізації (мало), Тестування сумісності (мало), конфігураційне тестування (мало), крос-браузерне тестування (мало), тестування даних і баз даних, тестування використання ресурсів (дуже рідко), вичерпне тестування (дуже рідко), тестування надійності (дуже рідко), тестування відновлюваності (дуже рідко), тестування відмово стійкості (дуже рідко), тестування продуктивності (дуже рідко), навантажувальне тестування (дуже рідко), тестування масштабуємості (дуже рідко), об’ємне тестування (дуже рідко), стресове тестування (дуже рідко), конкурентне тестування (дуже рідко), інвазивне тестування, неінвазивне тестування, тестування під управлінням даними, тестування під управлінням ключовими словами, тестування передбачуванням помилок (дуже рідко), мутаційне оцінювання, тестування додаванням помилок, тестування на основі класів еквівалентності, тестування на основі граничних умов, доменне тестування, попарне тестування, тестування на основі ортогональних масивів, тестування в процесі розробки, тестування по потоку управління, тестування по потоку даних, тестування по діаграмі або таблиці станів (рідко), інспекція (аудит) коду, тестування на основі виразів, тестування на основі гілок, тестування на основі умов, тестування на основі комбінацій умов, тестування на основі окремих умов, що породжують розгалуження («рішаючих умов»), тестування на

основі рішень, тестування на основі шляхів, тестування по таблиці прийняття рішень, тестування по моделям поведінки додатку, тестування на основі варіантів використання, паралельне тестування, тестування на основі випадкових даних, висхідне тестування, низхідне тестування, гібридне тестування, тестування на основі дерева класифікацій, тестування на основі синтаксису, комбінаторні техніки (комбінаторне тестування), тестування всіх комбінацій, тестування з вибором значень-представників, тестування з вибором базового набору значень, тестування по графу причинно-наслідкових зв'язків (мало), перевірка використання всіх оголошень, перевірка всіх обчислень на основі всіх оголошень, перевірка всіх розгалужень на основі всіх оголошень, перевірка всіх обчислень і розгалужень на основі всіх оголошень, перевірка використання всіх оголошень і всіх шляхів без пере оголошень (без циклів або з одноразовими повтореннями циклів).

- **Метод чорної скриньки**: динамічне тестування, ручне тестування, автоматизоване тестування, інтеграційне тестування, системне тестування, димове тестування, тестування критичного шляху, розширене тестування, позитивне тестування, негативне тестування, тестування веб-додатків, тестування мобільних додатків, тестування настільних додатків, тестування рівня представлення, тестування рівня бізнес-логіки, тестування рівня даних (мало), альфа-тестування, бета-тестування, гама-тестування, тестування на рівні тест-кейсів, дослідницьке тестування, вільне (інтуїтивне) тестування, функціональне тестування, інсталяційне тестування, регресійне тестування, повторне тестування, приймальне тестування, операційне тестування, тестування зручності використання, тестування доступності, тестування інтерфейсу, тестування безпеки, тестування інтернаціоналізації, тестування локалізації, тестування сумісності, конфігураційне тестування, крос-браузерне тестування, тестування даних і баз даних (мало), тестування використання ресурсів, порівняльне тестування, демонстраційне

тестування, тестування надійності, тестування відновлюваності, тестування відмовостійкості, тестування продуктивності, навантажувальне тестування, тестування масштабуємості, об'ємне тестування, стресове тестування, конкурентне тестування, інвазивне тестування, неінвазивне тестування, тестування під управлінням даними, тестування під управлінням ключовими словами, тестування передбачуванням помилок, евристична оцінка, мутаційне оцінювання, тестування додаванням помилок, тестування на основі класів еквівалентності, тестування на основі граничних умов, доменне тестування, попарне тестування, тестування на основі ортогональних масивів, тестування в процесі розробки, тестування по діаграмі або таблиці станів, тестування по таблиці прийняття рішень, тестування по моделям поведінки додатку, тестування на основі варіантів використання, паралельне тестування, тестування на основі випадкових даних, A/B-тестування, висхідне тестування, низхідне тестування, гібридне тестування, тестування на основі дерева класифікацій, тестування на основі синтаксису, комбінаторні техніки (комбінаторне тестування), тестування по графу причинно-наслідкових зв'язків,

Закріплення матеріалу

1. Як виглядає спрощена класифікація тестування?
2. Надати пояснення до кожної складової спрощеної класифікації тестування.
3. Як виглядає детальна класифікація тестування?
4. Надати пояснення до кожної складової детальної класифікації тестування.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.2 КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ

Лекція 5

ЧЕК-ЛИСТ, ТЕСТ-КЕЙСИ, НАБОРИ ТЕСТ-КЕЙСІВ

Мета лекції: Розгляд понять чек-листа, тест-кейсу та набору тест-кейсів.

Зміст лекції

1. Чек-лист.
    1.1. Функції, без яких існування додатку втрачає сенс.
    1.2. Функції, затребувані більшістю користувачів.
    1.3. Інші функції та особливі сценарії.
2. Тест-кейс та його життєвий цикл.
    2.1. Термінологія і загальні відомості.
    2.2. Мета написання тест-кейсу.
    2.3. Життєвий цикл тест-кейсу.
3. Атрибути (поля) тест-кейсу.
4. Інструментальні засоби управління тестуванням.
5. Властивості якісних тест-кейсів.
6. Набори тест-кейсів.
    6.1. Термінологія та загальні відомості.
    6.2. Детальна класифікація наборів тест-кейсів.
    6.3. Принципи побудови наборів тест-кейсів.
7. Логіка створення ефективних перевірок.
8. Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів.
    8.1 Помилки оформлення та формулювань.

8.2. Логічні помилки.

Чек-лист

- **Чек-лист** (checklist) – набір ідей [тест-кейсів]. Поняття «чек-листа» не зав'язане на тестуванні як такому – це універсальна техніка, яка може застосовуватися у будь-якій без винятку області життя. В українській мові поза контекстом інформаційних технологій частіше використовується зрозуміле і звичне слово «список», але в тестуванні прижилася калькована з англійської версія – «чек-лист» [1].
- Чек-лист – список:
  - у якому послідовність пунктів не має значення;
  - у якому послідовність пунктів важлива;
  - структурований (багаторівневий) список, який дозволяє відобразити ієрархію ідей.
- Іноді чек-листи можуть навіть надаватись графічно, хоча традиційно їх складають як багаторівневі списки.
- Випадки, в яких до чек-листа можуть додаватись очікувані результати:
  - у певному пункті чек-листа розглядається особлива, нетривіальна поведінка програми або складна перевірка, результат якої важливо відзначити вже зараз, щоби не забути;
  - в силу стислих термінів та/або нестачі інших ресурсів тестування проводиться безпосередньо за чек-листами без тест-кейсів.
- Властивості чек-листа:
  - логічність;
  - послідовність та структурованість;
  - повнота та надмірність.
- Типові варіанти логіки створення окремих чек-листів для:
  - типових сценаріїв користувача;
  - різних рівнів функціонального тестування;

  - окремих частин (модулів та підмодулів) додатку;
  - окремих вимог, груп вимог, рівнів та типів вимог;
  - частин або функцій програми, найбільш схильних до ризиків.
- Логіка розбиття функцій програми за ступенем їх важливості (на конкретному прикладі проєкту із кодовою назвою «Конвертер файлів»):
  - базові функції;
  - функції, затребувані більшістю користувачів у їх повсякденній роботі;
  - інші функції.

Функції, без яких існування додатку втрачає сенс

- Ключові функції додатку (на конкретному прикладі проєкту із кодовою назвою «Конвертер файлів»):
  - конфігурування та запуск;
  - обробка файлів;
  - зупинка.

Функції, затребувані більшістю користувачів

- Функції, необхідні користувачам (на конкретному прикладі проєкту із кодовою назвою «Конвертер файлів»):
  - Конфігурування та запуск:
    - з правильними параметрами;
    - без параметрів;
    - з недостатньою кількістю параметрів;
    - з неправильними параметрами.
  - Обробка файлів:
    - різні формати, кодування та розміри;
    - недоступні вхідні файли:
      - немає прав доступу;
      - файл відкритий та заблокований;
      - файл із атрибутом лише для читання.

- Зупинка:
  - закриттям вікна консолі.
- Журнал роботи програми:
  - автоматичне створення (за відсутності журналу);
  - продовження (доповнення журналу) при повторних запусках.
- • Продуктивність:
  - елементарний тест із грубою оцінкою.

Інші функції та особливі сценарії

- Інші функції для запобігання проблем (на конкретному прикладі проєкту із кодовою назвою «Конвертер файлів»):
  - Конфігурування та запуск;
  - Обробка файлів.

Тест-кейс та його життєвий цикл

Термінологія та загальні відомості

- **Тест** (test) – це набір з одного або кількох тестових випадків.
- **Тест-кейс** (test case) – це набір вхідних значень, передумов виконання, очікуваних результатів і постумов виконання, розроблених для конкретної мети або умови тестування, наприклад, для виконання певного шляху програми або для перевірки відповідності конкретній вимозі [1].
- **Високорівневий тест-кейс, логічний тест-кейс** (high level test case) – це тест-кейс без конкретних (рівень реалізації) значень для вхідних даних та очікуваних результатів. Використовуються логічні оператори; екземпляри фактичних значень ще не визначені та/або не доступні.
- **Низькорівневий тест-кейс** (low level test case) – це тест-кейс із конкретними (рівень реалізації) значеннями вхідних даних та очікуваними результатами. Логічні оператори з тест-кейсів високого рівня замінюються фактичними значеннями, які відповідають цілям логічних операторів.

- **Специфікація тест-кейсу** (test case specification) – це документ, що визначає набір тест-кейсів (мета, вхідні дані, тестові дії, очікувані результати та попередні умови виконання) для тестового завдання.
- **Тестовий елемент** (test item) – це окремий елемент, що підлягає перевірці. Зазвичай існує один тестовий об'єкт і багато тестових елементів.
- **Тестовий об'єкт** (test object) – це компонент або система, що підлягають перевірці.
- **Специфікація тесту** (test specification) – це документ, який складається зі специфікації проєкту тестування, специфікації тест-кейсу та/або специфікації процедури тестування.
- **Специфікація тест-дизайну** (test design specification) – це документ, що визначає умови тестування (елементи покриття) для тестового елемента, детальний підхід до тестування і ідентифікує відповідні тест-кейси високого рівня.
- **Специфікація тест-процедури, процедура тестування** (test procedure specification) – це документ, що визначає послідовність дій для виконання тесту. Також відомий як тестовий сценарій або сценарій ручного тестування.
- **Тестовий сценарій** (test scenario, test procedure specification, test script) – це документ, що визначає послідовність дій для виконання тесту. Також відомий сценарій ручного тестування.
- **Тестове покриття** (test coverage metrics) – це ступінь, виражена у відсотках, в якій певний об'єкт покриття (сутність або властивість), що використовується як основа для тестового покриття був реалізований набором тестів.

Мета написання тест-кейсів

- Тестування можна проводити і без тест-кейсів.
- Наявність тест-кейсів дозволяє:

- структурувати та систематизувати підхід до тестування;
- обчислювати метрики тестового покриття та вживати заходів щодо його збільшення;
- відслідковувати відповідність поточної ситуації плану;
- уточнити взаєморозуміння між замовником, розробниками та тестувальниками;
- зберігати інформацію для тривалого використання та обміну досвідом між співробітниками та командами;
- проводити регресійне та повторне тестування;
- підвищувати якість вимог;
- швидко вводити в курс справи нового співробітника, який нещодавно підключився до проєкту.

Життєвий цикл тест-кейсу

Життєвий цикл або набір станів тест-кейсу:

- створено (new);
- запланований (planned, ready for testing);
- не виконано (not tested);
- виконується (work in progress);
- пропущений (skipped);
- провалений (failed);
- пройдений успішно (passed);
- заблокований (blocked);
- закритий (closed);
- вимагає доопрацювання (not ready).

Атрибути (поля) тест-кейсу

- Термін «тест-кейс» може належати до формального запису тест-кейса як технічного документа. Цей запис має загальноприйняту структуру, компоненти якої називаються атрибутами (полями) тест-кейсу.

- **Ідентифікатор** (identifier) – це атрибут тест-кейсу, що є унікальним значенням, що дозволяє однозначно відрізнити один тест-кейс від іншого і використовується у різноманітних посиланнях.
- **Пріоритет** – це атрибут тест-кейсу, що показує важливість тест-кейсу.
- **Пов'язана з тест-кейсом вимога** (requirement) – це атрибут тест-кейсу, що показує ту основну вимогу, перевірці виконання якої присвячено тест-кейс.
- **Модуль та підмодуль додатку** (module and submodule) – це атрибути тест-кейсу, що вказують на частини додатку, до яких належить тест-кейс, і дозволяють краще зрозуміти його ціль.
- **Заголовок (суть) тест-кейсу** (title) – це атрибут тест-кейсу, що покликаний спростити та прискорити розуміння основної ідеї (мети) тест-кейсу без звернення до його інших атрибутів.
- **Вихідні дані, необхідні для виконання тест-кейсу** (precondition, preparation, initial data, setup) – це атрибут тест-кейсу, який дозволяє описати все те, що повинно бути підготовлено до початку виконання тест-кейсу.
- **Кроки тест-кейсу** (steps) – це атрибут тест-кейсу, що описує послідовність дій, які необхідно реалізувати у процесі виконання тест-кейсу.
- **Очікувані результати** (expected results) по кожному кроку тест-кейсу – це атрибут тест-кейсу, що демонструє реакцію програми на дії, описані в полі «кроки тест-кейсу».

Інструментальні засоби управління тестуванням

- **Інструментальні засоби керування тестуванням** (test management tool) – це інструмент, який забезпечує підтримку управління тестуванням та контроль частини процесу тестування. Він часто має кілька можливостей, таких як керування тестовим програмним забезпеченням,

планування тестів, реєстрація результатів, відстеження прогресу, керування інцидентами та звітування про тестування [1].

Загальний набір функцій, що реалізуються інструментальними засобами керування тестуванням:

- створення тест-кейсів та наборів тест-кейсів;
- контроль версій документів з можливістю визначити, хто вніс ті чи інші зміни, та скасувати ці зміни, якщо потрібно;
- формування та відстеження реалізації плану тестування, збирання та візуалізації різноманітних метрик, генерування звітів;
- інтеграція із системами управління дефектами, фіксація взаємозв'язку між виконанням тест-кейсів та створеними звітами про дефекти;
- інтеграція із системами управління проєктами;
- інтеграція з інструментами автоматизованого тестування, керування виконанням автоматизованих тест-кейсів.

Властивості якісних тест-кейсів

- Властивості для побудови якісних тест-кейсів:
  - правильна технічна мова, точність та одноманітність формулювань;
  - баланс між специфічністю та загальністю;
  - баланс між простотою та складністю;
  - «показовість» (висока ймовірність виявлення помилки);
  - послідовність у досягненні мети;
  - відсутність зайвих дій;
  - ненадмірність по відношенню до інших тест-кейсів;
  - демонстративність (здатність демонструвати виявлену помилку очевидним чином);
  - простежуваність;
  - можливість повторного використання;

- повторюваність;
- відповідність прийнятим шаблонам оформлення та традиціям.

Набори тест-кейсів

Термінологія та загальні відомості

- **Набір тест-кейсів** (test case suite, test suite, test set) – це набір з кількох тестових випадків для компонента або системи, що тестуються, де постумова одного тесту часто використовується як передумова для наступного [1].
- Часто замість «набір тест-кейсів» кажуть «тестовий сценарій». Формально це можна вважати помилкою, але явище набуло такого поширення, що стало варіантом норми.
- У загальному випадку набори тест-кейсів можна розділити на вільні (порядок виконання тест-кейсів не важливий) та послідовні (порядок виконання тест-кейс важливий).
- Переваги вільних наборів:
  - тест-кейси можна виконувати у будь-якому зручному порядку, а також створювати «набори всередині наборів»;
  - якщо тест-кейс завершився помилкою, це не вплине на можливість виконання інших тест-кейсів.
- Переваги послідовних наборів:
  - кожен наступний у наборі тест-кейс в якості вхідного стану додатку отримує результат роботи попереднього тест-кейсу, що дозволяє сильно скоротити кількість кроків у окремих тест-кейсах;
  - довгі послідовності дій куди краще імітують роботу реальних користувачів, ніж окремі «точкові» впливи на додаток.

Сценарії користувачів (сценарії використання)

- **Сценарій** – це гіпотетична історія, яка використовується, щоб допомогти людині продумати складну проблему або систему [1].
- Переваги сценаріїв:

- показують реальні і зрозумілі приклади використання продукту;
- зрозумілі кінцевим користувачам і добре підходять для обговорення та спільного покращення;
- легше оцінюються з огляду на важливість, ніж окремі пункти вимог;
- чудово показують недоробки у вимогах;
- у граничному випадку сценарії можна навіть не прописувати докладно, а просто називати – і саме найменування вже підкаже досвідченому фахівцю, що робити.

Детальна класифікація наборів тест-кейсів

- Тест-кейси за рівнем ізольованості один від одного поділяються на ізольовані та узагальнені.
- Тест-кейси за утворенням суворої послідовності поділяються на вільні та послідовні.
- Набір ізольованих вільних тест-кейсів: дії з розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейси можна виконувати у будь-якому порядку.
- Набір узагальнених вільних тест-кейсів: дії розділу «приготування» потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси можна виконувати у будь-якому порядку.
- Набір ізольованих послідовних тест-кейсів: дії з розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейси потрібно виконувати у строго визначеному порядку.
- Набір узагальнених послідовних тест-кейсів: дії з розділу «приготування» потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси потрібно виконувати в певному порядку.
- Головна перевага ізольованості: кожен тест-кейс виконується в «чистому середовищі», на нього не впливають результати роботи попередніх тест-кейсів.

- Головна перевага узагальненості: приготування не потрібно повторювати.
- Головна перевага послідовності: відчутне скорочення кроків у кожному тест-кейсі, оскільки результат виконання попереднього тест-кейсу є початковою ситуацією для наступного.
- Головна перевага свободи: можливість виконувати тест-кейси в будь-якому порядку, а також те, що при провалі якогось тест-кейсу, інші тест-кейси, як і раніше, можна виконувати.

Принципи побудови наборів тест-кейсів

- Єдине завдання наборів – підвищити ефективність тестування за рахунок прискорення і спрощення виконання тест-кейсів, збільшення глибини дослідження якоїсь області додатку або функціональності, дотримання типових сценаріїв користувача або зручної послідовності виконання тест-кейсів тощо.
- Найбільш типові підходи до складання наборів тест-кейсів:
  - на основі чек-листів;
  - на основі розбиття програми на модулі та під модулі;
  - за принципом перевірки найважливіших, менш важливих та інших функцій додатку;
  - за принципом групування тест-кейсів для перевірки певного рівня вимог або типу вимог, групи вимог або окремої вимоги;
  - за принципом частоти виявлення тест-кейсами дефектів у додатку;
  - за архітектурним принципом;
  - за областю внутрішньої роботи програми;
  - за видами тестування.

Логіка створення ефективних перевірок

- Вся суть роботи тестувальника спрямована на підвищення якості (процесів, продуктів тощо).

- **Якість** (quality) – це ступінь, в якій компонент, система чи процес відповідають визначеним вимогам та/або потребам та очікуванням користувача/замовника.
- **Завдання тестувальника** – знайти максимум ВАЖЛИВИХ помилок за наявний час. Під важливими помилками розуміють такі, які призводять до порушення важливих для користувача функцій або властивостей продукту.
- Питання, які тестувальнику необхідно задати собі та отримати чіткі відповіді, приступаючи до продумування чек-листа, тест-кейс або набору тест-кейсів:
  - Що перед вами?
  - Кому і навіщо воно потрібне (і наскільки це важливо)?
  - Як воно зазвичай використовується?
  - Як воно може зламатися, тобто почати працювати неправильно?
  - Перелік універсальних рекомендацій, які дозволять проводити тестування краще:
  - починати якомога раніше;
  - якщо потрібно тестувати щось велике і складне, доцільно розбити його на модулі та підмодулі, функціональність піддати функціональній декомпозиції;
  - обов’язково писати чек-листи;
  - по мірі створення чек-листів, тест-кейсів тощо прямо в текст вписувати питання, що виникають; коли питань накопичиться достатньо, зібрати їх окремо, уточнити формулювання та звернутися до того, хто може дати відповіді;
  - якщо використовуваний інструментальний засіб дозволяє застосовувати косметичне оформлення тексту – використати;
  - використати техніку швидкого перегляду для отримання відгуку від колег та покращення створеного документа;

- планувати час на покращення тест-кейсів;
- починати опрацювання (і виконання) тест-кейсів із простих позитивних перевірок найважливішої функціональності; потім поступово підвищувати складність перевірок, пам'ятаючи не тільки про позитивні, а й про негативні перевірки;
- пам'ятати, що основою тестування є мета – якщо тестувальник не може швидко і просто сформулювати ціль створеного ним тест-кейсу, він створив поганий тест-кейс;
- уникати надлишкових тест-кейсів, що дублюють один одного;
- якщо показник тест-кейсу можна збільшити, при цьому не сильно змінивши його складність і не відхилившись від вихідної мети, зробити це;
- пам'ятати, що багато тест-кейсів потребують окремої підготовки, яку потрібно описати у відповідному полі тест-кейсу;
- кілька позитивних тест-кейсів можна безбоязно об'єднувати, але об'єднання негативних тест-кейсів майже завжди заборонено;
- подумати, як можна оптимізувати створений тест-кейс (набір тест-кейсів тощо) так, щоб знизити трудовитрати на його виконання;
- перед тим, як надсилати фінальну версію створеного документа, ще раз перечитати написане.

Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів

Помилки оформлення та формулювань

- Перелік типових помилок оформлення та формулювань:
  - відсутність назви тест-кейсу або погано написана назва;
  - відсутність нумерації кроків та/або очікуваних результатів;
  - посилання на багато вимог;
  - використання особової форми дієслів;

- використання минулого чи майбутнього часу в очікуваних результатах;
- постійне використання слів «перевірити» (і йому подібних) у чек-листах;
- опис стандартних елементів інтерфейсу замість використання їх усталених назв;
- пунктуаційні, орфографічні, синтаксичні та подібні до них помилки.

Логічні помилки

- Перелік типових логічних помилок:
  - посилання на інші тест-кейс або кроки інших тест-кейсів;
  - деталізація, що не відповідає рівню функціонального тестування;
  - розпливчасті двозначні описи дій та очікуваних результатів;
  - опис дій в якості найменувань модуля/під модуля;
  - опис подій або процесів в якості кроків або очікуваних результатів;
  - «вигадування» особливостей поведінки програми;
  - відсутність опису приготування для виконання тест-кейсу;
  - повне дублювання (копіювання) одного і того ж тест-кейсу на рівнях димового тестування, тестування критичного шляху, розширеного тестування;
  - занадто довгий перелік кроків, що не належать до суті (мети) тест-кейсу;
  - некоректне найменування елементів інтерфейсу чи його властивостей;
  - нерозуміння принципів роботи додатку та викликана цим некоректність тест-кейсів;
  - перевірка типової «системної» функціональності;
  - неправильна поведінка додатку як очікуваний результат;
  - загальна некоректність тест-кейсів;

– неправильне розбиття наборів даних на класи еквівалентності;
– тест-кейси, що не належать до додатку, який тестується;
– формальні та/або суб'єктивні перевірки.

Закріплення матеріалу

1. Що таке чек-лист, тест-кейс, набір тест-кейсів?
2. Які властивості чек-листа вам відомі?
3. Які складові набору станів тест-кейсу вам відомі?
4. Що таке високорівневий та низькорівневий тест-кейс?
5. Що таке специфікація тест-кейсу, специфікація тесту, специфікація тест-дизайну, специфікація тест-процедури, процедура тестування?
6. Що таке тестовий елемент та тестовий об’єкт?
7. Що таке тестовий сценарій?
8. Що таке тестове покриття?
9. Яку загальноприйняту структуру має формальний запис тест-кейса як технічного документа?
10. Перелічити загальний набір функцій, що реалізуються інструментальними засобами керування тестуванням.
11. Які властивості для побудови якісних тест-кейсів вам відомі?
12. За якими ознаками проводиться детальна класифікація наборів тест-кейсів?
13. Назвати найбільш типові підходи до складання наборів тест-кейсів.
14. В чому полягає логіка створення ефективних перевірок?
15. Які типові помилки оформлення та формулювань при розробці чек-листів, тест-кейсів та наборів тест-кейсів вам відомі?
16. Які типові логічні помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів вам відомі?

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.3 ДЕФЕКТИ

Лекція 6

ЗВІТИ ПРО ДЕФЕКТИ

Мета лекції: Розгляд поняття дефекту та принципів формування ефективного звіту про дефект.

Зміст лекції

1. Помилки, дефекти, збої, відмови.
    1.1. Спрощений погляд на поняття дефекту.
    1.2. Розширений погляд на термінологію, що описує проблеми.
2. Звіт про дефект та його життєвий цикл.
3. Атрибути (поля) звіту про дефект.
4. Інструментальні засоби управління звітами про дефекти.
5. Властивості якісних звітів про дефекти.
6. Логіка створення ефективних звітів про дефекти.
7. Типові помилки при написанні звітів про дефекти.
    7.1. Помилки оформлення та формулювань.
    7.2. Логічні помилки.

Помилки, дефекти, збої, відмови

Спрощений погляд на поняття дефекту

- **Дефект** – розбіжність очікуваного та фактичного результату [1].
- **Очікуваний результат** – поведінка системи, описана у вимогах.
- **Фактичний результат** – поведінка системи, що спостерігається в процесі тестування.

Розширений погляд на термінологію, що описує проблеми

- **Помилка** (error, mistake) – це дія людини, яка дає неправильний результат [1].
- **Дефект, помилка, проблема, несправність** (defect, bug, problem, fault) – це несправність компонента або системи, яка може призвести до того, що компонент або система не зможуть виконувати свої необхідні функції. Дефект, якщо виникне під час виконання, може призвести до збою компонента або системи.
- **Переривання, збій, відмова** (interruption) – це призупинення процесу, наприклад виконання комп'ютерної програми, викликане подією, зовнішньою по відношенню до цього процесу, і виконується таким чином, що процес можна відновити [1].
- **Відмова** (failure) – це відхилення компонента або системи від очікуваної доставки, обслуговування або результату [1].
- **Аномалія** (anomaly) – це будь-яка умова, що відхиляється від очікувань, заснована на специфікаціях вимог, проєктної документації, документації користувача, стандартів тощо або з чийогось сприйняття чи досвіду. Аномалії можуть бути виявлені під час, перегляду, тестування, аналізу, компіляції або використання програмних продуктів чи відповідної документації [1].
- **Інцидент, відхилення** (deviation) – це будь-яка подія, яка потребує розслідування [1].
- **Фактичний результат** (actual result) – це поведінка, яка виникає/спостерігається під час тестування компонента або системи [1].
- **Очікуваний результат, прогнозований результат** (expected result) – це подія, передбачена специфікацією або іншим джерелом компонента або системи за певних умов [1].

Звіт про дефект та його життєвий цикл

- **Звіт про дефект, звіт про помилку** (defect report) – це документ, у якому повідомляється про будь-який недолік у компоненті або системі, який може призвести до того, що компонент або система не зможуть виконувати свої необхідні функції [1].
- Основні цілі написання звіту про дефект:
  - надати інформацію про проблему;
  - пріоритизувати проблему;
  - сприяти усуненню.
- Стадії життєвого циклу звіту про дефект:
  - **Виявлений** (submitted) – початковий стан звіту, в якому він знаходиться відразу після створення.
  - **Призначений** (assigned) – в цей стан звіт переходить з моменту, коли хтось із проєктної команди призначається відповідальним за виправлення дефекту.
  - **Виправлений** (fixed) – у цей стан звіт переводить відповідальний за виправлення дефекту член команди після виконання відповідних дій із виправлення.
  - **Перевірений** (verified) – у цей стан звіт переводить тестувальник, який переконався, що дефект насправді був усунений.
  - **Закритий** (closed) – стан звіту, який означає, що за цим дефектом не планується жодних подальших дій
  - **Відкритий знову** (reopened) – в цей стан звіт переводить тестувальник, який переконався, що дефект, як і раніше, відтворюється на білді, в якому він вже має бути виправлений.
  - **Рекомендований до відхилення** (to be declined) – в цей стан звіт про дефект може бути переведений з багатьох інших станів з метою винести на розгляд питання про відхилення звіту з тієї чи іншої причини.

- **Відхилений** (declined) – у цей стан звіт преводиться у випадках, докладно викладених у пункті «Закритий», якщо засіб керування звітами про дефекти передбачає використання цього стану замість стану «Закритий» для тих чи інших резолюцій щодо звіту.
- **Відкладений** (deferred) – у цей стан звіт переводиться у випадку, якщо виправлення дефекту найближчим часом є нераціональним чи не є можливим, однак є підстави вважати, що в найближчому майбутньому ситуація виправиться.

Атрибути (поля) звіту про дефект

- **Ідентифікатор** (identifier) – це унікальне значення, що дозволяє однозначно відрізнити один звіт про дефект від іншого і використовуване в у різних посиланнях.
- **Короткий опис** (summary) має у гранично лаконічній формі давати вичерпну відповідь на запитання «Що сталося?», «Де це сталося?», «За яких умов це сталося?».
- Алгоритм для створення хороших коротких описів дефектів:
  1. Повноцінно зрозуміти суть проблеми.
  2. Сформулювати докладний опис (description) дефекту – спочатку без огляду на довжину тексту, що вийшов.
  3. Прибрати з детального опису все зайве, уточнити важливі деталі.
  4. Виділити у докладному описі слова (словосполучення, фрагменти фраз), які відповідають на питання, «що, де і за яких умовах сталося».
  5. Оформити отримане у пункті 4 у вигляді закінченого граматично правильного речення.
  6. Якщо речення вийшло занадто довгим, переформулювати його, скоротивши довжину.

- **Детальний опис** (description) представляє у розгорнутому вигляді необхідну інформацію про дефект, а також (обов'язково!) опис фактичного результату, очікуваного результату та посилання на вимогу.
- **Кроки з відтворення** (steps to reproduce, STR) описують дії, які необхідно виконати для відтворення дефекту.
- **Відтворюваність** (reproducibility) показує, чи при кожному проходженні по кроках відтворення дефекту вдається викликати його прояв. Це поле приймає лише два значення: завжди (always) або іноді (sometimes). Відтворюваність «іноді» означає, що тестувальник не знайшов справжню причину виникнення дефекту.
- **Важливість** (severity) показує ступінь шкоди, що завдається проєкту існуванням дефекту.
- Градації важливості:
  - *Критична* (critical) – існування дефекту призводить до масштабних наслідків катастрофічного характеру.
  - *Висока* (major) – існування дефекту приносить відчутні незручності багатьом користувачам у межах їх типової діяльності.
  - *Середня* (medium) – існування дефекту слабко впливає на типові сценарії роботи користувачів та/або існує обхідний шлях досягнення.
  - *Низька* (minor) – існування дефекту рідко виявляється незначним відсотком користувачів і (майже) не впливає на їх роботу.
- **Терміновість** (priority) показує, як швидко дефект має бути усунений.
- Градації терміновості:
  - *Найвища* (ASAP, as soon as possible) терміновість вказує на необхідність усунути дефект настільки швидко, наскільки це можливо.
  - *Висока* (high) терміновість означає, що дефект слід виправити позачергово, оскільки його існування або вже об'єктивно заважає

роботі, або почне створювати такі перешкоди у найближчому майбутньому.

- *Звичайна* (normal) терміновість означає, що дефект слід виправити у порядку загальної черги.
- *Низька* (low) терміновість означає, що у найближчому майбутньому виправлення даного дефекту не вплине на підвищення якості продукту.

- **Симптом** (symptom) – дозволяє класифікувати дефекти за їх типовим проявом. Не існує загальноприйнятого списку симптомів. В одного дефекту може бути відразу кілька симптомів.
- Приклади значень симптомів дефекту:
  - *Косметичний дефект* (cosmetic flaw) – візуально помітний недолік інтерфейсу, що не впливає на функціональність програми.
  - *Пошкодження/втрата даних* (data corruption/loss) – внаслідок виникнення дефекту спотворюються, знищуються (або не зберігаються) деякі дані.
  - *Проблема в документації* (documentation issue) – дефект не стосується ні додатку, ні документації.
  - *Некоректна операція* (incorrect operation) – деяка операція виконується некоректно.
  - *Проблема інсталяції* (installation problem) – дефект проявляється на стадії встановлення та/або конфігурування програми.
  - *Помилка локалізації* (localization issue) – щось у програмі не перекладено або перекладено неправильно на вибрану мову інтерфейсу.
  - *Нереалізована функціональність* (missing feature) – якась функція програми не виконується або не може бути викликана.
  - *Проблема масштабованості* (scalability) – зі збільшенням кількості доступних додатку ресурсів не відбувається очікуваного приросту продуктивності програми.

- *Низька продуктивність* (low performance) – виконання деяких операцій займає неприпустимо багато часу.
- *Крах системи* (system crash) – програма припиняє роботу або втрачає здатність виконувати свої ключові функції.
- *Несподівана поведінка* (unexpected behavior) – у процесі виконання певної типової операції програма поводиться незвичайним (відмінним від загальноприйнятого) чином.
- *Недружня поведінка* (unfriendly behavior) – поведінка програми створює користувачеві незручності в роботі.
- *Розбіжність із вимогами* (variance from specs) – цей симптом вказують, якщо дефект складно співвіднести з іншими симптомами, проте додаток веде себе не так, як описано у вимогах.
- *Пропозиція щодо покращення* (enhancement) – у багатьох інструментальних засобах управління звітами про дефекти для цього випадку є окремий вид звіту, оскільки пропозицію щодо покращення формально не можна вважати дефектом: додаток поводиться відповідно до вимог, але у тестувальника є обґрунтована думка про те, як ту чи іншу функціональність можна покращити.

- **Можливість обійти** (workaround) – показує, чи існує альтернативна послідовність дій, виконання якої дозволило б користувачеві досягти поставленої мети.
- **Коментар** (comments, additional info) – може містити будь-які корисні для розуміння та виправлення дефекту дані.
- **Вкладення** (attachments) – це не так поле, як список прикріплених до звіту про дефект додатків.

Інструментальні засоби управління звітами про дефекти

- **Інструмент управління дефектами, інструмент управління інцидентами** – це інструмент, який полегшує запис і відстеження стану дефектів і змін. Вони часто мають засоби, орієнтовані на робочий

процес, для відстеження та контролю розподілу, виправлення та повторного тестування дефектів і надання засобів звітності. Найчастіше такі інструментальні засоби є частинами інструментальних засобів управління тестуванням [1].

- Загальний набір функцій, які зазвичай реалізуються інструментальними засобами управління звітами про дефекти:
  - створення звітів про дефекти, керування їх життєвим циклом з урахуванням контролю версій, прав доступу та дозволених переходів зі стану до стану;
  - збір, аналіз та надання статистики у зручній для сприйняття людиною формі;
  - розсилка повідомлень, нагадувань та інших артефактів відповідним співробітникам;
  - організація взаємозв'язків між звітами про дефекти, тест-кейсами, вимогами та аналіз таких зв'язків із можливістю формування рекомендацій;
  - підготовка інформації для включення до звіту про результати тестування;
  - інтеграція із системами управління проєктами.
- До найбільш вживаних інструментальних засобів управління звітами про дефекти Jira, Bugzilla, Mantis.

Властивості якісних звітів про дефекти

- Властивості, порушення яких ведуть до неякісних звітів про дефекти:
- **Ретельне заповнення всіх полів точною та коректною інформацією.** Причинами порушення цієї властивості можуть бути недостатній досвід тестувальника, неуважність, лінь тощо.
- **Правильна технічна мова.**
- **Специфіка опису кроків.** У звітах про дефекти перевагу, як правило, надають специфічності.

- **Відсутність зайвих дій та/або їх довгих описів.** Найчастіше ця властивість передбачає, що не потрібно в кроках відтворення дефекту довго і за пунктами розписувати те, що можна замінити однією фразою. Друга за частотою помилка – початок кожного звіту про дефект із запуску програми та докладного опису приведення його в той чи інший стан.
- **Відсутність дублікатів.** Коли в проєктній команді працює велика кількість тестувальників, може виникнути ситуація, за якої один і той самий дефект буде описаний кілька разів різними людьми.
- **Очевидність та зрозумілість.** Описуйте дефект так, щоб у читача вашого звіту не виникло жодного сумніву в тому, що це справді дефект.
- **Простежуваність.** З інформації, що міститься в якісному звіті про дефект, має бути зрозуміло, яку частину додатку, які функції і які вимоги зачіпає дефект.
- **Окремі звіти кожного нового дефекту.** Існує два непорушні правила:
    1. У кожному звіті описується **рівно один** дефект.
    2. При виявленні нового дефекту створюється новий звіт. **Не можна** для опису **нового** дефекту правити старі звіти, переводячи їх у стан «відкритий заново».
- **Відповідність прийнятим шаблонам оформлення та традиціям.**

Логіка створення ефективних звітів про дефекти

- Алгоритм створення ефективного звіту про дефект:
    0. Виявити дефект.
    1. Зрозуміти суть проблеми.
    2. Відтворити дефект.
    3. Перевірити наявність опису знайденого вами дефекту у системі управління дефектами.
    4. Сформулювати суть проблеми у вигляді «що зробили, що отримали, що очікували отримати».
    5. Заповнити поля звіту, починаючи з детального опису.

6. Після заповнення всіх полів уважно перечитати звіт, виправивши неточності та додавши подробиці.
7. Ще раз перечитати звіт, оскільки у пункті 6 ви точно щось пропустили.

Типові помилки при написанні звітів про дефекти

Помилки оформлення та формулювань

- Перелік можливих помилок щодо оформлення та формулювання:
  - погані короткі описи (summary);
  - ідентичні короткий та детальний опис (summary і description);
  - відсутність у докладному описі явного надання фактичного результату, очікуваного результату та посилання на вимогу;
  - ігнорування лапок, що призводить до спотворення сенсу;
  - загальні проблеми з формулюваннями фраз українською та англійською мовами;
  - зайві пункти на кроках відтворення;
  - копії екрана як «копії всього екрана повністю»;
  - копії екрана, на яких не позначено проблем;
  - копії екрана та інші артефакти, розміщені на сторонніх серверах;
  - відкладання написання звіту «на потім»;
  - пунктуаційні, орфографічні, синтаксичні та подібні до них помилки.

Логічні помилки

- Перелік можливих логічних помилок:
  - вигадані дефекти;
  - віднесення розширених можливостей застосування до дефектів;
  - невірно вказані симптоми;
  - надмірно занижені (або завищені) важливість та терміновість;
  - концентрація на дрібницях на шкоду головному;
  - технічна безграмотність;

- вказування на кроки відтворення інформації, неважливої для відтворення помилки;
- відсутність у кроках відтворення інформації, що є важливою для відтворення дефекту;
- ігнорування так званих «послідовних дефектів».

Закріплення матеріалу

1. Що таке дефект?
2. Які терміни, пов’язані з поняттям дефекту вам відомі?
3. Що таке звіт про дефект?
4. Які основні цілі переслідуються при написанні звіту про дефект?
5. Які стадії життєвого циклу звіту про дефект вам відомі? Пояснити, що відбувається на кожній з цих стадій.
6. Які атрибути (поля) звіту про дефект вам відомі?
7. Навести алгоритм для створення хороших коротких описів дефектів.
8. Які градації важливості та терміновості вам відомі?
9. Які значення симптомів дефектів вам відомі?
10. Чи може мати один дефект декілька симптомів?
11. Що таке інструментальний засіб управління звітами про дефекти?
12. Які найбільш розповсюджені інструментальні засоби управління звітами про дефекти вам відомі. Описати особливості створення в них звітів про дефекти.
13. Описати властивості, порушення яких ведуть до неякісних звітів про дефекти.
14. Навести алгоритм створення ефективного звіту про дефект.
15. Які різновиди помилок оформлення та формулювання при написанні звітів про дефекти вам відомі?
16. Які різновиди логічних помилок при написанні звітів про дефекти вам відомі?

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 7

ВИДИ ДОДАТКІВ

Мета лекції: Ознайомлення з різними видами програмного забезпечення, принципами взаємодії з ним та особливостями тестування кожного з видів.

Зміст лекції

1. Мобільні додатки.
2. Десктопні додатки.
3. Web-додатки.

Мобільні додатки

- **Мобільний додаток (mobile application)** – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних додатків передвстановлені на самому пристрої або можуть бути завантажені на нього з онлайнових магазинів додатків, таких як App Store, BlackBerry App World, Google Play, 1mobile market, Windows Phone Store тощо, безкоштовно або за плату [15].
- Різновиди мобільних додатків:
  - кросплатформені;
  - нативні;
  - гібридні.
- Основні принципи тестування мобільних додатків:
  - постійна мобільність;
  - знаходження Wi-Fi в екстремальних умовах;

  - переривання;
  - особливості операційних систем і апаратного забезпечення;
  - людський фактор.
- Мобільні операційні системи:
  - Android;
  - iOS;
  - BlackBerry;
  - Windows Phone.
- Переваги операційної системи Android:
  - різноманіття додатків та ігор;
  - відкритий вихідний код;
  - багатозадачність;
  - оперативні оновлення;
  - незалежність від апаратного вмісту мобільного пристрою;
  - широкі можливості індивідуалізації;
  - можливість заміни/видалення додатків, встановлених за замовчуванням.
- Недоліки операційної системи Android:
  - пристрій під управлінням ОС Android необхідно часто перезаряджати;
  - проблеми сумісності;
  - багато налаштувань.
- Переваги операційної системи iOS:
  - постійні оновлення та тривала підтримка застарілих пристроїв;
  - оптимізація і великий вибір додатків;
  - магазин додатків App Store;
  - дизайн;
  - багатозадачність;
  - акцент на надійність і якість.

- Недоліки операційної системи iOS:
  - закрита файлова система;
  - відсутнє пряме копіювання файлів;
  - вартість додатків;
  - обов’язкова наявність Internet;
  - працює лише на пристроях Apple.
- Переваги операційної системи BlackBerry:
  - підтримка всіх сервісів компанії;
  - шифрування даних та миттєвий обмін електронною поштою через сервер компанії;
  - зручний перегляд електронних документів всіх популярних форматів;
  - налаштування меню під кожного окремого користувача.
- Недоліки операційної системи BlackBerry:
  - недостатньо розвинені мультимедійні функції.
- Переваги операційної системи Windows Phone:
  - швидкість;
  - зручний дизайн інтерфейсу;
  - прості та зрозумілі налаштування;
  - підтримка великої кількості пристроїв;
  - відсутні проблеми з оперативною пам’яттю;
  - різноманіття додатків за замовчуванням.
- Недоліки операційної системи Windows Phone:
  - мало додатків або ж вони не повнофункціональні.

Десктопні додатки

- **Десктопні додатки** – це повнофункціональні програми, які працюють незалежно від інших додатків та вимагають наявності оператора. Для їх роботи необхідні достатні апаратні ресурси комп'ютера, сам додаток та набір функцій для роботи з додатком [15].

- Основні особливості тестування десктопних додатків:
  - не вимагають доступу до мережі;
  - повинний бути розгорнутий/встановлений;
  - стандартні інтерфейси та взаємодія;
  - залежність від платформи за виключенням кросплатформених;
  - швидка графіка, анімація;
  - незначні проблеми з аудіо та відео;
  - в наявності шрифти, які встановлені у користувача;
  - відсутній пошук за контентом;
  - для надання доступу іншим комп'ютерам необхідні додаткові налаштування;
  - наявність власних інструментів під кожну платформу;
  - повсюдне використання;
  - тестування виконується QA, групою QA.
- Види тестування, що необхідно проводити на десктопних додатках окрім основних:
  - тестування інсталяції;
  - тестування оновлення;
  - тестування деінсталяції.
- **Інсталятор** – це «звичайна» програма, основні функції якої – встановлення (інсталяція), оновлення та видалення (деінсталяція) програмного забезпечення.

Web-додатки

- **Web-додаток** – це клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером web-сервер, що є по суті двома різними програмами, які необхідно тестувати як окремо, так і у зв'язці [15].
- Особливості тестування Web-додатків:
  - технологічні відмінності;

- структурні відмінності;
- відмінності режимів роботи;
- відмінності формування інтерфейсу;
- відмінності роботи з мережею;
- відмінності запуску та зупинки;
- різниця у кількості користувачів;
- особливості збоїв та відмов;
- відмінності в інсталяції;
- відмінності в деінсталяції;
- особливості середовища функціонування.

Закріплення матеріалу

1. Описати особливості тестування мобільного додатку.
2. Які мобільні операційні системи вам відомі?
3. Описати особливості тестування десктопного додатку.
4. Описати особливості тестування Web-додатку.
5. Для чого потрібний інсталятор?
6. Які додатки не можуть працювати без інсталяції?
7. Описати структуру Web-додатку.
8. Назвати переваги та недоліки операційної системи Android.
9. Назвати переваги та недоліки операційної системи Windows Phone.
10. Назвати переваги та недоліки операційної системи iOS.
11. Назвати переваги та недоліки операційної системи BlackBerry.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 8

КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА

Мета лекції: Ознайомлення з клієнт-серверною архітектурою більшості додатків.

Зміст лекції

1. Загальні відомості.
2. Шаблони Web-архітектури.
3. Структура мобільного додатку.

Загальні відомості

- **Веб-додаток** – це клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер. Основна частина додатку, як правило, знаходиться на стороні веб-сервера, який обробляє отримані запити відповідно до бізнес-логіки продукту і формує відповідь, що надсилається користувачеві. На цьому етапі в роботу включається браузер, який перетворює отриману відповідь від сервера в графічний інтерфейс, зрозумілий користувачеві [15, 16].
- **Клієнт-серверна архітектура** визначає загальні принципи взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) і клієнти (споживачі цих функцій).
- **Клієнт-серверна технологія** – практична реалізація клієнт-серверної архітектури.
- Функції, орієнтовані на розв’язок різних підзадач:

  - функції введення та відображення даних;
  - прикладні функції;
  - функції управління ресурсами.
- Компоненти мережевого додатку:
  - надання даних;
  - прикладна логіка;
  - доступ до ресурсів.
- **Дволанкова архітектура** – розподіл трьох базових компонентів між двома вузлами (клієнтом та сервером). Дволанкова архітектура використовується в клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо та в повному обсязі.
- Розподіл компонентів в дволанковій архітектурі:
  - сервер терміналів – розподілене надання даних;
  - файл-сервер – доступ до віддаленої бази даних та файлових ресурсів;
  - сервер БД – віддалене надання даних;
  - сервер додатків – віддалена програма.
- **Клієнт** – це браузер, але трапляються й винятки (у разі, коли один веб-сервер (ВС1) виконує запит до іншого (ВС2), роль клієнта грає веб-сервер ВС1). У класичній ситуації (коли роль клієнта виконує браузер) для того, щоб користувач побачив графічний інтерфейс програми у вікні браузера, останній повинен обробити отриману відповідь веб-сервера, в якому буде міститися інформація, реалізована із застосуванням HTML, CSS, JS). Саме ці технології «дають зрозуміти» браузеру, як саме необхідно «відмалювати» все, що він отримав у відповіді.
- **Веб-сервер** – це сервер, який приймає HTTP-запити від клієнтів і видає HTTP-відповіді. Веб-сервером називають як програмне забезпечення, що виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює. Найбільш поширеними видами

програмного забезпечення веб-серверів є Apache, IIS і NGINX. На веб-сервері функціонує додаток, що тестується, який може бути реалізований із застосуванням найрізноманітніших мов програмування: PHP, Python, Ruby, Java, Perl тощо.

- **База даних** – це інформаційна модель, що дозволяє впорядковано зберігати дані про об'єкт або групу об'єктів, що мають набір властивостей, які можна категоризувати. Бази даних функціонують під керівництвом систем управління базами даних. Найпопулярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle (всі – клієнт-серверні). База даних фактично не є частиною веб-сервера, але більшість програм просто не можуть виконувати всі покладені на них функції без неї, тому що саме в базі даних зберігається вся динамічна інформація програми (облікові дані, дані користувача тощо).
- **Триланкова архітектура** – мережевий додаток розділений на дві або більше частин, кожна з яких може виконуватися на окремому комп'ютері. Виділені частини програми взаємодіють одна з одною, обмінюючись повідомленнями у заздалегідь узгодженому форматі. Третьою ланкою в триланковій архітектурі стає сервер додатків.
- Розподіл компонентів в триланковій архітектурі:
  - подання даних – на стороні клієнта;
  - прикладний компонент – на виділеному сервері програм;
  - управління ресурсами – на сервері БД, який і представляє дані, що запитуються.
- Триланкова архітектура може бути розширена до **багатоланкової** (N-tier, Multi-tier) шляхом виділення додаткових серверів, кожен з яких представлятиме власні сервіси та користуватиметься послугами інших серверів різного рівня.
- Переваги дволанкової архітектури:
  - проста, оскільки всі запити обслуговуються одним сервером.

- Недоліки дволанкової архітектури:
  - менш надійна;
  - висуває підвищені вимоги до продуктивності сервера.
- Переваги триланкової архітектури:
  - високий ступінь гнучкості та масштабованості;
  - висока безпека (захист можна визначити для кожного сервісу або рівня);
  - висока продуктивність (завдання розподілені між серверами).
- Недоліки триланкової архітектури:
  - складна.
- Клієнт-серверні технології:
  - Web-сервери;
  - сервери додатків;
  - сервери баз даних;
  - файл-сервери;
  - проксі-сервер;
  - файрволи (брандмауери);
  - поштові сервери.
- Основні протоколи «rich»-клієнтів на базі XML:
  - XAML (eXtensible Application Markup Language) – розроблений Microsoft, використовується в додатках на платформі .NET;
  - XUL (XML User Interface Language) – стандарт, розроблений в рамках проєкту Mozilla, використовується, наприклад, у поштовому клієнті Mozilla Thunderbird або браузері Mozilla Firefox;
  - Flex – мультимедійна технологія на основі XML, розроблена Macromedia/Adobe.

Шаблони Web-архітектури

- **Простий Web-клієнт** – найчастіше застосовується з додатками Internet, де неможливо керувати конфігурацією клієнта. Для клієнта потрібен лише звичайний Web-браузер, здатний обробляти форми. Уся бізнес-логіка виконується на сервері.
- **Розширений Web-клієнт** – значна частина бізнес-логіки виконується в системі клієнта. Зазвичай клієнт застосовує для роботи з бізнес-логікою динамічний HTML, аплети Java або елементи ActiveX. Обмін даними з сервером, як і раніше, здійснюється за протоколом HTTP.
- **Web-доставка** – поряд із протоколом HTTP для підтримки розподіленої системи об'єктів, що включає і клієнт, і сервер, можуть застосовуватись такі протоколи як IIOP та DCOM. Web-браузер, головним чином, виступає як агент зберігання та доставки об'єктів у розподіленій системі.

Структура мобільного додатку

- Архітектурні шари додатку, орієнтованого на роботу з сервером:
  - ядро додатку, яке включає компоненти системи, не доступні для взаємодії з користувачем;
  - графічний інтерфейс користувача;
  - компоненти повторного використання: бібліотеки, візуальні компоненти та інше;
  - файли оточення: AppDelegate, .plist тощо;
  - ресурси програми: графічні файли, звуки, потрібні бінарні файли.
- Шари ядра додатку:
  - стартовий шар (start layer) – шар, що визначає робочий процес, початку виконання програми;
  - мережевий шар (network layer) – шар, що забезпечує механізм транспортної взаємодії;

- шар API (API layer) – шар, що забезпечує єдину систему команд взаємодії між клієнтом та сервером;
- шар мережевого кешування (network cache layer) – шар, що забезпечує прискорення взаємодії клієнта і сервера;
- шар валідації даних (validation items layer) – шар валідації даних, отриманих з мережі;
- шар сутності даних (network items layer) – шар сутності даних, що передаються по мережі;
- модель даних (data model) – модель, що забезпечує взаємодію сутності даних;
- шар локального кешування (local cache layer) – шар, що забезпечує локальний доступ до вже отриманих мережевих ресурсів;
- шар робочих процесів (workflow layer) – шар, що включає класи та алгоритми, специфічні для цієї програми;
- локальне сховище (local storage).

Закріплення матеріалу

1. Що таке веб-додаток?
2. Що таке архітектура «клієнт-сервер»?
3. Що таке технологія «клієнт-сервер»?
4. З чого складаються компоненти підключення до мережі?
5. Що таке дволанкова клієнт-серверна архітектура?
6. Чим дволанкова відрізняється від триланкової клієнт-серверної архітектури?
7. Описати мінімальну архітектуру простого веб-клієнта.
8. Що таке розширений веб-клієнт?
9. Що таке веб-доставка?
10. Описати загальну структуру мобільного додатку.
11. Описати шари ядра додатку.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 9

ВЕБ-КЛІЄНТ

Мета лекції: Ознайомлення з поняттям веб-клієнта, його перевагами та недоліками.

Зміст лекції

1. Загальні відомості.
2. Найбільш розповсюджені різновиди браузерів.

Загальні відомості

- **Браузер** (веб-браузер, Web browser) – спеціальна програма, призначена для перегляду веб-сайтів. Перегляд відбувається за допомогою http запитів до сервера та отримання від нього даних, які обробляються за спеціальними затвердженими стандартами і таким чином формується веб-сторінка.
- **Основна мета Інтернет-браузера** – переведення коду, за допомогою якого комп'ютери створюють веб-сайти, у текст, графіку та інші функції веб-сторінок.
- Типи браузерів:
  - браузер режиму командного рядка;
  - повноекранний браузер;
  - браузер із підтримкою мультимедіа;
  - браузери-доповнення;
  - офлайн браузери.

- **Браузери режиму командного рядка** – найраніші браузери – не дають можливості переглядати текст та графіку, підтримують переміщення тільки з використанням цифрових адрес (IP), на даний час практично не використовуються.
- **Повноекраний браузер** – текстовий браузер без підтримки мультимедійних (картинки, анімація тощо) ресурсів мережі Інтернет. За допомогою нього можна переглядати лише текст та посилання. Браузери такого виду використовуються досить рідко, але швидкість завантаження сторінок в них вражає. Без графічних елементів, без таблиць, багато сторінок завантажуються майже миттєво. Один із найпопулярніших повноекранних браузерів – Lynx.
- **Браузер з підтримкою мультимедіа** – найпоширеніші та найпопулярніші браузери на теперішній час – дозволяють працювати практично з усіма видами інформації, представленої в Інтернеті. Найчастіше використовувані: Internet Explorer, Opera, Mozilla, Google Chrome.
- **Браузери-доповнення** – надбудови над повнофункціональними браузерами. Найчастіше розробниками доповнень використовується Internet Explorer. Надбудови використовують для відображення сайтів «движок» цього браузера. Тому їх можливості у цій галузі повністю ідентичні з Internet Explorer. Доповнення лише змінюють інтерфейс і додають деякі функції, які розробники з Microsoft обійшли своєю увагою.
- **Офлайн браузери** – це програми, які автоматично завантажують інформацію з Інтернету та зберігають її на локальному диску комп'ютера для подальшого перегляду і аналізу. Ідея, що лежить в основі роботи офлайн браузерів: користувач вказує адресу Web-сайту, що його цікавить, а програма у відповідь завантажує на його комп'ютер всі

файли, які необхідні для автономного (тобто відключившись від Internet) перегляду цього сайту.

Найбільш розповсюджені різновиди браузерів

- Браузери з підтримкою мультимедіа:
  - Google Chrome;
  - Mozilla Firefox;
  - Internet Explorer;
  - Opera;
  - Apple Safari;
  - Tor.
- **Google Chrome** – єдиний Інтернет-браузер, який швидко піднявся в рейтингах популярності, як тільки він був представлений в 2009 році. Це один з найчастіше використовуваних браузерів всіх часів, що займає понад 60 % завдяки швидкодії, зручному інтерфейсу та елементам управління безпекою.
- Переваги Google Chrome:
  - простий інтерфейс з обмеженою кількістю інструментів на головній сторінці;
  - висока швидкість завантаження веб-сторінок;
  - наявність корисних розширень для підвищення швидкості перегляду і безпеки;
  - нові вбудовані інструменти, такі як PDF Viewer, Language Translator тощо;
  - легка інтеграція з іншими програмами Google, такими як Gmail, Диск, AdSense, Реклама Google.
- Недоліки Google Chrome:
  - використовує багато системної пам'яті;
  - порівняно менше можливостей для налаштування кнопок та меню.

- **Mozilla Firefox** – один з найбільш використовуваних браузерів; сумісний з Windows, Android, OSX, iOS; дозволяє користувачеві здійснювати приватний перегляд, запобігаючи його відстеженню; має можливість потужного антивірусу, який блокує спливаючі вікна; дозволяє працювати без підключення до Інтернету, але з можливістю синхронізації всіх наявних у користувача пристроїв.
- Переваги Mozilla Firefox:
  - швидкий та якісно зроблений браузер;
  - підтримує всі стандарти W3C;
  - дозволяє працювати як з вікнами, так і з вкладками всередині кожного вікна;
  - підтримує плагіни;
  - має клієнтів для роботи з поштою, у news конференціях та IRC;
  - постійно оновлюється.
- Недоліки Mozilla Firefox:
  - вибагливий до ресурсів;
  - тривалий час завантаження програми;
  - помічені в закладках веб-сторінки нелегко знайти;
  - цільова сторінка заповнена безліччю рекомендованих посилань та оголошень.
- **Internet Explorer** та **Microsoft Edge**. Флагманський веб-браузер Internet Explorer, створений для Windows 95 компанією Microsoft, є одним із найпопулярніших браузерів на ринку завдяки інструментам забезпечення конфіденційності та безлічі опцій налаштування. Microsoft Edge – це один з найкращих і найшвидших браузерів для Windows 10.
- Переваги Microsoft Edge:
  - найлегший браузер для Windows 10;
  - надійний та простий у використанні;

  - один із найшвидших браузерів для Windows 10 у порівнянні з іншими варіантами;
  - один із найбезпечніших браузерів для ПК з Windows завдяки попередженням користувачів про небезпечні веб-сайти під час перегляду;
  - найкращий браузер для завантаження веб-сайтів у вигляді програм.
- Недоліки Microsoft Edge:
  - несумісний з комп'ютерами під управлінням попередніх версій Windows;
  - доступно менше розширень для перегляду, ніж в інших популярних браузерах.
- **Opera** – це один із найстаріших веб-браузерів, існуючих на сьогоднішній день, що отримав кілька активних розробок: робочі простори, інтегровані месенджери, конвертер одиниць вимірювання, інструмент для створення знімків, криптогаманець тощо.
- Переваги Opera:
  - дозволяє прикріплювати ярлики до улюблених сайтів та найчастіше використовуваних налаштувань для підвищення ефективності;
  - дозволяє переглядати веб-сторінки за допомогою голосових команд;
  - дозволяє синхронізувати всі пристрої користувача, на яких він його використовує;
  - постачається із вбудованою функцією безпеки, яка дозволяє користувачам перевіряти веб-сайти на наявність шкідливого контенту та інших інфекцій.
- Недоліки Opera:
  - повільніший у порівнянні з іншими найпопулярнішими браузерами.
- **Apple Safari** – це браузер, який можна інтегрувати в MAC OS x, крім того, що він має виконувані версії, які належать Microsoft Windows.

- Переваги Apple Safari:
  - дозволяє користувачам організовувати закладки;
  - має перевірку правопису, щоб полегшити введення тексту та уникнути помилок друку;
  - блокує спливаючі вікна;
  - суворий з точки зору управління паролями та безпеки;
  - додає програвач мультимедійних файлів;
  - має яскраву 3D-«зовнішність» і відмінні функції для неспішного читання веб-сторінок та зручні інструменти на кшталт RSS або вбудованого рідера.
- Недоліки Apple Safari:
  - безнадійно застарів у питаннях безпеки та функціональності;
  - велика кількість соціальних мереж і мультимедійних сервісів (YouTube, Google Map тощо) відмовляються працювати в браузері Safari;
  - не варто користуватися цим браузером там, де потрібна авторизація і, тим більше, є можливість онлайн-оплати послуг.
- **Tor** – це інструмент всеосяжного вирішення питань анонімності у мережі. По суті, це програмне забезпечення, яке з дуже високою ймовірністю дозволяє приховати від сторонніх очей все, що ви робите і робили в Інтернеті. На основі цієї технології працює Tor-браузер.
- Переваги Tor:
  - відкритий вихідний код;
  - найкраща пошукова система, яка дбає про конфіденційність – Surf Deep, Dark і навіть заблоковані веб-сайти доступні без жодних обмежень із цим настільним браузером;
  - повна анонімність, можливість приховати вихідну IP-адресу;
  - можливість перегляду непроіндексованих сторінок в Яндекс, Bing та Google;

  - є браузери для iPhone та Android.
- Недоліки Tor:
  - не рекомендується використовувати з BitTorrent або будь-якими іншими торент-клієнтами;
  - знижена швидкість смуги пропускання;
  - застарілий інтерфейс.
- Додаткові браузери:
  - Netsurf;
  - Neoplanet;
  - MyIE2 (Maxthon);
  - Internet Surfer;
  - Vivaldi;
  - Brave;
  - Chromium;
  - Comodo IceDragon;
  - Konqueror.
- Різновиди браузерів для мобільних пристроїв:
  - для Android;
  - для iOS;
  - без ОС.

Закріплення матеріалу

1. Що таке браузер?
2. Які бувають движки браузера?
3. Що таке мультимідійний браузер?
4. У чому особливість браузера Top?
5. На основі якого браузера створено Top?
6. Чому тестувальник має користуватися не одним браузером?

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 10

ДАНІ ТА ЇХ ВПЛИВ НА ПРОГРАМУ

Мета лекції: Ознайомлення з різними типами даних, які згодом прямо чи опосередковано впливатимуть на програмний продукт і які також потрібно буде протестувати.

Зміст лекції

1. Управління доступом на основі ролей.
2. Поняття конфігурацій.
3. Часові пояси.
4. Локалізації.

Управління доступом на основі ролей

- **Role Based Access Control (RBAC)** – це модель дозволів, що використовується в Microsoft Exchange Server. При використанні RBAC відпадає потреба у зміні та підтримці списків керування доступом (ACL) [17].
- Проблеми, пов’язані зі списками ACL:
  - важко змінювати списки ACL без непередбачених наслідків;
  - важко зберігати зміни ACL після оновлення;
  - важко діагностувати проблеми, що виникають через нестандартне використання.
- Модель RBAC дозволяє контролювати, які дії можуть здійснювати адміністратори та кінцеві користувачі.

- RBAC дозволяє більш точно зіставляти ролі, призначені користувачам та адміністраторам, зі своїми фактичними ролями в організації.
- Базова модель RBAC
  - S = Суб'єкт (Subject) = Людина або автоматизований агент (множина користувачів);
  - R = Роль (Role) = Робоча функція або назва, яка визначається на рівні авторизації (множина ролей);
  - P = Дозволи (Permissions) = Затвердження режиму доступу до ресурсу (множина прав доступу до об'єктів системи);
  - SE = Сесія (Session) = Відповідність між S, R та/або P;
  - SA = Призначення суб'єкта (Subject Assignment);
  - PA (Permission Assignment): $R \rightarrow 2^p$ – функція, що визначає для кожної ролі множину прав доступу; при цьому для кожного $p \in P$ існує $r \in R$ така, що $p \in PA(r)$;
  - RH = Частково впорядкована ієрархія ролей (Role Hierarchy). Ще один запис RH: $\geq$;
  - Один суб'єкт може мати кілька ролей.
  - Одну роль можуть мати кілька суб'єктів.
  - Одна роль може мати кілька дозволів.
  - Один дозвіл може належати кільком ролям.
- Можливість різних видів доступу окремих учасників організації до ресурсів залежить від прав доступу, наданих їм в організації. Типи користувачів дозволяють організаціям керувати правами доступу, які можуть надаватися учасником через ролі.
- Можливі типи користувачів:
  - **Viewer** – тип користувача, який може переглядати елементи, опубліковані для нього іншими користувачами і має доступ до

вибору додатків; не може створювати, редагувати, публікувати чи виконувати аналіз елементів або даних.

- **Editor** – тип користувача, який може переглядати та редагувати дані в картах та програмах, доступ до яких йому надали інші користувачі; не може виконувати аналіз, створювати чи публікувати елементи або дані.
- **Worker** – тип користувача, який може переглядати та редагувати дані, доступ до яких йому було надано іншими користувачами, та має доступ до вибору програм; не може виконувати аналіз, створювати чи публікувати елементи чи дані.
- **Creator** – тип користувача, який може створювати та редагувати ресурси, виконувати поглиблений просторовий аналіз за допомогою інструментів аналізу на порталі, збирати дані, а також співпрацювати та обмінюватися ресурсами для використання у додатках; має доступ до вибору програм, може переглядати ресурси, створені іншими учасниками організації, та керувати користувачами і ресурсами в організації.
- **Professional** – тип користувача, який може створювати розширені ресурси, використовувати спеціальне програмне забезпечення для створення, редагування, співпраці та обміну ресурсами; може керувати користувачами та ресурсами в організації.
- **Insights Analyst** – тип користувача, який має усі можливості, у тому числі створення та публікацію ресурсів і виконання аналізу.
- **Storyteller** – тип користувача, який дозволяє створювати історії за допомогою спеціального програмного забезпечення.

- Роль визначає набір прав, наданих учаснику. Права призначаються учасникам за допомогою ролі за замовчуванням або ролі користувача. Роль надається учаснику в останній момент, коли він запрошується в організацію.

- Різновиди ролей користувача:
  - вьюер;
  - редактор даних;
  - користувач;
  - видавець;
  - адміністратор.

Поняття конфігурацій

- **Конфігурація** – сукупність значень параметрів, які визначають роботу пристрою.
- **Конфігурація програмного забезпечення** – це сукупність налаштувань програми, що задається користувачем, а також процес зміни цих налаштувань відповідно до потреб користувача.
- **Конфігурація додатку** – це все, що може змінюватися між розгортаннями (середовище розробки, проміжне та робоче розгортання). Це включає:
  - ідентифікатори підключення до ресурсів типу бази даних, кеш-пам'яті та інших сторонніх служб;
  - реєстраційні дані для підключення до зовнішніх сервісів;
  - значення, залежні від середовища розгортання.
- Різновиди збереження конфігурацій:
  - константи в коді;
  - конфігураційні файли;
  - змінні оточення;
  - групування;
  - спеціальна база даних.

Часові пояси

- **Часовий пояс** – це географічний регіон, у якому використовується єдиний місцевий час, встановлений урядом країни. Багато країн цілком

належать до якихось конкретних часових поясів, а на територіях великих держав застосовується кілька часових поясів.

- **Адміністративний часовий пояс** – ділянка земної поверхні, на якій відповідно до певного закону встановлено певний офіційний час.
- **Time zone** – поняття, еквівалентне адміністративному часовому поясу.
- **GMT** (Greenwich Mean Time) – середній час за Грінвічем, тобто це час на годиннику Королівської обсерваторії в Грінвічі, Великобританія. Вона розташована на нульовому меридіані. Радіосигнал GMT-системи почав транслюватися з 5 лютого 1924 року, а сама вона перетворилася на світовий стандарт з 1 січня 1972 року.
- **Система UTC** – це система, яка з'явилася у 1972 році як спосіб компенсації ефекту обертання Землі. В основі системи лежить Міжнародний атомний час (International Atomic Time), що обчислюється за частотою електромагнітних коливань атомів цезію.
- UTC – більш точна заміна GMT.
- **DST** (Daylight Saving Time) – це переведення годинника в літній період на одну годину вперед відносного стандартного часу.
- **IANA Time Zone Database** – стандартна база даних, яка містить історичні дані про зміни стандартного часу та DST по всій земній кулі; організована так, щоб можна було перевірити всі історичні дані та переконатися в точності часу, починаючи з Unix time (1970.01/01 00:00:00) – хоча можна знайти в базі інформацію до 1970-го, проте її точність не гарантується.
- Windows використовує базу даних Microsoft, проте, вона неточна з погляду історичних даних і підтримується лише Microsoft. Менш надійна, ніж база IANA.

Локалізації

- **Локалізація програмного продукту** – приведення програмного продукту у відповідність із законами та іншими нормативно-правовими

актами, стандартами, нормами і правилами, що діють в країні, для якої проводиться локалізація.

- **Локалізація** (localization) – переклад і адаптація елементів інтерфейсу, допоміжних файлів та документації.
- **Локалізація бага** – пошук першопричини виникнення помилки.
- **Локалізація програмного забезпечення** – переклад на різні мови.
- Те, що тестується в багатомовних програмах:
  - адекватність перекладу;
  - пункти меню;
  - кнопки;
  - рисунки;
  - довгі фрази.
- Те, що важливо тестувати при локалізації:
  - всі взаємодії з користувачем;
  - взаємодії з обладнанням;
  - переходи по сторінках і назад зі зміною мови;
  - правильний вибір мови за замовчуванням.

Закріплення матеріалу

1. Що таке «ролі користувачів» та чим це зручно?
2. В яких випадках використання «ролей» не є виправданим?
3. Що таке «конфігурація» програми?
4. В яких випадках використання різних конфігурацій є виправданим?
5. Що таке time zone та для чого вони потрібні?
6. Що таке локалізація?
7. Що зазвичай робить тестувальник, якому слід протестувати локалізацію?
8. Чи потрібно знати мову локалізації, яку тестуєш, та чому?

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 11

БАЗИ ДАНИХ

Мета лекції: Ознайомлення з основами роботи з базами даних та набуття розуміння важливості мови SQL в роботі тестувальника.

Зміст лекції

1. Поняття бази даних.
2. Необхідність знання SQL для тестувальника.
3. Мови опису та маніпулювання даним.
    3.1. Оператори мови опису даних.
    3.2. Оператори мови маніпулювання даними.
    3.3. Найбільш вживані в тестуванні оператори SQL.

Поняття бази даних

- **База даних** – це сукупність взаємозалежних даних певної предметної галузі, збережених у пам'яті комп'ютера і організованих таким чином, що ці дані можуть бути використані для розв'язання різних завдань багатьма користувачами [18].
- **База даних** – набір логічно пов'язаних даних спільного використання (і опис цих даних), призначений для задоволення інформаційних потреб.
- **База даних** – сукупність зв’язаних даних, організована за певними правилами, які передбачають загальні принципи опису, збереження та маніпулювання даними, та незалежна від прикладних програм.

- **База даних** – сукупність даних, організованих відповідно до концептуальної структури, що описує характеристики цих даних та взаємовідносини між ними, причому таке накопичення даних, яке підтримує одну або більше областей застосування.
- Характерні ознаки БД:
  - єдине сховище даних, яке визначається одноразово, а потім одночасно використовується багатьма користувачами;
  - загальний корпоративний ресурс;
  - зберігає не лише дані, а і їх опис.
- Вимоги до бази даних:
  - ненадлишковість даних;
  - спільне використання даних;
  - можливість розширення бази даних;
  - простота роботи з базою даних;
  - ефективність доступу до бази даних;
  - цілісність бази даних;
  - захист даних.
- **Реляційна база даних** – це сукупність відношень, що містять всю інформацію, яка повинна зберігатися в БД [18].
- Вимоги до реляційної бази даних:
  - кожна таблиця складається з однотипних рядків і має унікальне ім'я;
  - рядки мають фіксоване число полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі);
  - рядки таблиці обов'язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці;

  - стовпцям таблиці однозначно присвоюються імена і в кожному з них розміщуються однорідні значення даних;
  - повний інформаційний зміст бази даних представляється у вигляді явних значень даних і такий метод представлення є єдиним;
  - при виконанні операцій з таблицею її рядки і стовпці можна обробляти в будь-якому порядку, не зважаючи на їх інформаційний зміст.
- Недоліки реляційних баз даних:
  - модель не надає достатніх засобів для представлення змісту даних;
  - для багатьох програм важко моделювати предметну галузь на основі плоских таблиць;
  - реляційна модель даних не пропонує апарату для поділу сутностей і зв'язків.

Необхідність знання SQL для тестувальника

- Для роботи з одними додатками тестувальнику необхідні глибокі навички перевірки SQL, для роботи з другими – середні, а для деяких додатків знання SQL взагалі не потрібні.
- Знання баз даних та SQL, необхідні тестувальнику:
  - вміння розпізнати різні типи баз даних;
  - вміння підключитися до бази даних за допомогою різних клієнтів SQL-з’єднань;
  - розуміння відносин між таблицями бази даних, ключами та індексами;
  - вміння написання простого оператору вибору або SQL запиту на з’єднання;
  - вміння інтерпретувати складніші запити.
- Обгрунтування важливості SQL для тестування програмного забезпечення:

- SQL дає розробникам більш адекватний, так би мовити, зворотний зв'язок;
- SQL допомагає зрозуміти, чи додаються на бекенд дані, які додаються до форми (на frontend);
- SQL допомагає отримати тестові дані;
- SQL допомагає в автоматизації тестування.

Мови опису та маніпулювання даним

- **SQL** – це стандартна комп'ютерна мова для управління базами даних та для обробки даних. SQL використовується для запиту, вставки, оновлення та зміни даних.
- **SQL** – це засіб зв'язку між користувачем та системою управління базою даних.
- **SQL** – це мова програмування, за допомогою якої можна звертатися до бази даних.
- **SQL** – це приклад мови з трансформуючою орієнтацією, або ж мови, призначеної для роботи з таблицями з метою перетворення вхідних даних до необхідного вихідного виду [18].
- Основні компоненти мови SQL:
  - **мова опису даних** (DDL, Data Definition Language) – призначена для визначення структур бази даних;
  - **мова маніпулювання даними** (DML, Data Manipu1ation Langllage) – призначена для вибірки і поновлення даних.
- Мова SQL включає тільки команди опису та маніпулювання даними – в ній відсутні будь-які команди управління ходом обчислень.
- Способи використання мови SQL:
  - інтерактивна робота – полягає у введенні користувачем окремих SQL-операторів;
  - впровадження SQL-операторів в програми на інших мовах.

- Характеристики мови SQL:
  - не вимагає вказування методів доступу до даних (непроцедурна мова);
  - підтримує вільний формат запису операторів (при введенні окремі елементи операторів не пов'язані фіксованими позиціями екрану);
  - структура команд задається набором ключових слів, які представляють собою звичайні слова англійської мови (CREATE TABLE (створити таблицю), INSERT (вставити), SELECT (вибрати), тощо);
  - може використовуватися широким колом користувачів.
- Типи даних мови SQL:
  - текстовий (CHAR або TEXT);
  - числовий (BYTE, INT, LONG, FLOAT, REAL);
  - часовий (DATE, TIME, DATETIME);
  - грошовий (MONEY);
  - лічильник (COUNTER);
  - логічний (LOGICAL);
  - поле MEMO;
  - об'єкт (IMAGE).

Оператори мови опису даних

- Оператори мови опису даних призначені для створення опису, зміни опису і знищення об'єктів бази даних.
- Види об'єктів в SQL:
  - база даних (database);
  - таблиця (table);
  - стовпець (column);
  - індекс (index);
  - представлення (view);

- синонім (synonym).

- Основні оператори мови SQL, призначені для опису даних:
  - створення об’єктів: CREATE SCHEMA, CREATE DOMAIN, CREATE DATABASE, CREATE TABLE, CREATE VIEW, CREATE INDEX;
  - зміна існуючого об’єкта: ALTER DOMAIN, ALTER TABLE;
  - видалення об'єкта: DROP SCHEMA, DROP DOMAIN, DROP DATABASE, DROP TABLE, DROP VIEW, DROP INDEX.

Оператори мови маніпулювання даними

- Оператори мови маніпулювання даними призначені для заповнення таблиць даними, вибірки з них інформації за допомогою запитів, внесення змін в дані, які зберігаються в БД, видалення даних з таблиць.
- Основні оператори мови SQL, призначені для маніпулювання даними:
  - SELECT – вибірка даних з бази;
  - INSERT – вставка даних в таблицю;
  - UPDATE – оновлення (зміна) даних таблиці;
  - DELETE – видалення даних з таблиці.

Найбільш вживані в тестуванні оператори SQL

- Оператори SQL, які найчастіше використовуються у тестуванні:
  - SELECT;
  - UPDATE;
  - INSERT;
  - CREATE;
  - ALTER;
  - DROP;
  - COMMIT – успішне завершення транзакції;
  - ROLLBACK – скасування змін, внесених під час поточної явної чи неявної транзакції до початку транзакції або точки збереження;

- INNER JOIN – вибірка зіставлених записів з обох таблиць;
- DISTINCT – вибірка різних значень з одного або кількох полів;
- IN – перевірка наявності значення у списку;
- BETWEEN – отримання значень у діапазоні;
- WHERE – отримання необхідних рядків;
- LIKE – виконання зіставлення з шаблоном (використовується з оператором WHERE);
- ORDER BY – сортування записів таблиці в порядку зростання або спадання;
- GROUP BY – групування рядків із загальною властивістю так, що агрегатна функція може бути застосована до кожної групи;
- HAVING – обрання з груп, визначених оператором GROUP BY;
- Aggregate Functions – обчислення для набору значень і повернення одного значення (AVG, MIN, MAX, SUM, COUNT тощо).

Закріплення матеріалу

1. Що таке база даних?
2. Назвати характерні ознаки баз даних.
3. Назвати вимоги до бази даних.
4. Назвати вимоги до реляційної бази даних.
5. Вказати недоліки реляційної баз даних.
6. Для яких видів робіт тестувальнику необхідні знання мови SQL?
7. З яких частин складається мова SQL?
8. Назвати основні оператори мови маніпулювання даними.
9. Назвати основні оператори мови опису даних.
10. Назвати найбільш вживані в тестуванні оператори SQL.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.4 БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ

Лекція 12

ТЕСТ-ДИЗАЙН

Мета лекції: Ознайомлення з поняттям та техніками тест-дизайну.

Зміст лекції

1. Поняття, цілі, завдання та навички, необхідні для тест-дизайну.
2. Техніки тест-дизайну.

Поняття, цілі, завдання та навички, необхідні для тест-дизайну

- **Тест-дизайн** – це етап процесу тестування програмного забезпечення, на якому проєктуються та створюються тестові випадки (тест-кейси), у відповідності до визначених раніше критеріїв якості та цілей тестування [15, 19].
- **Тест-дизайн** – процес перетворення загальних цілей тестування на реальні тестові умови та тестові приклади.
- **Тестове покриття** – це одна з метрик оцінки якості тестування, що представляє собою щільність покриття тестами вимог, або виконуваного коду.
- **Тест-дизайнер** повинен побудувати процес тестування всіх найважливіших елементів програмного продукту, використовуючи мінімально можливу кількість перевірок.
- Цілі тест-дизайну:
  - вигадати тести, які виявлять найбільш серйозні помилки продукту, в іншому випадку тестування буде неефективним;

  - мінімізувати кількість тестів, необхідних для знаходження більшості серйозних помилок – перед розробниками тестів завжди стоїть завдання зберегти ефективність тестів (тобто їх здатність виявляти серйозні помилки) без збільшення їх кількості.
- Задачі тест-дизайну:
  - проаналізувати вимоги до продукту;
  - оцінити ризики, можливі під час використання продукту;
  - написати достатню мінімальну кількість тестів;
  - розмежувати тести на приймальні, критичні, розширені.
- Необхідні навички:
  - вміння розділяти систему на складові (робити декомпозицію) – тобто треба вміти бачити систему як ціле, так і вміти розкладати її на складові;
  - вміння збирати та аналізувати вимоги до продукту – якщо немає формально описаних вимог, потрібно вміти їх збирати у розробників, аналітиків, користувачів;
  - вміння розставляти пріоритети – тест-дизайнер повинен вміти відрізняти більш важливе від менш важливого та розставляти пріоритети тестування;
  - вміння формулювати свої думки (письмово та усно) – це вміння важливе для тестувальника в принципі;
  - знання технік тест-дизайну;
  - вміння застосовувати техніки тест-дизайну на практиці.

Техніки тест-дизайну

- **Техніки тест-дизайну** – це рекомендації, поради та правила, за якими необхідно розробляти тест для проведення тестування.
- **Розбиття на класи еквівалентності** – це техніка, яка полягає в розбитті всього набору тестів на класи еквівалентності з подальшим скороченням числа тестів.

- Мета техніки – не лише скорочення кількості тестів, а й збереження прийнятного тестового покриття.
- Приблизний алгоритм використання техніки:
  1. Визначити класи еквівалентності.
  2. Вибрати одного представника кожного класу. На цьому кроці з кожного еквівалентного набору тестів обирається один тест.
  3. Виконати тести. На цьому кроці виконуються тести від кожного класу еквівалентності.
- **Аналіз граничних значень** – це перевірка значеннями, що знаходяться на границях класів еквівалентності. Необхідно перевірити кожну границю класу еквівалентності з обох боків.
- **Передбачення помилки** – це коли тест-аналітик використовує свої знання системи та здатність інтерпретувати специфікацію на предмет того, щоб «передбачити» за яких вхідних умов система може видати помилку.
- **Причина/Наслідок** – це, як правило, введення комбінацій умов (причин) для отримання відповіді від системи (наслідок).
- **Вичерпне тестування** – це крайній випадок. В межах даної техніки необхідно перевірити всі можливі комбінації вхідних значень, і, в принципі, це дозволить знайти всі проблеми. На практиці застосування цього методу не представляється можливим, через величезну кількість вхідних значень.
- **Попарне тестування** – техніка тестування, в якій замість перевірки всіх можливих комбінацій значень всіх параметрів перевіряються лише комбінації значень кожної пари параметрів. Полягає вона в наступному: формуються такі набори даних, в яких кожне тестоване значення кожного з параметрів, що перевіряються, хоча б один раз поєднується з кожним тестованим значенням всіх інших параметрів, що перевіряються.

- Утиліти для автоматизації pairwise testing:
  - Pairwise Independent Combinatorial Testing, provided by Microsoft Corp;
  - IBM FoCuS – “Functional Coverage Unified Solution“, provided by IBM;
  - ACTS – “Advanced Combinatorial Testing System“,an agency of the US Government;
  - Hexawise;
  - Jenny;
  - Pairwise by Inductive AS;
  - VPTag free All-Pair Testing Tool.
- **Таблиця прийняття рішень** – це зручний інструмент для фіксації вимог та опису функціональності програми. Таблицею зручно описувати бізнес-логіку програми і вони можуть служити чудовою основою для тест-кейсів.
- Таблиці прийняття рішень описують логіку додатку на основі умов системи, що характеризують її стани. Кожна таблиця повинна описувати один стан системи.
- **Тестування переходів станів** (state transition testing) – це розробка тестів методом чорної скриньки, в якій сценарії тестування будуються на основі виконання коректних та некоректних переходів станів.

Закріплення матеріалу

1. Що таке тест-дизайн?
2. Що таке покриття вимогами?
3. Що таке покриття коду?
4. Назвати цілі тест-дизайну.
5. Які існують техніки тест-дизайну?

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.5 ТРУДОВИТРАТИ

Лекція 13

ПЛАНУВАННЯ ТА ЗВІТНІСТЬ

Мета лекції: Ознайомлення з поняттями тест-плану та звітності, розгляд принципів створення звіту про результати тестування.

Зміст лекції

1. Поняття планування та звітності.
2. Тест-план та звіт про результати тестування.
    2.1. Тест-план.
    2.2. Звіт про результати тестування.
    2.3. Логіка побудови звіту про результати тестування.

Поняття планування та звітності

- Кожна ітерація починається з планування та закінчується звітністю, яка стає основою для планування наступної ітерації. Планування та звітність перебувають у тісному взаємозв'язку, і проблеми з одним з цих видів діяльності неминуче призводять до проблем з іншим видом, а зрештою і до проблем із проєктом в цілому.
- **Планування** (planning) – це безперервний процес прийняття управлінських рішень з орієнтацією на майбутнє та методичної організації зусиль, необхідних для виконання цих рішень [20].
- Високорівневі завдання планування:
    - зниження невизначеності;
    - підвищення ефективності;

  - покращення розуміння цілей;
  - створення основи для управління процесами.
- Основні причини планування проєкту:
  - усунення або зменшення невизначеності;
  - підвищення ефективності операції;
  - краще розуміння цілі;
  - створення основи для моніторингу та контролю роботи.
- **Звітність** (reporting) – збір та розповсюдження інформації про ефективність (включаючи звіти про стан, оцінку прогресу та прогнозування) [21].
- Високорівневі завдання звітності:
  - збір, агрегація та надання у зручній для сприйняття формі об'єктивної інформації про результати роботи;
  - формування оцінки поточного статусу та прогресу (порівняно з планом);
  - позначення існуючих та можливих проблем;
  - формування прогнозу розвитку ситуації та фіксація рекомендацій щодо усунення проблем та підвищення ефективності роботи.

Тест-план та звіт про результати тестування

Тест-план

- **Тест-план** (test plan) – це документ, що описує обсяг, підхід, ресурси та графік запланованих заходів з тестування. Він визначає серед іншого тестові елементи, функції, які підлягають тестуванню, завдання тестування, хто виконуватиме кожне завдання, ступінь незалежності тестувальника, тестове середовище, методи розробки тесту та критерії входу та виходу, які будуть використані, а також обґрунтування їх вибору та будь-які ризики, що вимагають планування на випадок непередбачених обставин [1].
- Низькорівневі завдання планування у тестуванні:

  - оцінка обсягу та складності робіт;
  - визначення необхідних ресурсів та джерел їх отримання;
  - визначення розкладу, термінів та ключових точок;
  - оцінка ризиків та підготовка превентивних контрзаходів;
  - розподіл обов'язків та відповідальності;
  - узгодження робіт із тестування з діяльністю учасників проектної команди, що займаються іншими завданнями.
- Якісний тест-план володіє більшістю властивостей якісних вимог, а також розширює їх набір наступними пунктами:
  - реалістичність;
  - узгодженість із загальним проєктним планом та іншими окремими планами;
  - гнучкість.
- Тест-план створюється на початку проєкту і доопрацьовується при необхідності протягом усього часу життя проєкту за участю найбільш кваліфікованих представників проєктної команди, задіяних у забезпеченні якості.
- Відповідальним за створення тест-плану, як правило, є ведучий тестувальник («тест-лід»).
- Розділи тест-плану:
  - ціль;
  - області, що піддаються тестуванню;
  - області, які не піддаються тестуванню;
  - тестова стратегія;
  - тестовий підхід;
  - критерії;
  - ресурси;
  - розклад;
  - ролі та відповідальність;

- оцінка ризиків;
- документація;
- метрики;
- покриття.

- **Ціль** (purpose) –це гранично короткий опис мети розробки додатку.
- **Області, що піддаються тестуванню** (features to be tested) – це список функцій та/або нефункціональних особливостей додатку, які будуть піддаватися тестуванню.
- **Області, які не піддаються тестуванню** (features not to be tested) – це перелік функцій та/або нефункціональних особливостей додатку, які не будуть піддаватися тестуванню.
- **Тестова стратегія** (test strategy) – це високорівневий опис тестових рівнів, які необхідно виконати, і тестування на цих рівнях для організації або додатку.
- **Тестовий підхід** (test approach) – це реалізація тестової стратегії для конкретного проєкту. Зазвичай він включає в себе рішення, прийняті на основі мети (тестового) проєкту та проведеної оцінки ризику, вихідні точки щодо процесу тестування, методи проєктування тестів, які потрібно застосувати, критерії виходу та типи тестів, які необхідно виконати.
- **Критерії** (criteria) включають в себе декілька різновидів.
- Види критеріїв:
  - приймальні критерії, критерії якості;
  - критерії початку тестування;
  - критерії призупинення тестування;
  - критерії відновлення тестування;
  - критерії завершення тестування.
- **Приймальні критерії, критерії якості** (acceptance criteria) – це критерії виходу, яким повинен задовольняти компонент або система, щоб бути

прийнятими користувачем, клієнтом або іншим уповноваженим органом.

- **Критерії початку тестування** (entry criteria) – це набір загальних і специфічних умов, що дозволяють процесу йти вперед із визначеним завданням. Мета критеріїв початку тестування – запобігти запуску завдання, яке спричинило б більше (марних) зусиль у порівнянні з зусиллями, необхідними для видалення невдалих критеріїв входу.
- **Критерії призупинення** (suspension criteria) – це критерії, які використовуються для (тимчасової) зупинки всіх або частини тестових заходів для елементів тестування.
- **Критерії відновлення тестування** (resumption criteria) – це критерії, які використовуються для перезапуску всіх або частини тестових заходів, що були призупинені раніше.
- **Критерії завершення тестування** (exit criteria) – набір загальних і специфічних умов, узгоджених із зацікавленими сторонами для дозволу офіційного завершення процесу. Метою критеріїв завершення тестування є запобігання тому, щоб завдання вважалося виконаним, якщо є ще невиконані частини. Критерії завершення використовуються для звітування та планування часу припинення тестування.
- **Ресурси** (resources) поділяються на:
  - програмні ресурси;
  - апаратні ресурси;
  - людські ресурси;
  - часові ресурси;
  - фінансові ресурси.
- **Розклад тесту** (test schedule) – це список дій, завдань або подій процесу тестування з визначенням передбачуваних дат і/або часу їх початку та завершення, а також взаємозалежностей.

- **Ролі та відповідальність** (roles and responsibility) – це перелік необхідних ролей та область відповідальності фахівців, які виконують ці ролі.
- **Оцінка ризиків** (risk evaluation) – це перелік ризиків, які з високою ймовірністю можуть виникнути в процесі роботи над проєктом.
- **Документація** (documentation) – це перелік використовуваної тестової документації із зазначенням, хто і коли повинен її готувати та кому передавати.
- **Метрика** (metric) – числова характеристика показника якості. Може включати опис способів оцінки та аналізу результату [1, 16, 22].
- **Метрика** – це шкала вимірювання та метод вимірювання.
- Види метрик:
  - прямі (не вимагають обчислень);
  - розрахункові (обчислюються за формулою).
- **Покриття, тестове покриття** (coverage) – це ступінь, виражена у відсотках, в якій до певного елементу покриття був використаний набір тестів.
- **Елемент покриття (coverage item)** – це сутність або властивість, які використовуються як основа для тестового покриття.
- Найпростіші представники метрик покриття:
  - метрика покриття вимог (вимога вважається «покритою», якщо на неї посилається хоча б один тест-кейс);
  - метрика щільності покриття вимог (враховується, скільки тест-кейсів посилається на кілька вимог);
  - метрика покриття класів еквівалентності (аналізується, скільки класів еквівалентності зачіпалося тест-кейсами);
  - метрика покриття граничних умов (аналізується, скільки значень з групи граничних умов зачіпалося тест-кейсами);

- метрики покриття коду модульними тест-кейсами (виявляється певна характеристика коду та визначається, який відсоток представників цієї характеристики покритий тест-кейсами).

Звіт про результати тестування

- **Звіт про хід тестування** (test progress report) – це документ, що підсумовує діяльність з тестування, підготований на регулярних умовах, для звіту про хід діяльності з тестування у порівнянні з вихідним планом та для інформування керівництва про ризики і альтернативи, що потребують прийняття рішення [1].
- **Звіт про підсумки тестування** (test summary report) – документ, що підсумовує результати тестування. Він також містить оцінку відповідних тестових завдань до критеріїв виходу [1].
- Низькорівневі завдання звітності у тестуванні:
  - оцінка обсягу та якості виконаних робіт;
  - порівняння поточного прогресу із тест-планом;
  - опис наявних складнощів та формування рекомендацій щодо їх усунення;
  - надання особам, зацікавленим у проєкті, повної та об'єктивної інформації про поточний стан якості проєкту, вираженої у конкретних фактах та числах.
- Якісний звіт про результати тестування має багато властивостей якісних вимог, а також розширює їх набір наступними пунктами:
  - інформативність;
  - точність та об'єктивність.
- Звіт про результати тестування створюється за заздалегідь обумовленим розкладом за участі більшості представників проєктної команди, задіяних у забезпеченні якості.
- Відповідальним за створення звіту, як правило, є провідний тестувальник (тест-лід).

- Особи, яким потрібний звіт про результати тестування:
  - менеджер проєкту;
  - керівник команди розробників («дев-ліду»);
  - керівник команди тестувальників («тест-лід»);
  - замовник.
- Розділи звіту про результати тестування:
  - короткий опис;
  - команда тестувальників;
  - опис процесу тестування;
  - розклад;
  - статистика щодо нових дефектів;
  - список нових дефектів;
  - статистика з усіх дефектів;
  - рекомендації;
  - програми.
- **Короткий опис** (summary) – це розділ, який відображає у гранично короткій формі основні досягнення, проблеми, висновки та рекомендації.
- **Команда тестувальників** (test team) – це список учасників проєктної команди, задіяних у забезпеченні якості, із зазначенням їх посад та ролей у підзвітний період.
- **Опис процесу тестування** (testing process description) – це послідовний опис того, які роботи було виконано за підзвітний період.
- **Розклад** (timetable) – це детальний розклад роботи команди тестувальників та/або особисті розклади учасників команди.
- **Статистика щодо нових дефектів** (new defects statistics) – це таблиця, в якій представлені дані щодо виявлених за підзвітний період дефектів.
- **Список нових дефектів** (new defects list) – це список виявлених за підзвітний період дефектів з їх короткими описами та важливістю.

- **Статистика з усіх дефектів** (overall defects statistics) – це таблиця, в якій представлені дані щодо виявлених за весь час існування проєкту дефектів.
- **Рекомендації** (recommendations) – це обґрунтовані висновки та рекомендації щодо прийняття тих чи інших управлінських рішень.
- **Програми** (appendixes) – це фактичні дані (як правило, значення метрик та графічне надання їх зміни у часі).

Логіка побудови звіту про результати тестування

- Універсальна логіка звітності:
  - висновки будуються на основі цілей;
  - висновки доповнюються рекомендаціями;
  - як висновки, так і рекомендації суворо обґрунтовуються;
  - обґрунтування спирається на об'єктивні факти.
- Вимоги до висновків:
  - короткі;
  - інформативні;
  - корисні для читача звіту.
- Вимоги до рекомендацій:
  - короткі;
  - реально виконувані;
  - такі, що дають як розуміння того, що треба зробити, так і деякий простір для прийняття власних рішень.
- **Обґрунтування** висновків та рекомендацій – це проміжна ланка між гранично стислими результатами аналізу та величезною кількістю фактичних даних.
- **Фактичний матеріал –** це матеріал, який містить найрізноманітніші дані, отримані в процесі тестування.

Закріплення матеріалу

1. Пояснити логіку виконання тест-плану та звіту про результат.

РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості

Тема 1.5 ТРУДОВИТРАТИ

Лекція 14

ОЦІНКА ТРУДОВИТРАТ

Мета лекції: Ознайомлення з принципами виконання оцінки трудовитрат.

Зміст лекції

1. Вимоги до оцінки трудовитрат.
2. Оцінка з використанням структурної декомпозиції.

Вимоги до оцінки трудовитрат [22-24]

- **Трудовитрати** – кількість робочого часу, необхідного для виконання роботи (виражається в людино-годинах).
- **Людино-година** (man-hours) – одиниця виміру роботи в промисловості, що дорівнює роботі, виконаній однією людиною за одну годину.
- Правила виконання оцінки трудовитрат:
  - будь-яка оцінка краща за її відсутність;
  - недооцінювання складності незнайомих завдань;
  - оцінка має бути аргументована;
  - простий спосіб навчитися оцінювати – оцінювати.
- Алгоритм навчання формувати оцінку:
  - сформувати оцінку;
  - запишіть отриману оцінку;
  - виконати роботу;
  - звірити реальні результати з раніше сформованою оцінкою;
  - зважити на помилки при формуванні нових оцінок;

  - повторювати цей алгоритм якнайчастіше для різних областей життя.
- Корисні ідеї щодо формування оцінки трудовитрат:
  - додати невеликий «буфер» (за часом, бюджетом чи іншими критичними ресурсами) на непередбачені обставини;
  - з'ясувати власний «коефіцієнт спотворення»;
  - зважати на обставини, що не залежать від вас;
  - заздалегідь замислюватись про необхідні ресурси;
  - шукати способи організувати паралельне виконання завдань;
  - періодично звірятись з планом, вносити коригування в оцінку та повідомляти зацікавлених осіб про внесені зміни завчасно;
  - використовувати інструментальні.

Оцінка з використанням структурної декомпозиції

- Структурна декомпозиція (work breakdown structure, WBS) – ієрархічна декомпозиція об'ємних завдань на дедалі більші підзадачі з метою спрощення оцінки, планування та моніторингу виконання роботи [21].
- **WBS** – це орієнтована на результат ієрархічна декомпозиція роботи, яка має бути виконана проєктною командою для досягнення цілей проєкту та створення необхідних результатів. WBS організовує та визначає загальний обсяг проєкту. WBS підрозділяє роботу над проєктом на менші, більш керовані частини роботи, причому кожен спадний рівень WBS представляє все більш детальне визначення роботи над проєктом. Запланована робота, що міститься в компонентах WBS найнижчого рівня, які називаються робочими пакетами, може плануватися, оцінюватися за вартістю, відстежуватися та контролюватися.
- В процесі виконання структурної декомпозиції великі завдання поділяються на все більш дрібні підзадачі, що дозволяє:

  - описати весь обсяг робіт з точністю, достатньою для чіткого розуміння суті завдань, формування досить точної оцінки трудовитрат та вироблення показників досягнення результатів;
  - визначити весь обсяг трудовитрат як суму трудових витрат за окремими завданнями (з урахуванням необхідних поправок);
  - перейти від інтуїтивного уявлення до конкретного переліку окремих дій, що спрощує побудову плану, прийняття рішень про розпаралелювання робіт тощо.
- Кроки при застосуванні структурної декомпозиції у поєднанні із спрощеним поглядом на оцінку трудовитрат на основі вимог та тест-кейсів:
  - декомпозиція вимог до рівня, на якому з'являється можливість створення якісних чек-листів;
  - декомпозиція завдань із тестування кожного пункту чек-листа до рівня «тестування дій» (створення тест-кейсів, виконання тест-кейсів, створення звітів про дефекти тощо);
  - виконання оцінки з урахуванням своєї продуктивності.
- Коефіцієнти, що уточнюють оцінку часу, необхідного на розробку та виконання одного тест-кейсу:
  - професіоналізм та досвід тестувальника;
  - складність та об'ємність тест-кейсів;
  - продуктивність додатка, що тестується, та тестового оточення;
  - вид тестування;
  - наявність та зручність засобів автоматизації;
  - стадія розробки проєкту.

Закріплення матеріалу

1. Що таке трудовитрати?
2. Що таке структурна декомпозиція?

РОЗДІЛ 2
Автоматизація тестування

Тема 2.1 ВИГОДИ ТА РИЗИКИ АВТОМАТИЗАЦІЇ

Лекція 15

ОБЛАСТІ ЗАСТОСУВАННЯ АВТОМАТИЗАЦІЇ

Мета лекції: Ознайомлення з областями застосування автоматизації при оцінюванні якості програмного забезпечення.

Зміст лекції

1. Переваги та недоліки автоматизації.
2. Випадки найбільшої застосованості тестування.

Переваги та недоліки автоматизації

- **Автоматизоване тестування** – це набір технік, підходів та інструментальних засобів, що дозволяє виключити людину з виконання деяких завдань у процесі тестування [15, 16].
- Переваги автоматизації:
  - швидкість виконання тест-кейсів може в рази та на порядки перевершувати можливості людини;
  - відсутній вплив людського фактора в процесі виконання тест-кейсів;
  - засоби автоматизації здатні виконати тест-кейси, непосильні для людини через свою складність, швидкість або інші фактори;
  - засоби автоматизації здатні збирати, зберігати, аналізувати, агрегувати та представляти у зручній для сприйняття людиною формі колосальні обсяги даних;
  - засоби автоматизації здатні виконувати низькорівневі дії з додатком, операційною системою, каналами передачі тощо.

- Використання автоматизації збільшує тестове покриття за рахунок:
  - виконання тест-кейсів, про які раніше не варто було й думати;
  - багаторазового повторення тест-кейсів із різними вхідними даними;
  - вивільнення часу створення нових тест-кейсів.
- Недоліки та ризики автоматизації:
  - необхідність наявності висококваліфікованого персоналу;
  - багато часу на розробку та супровід як самих автоматизованих тест-кейсів, так і всієї необхідної інфраструктури;
  - ретельніше планування та управління ризиками;
  - висока вартість комерційних засобів автоматизації;
  - різноманіття засобів автоматизації.
- Автоматизація тестування вимагає відчутних інвестицій та сильно підвищує проєктні ризики, тому існують спеціальні підходи з оцінки застосування та ефективності автоматизованого тестування.
- Параметри та фактори спеціальних підходів з оцінки застосування та ефективності автоматизованого тестування, що враховуються в першу чергу:
  - витрати часу на ручне виконання тест-кейсів та на виконання цих самих тест-кейсів, але вже автоматизованих;
  - кількість повторень виконання тих самих тест-кейсів;
  - витрати часу на налагодження, оновлення та підтримку автоматизованих тест-кейсів;
  - наявність у команді відповідних фахівців та їх робоче завантаження;
  - автоматизацією займаються найкваліфікованіші співробітники, які в цей час не можуть вирішувати інших задач.

Області застосування автоматизації

- Завдання, які допомагає вирішити автоматизація [15, 16]:

- виконання тест-кейсів, непідсильних людині;
- вирішення рутинних завдань;
- прискорення тестування;
- вивільнення людських ресурсів для інтелектуальної роботи;
- підвищення тестового покриття;
- покращення коду за рахунок збільшення тестового покриття та застосування спеціальних технік автоматизації.

- Випадки найбільшої застосовності автоматизації:
  - регресійне тестування;
  - інсталяційне тестування та налаштування тестового оточення;
  - конфігураційне тестування та тестування сумісності;
  - використання комбінаторних технік тестування (в тому числі доменного тестування);
  - модульне тестування;
  - інтеграційне тестування;
  - тестування безпеки;
  - тестування продуктивності;
  - димовий тест для великих систем;
  - додатки (або їх частини) без графічного інтерфейсу;
  - довгі, рутинні, стомлюючі для людини операції та/або операції, що вимагають підвищеної уваги;
  - перевірка «внутрішньої функціональності» веб-додатків (посилань, доступності сторінок тощо);
  - стандартна, однотипна для багатьох проєктів функціональність;
  - «технічні задачі».
- Випадки найменшої застосовності автоматизації:
  - планування;
  - розробка тест-кейсів;

- написання звітів про дефекти;
- аналіз результатів тестування та звітність;
- функціональність, яку потрібно (достатньо) перевірити лише кілька разів;
- тест-кейси, які потрібно виконати всього кілька разів (якщо людина може їх виконати);
- низький рівень абстракції у наявних інструментах автоматизації;
- слабкі можливості засобу автоматизації з протоколювання процесу тестування та збору технічних даних про додаток та оточення;
- низька стабільність вимог;
- складні комбінації великої кількості технологій;
- проблеми з плануванням та ручним тестуванням;
- нестача часу та загроза зриву термінів;
- області тестування, що вимагають оцінки ситуації людиною (тестування зручності використання, тестування доступності тощо).

- Ефект від автоматизації настає не відразу і не завжди. Автоматизація при правильному застосуванні може дати відчутну вигоду, але при неправильному принесе лише відчутні витрати.

Закріплення матеріалу

1. Що таке автоматизоване тестування?
2. Назвати переваги автоматизованого тестування.
3. Які фактори збільшують тестове покриття при застосуванні автоматизованого тестування?
4. Назвати недоліки та ризики автоматизованого тестування.
5. Назвати параметри та фактори спеціальних підходів з оцінки застосування та ефективності автоматизованого тестування.
6. Які завдання допомагає вирішити автоматизація?
7. Назвати випадки найбільшої застосовності автоматизації.
8. Назвати випадки найменшої застосовності автоматизації.

РОЗДІЛ 2
Автоматизація тестування

Тема 2.1 ВИГОДИ ТА РИЗИКИ АВТОМАТИЗАЦІЇ

Лекція 16

ОСОБЛИВОСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

Мета лекції: Ознайомлення з особливостями автоматизації тестування та автоматизацією поза межами прямих задач тестування.

Зміст лекції

1. Необхідні знання та навички.
2. Особливості тест-кейсів в автоматизації.
3. Автоматизація поза межами прямих задач тестування.

Необходимые знання і навики

- **Автоматизація тестування** представляє собою поєднання програмування і тестування в різних масштабах (в залежності від проєкту і конкретних задач).
- Навички та вміння спеціаліста з автоматизації тестування:
- вміння і програмувати, і тестувати;
- вміння адмініструвати операційні системи, мережі, різні сервери;
- вміння працювати з базами даних; розуміння мобільних платформ;
- вкрай висока здатність навчання;
- здатність в гранично стислий термін самостійно знайти, вивчити, зрозуміти та почати застосовувати на практиці абсолютно нову інформацію з, можливо, раніше абсолютно незнайомої області.

Особливості тест-кейсів в автоматизації

- Часто автоматизації піддаються тест-кейси, спочатку написані простою людською мовою.

- Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації [15, 16]:
  - очікуваний результат в автоматизованих тест-кейсах має бути описаний максимально чітко із зазначенням конкретних ознак його коректності;
  - опис тест-кейсу без специфічних для того чи іншого інструментального засобу рішень;
  - не варто прописувати в тест-кейс щось, характерне лише для однієї платформи;
  - синхронізація засобу автоматизації та додатку, що тестується, за часом;
  - опис словами значення та/або сенсу якоїсь змінної;
  - використовувати найбільш універсальні способи взаємодії з додатком, що тестується;
  - автоматизовані тест-кейси мають бути незалежними;
  - автоматизований тест-кейс – це програма і варто враховувати хороші практики програмування хоча б лише на рівні відсутності так званих «магічних значень», «хардкодінгу» тощо;
  - уважно вивчати документацію по використовуваному засобу автоматизації, щоб уникнути ситуації, коли через невірно обрану команду тест-кейс стає хибно позитивним;
  - уважно слідкувати, щоб не відбулася заміна перевірки дією та навпаки.
- **Хибно позитивний тест-кейс** – це тест-кейс, який успішно проходить в ситуації, коли програма працює неправильно.
- Тест-кейс, призначений для автоматизації, буде набагато схожим на мініатюрне технічне завдання з розробки невеликої програми, ніж опис коректної поведінки тестованого додатку, що тестується, зрозумілий людині.

- Джерела даних та способи їх генерації в автоматизованих тест-кейсів:
  - випадкові величини;
  - генерація (випадкових) даних щодо алгоритму;
  - отримання даних із зовнішніх джерел;
  - зібрані робочі дані, тобто дані, створені реальними користувачами в процесі їх реальної роботи;
  - ручна генерація.

Автоматизація поза межами прямих задач тестування

- Ми з вами вже розглянули, як автоматизація може допомогти у створенні та виконанні тест-кейсів. Але ті самі принципи можна перенести і на решту роботи тестувальника, в якій також бувають тривалі та стомлюючі завдання, рутинні задачі або завдання, що потребують пильної уваги, але не пов'язані з інтелектуальною роботою. Все перераховане також можна автоматизувати.
- Типові рішень автоматизації поза межами прямих задач тестування:
  - використання командних файлів для виконання послідовностей операцій;
  - генерація та оформлення даних з використанням офісних можливостей додатків, баз даних, невеликих програм на високорівневих мовах програмування;
  - підготовка та оформлення технічних розділів для звітів;
  - керування своїм робочим місцем;
  - сортування та обробка пошти;
  - віртуалізація як спосіб позбавлення необхідності щоразу встановлювати та налаштовувати необхідний набір програм.

Закріплення матеріалу

1. Які навички та вміння необхідні спеціалісту з автоматизації тестування?
2. Назвати рекомендації щодо автоматизації та підготовки до неї.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. International Software Testing Qualifications Board Glossary. URL : https://www.istqb.org/#glossary

2. Automated and Manual Software Testing Tools and Servises. URL : https://www.aptest.com

3. The History of Software Testing. URL : http://www.testingreferences.com/testinghistory.php

4. Gelperin David, Hetzel Bill The Growth of Software Testing // *Communications of the ACM*. Vol. 31, Issue 6. – pp 687–695. DOI : https://doi.org/10.1145/62959.62965

5. Whittaker James The 7 Plagues of Software Testing. URL : http://googletesting.blogspot.com/2009/06/by-james.html, http://googletesting.blogspot.com/2009/07/plague-of-amnesia.html, http://googletesting.blogspot.com/2009/07/plague-of-boredom.html, http://googletesting.blogspot.com/2009/07/plague-of-homelessness.html, http://googletesting.blogspot.com/2009/07/plague-of-blindness.html, http://googletesting.blogspot.com/2009/09/7th-plague-and-beyond.html, http://googletesting.blogspot.com/2009/09/plague-of-entropy.html

6. What are the Software Development Models? URL : http://istqbexamcertification.com/what-are-the-software-development-models/

7. Manifesto for Agile Software Development. URL : http://agilemanifesto.org/iso/en/manifesto.html

8. Software Testing Life Cycle. URL : http://softwaretestingfundamentals.com/software-testing-life-cycle/

9. Barker Thomas T. Documentation for Software and IS Development // *Encyclopedia of Information Systems*. Elsevier Press, 2002. pp. 684-694.

10. Wiegers Karl, Beatty Joy Software Requirements. Kyiv : Print2print, 2013. 672 с.

11. Documenting Software Architectures / Paul Clements and other. Pearson Education, 2010. 608 p.

12. Requirements Gathering vs. Elicitation / URL : http://www.bridging-the-gap.com/requirements-gathering-vs-elicitation/

13. Cadle James, Paul Debra, Turner Paul Business Analysis Techniques. 72 Essential Tools for Success. BCS, The Chartered Institute, 2010. 260 p.

14. Types of Software Testing: List of 100 Different Testing Types / URL : http://www.guru99.com/types-of-software-testing.html

15 Якість програмного забезпечення та тестування: базовий курс : навч. посіб. / за ред. С.Я. Крепич, І.Я. Співак. Тернопіль : ФОП Паляниця В.А., 2020. 478 с.

16. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення : навч. посіб. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с.

17. Role-based access control / URL : https://en.wikipedia.org/wiki/Role-based_access_control

18. Добролюбова М.В. Програмування баз даних: конспект лекцій. Київ : КПІ ім. Ігоря Сікорського, 2021. 275 с.

19. Якість та тестування інформаційних систем / Золотухіна О.А., Негоденко О.В., Резник С.Ю., Разіна С.Я. Київ- : ННІІТ ДУТ, 2020. 128 с.

20. Kerzner Harold Project Management: A Systems Approach to Planning, Scheduling, and Controlling. Wiley, 2013. 1296 p.

21. A Guide to the Project Management Body of Knowledge / Project Management Inst., 2004. 390 p.

22. Conte Samuel D. Software engineering metrics and models. Benjamin-Cummings Pub Co, 1986. 403 p.

23. Frederick Brooks Jr. The Mythical Man Month, The: Essays on Software Engineering. Addison-Wesley Professional, 1995. 336 p.

24. De Marco Tom Controlling Software Projects. Pearson College Div, 1986. 304 p.

ДОДАТОК А
ПРЕЗЕНТАЦІЇ ДО ЛЕКЦІЙ


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 1. МЕТА, ПОНЯТТЯ ТА ТЕРМІНИ
Лекція 1
«Основні поняття та визначення в області тестування програмного забезпечення»
@ М.В.Добролюбова

Визначення тестування

Тестування програмного забезпечення – процес аналізу програмного засобу та супутньої документації з метою виявлення дефектів та підвищення якості продукту.

Testing – процес, що складається з усіх дій життєвого циклу, як статичних, так і динамічних, пов'язаних з плануванням, підготовкою та оцінкою програмних продуктів і пов'язаних з ними робочих продуктів, щоб визначити, що вони задовольняють визначеним вимогам, з метою демонстрації їх відповідності призначенню та виявлення дефектів.

Тестування – процес, що підтверджує правильність програми та демонструє, що помилок у програмі немає.

Тестування – процес виконання програми (або частини програми) з наміром (або метою) знайти помилки.

Мета тестування – знайти помилки у програмі і, тим самим, підвищити її надійність та цінність.

«Не намагайтеся знайти якнайбільше помилок, намагайтеся пропустити якомога менше!»

@ М.В.Добролюбова

2

Епохи тестування

1950-60-і роки Процес тестування формалізований, багато уваги приділялося exhaustive «вичерпному» тестуванню.

1970-і роки Тестування програмного забезпечення позначалося як «процес, спрямований на демонстрацію коректності товару».

1980-і роки Тестування розширилося таким поняттям, як попередження дефектів. Проектування тестів – найбільш ефективний із відомих методів попередження помилок.

1990-і роки До поняття «тестування» стали включати планування, проектування, створення, підтримку та виконання тестів та тестових оточень.

2000-і роки З розвитком Інтернету та розробкою великої кількості веб-додатків особливу популярність стало набувати «гнучке тестування» та такі підходи, як «розробка під управлінням тестуванням (test-driven development, TDD)». З'явилося ще ширше визначення тестування, коли до нього було додано поняття «оптимізація бізнес-технологій» (BTO).

«Мистецтво тестування програм», Гленфорд Майерс (видання 1979, 2004, 2011)

«Ключові процеси тестування», Рекс Блек («Critical Testing Processes», Rex Black)

The History of Software Testing (на ресурсі Testing References)

The Growth of Software Testing (David Gelperin, Bill Hetzel)

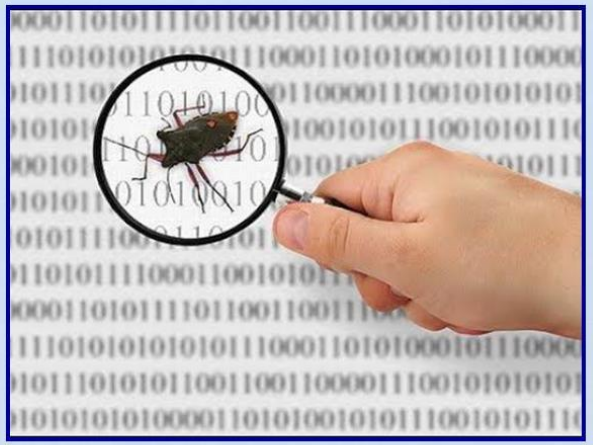
@ М.В.Добролюбова 3

Поняття, завдання, концепції

Філософія тестування

Види діяльності, що охоплює тестування програмного забезпечення:

- постановка задачі для тестування
- проектування, написання тестів
- тестування тестів
- виконання тестів
- вивчення результатів тестування



Щоб бути професіоналом, потрібно постійно

- перебувати в русі
- безперервно розвиватися
- завжди вміти ставити незручні питання

Завдання тестування

Завдання – відтворити якомога більше багів. Відтворювати баги тестувальнику необхідно, це видимий вимірюваний РЕЗУЛЬТАТ роботи, І чим їх більше, тим більше вас цінують як тестувальника.

Завдання – пропустити якнайменше пріоритетних для користувача багів. Чим менше багів пропущено, чим менше невдоволення клієнтом виражено – тим ви вище оцінюєте ефективність своєї роботи.

@ М.В.Добролюбова 4

Поняття, завдання, концепції

Концепції тестування

Концепція 1. Мета тестування – це 100% перевірка ПЗ.
Концепція 2. Критерій ефективності тестування – це кількість багів, знайдених до релізу.

Терміни

У програмуванні **баг** – жаргонне слово, що зазвичай означає помилку в програмі або системі, через яку програма видає несподівану поведінку і, як наслідок, результат.

Реліз – передача Програмного забезпечення користувачеві.

Тестування – це турбота про якість, у вигляді виявлення багів до того, як їх знайдуть юзери.

QA – це процес або результат формування необхідних властивостей та характеристик продукції в міру її створення, а також підтримка цих характеристик при зберіганні, транспортування та експлуатації продукції.

Загальне в тестуванні та QA Обидва покликані покращувати ПЗ.

Відмінності Тестування покликане покращити ПЗ через виявлення багів. QA покликане покращити ПЗ через покращення процесу розробки.

QC – це процес знаходження помилок у продукті, з метою їхнього подальшого виправлення.

Waterfall – модель процесу розробки програмного забезпечення, в якій процес розробки виглядає як потік, що послідовно проходить фази аналізу вимог, проектування, реалізації, тестування, інтеграції та підтримки.

Функціональне тестування – це тестування ПЗ з метою перевірки реалізованості функціональних вимог, тобто можливості ПЗ у певних умовах вирішувати завдання, потрібні користувачам.

Нефункціональне тестування – тестування властивостей, які не належать до функціональності системи.

@ М.В.Добролюбова

5

Тестувальник та його обов'язки

Типові види діяльності тестувальника

	Невеликі фірми	Великі фірми
Низька кваліфікація	Підмайстер, часто наданий сам собі у вирішенні завдань.	Пересічний учасник проектів, який одночасно проходить інтенсивне підвищення кваліфікації.
Висока кваліфікація	Майстер на всі руки з багатим, але не завжди структурованим досвідом.	Експерт в одній або кількох областях, консультант, керівник напряму.

@ М.В.Добролюбова

6

Тестувальник та його обов'язки

Стартові технічні навички

0) Знання іноземних мов
1) Впевнене володіння комп'ютером на рівні по-справжнього просунутого користувача та бажання постійно розвиватися у цій галузі
2) Програмування
3) Бази даних та мова SQL
4) Розуміння принципів роботи мереж та операційних систем
5) Розуміння принципів роботи веб-додатків та мобільних додатків
6) Розуміння принципів самого тестування

Особисті якості

1) підвищена відповідальність та старанність
2) добрі комунікативні навички, здатність ясно, швидко, чітко висловлювати свої думки
3) терпіння, усидливість, уважність до деталей, спостережливість
4) хороше абстрактне та аналітичне мислення
5) здатність ставити нестандартні експерименти, схильність до дослідницької діяльності

@ М.В.Добролюбова 7

Міфи про тестування

1. Не треба розумітися на комп'ютерах
2. Обов'язково треба добре знати програмування
3. В тестуванні все просто
4. У тестуванні купа рутини та нудьги
5. Тестувальника повинні всьому навчити

6. У тестувальники йдуть ті, хто не зміг стати програмістом
7. У тестуванні складно збудувати кар'єру
8. Тестувальник «винний у всьому»
9. Тестувальники скоро будуть не потрібні, оскільки все буде автоматизовано

«Сім бід тестування», Джеймс Віттакер

The Plagues of Software Testing (James Whittaker)

@ М.В.Добролюбова

8

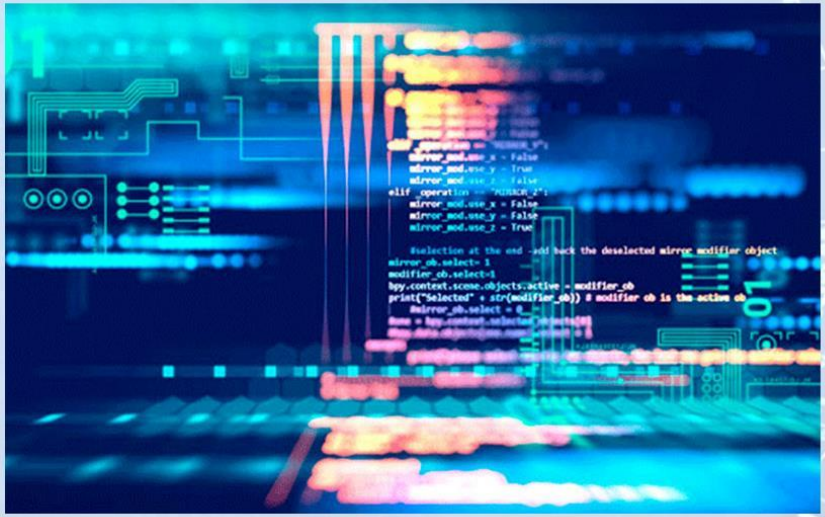

Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 2. КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ
Лекція 2
«Процеси тестування і розробки програмного забезпечення»
@ М.В.Добролюбова

Моделі розробки програмного забезпечення

Модель розробки програмного забезпечення (Software Development Model, SDM) – це структура, яка систематизує різні види проєктної діяльності, їх взаємодію та послідовність у процесі розробки програмного забезпечення (ПЗ).

Класичні моделі розробки:

- водоспадна
- v-подібна
- ітераційна інкрементальна
- спіральна
- гнучка



«What are the Software Development Models?»

«Scrum Tailoring», Максим Дорофеев

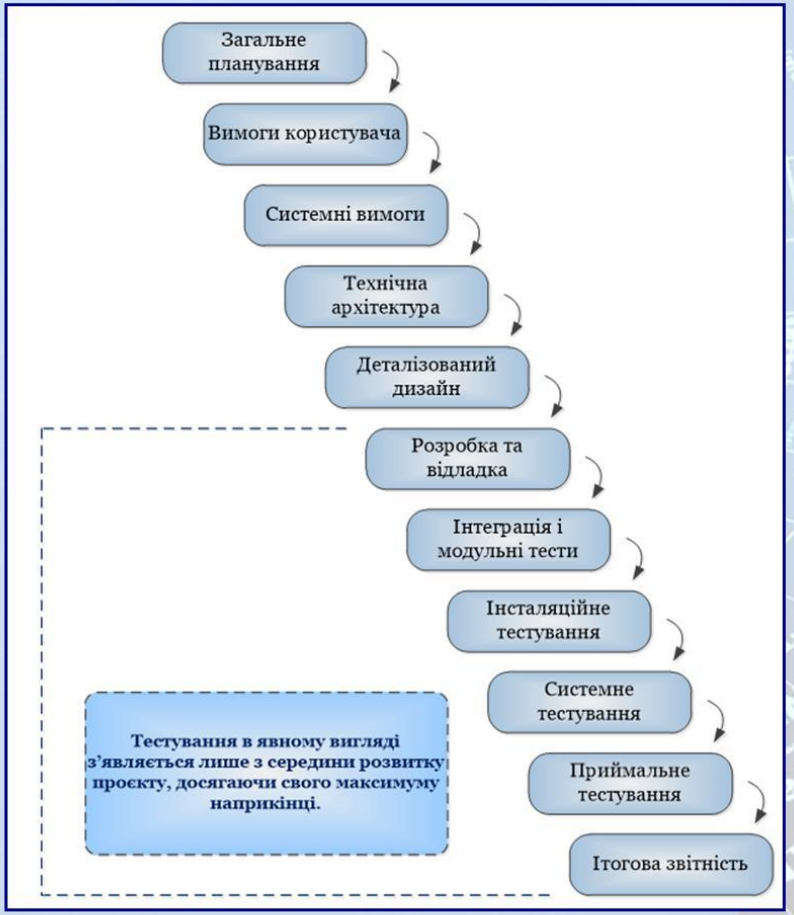
@ М.В.Добролюбова

2

Моделі розробки програмного забезпечення

Водоспадна модель розробки програмного забезпечення

Водоспадна модель (waterfall model) передбачає одноразове виконання кожної з фаз проєкту, які, в свою чергу, суворо слідують одна за одною.



«What is Waterfall model advantages, disadvantages and when to use it?»

«The Rise And Fall Of Waterfall», Максим Дорофєєв

@ М.В.Добролюбова

3

Моделі розробки програмного забезпечення

V-подібна модель розробки програмного забезпечення

V-подібна модель (V-model) – це різновид водоспадної моделі, який передбачає планування робіт з тестування на ранніх стадіях життєвого циклу.

«What is V-model advantages, disadvantages and when to use it?»

«Using V Models for Testing»

@ М.В.Добролюбова

4

Моделі розробки програмного забезпечення

Ітераційна інкрементальна модель розробки програмного забезпечення

Ітераційна інкрементальна модель (iterative model, incremental model) – модель, що передбачає розбиття проєкту на відносно невеликі проміжки (ітерації), кожен із яких у загальному випадку може включати всі класичні стадії, властиві водоспадній і v-подібній моделям.

Ітераційна модель розробки – життєвий цикл розробки, коли проєкт розбивається на зазвичай велику кількість ітерацій.

Ітерація – це повний цикл розробки, що призводить до випуску виконуваного продукту, підмножини кінцевого продукту у розробці, яка зростає від ітерації до ітерації, щоб стати кінцевим продуктом.

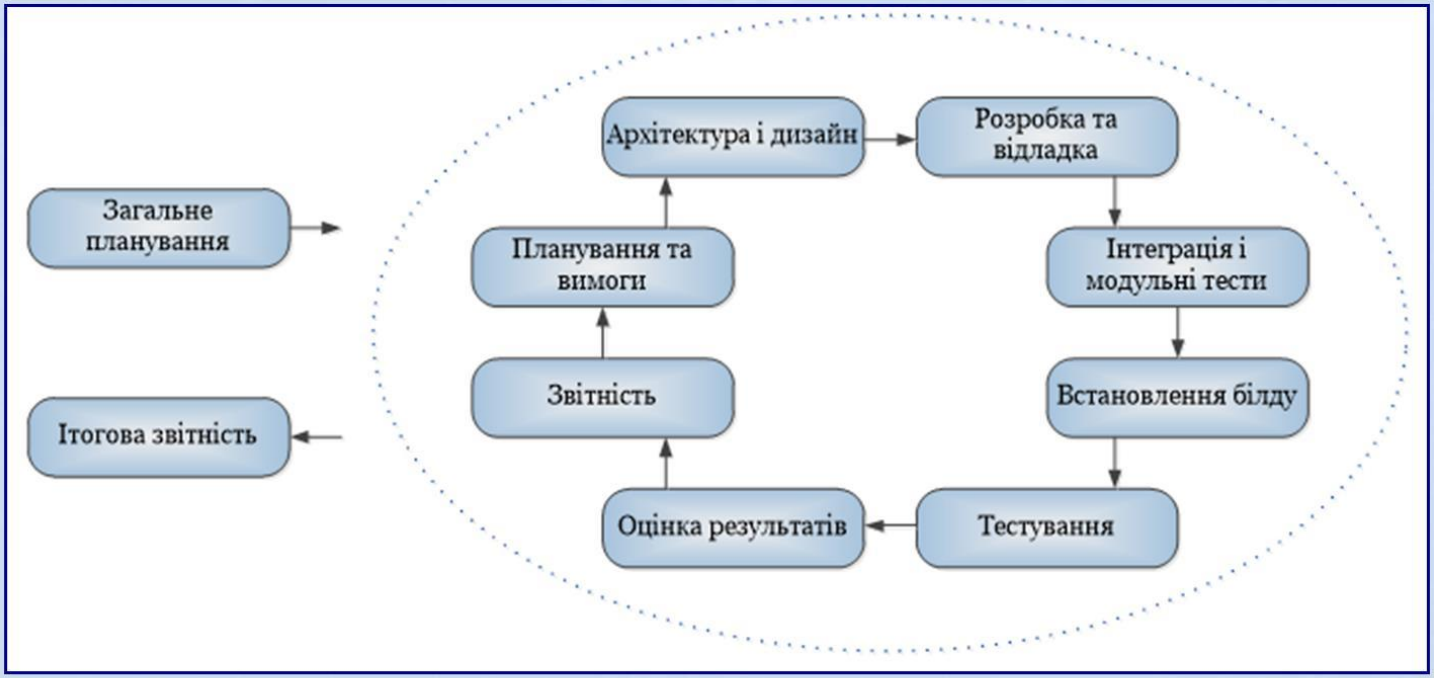
Інкрементальна модель (модель поступового розвитку) – це життєвий цикл розробки, коли проєкт розбивається на серію кроків, кожен з яких забезпечує частину функціональності в загальних вимогах проєкту.

Білд (build)– діяльність з розробки, за допомогою якої повна система компілюється та зв'язується, з метою отримання доступної узгодженої системи, включаючи всі останні зміни).

@ М.В.Добролюбова 5

Моделі розробки програмного забезпечення

Ітераційна інкрементальна модель розробки програмного забезпечення



«What is Iterative model advantages, disadvantages and when to use it?»

«What is Incremental model advantages, disadvantages and when to use it?»

@ М.В.Добролюбова

6

Моделі розробки програмного забезпечення

Спіральна модель розробки програмного забезпечення

Спіральна модель (spiral model) – це модель, яка відображає базову концепцію розробки, згідно з якою набору операцій у кожному циклі відповідає така ж кількість фаз, як і моделі водоспадного процесу, починаючи з формулювання вимог і закінчуючи кодуванням кожної окремої програми.

Прототип – це діючий програмний компонент, за допомогою якого реалізуються окремі функції та зовнішні інтерфейси програмного продукту, що розробляється.

Основні принципи спіральної моделі:

- розробка варіантів продукту з опрацюванням цілей, альтернатив та обмежень
- аналіз ризиків та створення прототипів програмного продукту
- планування наступних варіантів з оцінкою альтернатив та аналізом ризиків, пов'язаних із переходом до наступного варіанту
- перехід до розробки наступного варіанту до завершення попереднього у разі, коли ризик завершення чергового варіанта (прототипу) стає невиправдано високий
- активне залучення замовника до роботи над проєктом

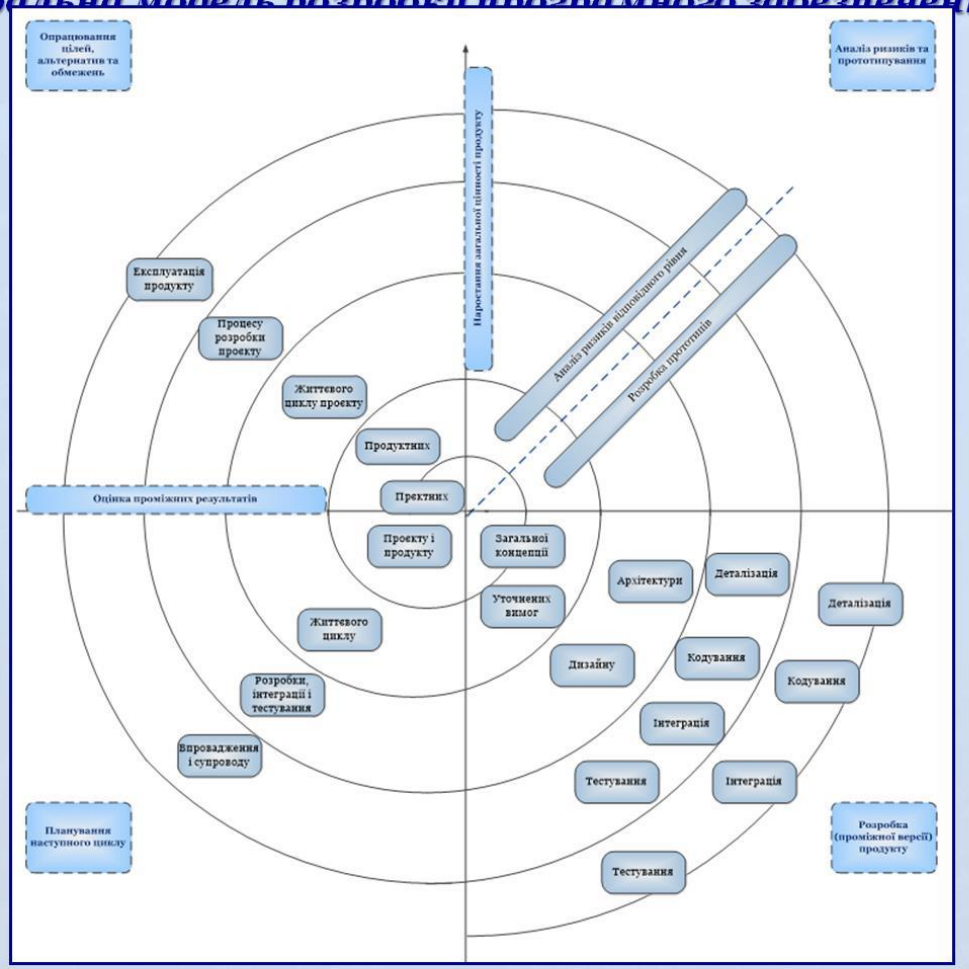
«A Spiral Model of Software Development and Enhancement», «Spiral Development: Experience, Principles, and Refinements», Barry Boehm

«What is Spiral model - advantages, disadvantages and when to use it?», «Spiral Model»

@ М.В.Добролюбова 7

Моделі розробки програмного забезпечення

Спіральна модель розробки програмного забезпечення



@ М.В.Добролюбова

8

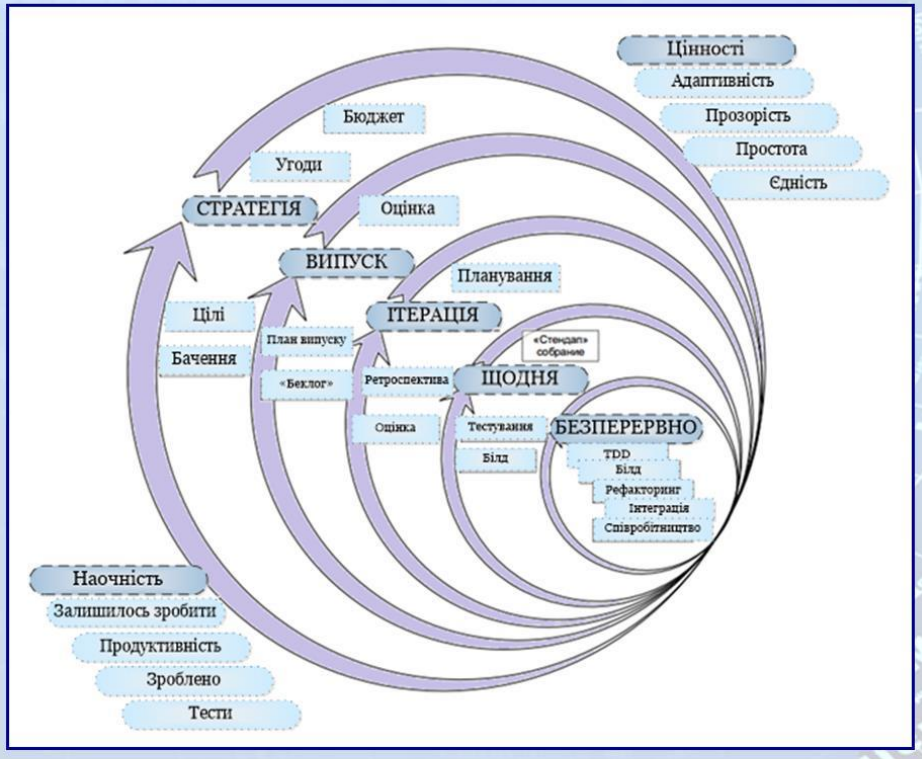
Моделі розробки програмного забезпечення

Гнучка модель розробки програмного забезпечення

Гнучка модель (agile model) – це модель, яка є сукупністю різних підходів до розробки програмного забезпечення та базується на «agile-маніфесті».

Постулати «agile-маніфесту»:

- люди та взаємодія важливіші за процеси та інструменти;
- працюючий продукт важливіший за вичерпну документацію;
- співпраця із замовником важливіша за погодження умов контракту;
- готовність до змін важливіша за дотримання початкового плану.



«Agile Testing», Lisa Crispin & Janet Gregory

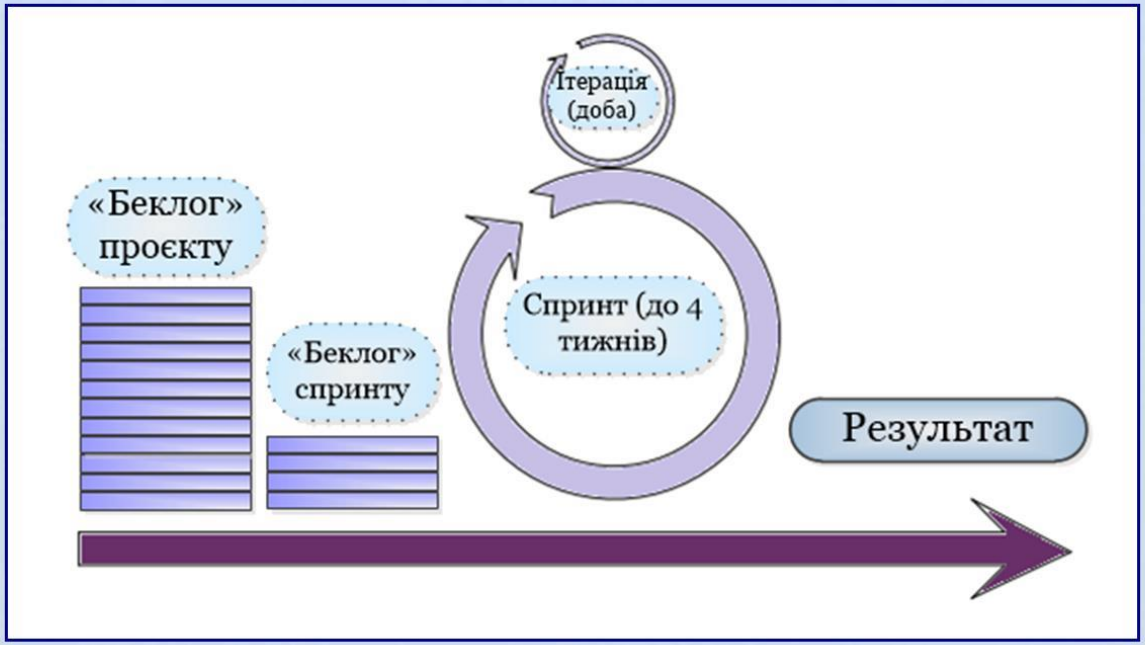
«Essential Scrum», Kenneth S. Rubin

@ М.В.Добролюбова

9

Моделі розробки програмного забезпечення

Ітераційний підхід у рамках гнучкої моделі та scrum



«The Agile System Development Life Cycle»

@ М.В.Добролюбова

10

Моделі розробки програмного забезпечення

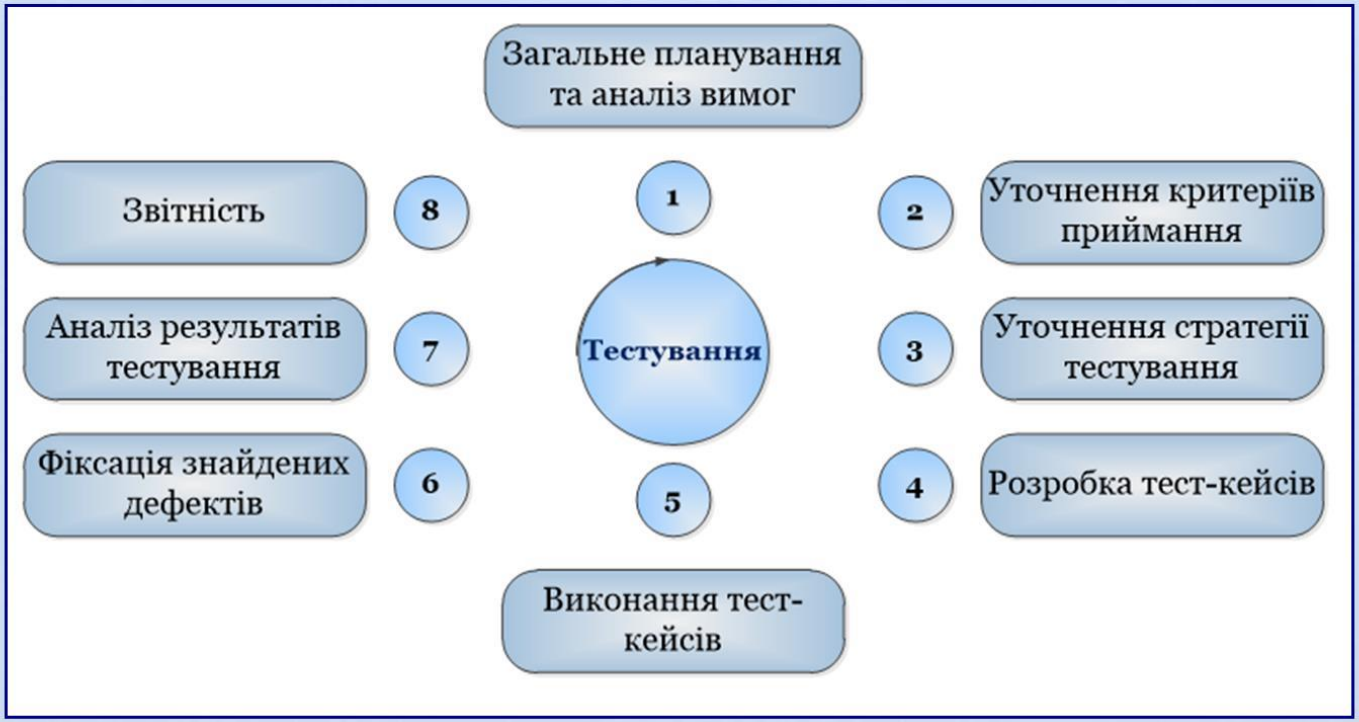
Порівняння моделей розробки програмного забезпечення

Модель	Переваги	Недоліки	Тестування
Водоспадна	• У кожної стадії є чіткий результат, що перевіряється. • У кожний момент часу команда виконує один вид роботи. • Добре працює для невеликих завдань.	• Повна нездатність адаптувати проект до змін у вимогах. • Вкрай пізнє створення працюючого продукту.	• Із середини проєкту.
V-подібна	• У кожної стадії є чіткий результат, що перевіряється. • Увага тестуванню приділяється з першої ж стадії. • Добре працює для проектів зі стабільними вимогами	• Недостатня гнучкість та адаптованість. • Відсутнє раннє прототипування. • Складність усунення проблем, пропущених на ранніх стадіях розвитку проекту.	• На переходах між стадіями.
Ітераційна інкрементальна	• Досить раннє прототипування. • Простота керування ітераціями. • Декомпозиція проекту на керовані ітерації.	• Недостатня гнучкість усередині ітерацій. • Складність усунення проблем, пропущених на ранніх стадіях розвитку проекту.	• У певні моменти ітерацій. • Повторне тестування (після доопрацювання) вже перевіреного раніше.
Спіральна	• Глибокий аналіз ризиків. • Підходить для великих проектів. • Досить раннє прототипування.	• Високі накладні витрати. • Складність застосування для невеликих проектів. • Висока залежність успіху від якості аналізу ризиків.	
Гнучка	• Максимальне залучення замовника. • Багато роботи з вимогами. • Тісна інтеграція тестування та розробки. • Мінімізація документації.	• Складність реалізації для великих проектів. • Складність побудови стабільних процесів.	• У певні моменти ітерацій та у **будь-який необхідний момент**.

«Project Lifecycle Models: How They Differ and When to Use Them», «Блок-схема вибору оптимальної методології розробки ПЗ» «What are the Software Development Models?»

@ М.В.Добролюбова **11**

Життєвий цикл тестування



@ М.В.Добролюбова

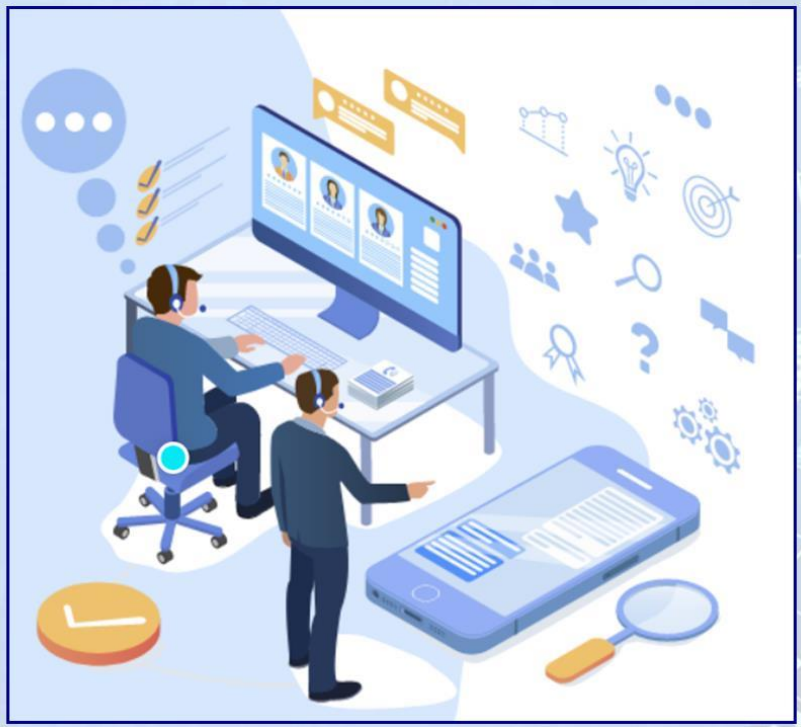
12


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 2. КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ
Лекція 3
«Тестування документації і вимог»
@ М.В.Добролюбова

Поняття «вимоги»

Вимога (requirement) – опис того, які функції та з дотриманням яких умов має виконувати програма в процесі розв'язання необхідної для користувача задачі.

Вимога (requirement) – умова або здатність, необхідна користувачеві для вирішення проблеми чи досягнення мети, якою має володіти система або компонент системи, щоб задовольнити контракт, стандарт, специфікацію або інший офіційно встановлений документ.



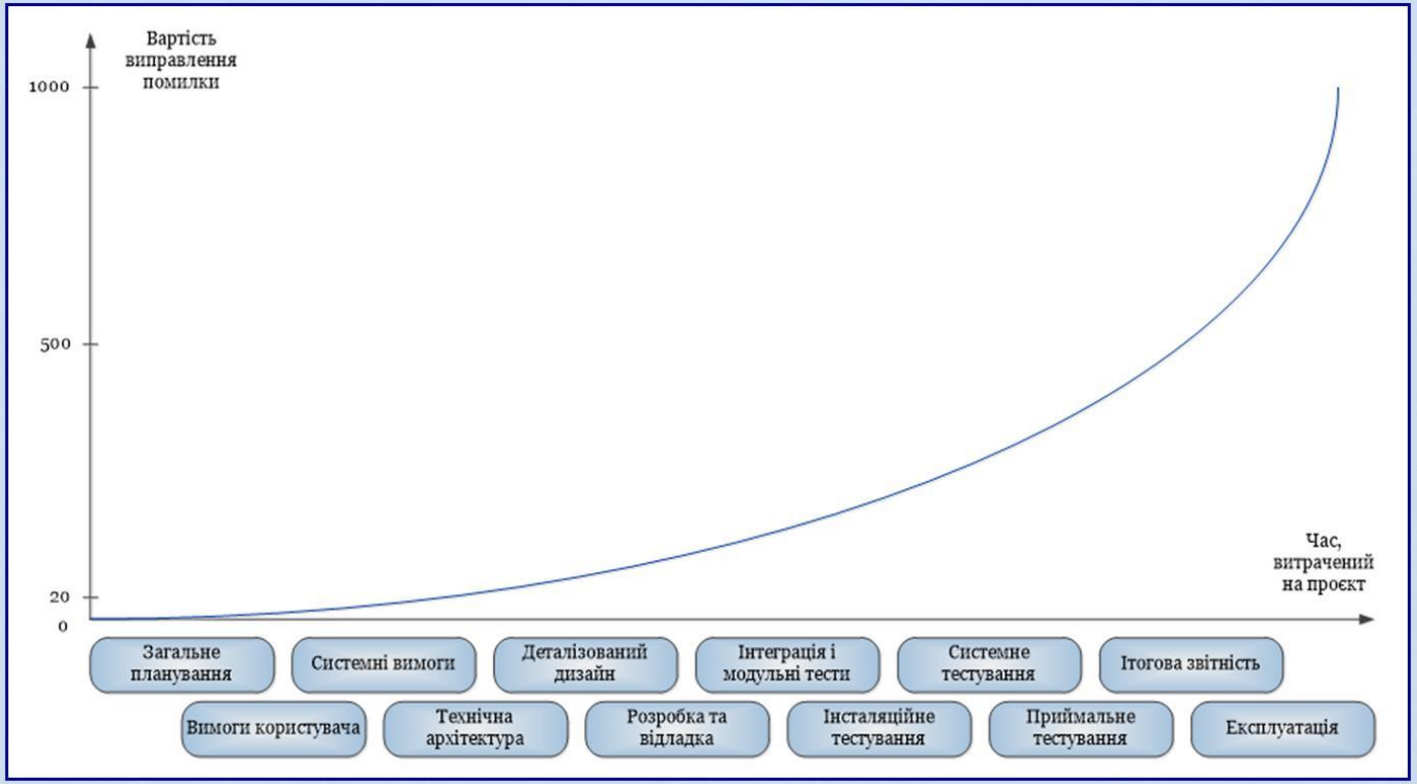
«What is documentation testing in software testing»

@ М.В.Добролюбова

2

Важливість вимог

Вартість виправлення помилки залежно від моменту її виявлення



Вимоги дозволяють зрозуміти, що та з дотриманням яких умов система повинна робити

Вимоги надають можливість оцінити масштаб змін та керувати змінами

Вимоги є основою формування плану проєкту (зокрема плану тестування)

Вимоги допомагають запобігати або вирішувати конфліктні ситуації

Вимоги спрощують розміщення пріоритетів у наборі завдань

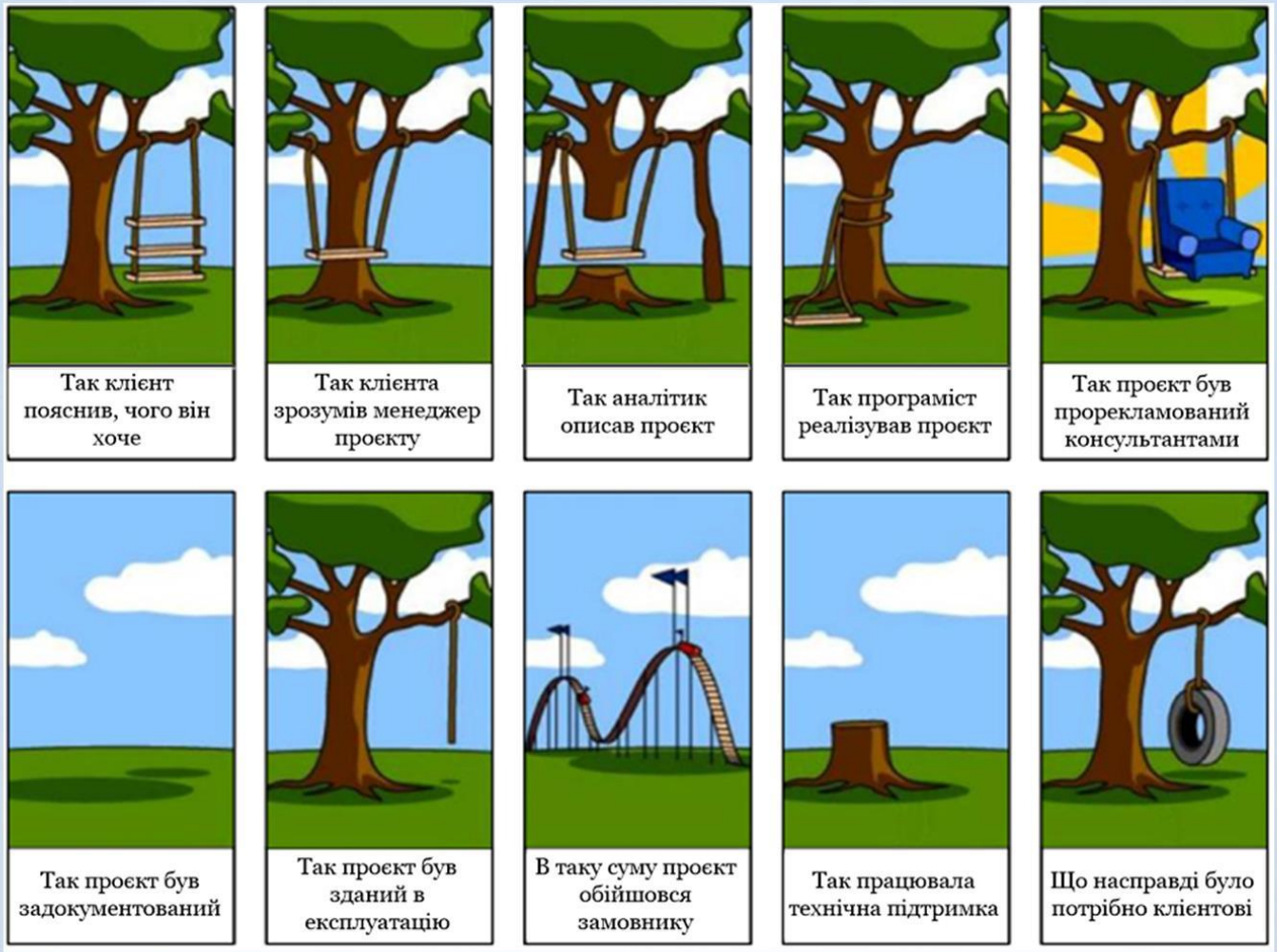
Вимоги дозволяють об'єктивно оцінити рівень прогресу у розробці проєкту

@ М.В.Добролюбова

3

Важливість вимог

Типовий проект із поганими вимогами



@ М.В.Добролюбова 4

Важливість вимог

Види документації

Продуктна документація

- план проєкту (project management plan) та тестовий план (test plan)
- вимоги до програмного продукту (product requirements document, PRD) та функціональні специфікації (FSD, SRS)
- архітектура та дизайн (architecture and design)
- тест-кейси та набори тест-кейсів (test cases, test suites)
- технічні специфікації (technical specifications)

Проєктна документація

- супровідна документація та документація користувача (user and accompanying documentation)
- маркетингова документація (market requirements document, MRD)

Співвідношення понять «продуктна документація» та «проєктна документація»

@ М.В.Добролюбова

5

Важливість вимог

Документація по розробці включає ті документи, які пропонують, уточнюють, планують, переглядають, тестують і впроваджують продукти команд розробників в індустрії програмного забезпечення.

План управління проєктом – це офіційний затверджений документ, який визначає, як проєкт виконується, контролюється та управляється.

План тестування – це документ, що описує обсяг, підхід, ресурси та графік запланованих тестових заходів.

Документ з вимогами до продукту, PRD – це документ, що описує продукт, який буде створювати компанія.

Специфікація (технічні характеристики) – це документ, який визначає вимоги, дизайн, поведінку, або інші характеристики компонента або системи та процедури для визначення того, чи були виконані ці положення.

Документ з функціональними специфікаціями, FSD – це документ, який використовується для детального опису передбачуваних можливостей продукту, зовнішнього вигляду та взаємодії з користувачами і розробникам програмного забезпечення.

Специфікація вимог до програмного забезпечення, SRS – це документ, який максимально повно описує очікувану поведінку програмної системи.

Архітектура та Дизайн. Архітектура програмного забезпечення для системи – це структура або структури системи, які містять елементи, їх зовні видиму поведінку та відношення між ними.

Тестовий приклад – це набір вхідних значень, передумов виконання, очікуваних результатів і пост-умов виконання, розроблених для конкретної мети або умови тестування.

Набір тестів – це набір із кількох тестових випадків для компонента або системи, що тестується, де пост-умова одного тесту часто використовується як умова для наступного.

Технічні характеристики – це скрипти, вихідний код, мова визначення даних тощо.

Проєктна документація – це інші очікування та результати, які не є частиною програмного забезпечення, що реалізує команда, але є необхідними для успішного завершення проєкту в цілому.

Документація користувача – це документація, яка належить до документації по продукту або послузі, що надаються кінцевим користувачам.

Документ з ринкових вимог, MRD. MRD детально описує цільові сегменти ринку та питання, пов'язані з комерційним успіхом.

@ М.В.Добролюбова 6

Джерела та шляхи виявлення вимог

Інтерв'ю	Робота з фокусними групами	Анкетування
Семінари та мозковий штурм	Спостереження	Прототипування
Аналіз документів	Моделювання процесів та взаємодій	Самостійний опис

«Requirements Gathering vs. Elicitation» (Laura Brandenburg)

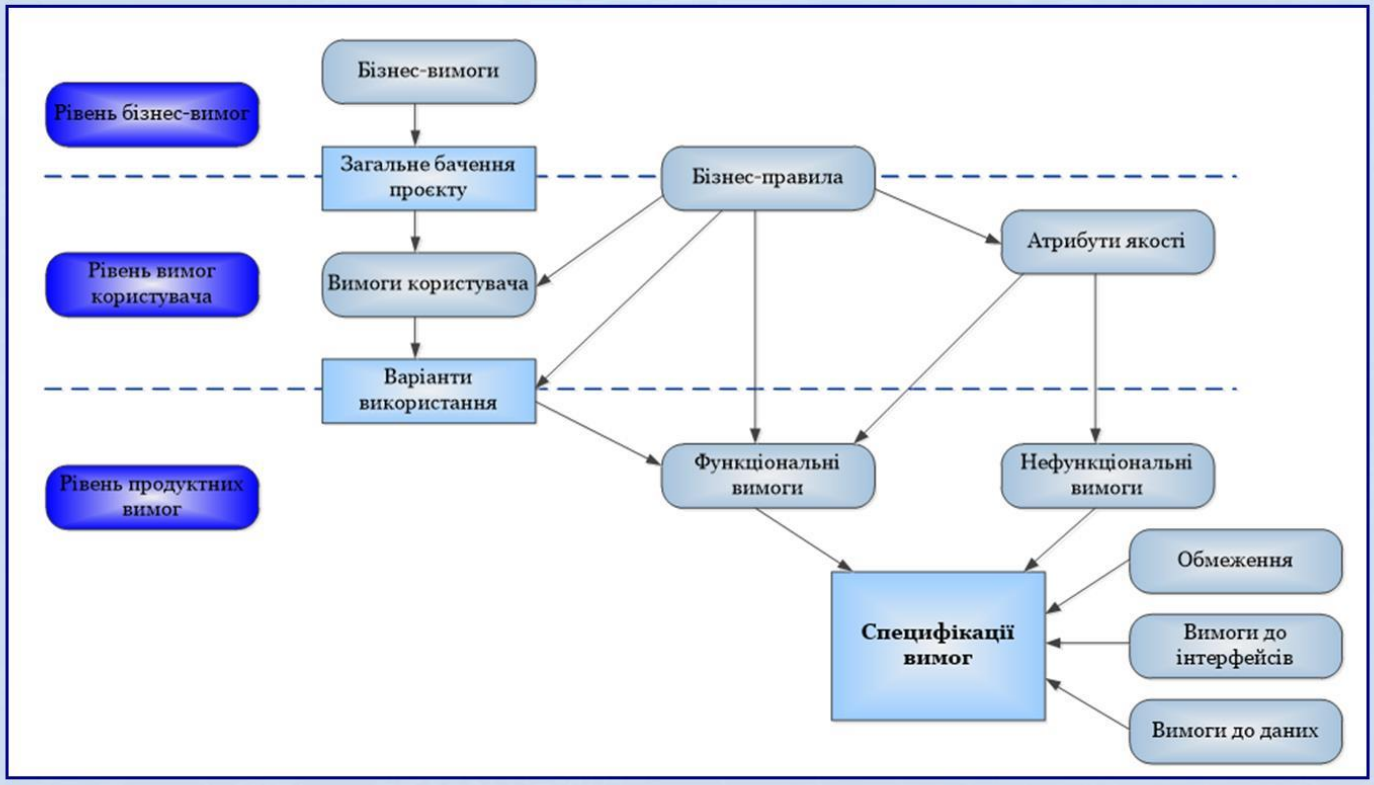
«Вимоги для програмного забезпечення: рекомендації щодо збору та документування» (Ілля Корніпаєв)

«Business Analysis Techniques. Essential Tools for Success» (James Cadle, Debra Paul, Paul Turner)

Business Analysis (Second Edition) (Debra Paul, Donald Yeates, James Cadle)

@ М.В.Добролюбова 7

Рівні та типи вимог



«Software Requirements Engineering: What, Why, Who, When, and How», Linda Westfall

«Software Requirements (3rd edition)» (Karl Wiegers and Joy Beatty)

«Developer Best Practices», Karl Wiegers, Joy Beatty

@ М.В.Добролюбова

8

Рівні та типи вимог

Бізнес-вимога – це все, що описує фінансову, ринкову чи іншу бізнес-вигоду, яку клієнти чи організація, що розвивається, бажають отримати від продукту.

Бачення та масштаби. У документі про бачення та обсяги бізнес-вимоги об'єднані в єдиний продукт, який створює основу для подальших робіт з розробки.

Вимоги користувачів – це загальні формулювання цілей користувачів або бізнес-завдань, які користувачі повинні виконувати.

Варіант використання – це послідовність транзакцій у діалозі між дійовою особою та компонентом або системою з відчутним результатом, де дійова особа може бути користувачем або будь-що, що може обмінюватися інформацією з системою.

Історія користувача – це вимога користувача або бізнес-вимога високого рівня, яка зазвичай використовується в гнучкій розробці програмного забезпечення, та зазвичай складається з одного або більше речень повсякденною чи діловою мовою, що відображає, яка функціональність потрібна користувачеві, будь-які нефункціональні критерії, а також містить критерії прийнятності.

Сценарій – це гіпотетична історія, яка використовується, щоб допомогти людині зрозуміти складну проблему або систему.

Бізнес-правило – це твердження, яке визначає або обмежує певний аспект бізнесу. Воно призначене для затвердження бізнес-структури або для контролю або впливу на поведінку бізнесу. Бізнес-правило виражає конкретні обмеження для створення, оновлення та видалення постійних даних в інформаційній системі.

Атрибут якості – це особливість або характеристика, яка впливає на якість продукту.

@ М.В.Добролюбова 9

Рівні та типи вимог

Функціональна вимога – це вимога, що визначає функцію, яку повинен виконувати компонент або система. Функціональні вимоги описують спостережувану поведінку системи за певних умов, і дії, які система дозволить виконувати користувачам.

Нефункціональна вимога – це вимога, яка стосується не функціональності, а таких атрибутів, як надійність, ефективність, зручність використання, ремонтопридатність та портативність.

Обмеження, стиснення – це обмеження дизайну та реалізації, що законно мінімізують можливості, доступні розробнику.

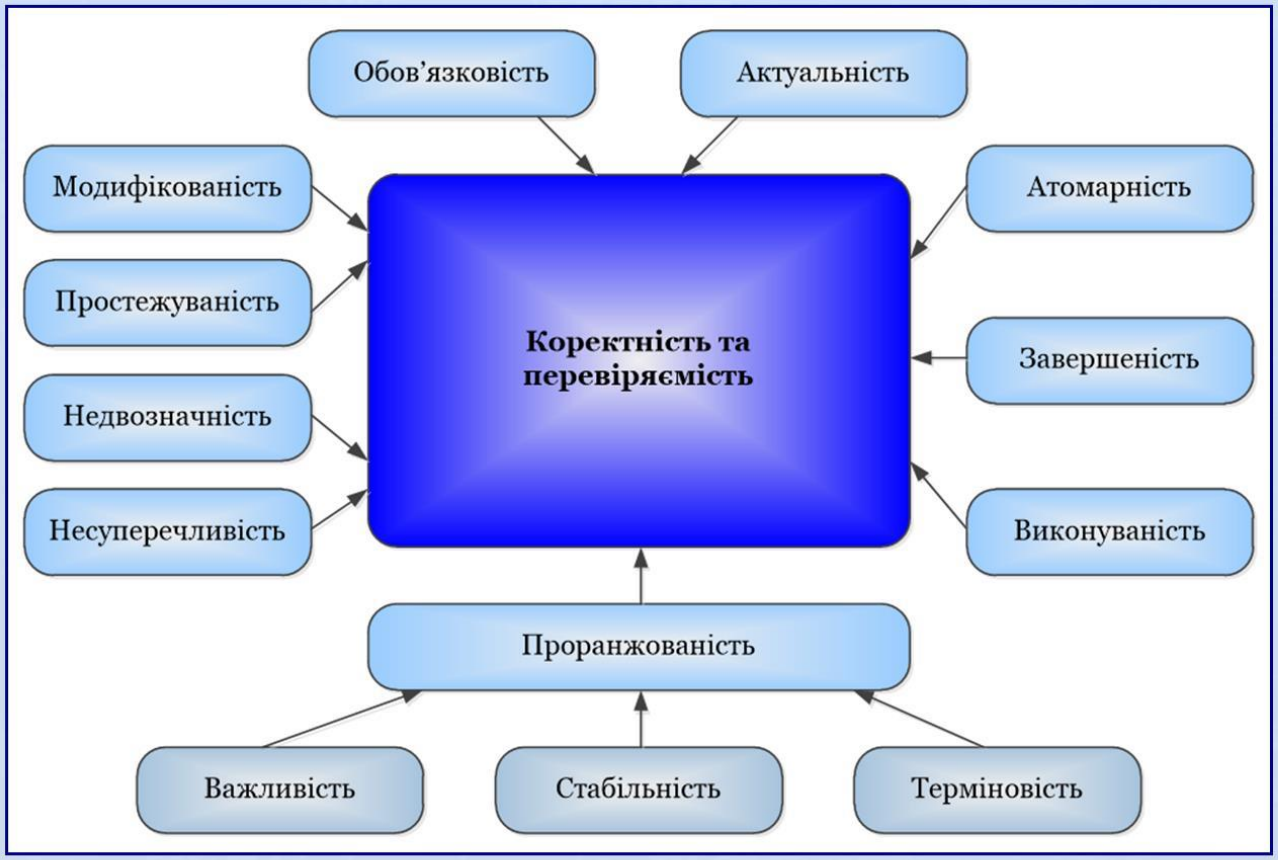
Вимоги до зовнішнього інтерфейсу – це вимоги, які описують зв'язки між системою та рештою всесвіту. Вони включають інтерфейси для користувачів, обладнання та інші програмні системи.

Вимоги до даних – це вимоги, які описують формат, тип даних, дозволені значення або значення за замовчуванням для елемента даних; склад складної структури бізнес-даних або звіт, який буде створено.

Інструмент управління вимогами – це інструмент, який підтримує запис вимог, атрибутів вимог (наприклад, пріоритет, відповідальний за знання) та анотації, а також полегшує відстеження через рівні вимог та управління змінами вимог. Деякі інструменти керування вимогами також надають засоби для статичного аналізу, наприклад, перевірку узгодженості та порушення попередньо визначених правил вимог.

@ М.В.Добролюбова 10

Властивості якісних вимог



@ М.В.Добролюбова

11

Властивості якісних вимог

Завершеність

Способи виявлення проблем	Способи усунення проблем
Застосовні майже всі техніки тестування вимог, але найкраще допомагає задавання питань та використання графічного представлення системи, що розробляється. Також дуже допомагає глибоке знання предметної галузі, що дозволяє помічати пропущені фрагменти інформації.	Як тільки було з'ясовано, що чогось не вистачає, потрібно отримати необхідну інформацію та дописати її до вимог. Можливо, знадобиться невелика переробка вимог.

Атомарність

Способи виявлення проблем	Способи усунення проблем
Обмірковування, обговорення з колегами та здоровий глузд: якщо ми вважаємо, що якийсь розділ вимог перевантажений і потребує декомпозиції, швидше за все, так і є.	Переробка та структурування вимог: розбиття їх у розділи, підрозділи, пункти, підпункти тощо.

@ М.В.Добролюбова

12

Властивості якісних вимог

Несуперечність

Способи виявлення проблем	Способи усунення проблем
Найкраще виявити суперечливість допомагає гарна пам'ять, але навіть за її наявності незамінним інструментом є графічне надання розроблюваної системи, що дозволяє представити всю ключову інформацію у вигляді єдиної узгодженої схеми (на якій суперечності дуже помітні).	Після виявлення протиріччя потрібно прояснити ситуацію із замовником та внести необхідні виправлення до вимог.

Недвозначність

Способи виявлення проблем	Способи усунення проблем
Побачити у вимогах двозначність добре допомагають перелічені вище слова-індикатори. Настільки ж ефективним є продумування перевірок: дуже важко придумати об'єктивну перевірку для вимоги, що припускає різночитання.	Найстрашніший ворог двозначності – числа та формули: якщо щось можна висловити у формульному або числовому вигляді (замість словесного опису), обов'язково варто це зробити. Якщо це неможливо, варто хоча б використовувати максимально точні технічні терміни, посилання до стандартів тощо.

@ М.В.Добролюбова 13

Властивості якісних вимог

Виконуваність

Способи виявлення проблем	Способи усунення проблем
На жаль, тут є лише один шлях: максимально напрацьовувати досвід та виходити із нього. Неможливо зрозуміти, що деяка вимога «коштує» занадто багато або зовсім нездійсненно, якщо немає розуміння процесу розробки ПЗ, розуміння предметної області та інших супутніх знань.	При виявленні нездійсненності вимоги не залишається нічого іншого, як докладно обговорити ситуацію із замовником та/або змінити вимогу (можливо – відмовитись від неї), або переглянути умови виконання проекту (зробивши виконання цієї вимоги можливим).

Обов'язковість

Способи виявлення проблем	Способи усунення проблем
Постійний (періодичний) перегляд вимог (бажано – з участю замовника) дозволяє помітити фрагменти, що втратили актуальність або стали низькопріоритетними.	Переробка вимог (з усуненням фрагментів, що втратили актуальність) та переробка фрагментів, у яких змінився пріоритет (часто зміна пріоритету веде і до зміни формулювання вимоги).

@ М.В.Добролюбова

14

Властивості якісних вимог

Простежуваність

Способи виявлення проблем	Способи усунення проблем
Порушення простежуваності стають помітними у процесі роботи з вимогами, як тільки у нас виникають питання, що залишаються без відповіді, виду «звідки взялася ця вимога?», «де описані супутні (пов'язані) вимоги?», «на що це впливає?».	Переробка вимог. Можливо, доведеться навіть змінювати структуру набору вимог, але все точно почнеться з розміщення безлічі перехресних посилань, що дозволяють здійснювати швидку і прозору навігацію за набором вимог.

Модифікованість

Способи виявлення проблем	Способи усунення проблем
Якщо при внесенні змін до набору вимог, ми стикаємося з проблемами, характерними для ситуації втрати простежуваності, отже – ми виявили проблему з модифікованістю. Також модифікованість погіршується за наявності практично будь-якої з розглянутих у розділі проблем з вимогами.	Переробка вимог із першорядною метою підвищити їх простежуваність. Паралельно можна усувати інші проблеми.

@ М.В.Добролюбова 15

Властивості якісних вимог

Проранжованість за важливістю, стабільністю, терміновістю

Способи виявлення проблем	Способи усунення проблем
Як і у випадку з актуальністю та обов'язковістю вимог, тут найкращим способом виявлення недоробок є постійний (періодичний) перегляд вимог (бажано – з участю замовника), у процесі якого можна виявити невірні значення показників терміновості, важливості та стабільності обговорюваних вимог.	Прямо у процесі обговорення вимог із замовником (під час перегляду вимог) варто вносити правки до значення показників терміновості, важливості та стабільності обговорюваних вимог.

Коректність і перевіряємість

Способи виявлення проблем	Способи усунення проблем
Оскільки тут ми маємо справу з «інтегральною» проблемою, то виявляється вона з використанням раніше описаних способів. Окремих унікальних методик тут нема.	Внесення до вимог необхідних змін – від елементарного виправлення виявленої помилки, до глобальної переробки всього набору вимог.

«Writing Good Requirements – The Big Ten Rules

@ М.В.Добролюбова

16

Техніки тестування вимог

Основні техніки тестування вимог:

- взаємний перегляд (peer review): побіжний перегляд (walkthrough), технічний перегляд (technical review), формальна інспекція (inspection)
- запитання
- тест-кейси та чек-листи
- вивчення поведінки системи
- рисунки (графічне надання)
- прототипування

Техніки тестування вимог

Приклад поганих та гарних питань до вимог

Погана вимога	Погані питання	Гарні питання
«Додаток повинен швидко запускатись»	«Наскільки швидко?» «А якщо не вийде швидко?» «Завжди?»	«Який максимально допустимий час запуску програми, на якому обладнанні та при якій завантаженості цього обладнання операційною системою та іншими програмами? На досягнення яких цілей впливає швидкість запуску програми? Чи допускається фонове завантаження окремих компонентів програми? Що є критерієм того, що програма закінчила запуск?»
«Опціонально повинен підтримуватися експорт документів у формат PDF»	«Будь-яких документів?» «У PDF якої версії повинен виконуватись експорт?» «Навіщо?»	«Наскільки можливість експорту до PDF важлива? Як часто, ким і з якою метою вона використовуватиметься? Чи є PDF єдиним допустимим форматом для цих цілей чи є альтернативи? Чи допускається використання зовнішніх утиліт (наприклад, віртуальних PDF-принтерів) для експорту документів у PDF?»
«Якщо дата події не вказана, вона вибирається автоматично».	«А якщо вказана?» «А якщо дату неможливо вибрати автоматично?» «А якщо у події немає дати?»	«Можливо, йшлося про те, що дата генерується автоматично, а не вибирається? Якщо так, то яким алгоритмом вона генерується? Якщо ні, то з якого набору вибирається дата та як генерується цей набір? P.S. Можливо, чи варто використовувати поточну дату?»

@ М.В.Добролюбова

18

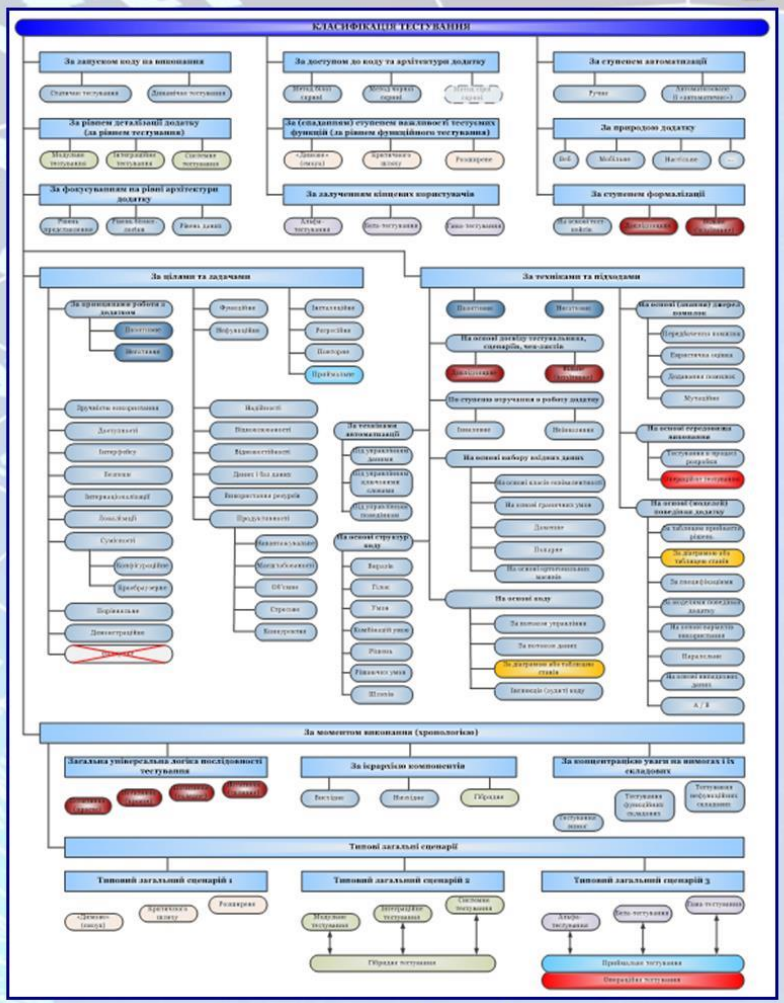

Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 2. КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ
Лекція 4
«Різновиди та напрями тестування»
@ М.В.Добролюбова

Спрощена класифікація тестування

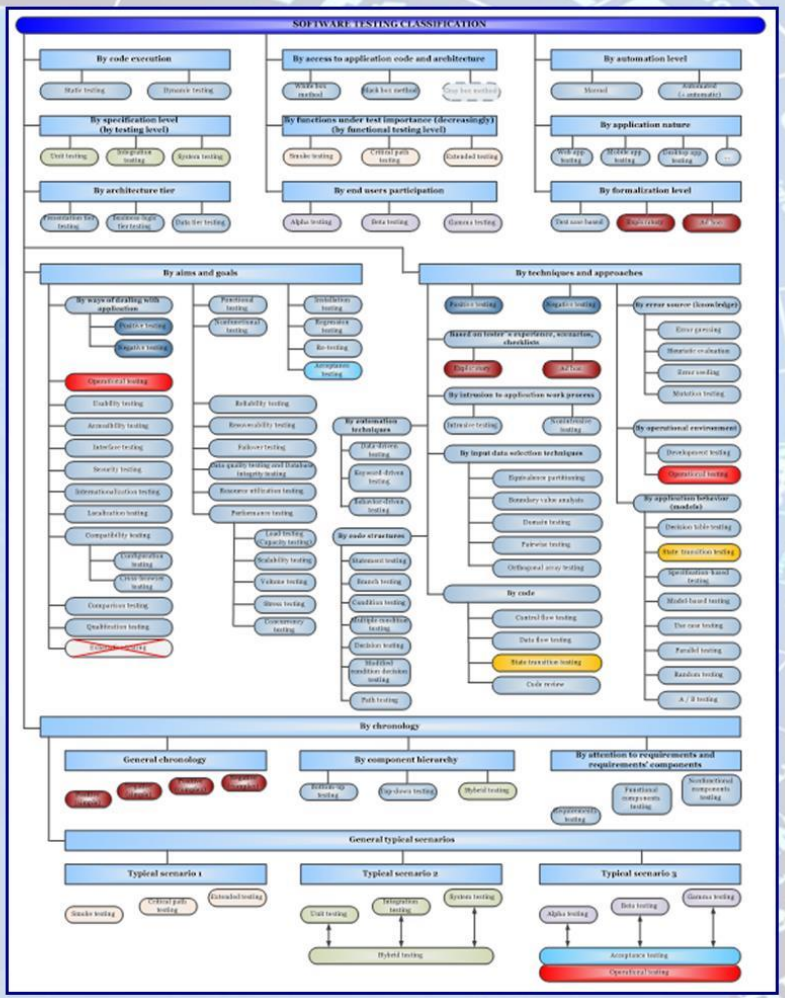
- **за запуском коду на виконання:** статичне тестування, динамічне тестування
- **за доступом до коду та архітектури додатку:** метод білої скриньки, метод чорної скриньки, метод сірої скриньки
- **за ступенем автоматизації:** ручне тестування, автоматизоване тестування
- **за рівнем деталізації додатку (за рівнем тестування):** модульне (компонентне) тестування, інтеграційне тестування, системне тестування
- **за (спаданням) ступеня важливості функцій, що тестуються (за рівнем функціонального тестування):** димове тестування, тестування критичного шляху, розширене тестування
- **за принципами роботи з додатком:** позитивне тестування, негативне тестування

@ М.В.Добролюбова 2

Детальна класифікація тестування



Україномовний варіант



Англомовний варіант

@ М.В.Добролюбова

3

Детальна класифікація тестування

- **за запуском коду на виконання:** статичне, динамічне
- **за доступом до коду та архітектури додатку:** метод білої скриньки, метод чорної скриньки, метод сірої скриньки
- **за ступенем автоматизації:** ручне, автоматизоване
- **за рівнем деталізації додатку (за рівнем тестування):** модульне (компонентне), інтеграційне, системне
- **за (спаданням) ступеня важливості функцій, що тестуються (за рівнем функціонального тестування):** димове, критичного шляху, розширене
- **за природою додатку:** веб, мобільне, настільне
- **за фокусуванням на рівні архітектури додатку:** рівень представлення, рівень бізнес-логіки, рівень даних
- **за залученням кінцевих користувачів:** альфа, бета, гама
- **за ступенем формалізації:** на основі тест-кейсів, дослідницьке, вільне (інтуїтивне)
- **за цілями та задачами:** за принципами роботи з додатком (позитивне, негативне), функційне, нефункційне, інсталяційне, регресійне, повторне, приймальне, за зручністю використання, за доступністю, за інтерфейсом, за безпекою, за інтернаціоналізацією, за локалізацією, за сумісністю (конфігураційне, кросбраузерне), порівняльне, демонстраційне, вичерпне, за надійністю, за відновлюваністю, за даними і базами даних, за використанням ресурсів, за продуктивністю (навантажувальне, масштабованості, об'ємне, стресове, конкурентне), за техніками автоматизації (під управлінням даних, ключовими словами, поведінкою), на основі структур коду (виразів, гілок, умов, комбінацій умов, рішень, рішаючи умов, шляхів)
- **за техніками та підходами:** позитивне, негативне, на основі досвіду тестувальника (дослідницьке, вільне), по ступеню втручання в роботу додатку (інвазивне, неінвазивне), на основі вибору вхідних даних (класів еквівалентності, граничних умов, ортогональних масивів, доменне, попарне), на основі коду (за потоком управління, за потоком даних, за діаграмою або таблицею станів, інспекція коду), на основі джерел помилок (передбачення помилок, евристична оцінка, додавання помилок, мутаційне), на основі середовища виконання (тестування в процесі розробки, операційне), на основі поведінки додатку (за таблицею прийняття рішень, за діаграмою або таблицею станів, за специфікаціями, за моделями поведінки додатку, на основі варіантів використання, паралельне, на основі випадкових даних, A/B)
- **за моментом виконання:** загальна універсальна логіка послідовності тестування (позитивне просте, негативне просте, позитивне складне, негативне складне), за ієрархією компонентів (висхідне, низхідне, гібридне), за концентрацією уваги на вимогах і їх складових (тестування вимог, тестування функцій них складових, тестування нефункційних складових)
- **типові загальні сценарії:** типовий загальний сценарій 1 (димове, критичного шляху, розширене), типовий загальний сценарій 1 (модульне, інтеграційне, системне), типовий загальний сценарій 3 (альфа, бета, гама)

@ М.В.Добролюбова 4

Детальна класифікація тестування

Класифікація із запуску коду на виконання

Статичне тестування (static testing) – це тестування артефакту розробки програмного забезпечення, наприклад, вимог, дизайну або коду, без виконання цих артефактів, наприклад, огляди чи статичний аналіз.

Динамічне тестування (dynamic testing) –це тестування, що включає програмне забезпечення компонента або системи.

Джейсон Коен, «Найпотаємніші секрети експертної перевірки коду (Сучасний підхід. Практичні поради)»

@ М.В.Добролюбова

5

Детальна класифікація тестування

Класифікація з доступу до коду та архітектури програми

Метод білої скриньки (white box testing, open box testing, clear box testing, glass box testing) – це тестування на основі аналізу внутрішньої структури компонента чи системи.

Метод чорної скриньки (black box testing, closed box testing, specificationbased testing) – це тестування, функціональне або нефункціональне, без посилання на внутрішню структуру компонента або системи.

Метод сірої скриньки (gray box testing) – це метод тестування програмного забезпечення, який є комбінацією методу тестування чорної скриньки та методу тестування білої скриньки. У Grey Box Testing внутрішня структура частково відома. Це передбачає доступ до внутрішніх структур даних і алгоритмів для цілей проєктування тестових випадків, але тестування проводиться на рівні користувача або на рівні чорної скриньки.

Проєктне тестування – це підхід до тестування, при якому тестові сценарії розробляються на основі архітектури та/або детального проєкту компонента або системи.

Тестування на основі вимог – це підхід до тестування, у якому тестові випадки розробляються на основі цілей тестування та умов тестування, що випливають із вимог.

«Основи тестування Gray Box», http://softwaretestingfundamentals.com/gray-box-testing

@ М.В.Добролюбова 6

Детальна класифікація тестування

Класифікація з доступу до коду та архітектури програми

Переваги та недоліки методів білої, чорної та сірої скринь

	Переваги	Недоліки		Переваги	Недоліки		Переваги	Недоліки
Метод білої скриньки	• Показує приховані проблеми та спрощує їх діагностику. • Допускає досить просту автоматизацію тест-кейсів та їх виконання на ранніх стадіях розвитку проєкту. • Має розвинену систему метрик, збір та аналіз яких легко автоматизується. • Стимулює розробників до написання якісного коду. • Багато технік цього методу є перевіреними, добре себе зарекомендували рішеннями, що базуються на строгому технічному підході.	• Не може виконуватися тестувальниками, які не мають достатніх знань у галузі програмування. • Тестування сфокусовано на реалізованій функціональності, що підвищує ймовірність пропуску нереалізованих вимог. • Поведінка додатку досліджується у відриві від реального середовища виконання та не враховує його вплив. • Поведінка додатку досліджується у відриві від реальних сценаріїв користувача.	**Метод чорної скриньки**	• Тестувальник не зобов'язаний мати (глибокі) знання в області програмування. • Поведінка додатку досліджується в контексті реального середовища виконання та враховує його вплив. • Поведінка програми досліджується в контексті реальних сценаріїв користувача. • Тест-кейси можна створювати вже на стадії появи стабільних вимог. • Процес створення тест-кейсів дозволяє виявити дефекти вимог. • Допускає створення тест-кейсів, які можна багаторазово використати на різних проєктах.	• Можливе повторення частини тест-кейсів, які вже виконані розробниками. • Висока ймовірність того, що частина можливих варіантів поведінки програми залишиться непротестованою. • Для розробки високоефективних тест-кейсів потрібна якісна документація. • Діагностика виявлених дефектів складніша порівняно з техніками методу білої скриньки. • У зв'язку з широким вибором технік та підходів ускладнюється планування та оцінка трудовитрат. • У разі автоматизації можуть знадобитися складні дорогі інструментальні засоби.	**Метод сірої скриньки**	Поєднує переваги та недоліки методів білої та чорної скриньки.	

@ М.В.Добролюбова 7

Детальна класифікація тестування

Класифікація за рівнем автоматизації

Ручне тестування (manual testing) – це тестування, що здійснюється тестувальником, який виконує всі дії з додатком, що тестується вручну, крок за кроком, і вказує, чи був конкретний крок виконаний успішно чи ні. Ручне тестування завжди є частиною будь-якого тестування. Це особливо корисно на початковому етапі розробки програмного забезпечення, коли програмне забезпечення та його інтерфейс користувача недостатньо стабільні, і починати автоматизацію не має сенсу.

Автоматизоване тестування (automated testing, test automation) – це використання програмного забезпечення для керування виконанням тестів, порівняння фактичних результатів із прогнозованими результатами, налаштування попередніх умов тестування та інших функцій управління тестуванням та звітів про тестування. Зазвичай автоматизація тестування включає автоматизацію вже існуючого ручного процесу, в якому використовується формалізований процес тестування.

Переваги та недоліки автоматизованого тестування

Переваги	Недоліки
• Швидкість виконання тест-кейсів може в рази та на порядки перевищувати можливості людини. • Відсутність впливу людського фактора у процесі виконання тест-кейсів (втоми, неуважності тощо). • Мінімізація витрат при багаторазовому виконанні тест-кейсів (участь людини тут потрібна лише епізодично). • Здатність засобів автоматизації виконати тест-кейси, у принципі непосильні для людини через свою складність, швидкість або інші фактори. • Здатність засобів автоматизації збирати, зберігати, аналізувати, агрегувати та представляти у зручній для сприйняття людиною формі колосальні обсяги даних. • Здатність засобів автоматизації виконувати низькорівневі дії з програмою, операційною системою, каналами передачі даних тощо.	• Необхідний високо-кваліфікований персонал через те, що автоматизація – це «проєкт усередині проєкту» (зі своїми вимогами, планами, кодом тощо). • Високі витрати на складні засоби автоматизації, розробку та супровід коду тест-кейсів. • Автоматизація вимагає більш ретельного планування та управління ризиками, оскільки в іншому випадку проєкту може бути завдано серйозної шкоди. • Засобів автоматизації дуже багато, що ускладнює проблему вибору того чи іншого засобу і може спричинити фінансові витрати (і ризики), необхідність навчання персоналу (або пошуку фахівців). • У разі відчутної зміни вимог, зміни технологічного домену, переробки інтерфейсів (як користувача, так і програмних) багато тест-кейси стають безнадійно застарілими і вимагають створення наново.

Равіндер Вір Худа, «Автоматизація тестування програмного забезпечення: основа для майбутнього»

@ М.В.Добролюбова 8

Детальна класифікація тестування

Класифікація за рівнем деталізації програми (за рівнем тестування)

Модульне (компонентне) тестування (unit testing, module testing, component testing) – це тестування окремих програмних компонентів.

Інтеграційне тестування (integration testing) – це тестування, яке проводиться для виявлення дефектів інтерфейсів і взаємодій між інтегрованими компонентами чи системами.

Компонентне інтеграційне тестування (component integration testing) – це тестування, яке проводиться для виявлення дефектів в інтерфейсах і взаємодії інтегрованих компонентів.

Парне інтеграційне тестування (pairwise integration testing) – це форма інтеграційного тестування, призначена для пар компонентів, які працюють разом.

Системне інтеграційне тестування (system integration testing) – це тестування інтеграції систем та пакетів; тестування інтерфейсів із зовнішніми організаціями.

Інкрементне тестування (incremental testing) – це тестування, при якому компоненти або системи інтегруються і тестуються по одному або кілька за раз, поки всі компоненти або системи не будуть проінтегровані і протестовані.

Тестування інтерфейсу (interface testing) – це тип інтеграційного тесту, пов'язаний із тестуванням інтерфейсів між компонентами або системами.

Потокове тестування (thread testing) – це підхід до компонентного інтеграційного тестування, у якому послідовна інтеграція компонентів слідує за реалізацією підмножин вимог, на відміну від інтеграції компонентів за рівнями ієрархії.

Тестування системи (system testing) – це процес тестування інтегрованої системи для перевірки відповідності заданим вимогам.

Рівень тестування (test level) – це група тестових дій, які організовуються та управляються разом. Рівень тестування пов'язаний з обов'язками у проєкті. Прикладами рівнів тестування є компонентне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

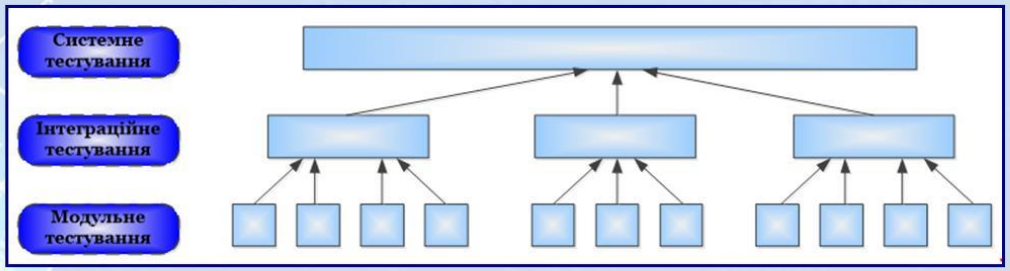
@ М.В.Добролюбова 9

Детальна класифікація тестування

Класифікація за рівнем деталізації програми (за рівнем тестування)

Самий повний варіант класифікації тестування за рівнем тестування

Схематичне представлення класифікації тестування за рівнем деталізації програми



Стаття «What are Software Testing Levels?»

@ М.В.Добролюбова

10

Детальна класифікація тестування

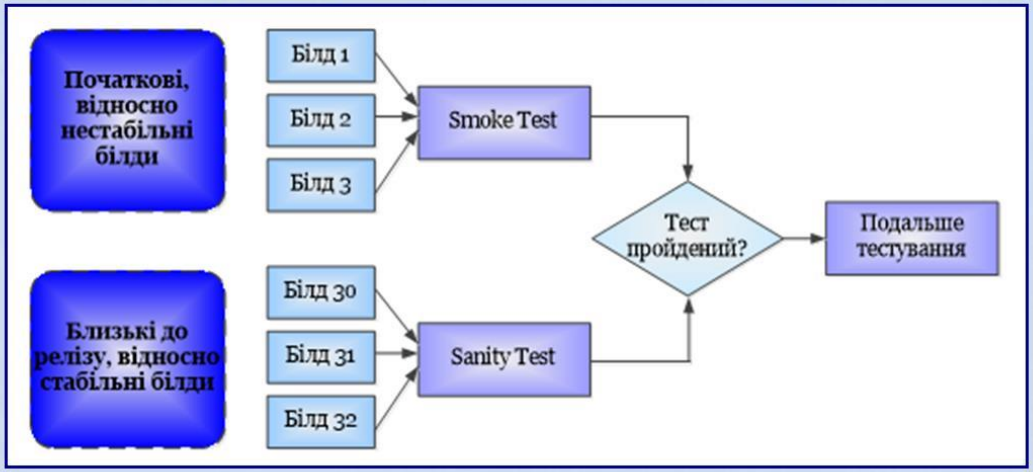
Класифікація за (спаданням) ступенем важливості функцій, що тестуються (за рівнем функціонального тестування)

Димове тестування (smoke test) – це підмножина всіх певних/запланованих тестових випадків, які охоплюють основні функціональні можливості компонента або системи, щоб переконатися, що найважливіші функції програми працюють, але не торкаючись дрібніших деталей.

Вступне тестування (intake test) – це особливий випадок димового тестування, щоб вирішити, чи готовий компонент або система до детального і подальшого тестування. Вступний тест зазвичай проводиться на початку етапу виконання тесту.

Тестування з перевірки збірки (build verification test) – це набір автоматичних тестів, які підтверджують цілісність кожної нової збірки та перевіряють її ключову/основну функціональність, стабільність та тестованість. Це галузева практика, коли збірки випускаються дуже часто (наприклад, agile-проєкти), і таке тестування запускається при кожній новій збірці до того, як збірку буде випущено для подальшого тестування.

Трактування різниці між smoke test та sanity test



@ М.В.Добролюбова

11

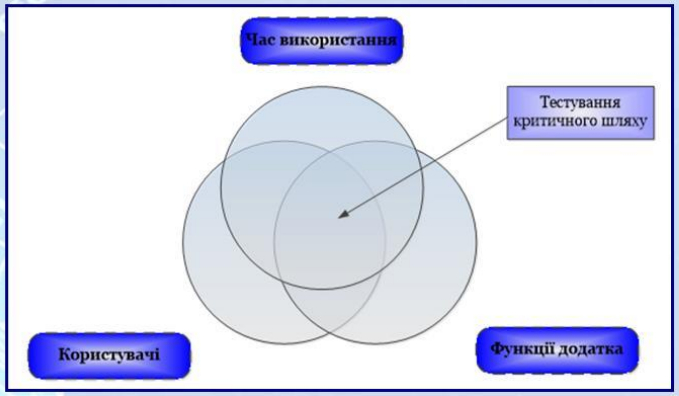
Детальна класифікація тестування

Класифікація за (спаданням) ступенем важливості функцій, що тестуються (за рівнем функціонального тестування)

Критичний шлях (critical path) – це найдовша послідовність заходів у плані проєкту, яка має бути виконана вчасно, щоб проєкт завершився у встановлений термін. Дія на критичному шляху не може бути розпочата, поки попередня дія не буде завершена; якщо вона затримується на один день, весь проєкт буде відкладено на день, якщо тільки дія, що слідує за відкладеною дією, не буде завершена на день раніше.

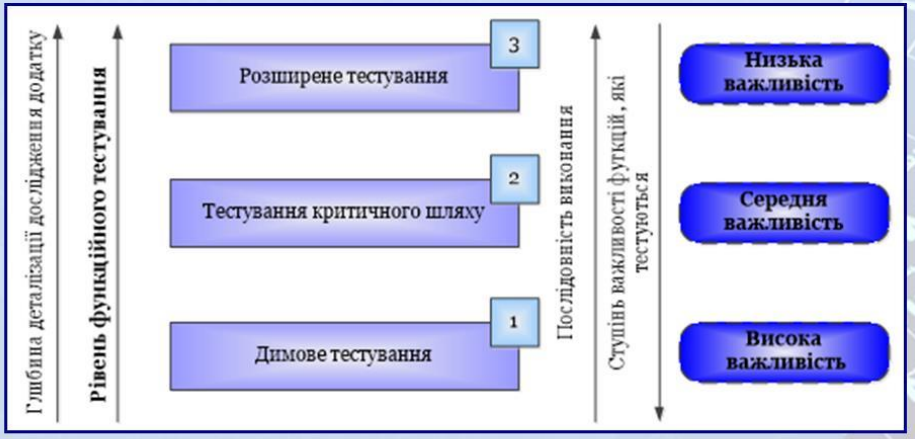
Розширений тест (extended test) – ідея полягає в тому, щоб розробити комплексний набір тестів прикладної системи шляхом моделювання основних можливостей у вигляді розширених варіантів використання.

Суть тестування критичного шляху



«Шаблон проектування тестів розширеного використання», Роб Куйт

Класифікація тестування за (спаданням) ступенем важливості функцій, що тестуються (за рівнем функціонального тестування)



@ М.В.Добролюбова

12

Детальна класифікація тестування

Класифікація за принципами роботи із додатком

Позитивне тестування (positive testing) – це процес тестування, при якому система перевіряється на відповідність припустимим вхідним даним. У цьому тестуванні тестувальник завжди перевіряє лише припустимий набір значень і перевіряє, чи очікувано веде себе додаток при очікуваних вхідних даних. Основна мета цього тестування полягає в тому, щоб перевірити, чи програмне забезпечення не показує помилку, коли цього бути не повинно, і показує помилку, коли це має бути. Таке тестування має проводитися, з позитивної точки зору та виконувати лише позитивний сценарій. Позитивне тестування завжди намагається довести, що даний продукт і проєкт завжди відповідають вимогам і специфікаціям.

Негативне тестування (negative testing) – це тести, спрямовані на те, щоб показати, що компонент або система не працює. Негативне тестування пов'язане зі ставленням тестувальників, а не з конкретним підходом до тестування або технікою розробки тестів, наприклад, тестування з неприпустимими вхідними значеннями або виключеннями.

Недійсне тестування (invalid testing) – це тестування з використанням вхідних значень, які повинні бути відхилені компонентом або системою.

@ М.В.Добролюбова

13

Детальна класифікація тестування

Класифікація за природою програми

Тестування веб-додатків (web-applications testing) – це тестування, яке пов'язане з інтенсивною діяльністю в області тестування сумісності, тестування продуктивності, автоматизації тестування з використанням широкого спектру інструментальних засобів.

Тестування мобільних додатків (mobile applications testing) – це тестування, яке вимагає підвищеної уваги до тестування сумісності, оптимізації продуктивності, автоматизації тестування із застосуванням емуляторів мобільних пристроїв.

Тестування настільних додатків (desktop applications testing) – це тестування з особливостями, які залежать від предметної області додатку, нюансів архітектури, ключових показників якості тощо.

@ М.В.Добролюбова 14

Детальна класифікація тестування

Класифікація за фокусуванням на рівні архітектури програми

Тестування рівня представлення (presentation tier testing) – це тестування, яке сконцентроване на тій частині програми, що відповідає за взаємодію із «зовнішнім світом». Тут досліджуються питання зручності використання, швидкості відгуку інтерфейсу, сумісності із браузерами, коректності роботи інтерфейсів.

Тестування рівня бізнес-логіки (business logic tier testing) – це тестування, яке відповідає за перевірку основного набору функцій програми та будується на базі ключових вимог до додатку, бізнес-правил та загальної перевірки функціональності.

Тестування рівня даних (data tier testing) – це тестування, яке сконцентровано на тій частини програми, що відповідає за збереження та деяку обробку даних. Тут особливий інтерес має тестування даних, перевірка дотримання бізнес-правил, тестування продуктивності.

Багаторівнева архітектура додатків – це клієнт-серверна архітектура, в якій поділяються функції представлення, обробки та зберігання даних.

@ М.В.Добролюбова

15

Детальна класифікація тестування

Класифікація із залученням кінцевих користувачів

Альфа-тестування (alpha testing) – це імітаційне або фактичне експлуатаційне тестування потенційними користувачами/замовниками або незалежною тестовою групою на сайті розробників, але за межами організації розробників. Альфа-тестування часто використовується для готового програмного забезпечення як форма внутрішнього приймального тестування.

Бета-тестування (beta testing) – це операційне тестування потенційними та/або існуючими користувачами/замовниками на зовнішньому сайті, не пов'язаному з розробниками, для визначення того, чи задовольняє компонент або система потребам користувача/замовника та чи вписується в бізнес-процеси. Бета-тестування часто використовується як форма зовнішнього приймального тестування готового програмного забезпечення для отримання зворотного зв'язку від ринку.

Гамма-тестування (gamma testing) – це тестування, яке виконується, коли програмне забезпечення готове до випуску із зазначеними вимогами, це тестування здійснюється безпосередньо, пропускаючи всі внутрішні дії з тестування. Програмне забезпечення майже готове до фінальної версії. Ніякої розробки функцій або вдосконалення програмного забезпечення не проводиться, а виправлення помилок є єдиним кодом. Гамма-перевірка виконується, коли додаток готовий до випуску відповідно до зазначених вимог, і ця перевірка виконується безпосередньо.

Детальна класифікація тестування

Класифікація за ступенем формалізації

Тестування на основі тест-кейсів (скриптове тестування, scripted testing, test case based testing) – тестування, при якому виконання тесту здійснюється шляхом дотримання попередньо задокументованої послідовності тестів.

Дослідницьке тестування (exploratory testing) – неформальна техніка розробки тестів, коли тестувальник активно контролює дизайн тестів під час виконання цих тестів і використовує інформацію, отриману під час тестування, для розробки нових і кращих тестів.

Сесійне тестування (session-based testing) – підхід до тестування, при якому тестові дії плануються як безперервні сеанси розробки та виконання тестів; часто використовується разом із дослідницьким тестуванням.

Тестування на основі чек-листів (checklist-based testing) – метод розробки тестів, заснований на досвіді, за допомогою якого досвідчений тестувальник використовує високорівневий список елементів, які потрібно визначити, перевірити чи запам'ятати, або набір правил чи критеріїв, за якими продукт має бути перевірений.

Вільне (інтуїтивне) тестування (ad hoc testing) – тестування, яке проводиться неофіційно; формальна підготовка до тесту не проводиться, не використовується визнана техніка розробки тестів, немає очікувань на результати та діяльність з виконання тестів визначається довільно.

Тестування на основі досвіду (experience-based testing) – це тестування на основі досвіду, знань та інтуїції тестувальника.

Стаття «Що таке дослідницьке тестування?», Джеймс Бах

@ М.В.Добролюбова 17

Детальна класифікація тестування

Класифікація за цілями та завданнями

Функціональне тестування (functional testing) – тестування на основі аналізу специфікації функціональності компонента або системи.

Тестування функціональності (functionality testing) – процес тестування з метою визначення функціональності програмного продукту (здатність програмного продукту забезпечувати функції, які задовольняють заявленим і передбачуваним потребам, коли програмне забезпечення використовується за певних умов).

Нефункціональне тестування (non-functional testing) – тестування атрибутів компонента або системи, які не стосуються функціональності, наприклад надійність, ефективність, зручність використання, ремонтопридатність і портативність.

Інсталяційне тестування (installation testing, installability testing) – процес перевірки можливості встановлення програмного продукту.

Регресійне тестування (regression testing) – тестування попередньо протестованої програми після модифікації, щоб переконатися, що дефекти не були внесені або виявлені в незмінних областях програмного забезпечення в результаті внесених змін. Виконується при зміні програмного забезпечення або його середовища.

Повторне тестування, тестування на підтвердження (re-testing, confirmation testing) – тестування, що запускає тестові випадки, які не вдалося виконати під час останнього запуску, щоб перевірити успішність коригувальних дій.

Приймальне тестування (acceptance testing) – формальне тестування з врахуванням потреб користувачів, вимог та бізнес-процесів, яке проводиться, щоб визначити, чи задовольняє система критеріям прийняття, і щоб користувач, клієнти чи інший уповноважений орган могли визначити, чи приймати систему чи ні.

Стаття «What is Functional testing (Testing of functions) in software?»
Стаття «What is functionality testing in software?»
Frederick Brooks, «The Mythical Man-Month»

@ М.В.Добролюбова 18

Детальна класифікація тестування

Класифікація за цілями та завданнями

Виробниче приймальне тестування (factory aceptance testing) – приймальні випробування, що проводяться на місці, де продукт розробляється, та виконуються працівниками організації-постачальника, щоб визначити, чи відповідає компонент або система вимогам, як правило, включаючи апаратне та програмне забезпечення.

Операційне приймальне тестування (operational aceptance testing, production acceptance testing) – операційне тестування на етапі приймального випробування, яке зазвичай виконується в (симульованому) операційному середовищі персоналом операційного та/або системного адміністрування, з особливою увагою до експлуатаційних аспектів, наприклад можливості відновлення, поведінки ресурсів, можливост встановлення та технічної відповідності.

Підсумкове приймальне тестування (site aceptance testing) – приймальне тестування користувачами/замовниками на їх сайті, щоб визначити, чи відповідає компонент або система потребам користувача/замовника та чи вписується в бізнес-процеси, як правило, включаючи апаратне та програмне забезпечення.

Операційне тестування (operational testing) – тестування, проведене для оцінки компонента або системи в робочому середовищі.

Операційне середовище (operational environment) – апаратні та програмні продукти, які встановлюються на сайтах користувачів або клієнтів, де будуть використовуватися компонент або система, що тестується. Програмне забезпечення може включати операційні системи, системи управління базами даних та інші програми.

Зручність використання (usability testing) – здатність програмного забезпечення бути зрозумілим, засвоєним, використаним і привабливим для користувача при використанні за певних умов.

Зрозумілість (understandability) – здатність програмного продукту дозволити користувачеві зрозуміти, чи підходить програмне забезпечення, і як його можна використовувати для певних завдань і умов застосування.

@ М.В.Добролюбова 19

Детальна класифікація тестування

Класифікація за цілями та завданнями

Навчання (learnability) – здатність програмного продукту дозволити користувачеві вивчити його застосування.

Працездатність (operability) – здатність програмного продукту дозволити користувачеві керувати ним.

Привабливість (attractiveness) – здатність програмного продукту бути привабливим для користувача.

Тестування доступності (accessibility testing) – тестування для визначення простоти, з якою користувачі з обмеженими можливостями можуть використовувати компонент або систему.

Тестування інтерфейсу (interface testing) – тип інтеграційного тесту, який стосується тестування інтерфейсів між компонентами або системами.

Тестування інтерфейсу прикладного програмування, тестування API (API testing) – тестування, яке здійснюється шляхом подання команд програмному забезпеченню, що тестується, безпосередньо за допомогою програмних інтерфейсів додатку.

Тестування інтерфейсу командного рядка, тестування CLI (CLI testing) – тестування, яке виконується шляхом подання команд програмному забезпеченню, яке тестується, за допомогою спеціального інтерфейсу командного рядка.

Тестування графічного інтерфейсу (GUI testing) – тестування, яке здійснюється шляхом взаємодії з тестованим програмним забезпеченням через графічний інтерфейс користувача.

Тестування безпеки (security testing) – тестування для визначення безпеки програмного продукту.

Глобалізація (globalization) – процес розробки ядра програми, чиї функції та дизайн коду не ґрунтуються лише на одній мові чи локалі.

Можливість локалізації (localizability) – розробка базового програмного коду та ресурсів, що дозволяє локалізувати програму на різні мовні версії без будь-яких змін у вихідному коді.

«Що таке тестування безпеки в тестуванні програмного забезпечення?» [http://istqbexamcertification.com/what-is-security-testing-in-software/]
«Глобалізація крок за кроком», https://docs.microsoft.com/en-us/globalization/

@ М.В.Добролюбова 20

Детальна класифікація тестування

Класифікація за цілями та завданнями

Тестування локалізації (localization testing) перевіряє якість локалізації продукту для певної цільової культури/локалу. Цей тест базується на результатах тестування глобалізації, яке перевіряє функціональну підтримку цієї конкретної культури/мови. Тестування локалізації можна виконати лише на локалізованій версії продукту

Тестування на сумісність, тестування функціональної сумісності (compatibility testing, interoperability testing) – процес тестування для визначення функціональної сумісності програмного продукту (здатності взаємодіяти з одним або кількома вказаними компонентами або системами).

Конфігураційне тестування, портативне тестування (configuration testing) – процес тестування для визначення портативності програмного продукту (легкість, з якою програмний продукт може бути перенесений з одного апаратного чи програмного середовища в інше).

Міжбраузерне тестування (cross-browser testing) – тестування, яке допомагає вам переконатися, що ваш веб-сайт або веб-програма правильно функціонують у різних веб-браузерах. Як правило, QA інженери створюють окремі тести для кожного браузера або створюють тести, які використовують багато умовних операторів, що перевіряють тип використовуваного браузера та виконують команди, які його стосуються.

Мобільне тестування (mobile testing) – це тестування з кількома операційними системами (і різними версіями кожної ОС, особливо з Android), декількома пристроями (різні марки та моделі телефонів, планшетів, фаблетів), кількома операторами зв'язку (у тому числі міжнародними), різними швидкостями передачі даних (3-5G, LTE, Wi-Fi), кількома розмірами екрана (а також роздільними здатностями і співвідношеннями сторін), кількома елементами керування вводом (включаючи вічні фізичні клавіатури BlackBerry) і безліч технологій – GPS, акселерометри – які веб- та настільні програми майже ніколи не використовують.

Тестування відповідності, тестування на відповідність, тестування нормативів (compliance testing, conformance testing, regulation testing) – процес тестування для визначення відповідності компонента або системи (здатність дотримуватися стандартів, конвенцій або правил у законах та подібних приписах).

«Що таке мобільне тестування?» [https://www.perfecto.io/blog/mobile-testing]
«Посібник для початківців з тестування мобільних додатків» [http://www.softwaretestinghelp.com/beginners-guide-to-mobile-applicationtesting/]

@ М.В.Добролюбова

21

Детальна класифікація тестування

Класифікація за цілями та завданнями

Якість даних (data quality) – атрибут даних, який вказує на правильність деяких попередньо визначених критеріїв, наприклад, бізнес-очікувань, вимог до цілісності даних, узгодженості даних.

Тестування цілісності бази даних (database integrity testing) – тестування методів і процесів, які використовуються для доступу до даних (бази) та керування ними, щоб переконатися, що методи доступу, процеси та правила даних функціонують належним чином і що під час доступу до бази даних дані не пошкоджені, несподівано не видалені, оновлені чи створені.

Тестування використання ресурсів, тестування зберігання (resource utilization testing) – процес тестування для визначення використання ресурсів програмного продукту.

Тестування ефективності (efficiency testing) – процес тестування для визначення ефективності програмного забезпечення прпродукт (здатність процесу надавати передбачуваний результат відносно кількості використаних ресурсів).

Тестування зберігання (storage testing) – це визначення того, чи використовують певні умови обробки більше пам'яті, ніж передбачалося.

Порівняльне тестування (comparison testing) – тестування, під час якого порівнюються слабкі та сильні сторони програмного забезпечення з продуктами конкурентів.

Демонстраційне тестування (qualification testing) – формальне (офіційне) тестування, яке зазвичай проводить розробник для споживача, щоб продемонструвати, що програмне забезпечення відповідає встановленим вимогам.

Вичерпне тестування (exhaustive testing) – тестовий підхід, у якому набір тестів містить усі комбінації вхідних значень та передумов.

«Software Testing Concepts And Tools», Nageshwar Rao Pusuluri
«Software Testing and Quality Assurance», Jyoti J. Malhotra, Bhavana S. Tiple

@ М.В.Добролюбова 22

Детальна класифікація тестування

Класифікація за цілями та завданнями

Тестування надійності (reliability testing) – процес тестування для визначення надійності програмного продукту (здатності програмного продукту виконувати свої необхідні функції за встановлених умов протягом певного періоду часу, або за задану кількість операцій).

Тестування відновлюваності (recoverability testing) – процес тестування для визначення можливості відновлення програмного продукту (здатність програмного продукту відновити заданий рівень продуктивності та відновити дані, які безпосередньо постраждали в разі збою).

Тестування відмовостійкості (failover testing) – тестування шляхом імітування режимів відмови або фактичного виникнення відмов у контрольованому середовищі. Після збою механізм аварійного перемикання тестується, щоб гарантувати, що дані не будуть втрачені або пошкоджені, а також будуть підтримуватись всі узгоджені рівні обслуговування (наприклад, доступність функцій або час відгуку).

Тестування продуктивності (performance testing) – процес тестування для визначення продуктивності програмного продукту.

Навантажувальне тестування (load testing) – тип тестування продуктивності, яке проводиться для оцінки поведінки компонента або системи зі збільшенням навантаження, наприклад кількості паралельних користувачів і/або кількості транзакцій, щоб визначити, яке навантаження може обробляти компонент або система.

Ємнісне тестування (capacity testing) – тестування, щоб визначити, скільки користувачів та/або транзакцій дана система підтримуватиме та при цьому відповідатиме вимогам продуктивності.

Тестування масштабованості (scalability testing) – тестування для визначення масштабованості програмного продукту (можливість оновлення програмного продукту задля роботи з підвищеними навантаженнями).

Об'ємне тестування (volume testing) – тестування, коли система піддається впливу великих об’ємів даних.

@ М.В.Добролюбова 23

Детальна класифікація тестування

Класифікація за цілями та завданнями

Стресове тестування (stress testing) – тип тестування продуктивності, що проводиться для оцінки системи або компонента на рівні або за межами очікуваних чи визначених робочих навантажень, або з обмеженою доступністю ресурсів, таких як доступ до пам'яті чи серверів.

Тестуванням на руйнування програмного забезпечення (destructive testing) –тестування, що забезпечує правильну або передбачувану поведінку програмного забезпечення, коли програмне забезпечення піддається неналежному використанню або неправильному вводу, спробам призвести до збою програмного продукту, спробам зламати програмний продукт, перевірити надійність програмного продукту.

Конкурентне тестування (concurrency testing) – тестування для визначення того, як компонент або система обробляє появу двох або більше дій протягом одного і того ж проміжку часу, яке досягається шляхом чергування дій або одночасного виконання.

«Towards Destructive Software Testing», Kiumi Akingbehin
«Автоматизація тестування ефективності: основні положення та області застосування» [http://svyatoslav.biz/technologies/performance_testing/]

@ М.В.Добролюбова 24

Детальна класифікація тестування

Класифікація за техніками та підходами

Тестування на основі досвіду тестувальника, сценаріїв, чек-листів: дослідницьке, вільне (інтуїтивне) тестування.

Класифікація за ступенем втручання у роботу додатку: інвазивне, неінвазивне тестування.

Класифікація з технік автоматизації: тестування під управлінням даними, тестування під управлінням ключовими словами, тестування під управлінням поведінкою.

Класифікація на основі (знання) джерел помилок: тестування передбаченням помилок, евристична оцінка, мутаційне тестування, тестування додаванням помилок.

Класифікація на основі вибору вхідних даних: тестування на основі класів еквівалентності, тестування на основі граничних умов, доменне тестування, попарне тестування, тестування на основі ортогональних масивів.

Класифікація на основі середовища виконання: тестування в процесі розробки, операційне тестування, тестування на основі коду (тестування по потоку управління, тестування по потоку даних, тестування за діаграмою або таблицею станів, інспекція коду), тестування на основі структур коду (тестування на основі виразів, тестування на основі гілок, тестування на основі умов, тестування на основі комбінацій умов, тестування на основі окремих умов, що породжують розгалуження, тестування на основі рішень, тестування на основі шляхів), тестування на основі (моделей) поведінки програми (тестування за таблицею прийняття рішень, тестування за діаграмою або таблицею станів, тестування за специфікаціями, тестування за моделями поведінки програми, тестування на основі варіантів використання, паралельне тестування, тестування на основі випадкових даних, A/B-тестування).

«A Practitioner's Guide to Software Test Design», Lee Copeland

@ М.В.Добролюбова

25

Детальна класифікація тестування

Класифікація за техніками та підходами

Інвазивне тестування (нав'язливе) (intrusive testing) – тестування, яке збирає інформацію про час та обробку під час виконання програми, що може змінити поведінку програмного забезпечення у порівнянні з його поведінкою в реальному середовищі.

Рівень проникнення (level of intrusion) – рівень, до якого модифікується тестовий об'єкт шляхом налаштування його для тестування.

Неінвазивне тестування (ненав'язливе) (nonintrusive testing) – тестування, яке є прозорим для тестованого програмного забезпечення, тобто не змінює його часові характеристики чи характеристики обробки.

Тестування під управлінням даними (DDT) (data-driven testing) – метод створення сценаріїв, за якого вхідні дані тесту та очікувані результати зберігаються в таблиці або електронній таблиці, щоб один керуючий сценарій міг виконувати всі тести в таблиці.

Тестування під управлінням ключовими словами (KDT) (keyworddriven testing) – метод створення сценаріїв, за якого файли даних використовуються для збереження не лише текстових даних та очікуваних результатів, але й ключових слів, пов'язаних з додатком, що тестується.

Тестування під управлінням поведінкою (поведінкове тестування, BDT) (behavior-driven testing). **Поведінкові тести** – тести, що фокусуються на поведінці, а не на технічному виконанні програмного забезпечення.

Тестування передбаченням помилок (error guessing) – метод проектування тестів, при якому досвід тестувальника використовується для прогнозування дефектів, які можуть бути присутніми в компоненті або системі, що тестуються, в результаті допущених помилок, і для розроблення тестів спеціально для їх виявлення.

Помилково-орієнтоване тестування (failure-directed testing) – тестування програмного забезпечення, засноване на знанні типів помилок, допущених у минулому, які є ймовірними для системи, що тестується.

Евристична оцінка (heuristic evaluation) – метод перевірки зручності використання, спрямований на усунення проблем зручності використання в інтерфейсі користувача або дизайні інтерфейсу користувача.

@ М.В.Добролюбова 26

Детальна класифікація тестування

Класифікація за техніками та підходами

Мутаційне тестування (mutation testing) – тестування, при якому два або більше варіантів компонента або системи виконуються з однаковими вхідними даними, вихідні дані порівнюються та аналізуються у випадках розбіжностей.

Тестування додаванням помилок (error seeding) – процес навмисного додавання відомих помилок до тих, які вже є в комп'ютерній програмі, з метою моніторингу швидкості виявлення та видалення, а також оцінки кількості помилок, що залишилися в програмі.

Тестування на основі класів еквівалентності (equivalence partitioning) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання представників із розділів еквівалентності. У принципі тестові випадки розроблені таким чином, щоб охопити кожен розділ принаймні один раз.

Тестування на основі граничних умов (boundary value analysis) – метод розробки тесту чорної скриньки, в якому тестові приклади розробляються на основі граничних значень (вхідних значень або вихідних значень, які знаходяться на границі розділу еквівалентності або на найменшій відстані по обидва боки від границі, наприклад, мінімальне або максимальне значення діапазону).

Доменне тестування (domain analysis, domain testing) – метод розробки тесту чорної скриньки, який використовується для визначення ефективних та дієвих тестових випадків, коли кілька змінних можна або потрібно тестувати разом. Він спирається на розподіл еквівалентності та аналіз граничних значень і узагальнює його.

Попарне тестування (pairwise testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання всіх можливих дискретних комбінацій кожної пари вхідних параметрів.

N-комбінаційне тестування (n-wise testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання всіх можливих дискретних комбінацій будь-якого набору з n вхідних параметрів.

Тестування на основі ортогональних масивів (orthogonal array testing) – систематичний спосіб перевірки комбінацій усіх пар змінних за допомогою ортогональних масивів.

27

Детальна класифікація тестування

Класифікація за техніками та підходами

Тестування в процесі розробки (development testing) – формальне або неформальне тестування, що проводиться під час реалізації компонента або системи, зазвичай у середовищі розробки розробниками.

Тестування по потоку управління (control flow testing) – підхід до структурного тестування, в якому тестові випадки призначені для виконання певних послідовностей подій.

Тестування по потоку даних (data-flow testing) – техніка розробки тесту білої скриньки, в якій тестові випадки розроблені для виконання пар визначення-використання змінних.

Тестування за діаграмою або таблицею станів (state transition testing) – метод розробки тесту чорної скриньки, в якому тестові випадки розроблені для виконання припустимих і неприпустимих переходів між станами.

Діаграма станів (state diagram) – діаграма, яка зображує стани, які може приймати компонент або система, а також події або обставини, які викликають і/або є результатом переходу одного стану в інший.

Таблиця станів (state table) – сітка, що показує результуючі переходи для кожного стану в поєднанні з кожною можливою подією, показуючи як припустимі, так і неприпустимі переходи.

Кінцевий автомат (finite state machine) – обчислювальна модель, що складається з кінцевої кількості станів і переходів між цими станами, можливо, із супутніми діями.

Інспекція (аудит) коду (code review, code inspection) – тип експертної перевірки, яка заснована на візуальному огляді документів для виявлення дефектів, тобто порушення стандартів розробки та невідповідність документації вищого рівня.

Стаття «What is State transition testing in software testing?»

@ М.В.Добролюбова 28

Детальна класифікація тестування

Класифікація за техніками та підходами

Тестування на основі виразів (statement testing) – метод проектування тестів білої скриньки, в якому тестові випадки розроблені для виконання операторів.

Тестування на основі гілок (branch testing) – метод проектування тестів білої скриньки, в якому тестові приклади розроблені для виконання розгалужень.

Тестування на основі умов (condition testing) – метод проектування тестів білої скриньки, в якому тестові набори призначені для виконання умов.

Тестування на основі комбінацій умов (multiple condition testing) – метод проектування тестів білої скриньки, в якому тестові набори призначені для виконання комбінацій результатів однієї умови .

Тестування на основі окремих умов, що породжують розгалуження («вирішальних умов») (modified condition decision coverage testing) – техніка розробки тестових випадків для виконання результатів умов розгалуження, які незалежно впливають на результат рішення, і відкидання умов, які не впливають на кінцевий результат.

Тестування на основі рішень (decision testing) – метод проектування тестів білої скриньки, в якому тестові випадки призначені для виконання результатів прийняття рішень.

Тестування на основі шляхів (path testing) – метод проектування тестів білої скриньки, в якому тестові випадки призначені для виконання шляхів.

Замітка «What is the difference between a Decision and a Condition?»

@ М.В.Добролюбова 29

Детальна класифікація тестування

Класифікація за техніками та підходами

Види тестування на основі структур коду

Україномовна назва	Англомовна назва	Суть (що перевіряється)
Тестування на основі виразів	Statement testing	Окремі атомарні ділянки коду, наприклад `x = 10`
Тестування на основі гілок	Branch testing	Прохід по гілках виконання коду
Тестування на основі умов	Condition testing, Branch Condition Testing	Окремі умовні конструкції, наприклад `if (a == b)`
Тестування на основі комбінацій умов	Multiple condition testing, Branch Condition Combination Testing	Складові умовні конструкції, наприклад `if ((a == b) || (c == d))`
Тестування на основі окремих умов, що породжують розгалуження («вирішальних умов»)	Modified Condition Decision Coverage Testing	Окремі умови, що поодинці впливають на результат обчислення складної умови, наприклад, в умові `if ((x == y) && (n == m))` хибне значення в кожній з окремих умов саме собою призводить до результату `false` незалежно від результату обчислення другої умови
Тестування на основі рішень	Decision testing	Складні розгалуження, наприклад оператор `switch`
Тестування на основі шляхів	Path testing	Всі або спеціально вибрані шляхи

@ М.В.Добролюбова

30

Детальна класифікація тестування

Класифікація за техніками та підходами

Тестування за таблицею прийняття рішень (decision table testing) – метод проєктування тестів чорної скриньки, у якому тестові набори призначені для виконання комбінацій вхідних даних та/або стимулів (причин), показаних у таблиці рішень.

Тестування за моделями поведінки програми (model-based testing) – тестування на основі моделі компонента або системи, що тестується, наприклад, моделі зростання надійності, використання моделей, таких як робочі профілі або поведінкові моделі, такі як таблиця рішень або діаграма переходу станів.

Тестування на основі варіантів використання (use case testing) – метод проектування тестів чорної скриньки, у якому тестові варіанти розроблені для виконання сценаріїв варіантів використання.

Тестування на основі історії користувача (user story testing) – метод проектування тестів чорної скриньки, у якому тестові випадки розробляються на основі історій користувачів для перевірки їх правильної реалізації.

Паралельне тестування (parallel testing) – тестування нової або зміненої системи обробки даних з тими ж вхідними даними, які використовуються в іншій системі. Інша система розглядається як еталон порівняння.

Тестування на основі випадкових даних (random testing) – метод проектування тестів чорної скриньки, у якому тестові випадки вибираються, можливо, з використанням псевдовипадкового алгоритму генерації, щоб відповідати робочому профілю.

Операційний профіль (operational profile) – представлення окремого набору завдань, що виконується компонентом або системою, можливо, на основі поведінки користувача під час взаємодії з компонентом або системою та ймовірності їх виникнення.

«Мавпяче тестування» (monkey testing) – тестування за допомогою випадкового вибору з великого діапазону вхідних даних і шляхом випадкового натискання кнопок, не знаючи, як використовується продукт.

A/B-тестування, спліт-тестування (A/B testing, split testing) – це дизайн для встановлення причинно-наслідкового зв'язку між змінами та їх впливом на поведінку, яку спостерігає користувач.

«Controlled experiments on the web: survey and practical guide», Ron Kohavi

@ М.В.Добролюбова

31

Детальна класифікація тестування

Класифікація за моментом виконання (хронології)

Висхідне тестування (bottom-up testing) – поступовий підхід до інтеграційного тестування, коли спочатку тестуються компоненти нижчого рівня, а потім використовуються для полегшення тестування компонентів вищого рівня. Цей процес повторюється до тих пір, поки компонент у верхній частині ієрархії не буде перевірено.

Низхідне тестування (top-down testing) – інкрементальний підхід до інтеграційного тестування, коли компонент у верхній частині ієрархії компонентів тестується першим, а компоненти нижнього рівня моделюються заглушками. Перевірені компоненти потім використовуються для тестування компонентів нижнього рівня. Процес повторюється до тих пір, поки компоненти найнижчого рівня не будуть перевірені.

Гібридне тестування, сендвіч-тестування (hybrid testing) – це процес, в якому, по-перше, вхідні дані для функцій інтегруються у висхідному шаблоні. Потім вихідні дані для кожної функції інтегруються зверху вниз. Основною перевагою цього підходу є ступінь підтримки раннього випуску обмеженої функціональності.

Типові загальні сценарії

Типовий загальний сценарій 1
1) Димове тестування
2) Тестування критичного шляху
3) Розширене тестування

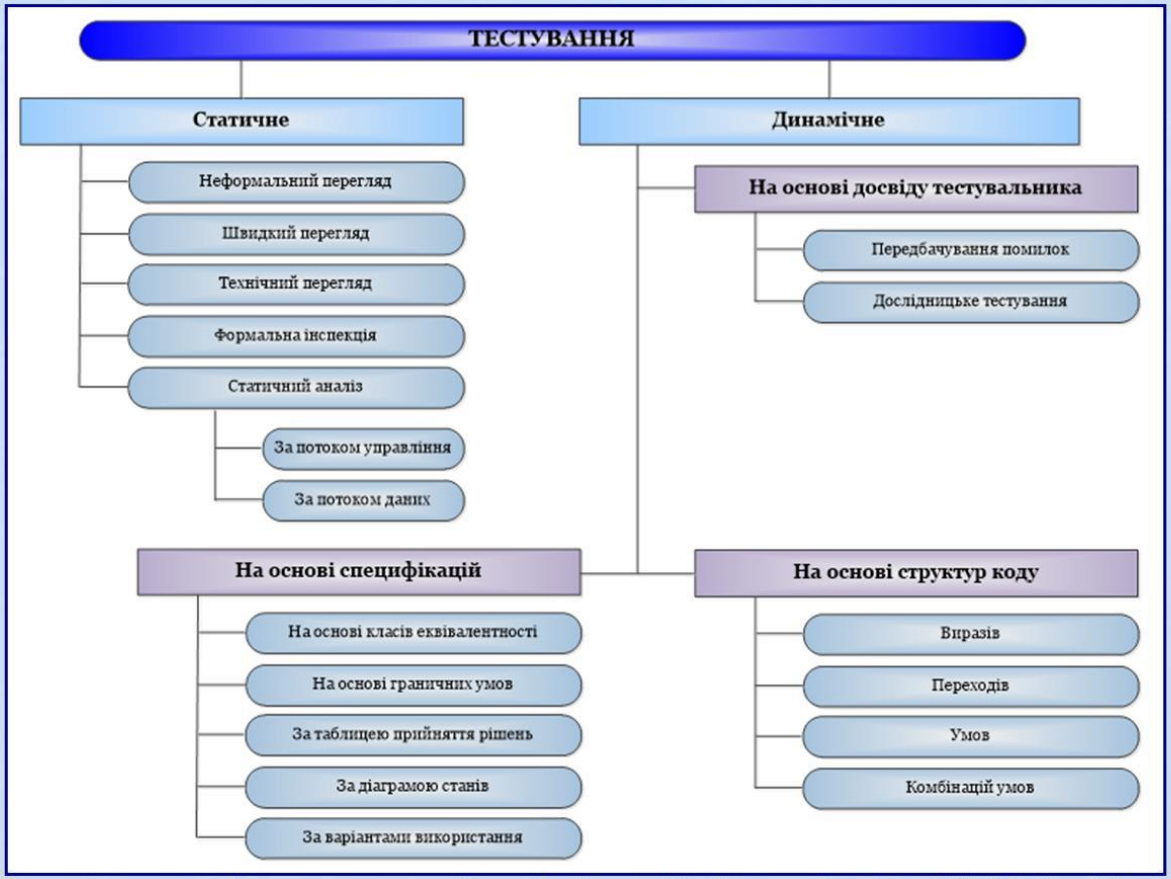
Типовий загальний сценарій 2
1) Модульне тестування
2) Інтеграційне тестування
3) Системне тестування

Типовий загальний сценарій 3
1) Альфа-тестування
2) Бета-тестування
3) Гамма-тестування6

@ М.В.Добролюбова

32

Альтернативні та додаткові класифікації тестування

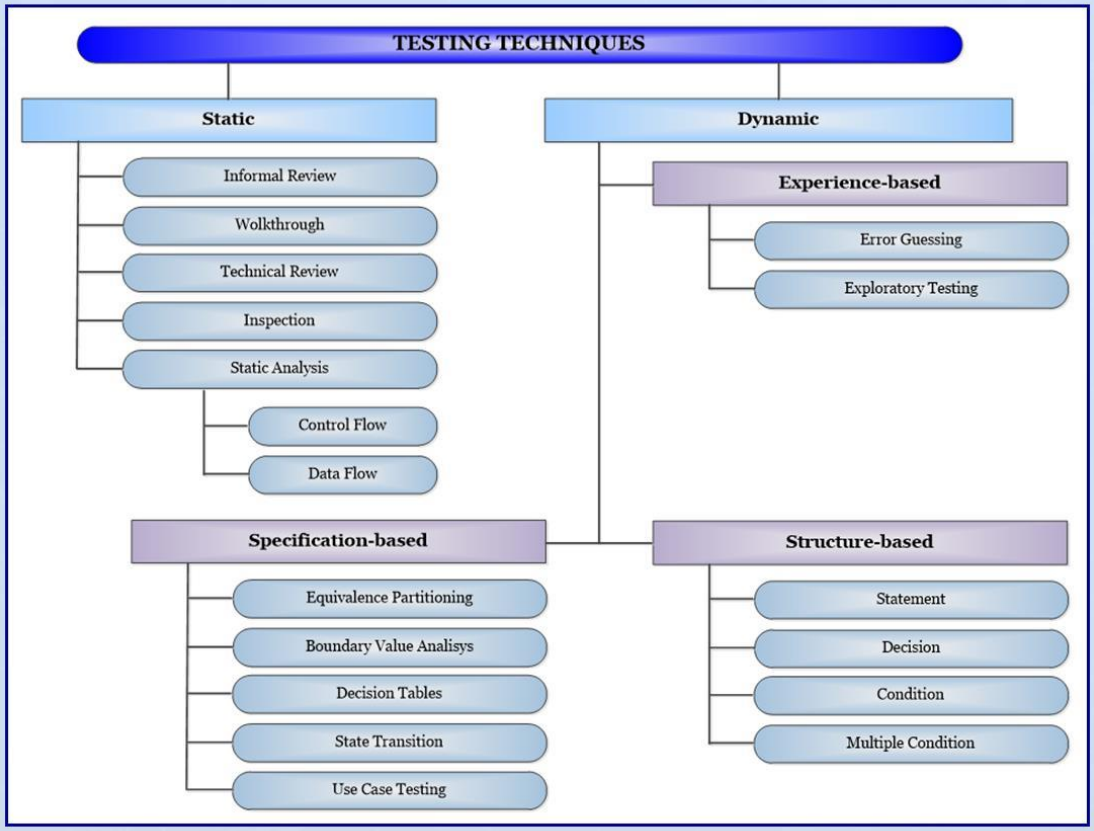


Класифікація тестування згідно з книгою «Foundations of Software Testing: ISTQB Certification» (Erik Van Veenendaal, Isabel Evans) (україномовний варіант)

@ М.В.Добролюбова

33

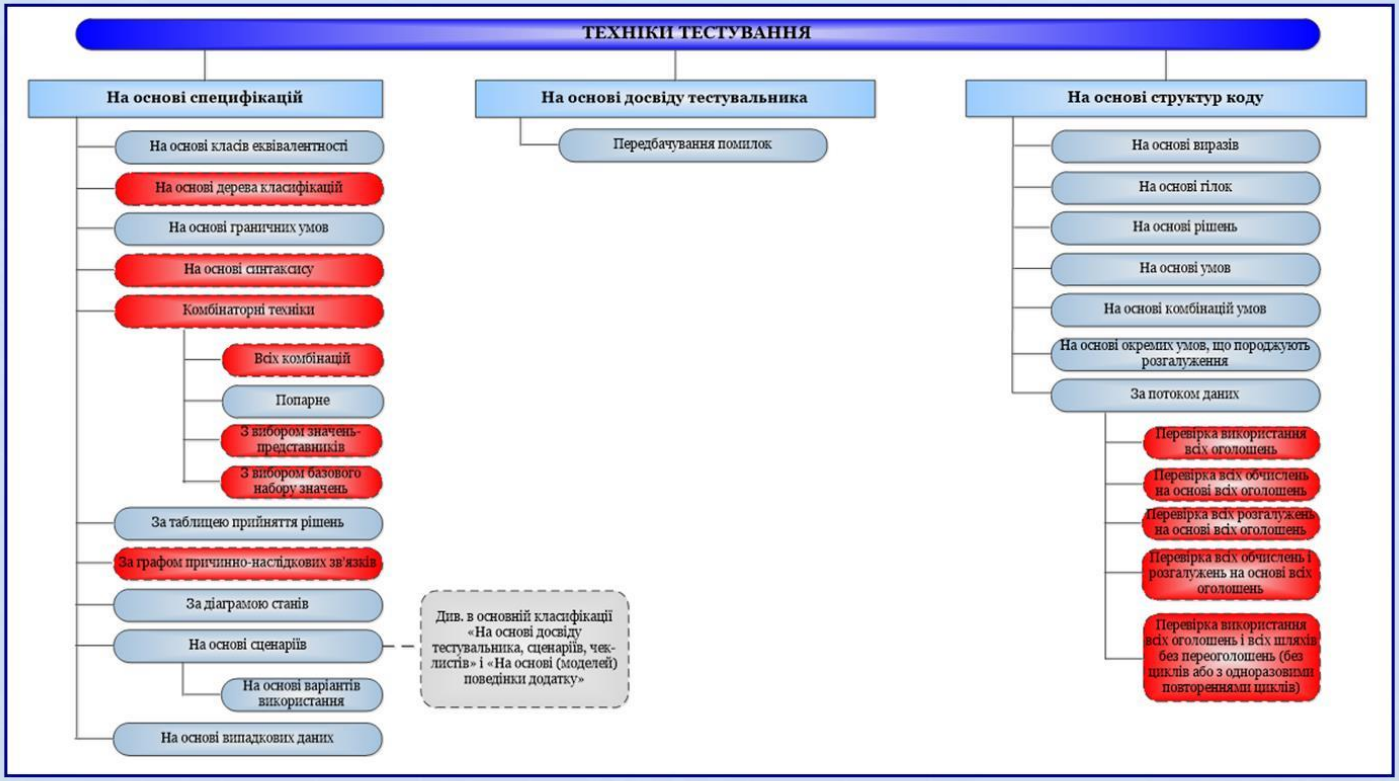
Альтернативні та додаткові класифікації тестування



Класифікація тестування згідно з книгою «Foundations of Software Testing: ISTQB Certification» (Erik Van Veenendaal, Isabel Evans) (англомовний варіант)

@ М.В.Добролюбова 34

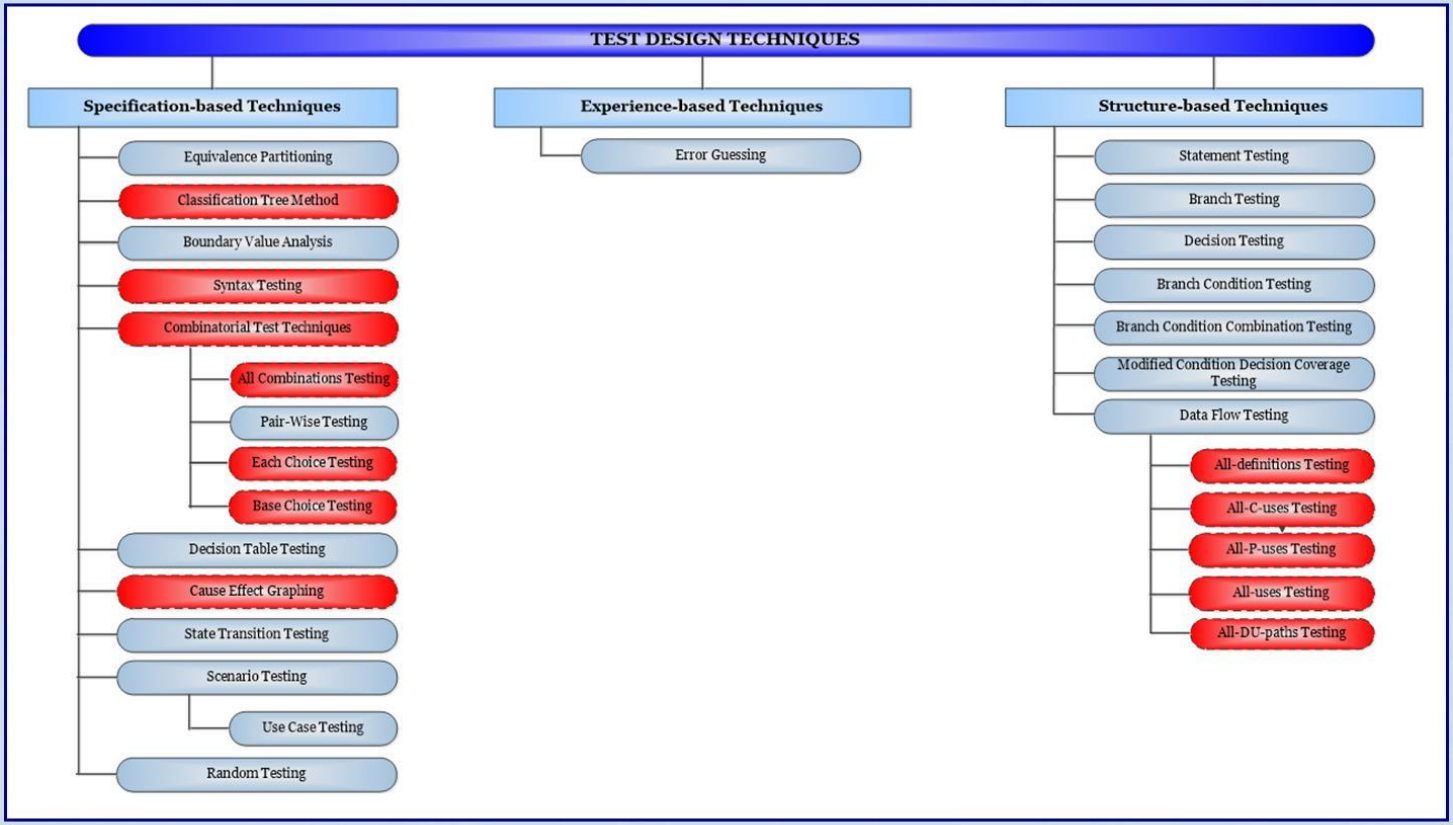
Альтернативні та додаткові класифікації тестування



Класифікація тестування згідно з ISO/IEC/IEEE 29119-4 (україномовний варіант)

@ М.В.Добролюбова 35

Альтернативні та додаткові класифікації тестування



Класифікація тестування згідно з ISO/IEC/IEEE 29119-4 (англомовний варіант)

@ М.В.Добролюбова 36

Альтернативні та додаткові класифікації тестування

Дерево класифікації (classification tree) – це дерево, що показує ієрархічно впорядковані розділи еквівалентності, яке використовується для розробки тестових випадків у методі тестування на основі дерева класифікацій.

Тестування на основі дерева класифікацій (classification tree method) – метод проектування тестів чорної скриньки, в якому тестові випадки, описані за допомогою дерева класифікації і призначені для виконання комбінацій представників вхідних та/або вихідних доменів.

Тестування на основі синтаксису (syntax testing) – метод проектування тестів чорної скриньки, в якому тестові випадки розробляються на основі визначення вхідної та/або вихідної області.

Комбінаторне тестування (combinatorial testing) – засіб для визначення відповідної підмножини тестових комбінацій для досягнення заздалегідь визначеного рівня охоплення під час тестування об'єкта з кількома параметрами, де кожен із цих параметрів має кілька значень, що призводить до більшої кількості комбінацій, ніж можливо перевірити протягом дозволеного часу.

Тестування всіх комбінацій (all combinations testing) – тестування всіх можливих комбінацій всіх значень для всіх параметрів.

Тестування з вибором значень-представників (each choice testing) – одне значення з кожного блоку для кожного розділу має використовуватися принаймні в одному тестовому випадку.

Тестування з вибором базового набору значень (base choice testing) – для кожного розділу вибирається блок базового вибору і базовий тест формується з використанням базового вибору для кожного розділу.

Тестування за графом причинно-наслідкових зв'язків (cause-effect graphing) – метод проектування тестів чорної скриньки, в якому тестові сценарії розробляються на основі причинно-наслідкових графів (графічне представлення вхідних даних та/або стимулів (причин) з відповідними вихідними даними (ефектами), які можна використовувати для розробки тестових сценаріїв).

Альтернативні та додаткові класифікації тестування

Тестування по потоку даних (data-flow testing) – метод проєктування тестів білої скриньки, в якому тестові випадки розроблені для виконання пар визначення-використання змінних.

Дії над змінною в загальному випадку

- оголошення (declaration): `int x;`
- визначення (definition, d-use): `x = 99;`
- використання в обчисленнях (computation use, c-use): `z = x + 1;`
- використання за умов (predicate use, p-use): `if (x > 17) { … };`
- вилучення (kill, k-use): `x = null;`

Техніки тестування на основі потоку даних

Стратегія з усіма визначеннями (all-definitions testing) – набір тестів вимагає, щоб кожне визначення кожної змінної охоплювалося принаймні одним використанням цієї змінної (c-use або p-use).

Стратегія використання всіх обчислень (all-c-uses testing) – для кожної змінної та кожного визначення цієї змінної включіть принаймні один шлях без визначення від визначення до кожного використання обчислень.

Стратегія використання всіх предикатів (all-p-uses testing) – для кожної змінної та кожного визначення цієї змінної включіть принаймні один шлях без визначення від визначення до кожного використання предиката.

Стратегія для всіх видів використання (all-uses testing) – тестовий набір включає принаймні один сегмент шляху від кожного визначення до кожного використання, яке може бути досягнуто цим визначенням.

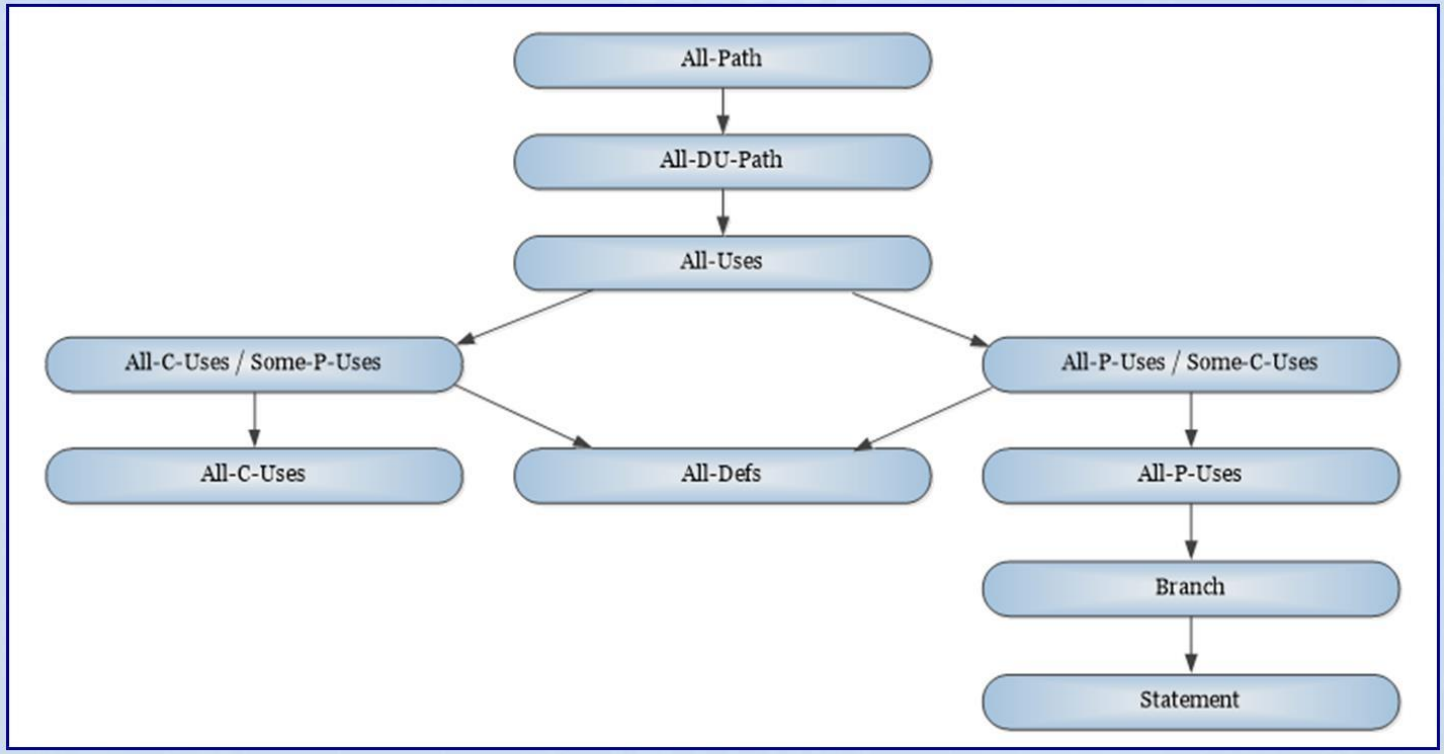
Стратегія All-DU-path (all-du-paths testing) – тестовий набір включає кожен шлях від кожного визначення кожної змінної до кожного використання цього визначення.

«Software Testing Techniques, Second Edition», Boris Beizer

@ М.В.Добролюбова

38

Альтернативні та додаткові класифікації тестування



Взаємозв'язок та відносна потужність стратегій тестування на основі потоку даних (за книгою Бориса Бейзера «Техніки тестування ПЗ»)

@ М.В.Добролюбова

39

Класифікація за приналежністю до тестування за методом білої та чорної скриньок

Вид тестування (україномовна назва)	Вид тестування (англомовна назва)	Біла скринька	Чорна скринька
Статичне тестування	Static testing	так	ні
Динамічне тестування	Dynamic testing	рідко	так
Ручне тестування	Manual testing	мало	так
Автоматизоване тестування	Automated testing	так	так
Модульне (компонентне) тестування	Unit testing, Module testing, Component testing	так	ні
Інтеграційне тестування	Integration testing	так	так
Системне тестування	System testing	мало	так
Димове тестування	Smoke test, Intake test, Build verification test	мало	так
Тестування критичного шляху	Critical path test	мало	так
Розширене тестування	Extended test	мало	так
Позитивне тестування	Positive testing	так	так
Негативне тестування	Negative testing, Invalid testing	так	так
Тестування веб-додатків	Web-applications testing	так	так
Тестування мобільних додатків	Mobile applications testing	так	так
Тестування настільних додатків	Desktop applications testing	так	так
Тестування рівня представлення	Presentation tier testing	мало	так
Тестування рівня бізнес-логіки	Business logic tier testing	так	так
Тестування рівня даних	Data tier testing	так	мало
Альфа-тестування	Alpha testing	мало	так
Бета-тестування	Beta testing	майже ніколи	так
Гама-тестування	Gamma testing	майже ніколи	так
Тестування на рівні тест-кейсів	Scripted testing, Test case based testing	так	так
Дослідницьке тестування	Exploratory testing	ні	так
Вільне (інтуїтивне) тестування	Ad hoc testing	ні	так
Функціональне тестування	Functional testing	так	так
Нефункціональне тестування	Non-functional testing	так	так
Інсталяційне тестування	Installation testing	рідко	так
Регресійне тестування	Regression testing	так	так
Повторне тестування	Re-testing, Confirmation testing	так	так
Приймальне тестування	Acceptance testing	дуже рідко	так
Операційне тестування	Operational testing	дуже рідко	так
Тестування зручності використання	Usability testing	дуже рідко	так
Тестування доступності	Accessibility testing	дуже рідко	так
Тестування інтерфейсу	Interface testing	так	так
Тестування безпеки	Security testing	так	так
Тестування інтернаціоналізації	Internationalization testing	мало	так
Тестування локалізації	Localization testing	мало	так

Вид тестування (україномовна назва)	Вид тестування (англомовна назва)	Біла скринька	Чорна скринька
Тестування сумісності	Compatibility testing	мало	так
Конфігураційне тестування	Configuration testing	мало	так
Крос-браузерне тестування	Cross-browser testing	мало	так
Тестування даних і баз даних	Data quality testing and Database integrity testing	так	мало
Тестування використання ресурсів	Resource utilization testing	дуже рідко	так
Порівняльне тестування	Comparison testing	ні	так
Демонстраційне тестування	Qualification testing	ні	так
Вичерпне тестування	Exhaustive testing	дуже рідко	ні
Тестування надійності	Reliability testing	дуже рідко	так
Тестування відновлюваності	Recoverability testing	дуже рідко	так
Тестування відмовостійкості	Failover testing	дуже рідко	так
Тестування продуктивності	Performance testing	дуже рідко	так
Навантажувальне тестування	Load testing, Capacity testing	дуже рідко	так
Тестування масштабуємості	Scalability testing	дуже рідко	так
Об'ємне тестування	Volume testing	дуже рідко	так
Стресове тестування	Stress testing	дуже рідко	так
Конкурентне тестування	Concurrency testing	дуже рідко	так
Інвазивне тестування	Intrusive testing	так	так
Неінвазивне тестування	Nonintrusive testing	так	так
Тестування під управлінням даними	Data-driven testing	так	так
Тестування під управлінням ключовими словами	Keyword-driven testing	так	так
Тестування передбачуванням помилок	Error guessing	дуже рідко	так
Евристична оцінка	Heuristic evaluation	ні	так
Мутаційне оцінювання	Mutation testing	так	так
Тестування додаванням помилок	Error seeding	так	так
Тестування на основі класів еквівалентності	Equivalence partitioning	так	так
Тестування на основі граничних умов	Boundary value analysis	так	так
Доменне тестування	Domain testing, Domain analysis	так	так
Попарне тестування	Pairwise testing	так	так
Тестування на основі ортогональних масивів	Orthogonal array testing	так	так
Тестування в процесі розробки	Development testing	так	так
Тестування по потоку управління	Control flow testing	так	ні
Тестування по потоку даних	Data flow testing	так	ні
Тестування по діаграмі або таблиці станів	State transition testing	рідко	так
Інспекція (аудит) коду	Code review, code inspection	так	ні
Тестування на основі виразів	Statement testing	так	ні
Тестування на основі гілок	Branch testing	так	ні

@ М.В.Добролюбова

40

Класифікація за приналежністю до тестування за методом білої та чорної скриньок

Вид тестування (українською мовою)	Вид тестування (англійською мовою)	Біла скринька	Чорна скринька
Тестування на основі умов	Condition testing	так	ні
Тестування на основі комбінацій умов	Multiple condition testing	так	ні
Тестування на основі вирішення умов, що передують розгалуженню (=більш-чим умов)	Modified condition decision coverage testing	так	ні
Тестування на основі рішень	Decision testing	так	ні
Тестування на основі шляхів	Path testing	так	ні
Тестування на таблиці прийняття рішень	Decision table testing	так	так
Тестування на моделях поведінки додатку	Model-based testing	так	так
Тестування на основі варіантів використання	Use case testing	так	так
Паралельне тестування	Parallel testing	так	так
Тестування на основі випадкових даних	Random testing	так	так
A/B-тестування	A/B testing, Split testing	ні	так
Висхідне тестування	Bottom-up testing	так	так
Низхідне тестування	Top-down testing	так	так
Гібридне тестування	Hybrid testing	так	так
Тестування на основі дерева класифікацій	Classification tree method	так	так
Тестування на основі синтаксису	Syntax testing	так	так
Комбінаторне тестування (комбінаторне тестування)	Combinatorial testing	так	так
Тестування всіх комбінацій	All combinations testing	так	ні
Тестування з вибором значень представників	Each choice testing	так	ні
Тестування з вибором базового набору значень	Base choice testing	так	ні
Тестування по графу причинно-наслідкових зв'язків	Cause-effect graphing	ні/ні	так
Перевірка використання всіх оголошень	All-definitions testing	так	ні
Перевірка всіх обчислень на основі всіх оголошень	All-c-uses testing	так	ні
Перевірка всіх розгалужень на основі всіх оголошень	All-p-uses testing	так	ні
Перевірка всіх обчислень і розгалужень на основі всіх оголошень	All-uses testing	так	ні
Перевірка використання всіх оголошень і всіх шляхів без циклів або з одноразовим повторенням циклів	All-du-paths testing	так	ні

@ М.В.Добролюбова

41


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 2. КЛАСИФІКАЦІЯ ТА ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ
Лекція 5
«Чек-лист, тест-кейси, набори тест-кейсів»
@ М.В.Добролюбова

Чек-лист

Чек-лист (checklist) – набір ідей [тест-кейсів]. Поняття «чек-листа» не зав'язане на тестуванні як такому – це універсальна техніка, яка може застосовуватися у будь-якій без винятку області життя. В українській мові поза контекстом інформаційних технологій частіше використовується зрозуміле і звичне слово «список», але в тестуванні прижилася калькована з англійської версія – «чек-лист».

Чек-лист – список:

- у якому послідовність пунктів не має значення
- у якому послідовність пунктів важлива
- структурований (багаторівневий) список, який дозволяє відобразити ієрархію ідей

Випадки, в яких до чек-листа можуть додаватись очікувані результати:

- у певному пункті чек-листа розглядається особлива, нетривіальна поведінка програми або складна перевірка, результат якої важливо відзначити вже зараз, щоби не забути
- в силу стислих термінів та/або нестачі інших ресурсів тестування проводиться безпосередньо за чек-листами без тест-кейсів

@ М.В.Добролюбова

2

Чек-лист

Властивості чек-листа:

- логічність
- послідовність та структурованість
- повнота та надмірність

Типові варіанти логіки створення окремих чек-листів для:

- типових сценаріїв користувача
- різних рівнів функціонального тестування
- окремих частин (модулів та підмодулів) додатку
- окремих вимог, груп вимог, рівнів та типів вимог
- частин або функцій програми, найбільш схильних до ризиків

Логіка розбиття функцій програми за ступенем їх важливості
(на конкретному прикладі проєкту із кодовою назвою «Конвертер файлів»):

- базові функції
- функції, затребувані більшістю користувачів у їх повсякденній роботі
- інші функції

@ М.В.Добролюбова 3

Чек-лист

Функції, без яких існування додатку втрачає сенс

Ключові функції додатку:

- конфігурування та запуск
- обробка файлів
- зупинка

Функції, затребувані більшістю користувачів

Функції, необхідні користувачам:

Конфігурування та запуск:

- з правильними параметрами
- без параметрів
- з недостатньою кількістю параметрів
- з неправильними параметрами

Обробка файлів:

- різні формати, кодування та розміри
- недоступні вхідні файли:
  - немає прав доступу
  - файл відкритий та заблокований
  - файл із атрибутом лише для читання

Зупинка:

- закриттям вікна консолі

Продуктивність:

- елементарний тест із грубою оцінкою

Журнал роботи програми:

- автоматичне створення (за відсутності журналу)
- продовження (доповнення журналу) при повторних запусках

Інші функції та особливі сценарії

Інші функції для запобігання проблем:

Конфігурування та запуск;

Обробка файлів:

@ М.В.Добролюбова 4

Тест-кейс та його життєвий цикл

Тест (test) – це набір з одного або кількох тестових випадків.

Тест-кейс (test case) – це набір вхідних значень, передумов виконання, очікуваних результатів і постумов виконання, розроблених для конкретної мети або умови тестування, наприклад, для виконання певного шляху програми або для перевірки відповідності конкретній вимозі.

Високорівневий тест-кейс, логічний тест-кейс (high level test case) – це тест-кейс без конкретних (рівень реалізації) значень для вхідних даних та очікуваних результатів. Використовуються логічні оператори; екземпляри фактичних значень ще не визначені та/або не доступні.

Низькорівневий тест-кейс (low level test case) – це тест-кейс із конкретними (рівень реалізації) значеннями вхідних даних та очікуваними результатами. Логічні оператори з тест-кейсів високого рівня замінюються фактичними значеннями, які відповідають цілям логічних операторів.

Специфікація тест-кейсу (test case specification) – це документ, що визначає набір тест-кейсів (мета, вхідні дані, тестові дії, очікувані результати та попередні умови виконання) для тестового завдання.

Тестовий елемент (test item) – це окремий елемент, що підлягає перевірці. Зазвичай існує один тестовий об'єкт і багато тестових елементів.

Тестовий об'єкт (test object) – це компонент або система, що підлягають перевірці.

Тест-кейс та його життєвий цикл

Специфікація тесту (test specification) – це документ, який складається зі специфікації проєкту тестування, специфікації тест-кейсу та/або специфікації процедури тестування.

Специфікація тест-дизайну (test design specification) – це документ, що визначає умови тестування (елементи покриття) для тестового елемента, детальний підхід до тестування і ідентифікує відповідні тест-кейси високого рівня.

Специфікація тест-процедури, процедура тестування (test procedure specification) – це документ, що визначає послідовність дій для виконання тесту. Також відомий як тестовий сценарій або сценарій ручного тестування.

Тестовий сценарій (test scenario, test procedure specification, test script) – це документ, що визначає послідовність дій для виконання тесту. Також відомий сценарій ручного тестування.

Тестове покриття (test coverage metrics) – це ступінь, виражена у відсотках, в якій певний об'єкт покриття (сутність або властивість), що використовується як основа для тестового покриття був реалізований набором тестів.

@ М.В.Добролюбова 6

Тест-кейс та його життєвий цикл

Мета написання тест-кейсів

Наявність тест-кейсів дозволяє:

- структурувати та систематизувати підхід до тестування
- обчислювати метрики тестового покриття та вживати заходів щодо його збільшення
- відслідковувати відповідність поточної ситуації плану
- уточнити взаєморозуміння між замовником, розробниками та тестувальниками
- зберігати інформацію для тривалого використання та обміну досвідом між співробітниками та командами
- проводити регресійне та повторне тестування
- підвищувати якість вимог
- швидко вводити в курс справи нового співробітника, який нещодавно підключився до проєкту

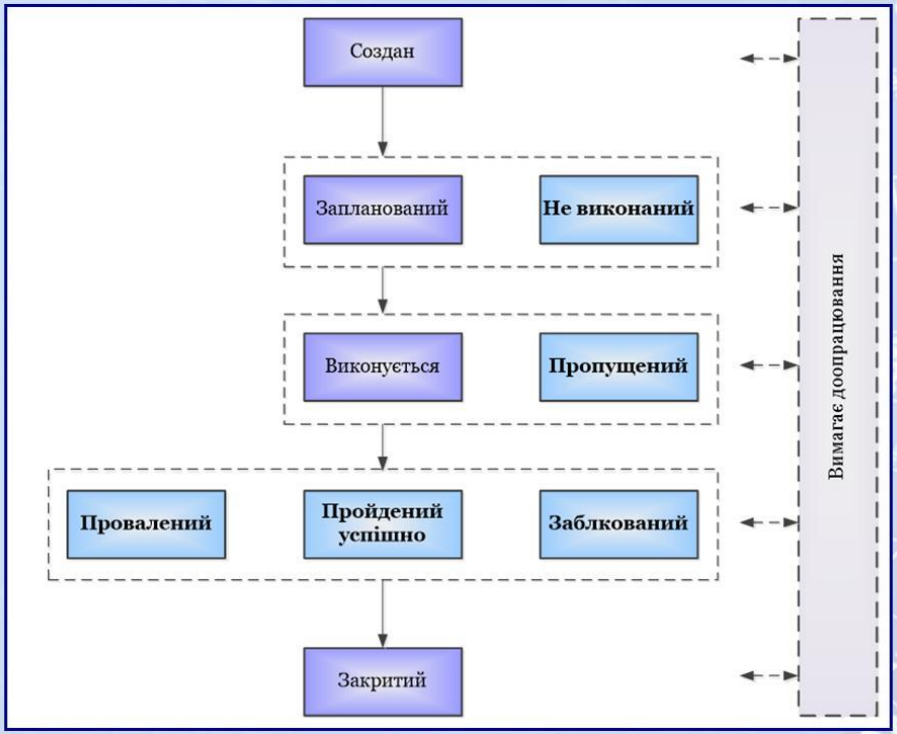
Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Area should accommodate up to 20 characters	Input up to 20 characters	All 20 characters in the request should be appropriate	Pass or Fail
Security	Verify password rules are working	Create a new password in accordance with rules	The user's password will be accepted if it adheres to the rules	Pass or Fail
Usability	Ensure all links are working properly	Have users click on various links on the page	Links will take users to another web page according to the on-page URL	Pass or Fail

@ М.В.Добролюбова 7

Тест-кейс та його життєвий цикл

Життєвий цикл або набір станів тест-кейсу:

- створено (new)
- запланований (planned, ready for testing)
- **не виконано (not tested)**
- виконується (work in progress)
- **пропущений (skipped)**
- **провалений (failed)**
- **пройдений успішно (passed)**
- **заблокований (blocked)**
- закритий (closed)
- вимагає доопрацювання (not ready)

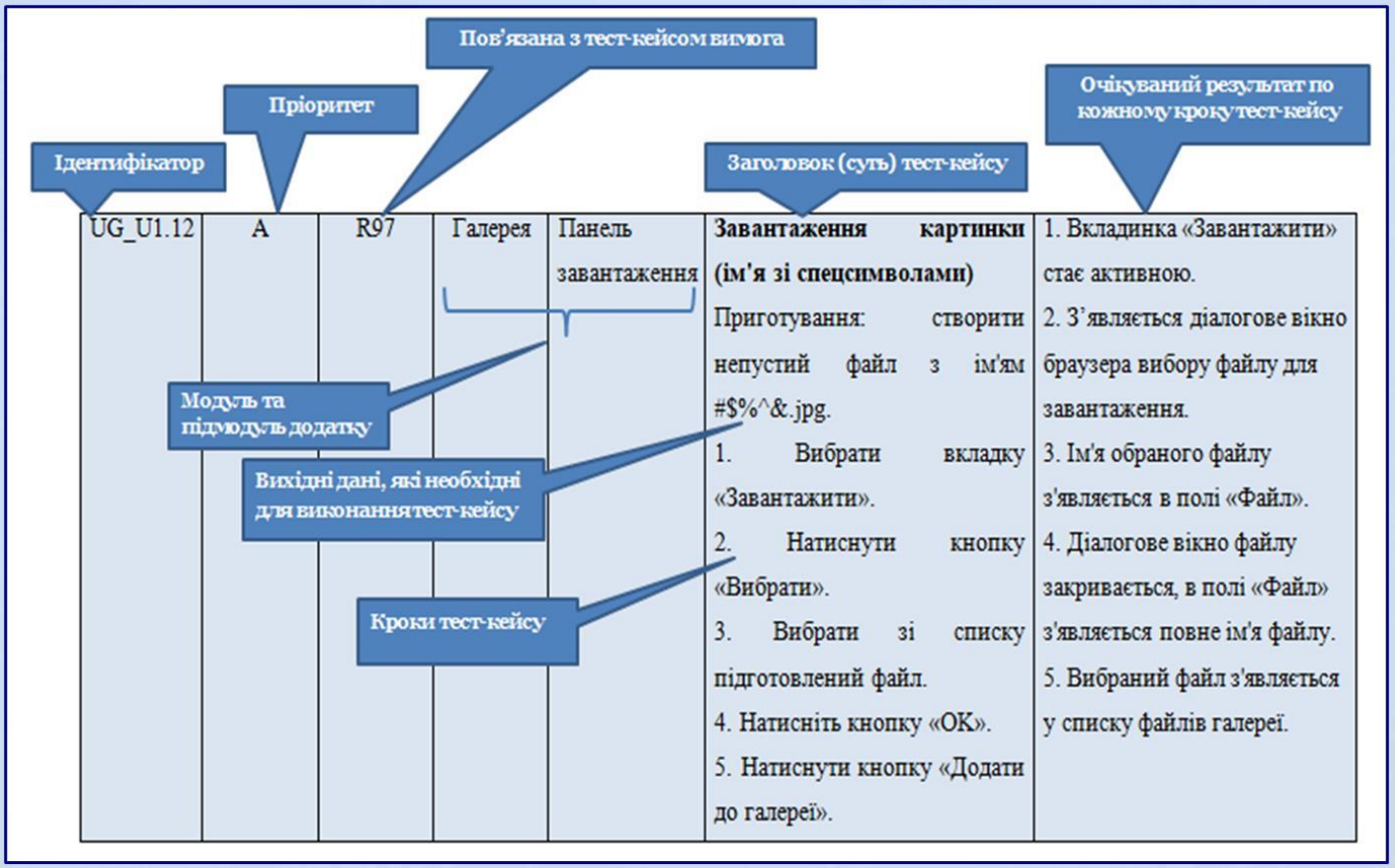


Життєвий цикл (набір станів) тест-кейсу

@ М.В.Добролюбова

8

Атрибути (поля) тест-кейсу



UG_U1.12	A	R97	Галерея	Панель завантаження	**Завантаження картинки (ім'я зі спецсимволами)** Приготування: створити непустий файл з ім'ям #$%^&.jpg. 1. Вибрати вкладку «Завантажити». 2. Натиснути кнопку «Вибрати». 3. Вибрати зі списку підготовлений файл. 4. Натисніть кнопку «ОК». 5. Натиснути кнопку «Додати до галереї».	1. Вкладинка «Завантажити» стає активною. 2. З'являється діалогове вікно браузера вибору файлу для завантаження. 3. Ім'я обраного файлу з'являється в полі «Файл». 4. Діалогове вікно файлу закривається, в полі «Файл» з'являється повне ім'я файлу. 5. Вибраний файл з'являється у списку файлів галереї.

Загальний вигляд тест-кейсу

@ М.В.Добролюбова

9

Атрибути (поля) тест-кейсу

Ідентифікатор (identifier) – це атрибут тест-кейсу, що є унікальним значенням, що дозволяє однозначно відрізнити один тест-кейс від іншого і використовується у різноманітних посиланнях.

Пріоритет – це атрибут тест-кейсу, що показує важливість тест-кейсу.

Пов'язана з тест-кейсом вимога (requirement) – це атрибут тест-кейсу, що показує ту основну вимогу, перевірці виконання якої присвячено тест-кейс.

Модуль та підмодуль додатку (module and submodule) – це атрибути тест-кейсу, що вказують на частини додатку, до яких належить тест-кейс, і дозволяють краще зрозуміти його ціль.

Заголовок (суть) тест-кейсу (title) – це атрибут тест-кейсу, що покликаний спростити та прискорити розуміння основної ідеї (мети) тест-кейсу без звернення до його інших атрибутів.

Погано	Добре
Тест 1	Запуск, одна копія, правильні параметри
Тест 2	Запуск однієї копії з неправильними шляхами
Тест 78 (покращений)	Запуск, багато копій, без конфліктів
Зупинка	Зупинка по Ctrl+C
Закриття	Зупинка закриттям консолі
…	…

Вихідні дані, необхідні для виконання тест-кейсу (precondition, preparation, initial data, setup) – це атрибут тест-кейсу, який дозволяє описати все те, що повинно бути підготовлено до початку виконання тест-кейсу.

Кроки тест-кейсу (steps) – це атрибут тест-кейсу, що описує послідовність дій, які необхідно реалізувати у процесі виконання тест-кейсу.

Очікувані результати (expected results) по кожному кроку тест-кейсу – це атрибут тест-кейсу, що демонструє реакцію програми на дії, описані в полі «кроки тест-кейсу».

Інструментальні засоби управління тестуванням

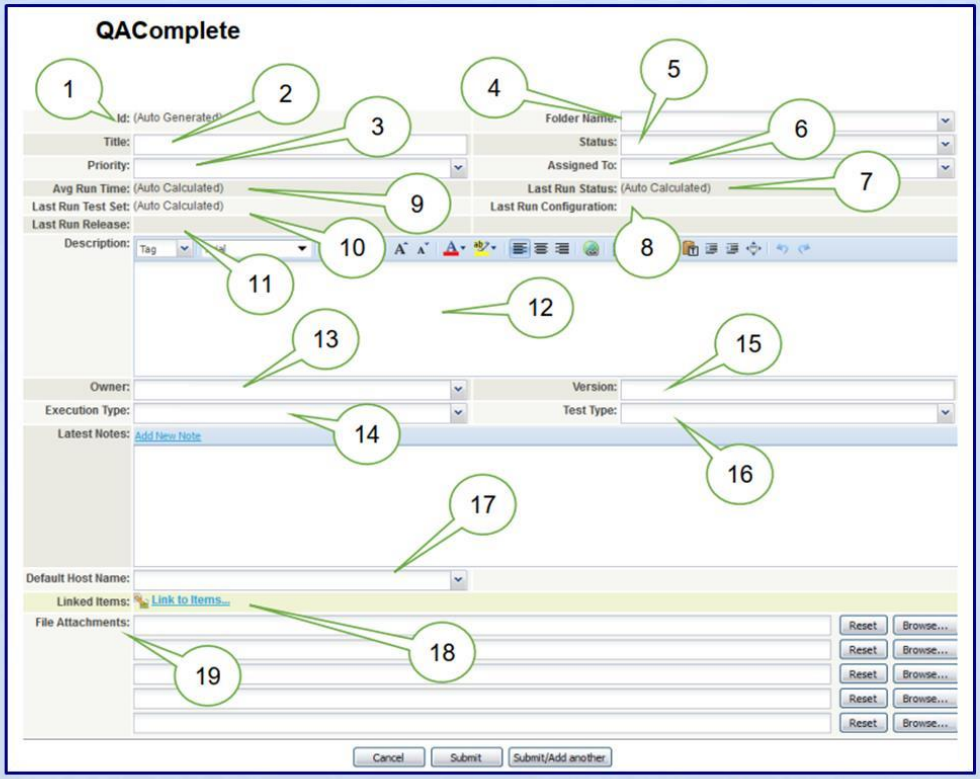
Інструментальні засоби керування тестуванням (test management tool) – це інструмент, який забезпечує підтримку управління тестуванням та контроль частини процесу тестування. Він часто має кілька можливостей, таких як керування тестовим програмним забезпеченням, планування тестів, реєстрація результатів, відстеження прогресу, керування інцидентами та звітування про тестування.

Загальний набір функцій, що реалізуються інструментальними засобами керування тестуванням:

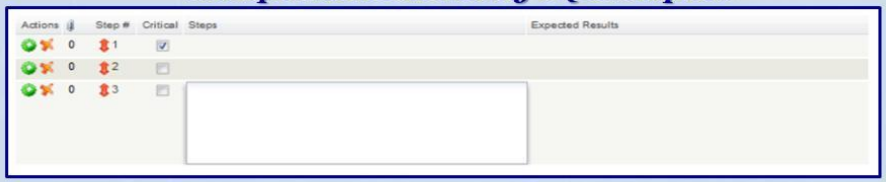
- створення тест-кейсів та наборів тест-кейсів
- контроль версій документів з можливістю визначити, хто вніс ті чи інші зміни, та скасувати ці зміни, якщо потрібно
- формування та відстеження реалізації плану тестування, збирання та візуалізації різноманітних метрик, генерування звітів
- інтеграція із системами управління дефектами, фіксація взаємозв'язку між виконанням тест-кейсів та створеними звітами про дефекти
- інтеграція із системами управління проєктами
- інтеграція з інструментами автоматизованого тестування, керування виконанням автоматизованих тест-кейсів

@ М.В.Добролюбова 11

Інструментальні засоби управління тестуванням



Створення тест-кейсу в QAComplete

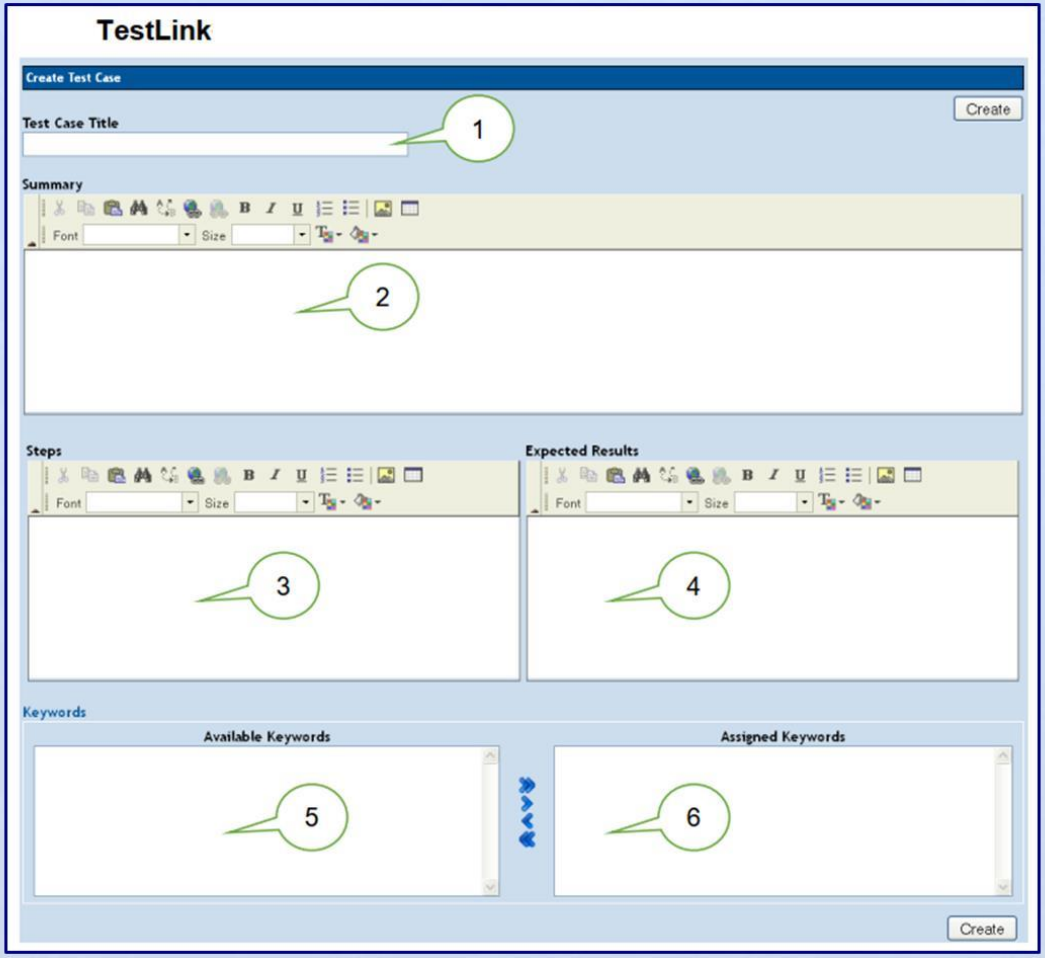


Додавання кроків тест-кейсу в QAComplete

@ М.В.Добролюбова

12

Інструментальні засоби управління тестуванням

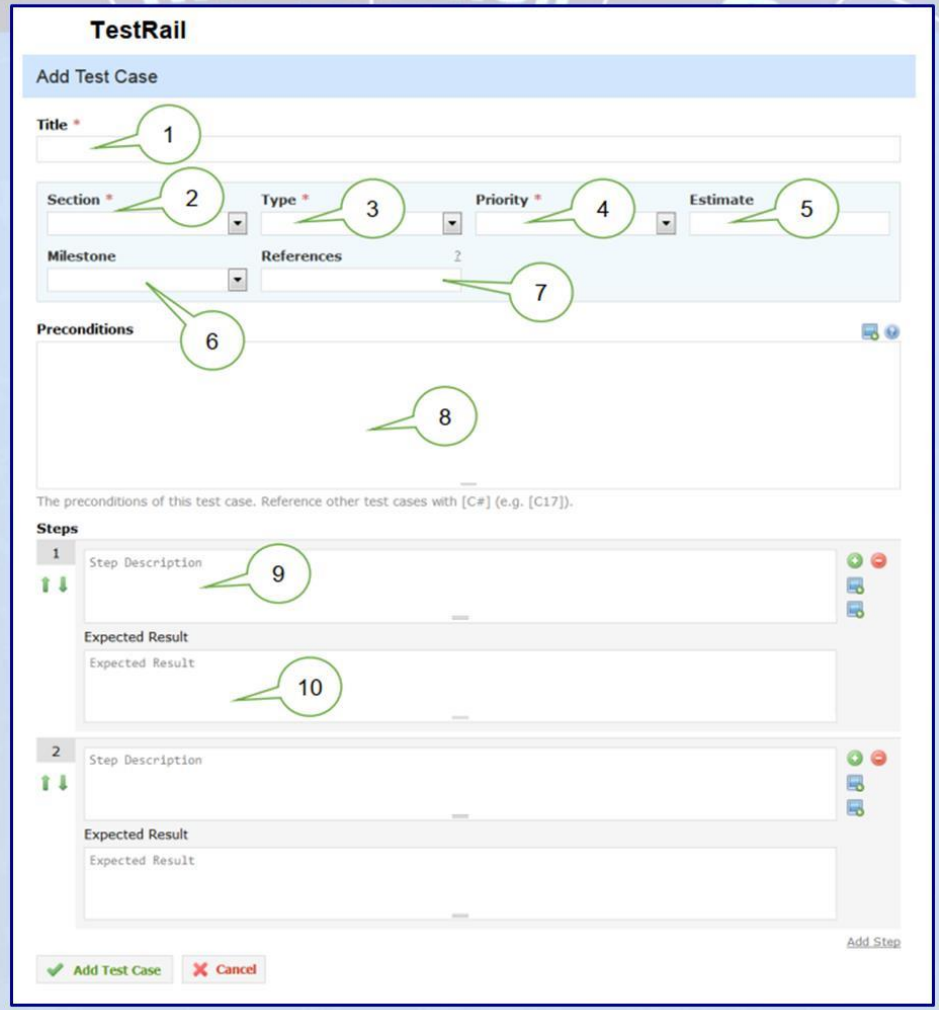


Створення тест-кейсу в TestLink

@ М.В.Добролюбова

13

Інструментальні засоби управління



Створення тест-кейсу в TestRail

@ М.В.Добролюбова

14

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

1. Правильна технічна мова, точність та одноманітність формулювань.
2. Баланс між специфічністю та загальністю.

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: • Створити папки C:/A, C:/B, C:/C, C:/D. • Розмістити в папці C:/D файли 1.html, 2.txt, 3.md з архіву, що додається. 1. Запустити програму, виконавши команду «php converter.php c:/a c:/b c:/c/converter.log». 2. Скопіювати файли 1.html, 2.txt, 3.md з папки C:/D до папки C:/A. 3. Зупинити програму натисканням Ctrl+C.	1. Відображається консольний журнал програми з повідомленням «поточний час started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log», у папці C:/C з'являється файл converter.log, в якому з'являється запис «поточний час started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log». 2. Файли 1.html, 2.txt, 3.md з'являються у папці C:/A, потім зникають і з'являються у папці C:/B. У консольному журналі та файлі C:/C/converter.log з'являються повідомлення (записи) «поточний_час processing 1.html (KOI8-R)», «поточний_час processing 2.txt (CP-1251)», «поточний_час processing 3.md (CP-866)». 3. У файлі C:/C/converter.log з'являється запис «поточний час closed». Додаток завершує роботу.

Тест-кейс 1

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань 1. Виконати конвертацію трьох файлів допустимого розміру в трьох різних кодуваннях всіх трьох допустимих форматів.	1. Файли переміщуються до папки-приймача, кодування всіх файлів змінюється на UTF-8.

Тест-кейс 2

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: • Створити в корені будь-якого диска чотири окремі папки для вхідних файлів, вихідних файлів, файлу журналу та тимчасового збереження тестових файлів. • Розпакувати вміст архіву, що додається, в папку для тимчасового зберігання тестових файлів. 1. Запустити програму, вказавши в параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіювати файли з папки для тимчасового зберігання до папки для вхідних файлів. 3. Зупинити додаток.	1. Додаток запускається та виводить повідомлення про свій запуск у консоль та файл журналу. 2. Файли з папки для вхідних файлів переміщуються до папки для вихідних файлів, в консолі та файлі журналу відображаються повідомлення про конвертацію кожного з файлів із зазначенням його вихідного кодування. 3. Додаток виводить повідомлення про завершення роботи у файл журналу та завершує роботу.

Тест-кейс 3

@ М.В.Добролюбова

15

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

3. Баланс між простотою та складністю.

Переваги простих тест-кейсів:

- їх можна швидко прочитати, легко зрозуміти та виконати
- вони зрозумілі початківцям-тестувальникам та новим людям на проекті
- вони роблять наявність помилки очевидною (як правило, у них передбачається виконання повсякденних тривіальних дій, проблеми з якими видно неозброєним поглядом і викликають дискусії)
- вони полегшують початкову діагностику помилки, оскільки звужують коло пошуку

Переваги складних тест-кейсів:

- при взаємодії багатьох об'єктів підвищується ймовірність виникнення помилки
- користувачі, як правило, використовують складні сценарії, тому складні тести повноцінніше емулюють роботу користувачів
- програмісти рідко перевіряють такі складні випадки (і вони зовсім не повинні це робити)

@ М.В.Добролюбова 16

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

Занадто простий тест-кейс:

Кроки	Очікувані результати
Запуск програми 1. Запустити програму.	1. Програма запускається.

Занадто складний тест-кейс:

Кроки	Очікувані результати
Повторна конвертація Приготування: • Створити в корені будь-якого диска три окремі папки для вхідних файлів, вихідних файлів, файл журналу. • Підготувати набір з кількох файлів максимального підтримуваного розміру підтримуваних форматів з підтримуваними кодуваннями, а також декількох файлів допустимого розміру, але неприпустимого формату. 1. Запустити програму, вказавши в параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіювати до папки для вхідних файлів кілька файлів допустимого формату. 3. Перемістити сконвертовані файли з папки з результуючими файлами до папки для вхідних файлів 4. Перемістити сконвертовані файли з папки з результуючими файлами до папки з набором файлів для тесту. 5. Перемістити всі файли з папки з набором файлів для тесту до папки для вхідних файлів. 6. Перемістити сконвертовані файли з папки з результуючими файлами до папки для вхідних файлів.	2. Файли поступово переміщуються з вхідної у вихідну папку, в консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів. 3. Файли поступово переміщуються з вхідної у вихідну папку, в консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів. 5. Файли поступово переміщуються з вхідної у вихідну папку, в консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів допустимого формату та повідомлення про ігнорування файлів неприпустимого формату. 6. Файли поступово переміщуються з вхідної у вихідну папку, в консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів допустимого формату та повідомлення про ігнорування файлів неприпустимого формату.

Приклад гарного складного тест-кейсу:

Кроки	Очікувані результати
Багато копій додатку, конфлікт файлових операцій Приготування: • Створити в корені будь-якого диска три окремі папки для вхідних файлів, вихідних файлів, файл журналу. • Підготувати набір з декількох файлів максимального підтримуваного розміру підтримуваних форматів з підтримуваним кодуванням. 1. Запустити першу копію додатка, вказавши у параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Запустити другу копію програми з тими самими параметрами (див. крок 1). 3. Запустити третю копію програми з тими самими параметрами (див. крок 1). 4. Змінити пріоритет процесів другої ("high") та третьої ("low") копій. 5. Скопіювати підготовлений набір вихідних файлів до папки для вхідних файлів.	3. Всі три копії додатку запускаються, в файлі журналу з'являються послідовно три записи про запуск програми. 5. Файли поступово переміщуються з вхідної у вихідну папку, в консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів, а також (можливо) повідомлення виду: a. "source file inaccessible, retrying". b. "destination file inaccessible, retrying". c. "log file inaccessible, retrying". Ключовим показником коректної роботи є успішна конвертація всіх файлів, а також поява в консолі та файлі журналу повідомлень про успішну конвертацію кожного файлу (від одного до трьох записів на кожен файл). Повідомлення (попередження) про недоступність вхідного файлу, вихідного файлу або файлу журналу також є показником коректної роботи програми, проте їх кількість залежить від багатьох зовнішніх чинників і може бути спрогнозована заздалегідь.

@ М.В.Добролюбова

17

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

4. «Показовість» (висока ймовірність виявлення помилки).

Приклад непоказового (поганого) тест-кейсу:

Кроки	Очікувані результати
Запуск та зупинка додатку 1. Запустити програму з коректними параметрами. 2. Завершити роботу додатка.	1. Додаток запускається. 2. Додаток завершує роботу.

Приклад показового (хорошого) тест-кейсу:

Кроки	Очікувані результати
Запуск із некоректними параметрами, неіснуючі шляхи 1. Запустити додаток з усіма трьома параметрами (SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME), значення яких вказують на неіснуючі в файловій системи шляхи (наприклад: z:\src\, z:\dst\, z:\log.txt за умови, що в системі немає логічного диска z).	1. В консолі відображаються наведені нижче повідомлення, програма завершує роботу. Повідомлення: a. Повідомлення про використання. b. SOURCE_DIR [z:\src\]: directory not exists or inaccessible. c. DESTINATION_DIR [z:\dst\]: directory not exists or inaccessible. d. LOG_FILE_NAME [z:\log.txt]: wrong file name or inaccessible path.

@ М.В.Добролюбова

18

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

5. **Послідовність у досягненні мети.**

Приклади правильної реалізації цієї якості - це приклади хороших тест-кейсів. Порушення може виглядати так:

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: • Створити в корені будь-якого диска чотири окремі папки для вхідних файлів, вихідних файлів, файлу журналу та тимчасового збереження тестових файлів. • Розпакувати вміст архіву, що додається, в папку для тимчасового зберігання тестових файлів. 1. Запустити програму, вказавши в параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіювати файли з папки для тимчасового зберігання до папки для вхідних файлів. 3. Зупинити додаток. 4. Видалити файл журналу. 5. Повторно запустити додаток з тими ж параметрами. 6. Зупинити додаток.	1. Програма запускається та виводить повідомлення про свій запуск у консоль та файл журналу. 2. Файли з папки для вхідних файлів переміщуються до папки для вихідних файлів, в консолі та файлі журналу відображаються повідомлення про конвертацію кожного файлу із зазначенням його вихідного кодування. 3. Додаток виводить повідомлення про завершення роботи у файл журналу та завершує роботу. 5. Додаток запускається та виводить повідомлення про свій запуск у консоль та заново створений файл журналу. 6. Додаток виводить повідомлення про завершення роботи у файл журналу та завершує роботу.

@ М.В.Добролюбова 19

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

6. **Відсутність зайвих дій.**

Погано	Добре
1. Вказати як перший параметр додатку шлях до папки з вихідними файлами. 2. Вказати як другий параметр додатку шлях до папки з кінцевими файлами. 3. Вказати як третій параметр програми шлях до файлу журналу. 4. Запустити додаток.	1. Запустити програму з усіма трьома коректними параметрами (наприклад, c:\src\, c:\dst\, c:\log.txt за умови, що відповідні папки існують і доступні додатку).

Погано	Добре
1. Запустити додаток. 2. Вибрати меню «Файл». 3. Вибрати підпункт «Відкрити». 4. Перейти до папки, в якій знаходиться хоча б один файл формату DOCX з трьома і більше сторінками.	1. Відкрити DOCX-файл із трьома та більше сторінками.

7. **Ненадмірність по відношенню до інших тест-кейсів.**

@ М.В.Добролюбова

20

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

8. **Демонстративність (здатність демонструвати виявлену помилку очевидним чином).**

Витяг з недемонстративного тест-кейсу:

Кроки	Очікувані результати
5. Розмістити у файлі текст «Приклад довгого тексту, що містить символи українського та англійського алфавіту впереміш.» у кодуванні KOI8-R (у слові «Приклад» літера «р» – англійська). 6. Зберегти файл під назвою «test. txt» і надіслати файл на конвертацію. 7. Перейменувати файл на «test.txt».	6. Додаток ігнорує файл. 7. Текст набуває коректного вигляду в кодуванні UTF-8 з урахуванням англійських букв.

Витяг з демонстративного тест-кейсу:

Кроки	Очікувані результати
5. Розмістити у файлі текст «Е╥╒═╪╥ └┼╥╙└.» (Ці символи є словосполученням «Приклад тексту.» у кодуванні KOI8-R, прочитаному як CP866). 6. Надіслати файл на конвертацію.	6. Текст набуває вигляду: «Приклад тексту.» (кодування UTF8).

@ М.В.Добролюбова

21

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

9. **Простежуваність.**

Приклад непростежуваного тест-кейсу:

Вимога	Модуль	Підмодуль	Кроки	Очікувані результати
ПТ-4	Додаток		**Поєднання кодувань** Приготування: файл з декількома допустимими та неприпустимими кодуваннями. 1. Передати файл на конвертацію.	1. Допустимі кодування конвертуються правильно, неприпустимі залишаються без змін.

Приклад простежуваного тест-кейсу:

Вимога	Модуль	Підмодуль	Кроки	Очікувані результати
ДС-2.4 **ДС-3.2**	Стартер	Обробник помилок	**Запуск із некоректними параметрами, неіснуючі шляхи** 1. Запустити додаток з усіма трьома параметрами, значення яких вказують на неіснуючі у файловій системі шляхи.	1. У консолі відображаються нижчезазначені повідомлення, додаток завершує роботу. Повідомлення a. SOURCE_DIR [шлях]: directory not exists or inaccessible. b. DESTINATION_DIR [шлях]: directory not exists or inaccessible. c. LOG_FILE_NAME [ім'я]: wrong file name or inaccessible path.

@ М.В.Добролюбова

22

Властивості якісних тест-кейсів

Властивості для побудови якісних тест-кейсів

10. Можливість повторного використання.

Кроки	Очікувані результати
Запуск, всі параметри некоректні 1. Запустити додаток, вказавши у якості всіх параметрів свідомо некоректні значення.	1. Додаток запускається, після чого виводить повідомлення з описом суті проблеми з кожним параметром і завершує роботу.

11. Повторюваність.
12. Відповідність прийнятим шаблонам оформлення та традиціям.

@ М.В.Добролюбова 23

Набори тест-кейсів

Набір тест-кейсів (test case suite, test suite, test set) – це набір з кількох тестових випадків для компонента або системи, що тестуються, де постумова одного тесту часто використовується як передумова для наступного.

Переваги вільних наборів:

- тест-кейси можна виконувати у будь-якому зручному порядку, а також створювати «набори всередині наборів»
- якщо тест-кейс завершився помилкою, це не вплине на можливість виконання інших тест-кейсів

Переваги послідовних наборів:

- кожен наступний у наборі тест-кейс в якості вхідного стану додатку отримує результат роботи попереднього тест-кейсу, що дозволяє сильно скоротити кількість кроків у окремих тест-кейсах
- довгі послідовності дій куди краще імітують роботу реальних користувачів, ніж окремі «точкові» впливи на додаток

@ М.В.Добролюбова 24

Набори тест-кейсів

Сценарії користувачів (сценарії використання)

Сценарій – це гіпотетична історія, яка використовується, щоб допомогти людині продумати складну проблему або систему.

Сценарій користувача, пункти якого можуть стати основою для кроків тест-кейсу або набору окремих тест-кейсів, на прикладі друку таблички на дверях кабінету з текстом «Йде робота, не стукати!»

1) Запустити текстовий редактор.
2) Створити новий документ (*якщо редактор не робить це самостійно*).
3) Набрати у документі текст.
4) Відформатувати текст належним чином.
5) Надіслати документ на друк.
6) Зберегти документ (*спірно, але допустимо*).
7) Закрити текстовий редактор.

Переваги сценаріїв:

- показують реальні і зрозумілі приклади використання продукту
- зрозумілі кінцевим користувачам і добре підходять для обговорення та спільного покращення;
- легше оцінюються з огляду на важливість, ніж окремі пункти вимог
- чудово показують недоробки у вимогах
- у граничному випадку сценарії можна навіть не прописувати докладно, а просто називати – і саме найменування вже підкаже досвідченому фахівцю, що робити

@ М.В.Добролюбова 25

Набори тест-кейсів

Сценарії користувачів (сценарії використання)

Переваги сценаріїв:

- у граничному випадку сценарії можна навіть не прописувати докладно, а просто називати – і саме найменування вже підкаже досвідченому фахівцю, що робити

Класифікація користувачів

	Низька кваліфікація	Висока кваліфікація
Не схильний до експериментів	«Обережний»	«Консервативний»
Схильний до експериментів	«Відчайдушний»	«Витончений»

Сценарії поведінки на основі класифікації користувачів

	«Обережний»	«Консервативний»	«Відчайдушний»	«Витончений»
Позитивно	«А можна ось так?»	«Почнемо з інструкції!»	«Дивіться, що я придумав!»	«Я все оптимізую!»
Негативно	«Я нічого не розумію.»	«У вас ось тут невідповідність ... »	«Все одно поламаю!»	«А я говорив!»

«An Introduction to Scenario Testing», Cem Kaner http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf]

@ М.В.Добролюбова

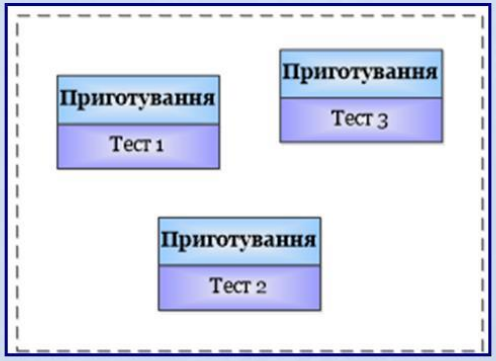
26

Набори тест-кейсів

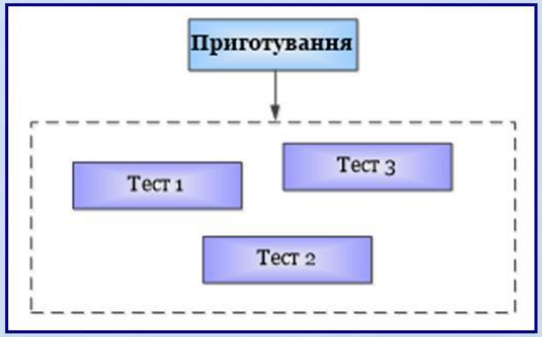
Детальна класифікація наборів тест-кейсів

		По ізольованості тест-кейсів один від одного	
		Ізольовані	Узагальнені
За утворенням тест-кейсами суворої послідовності	Вільні	Ізольовані вільні	Узагальнені вільні
	Послідовні	Ізольовані послідовні	Узагальнені послідовні

Набір ізольованих вільних тест-кейсів: дії з розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейси можна виконувати у будь-якому порядку.



Набір узагальнених вільних тест-кейсів: дії розділу «приготування» потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси можна виконувати у будь-якому порядку.



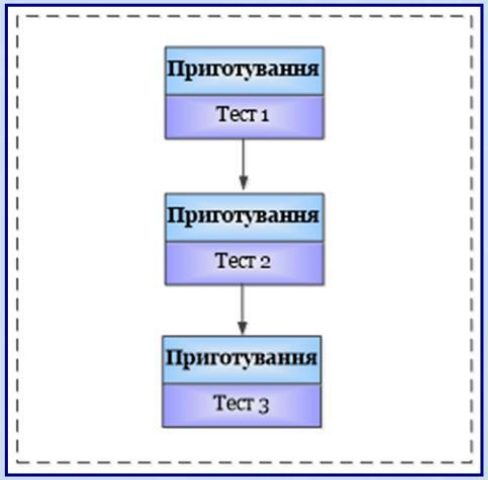
@ М.В.Добролюбова

27

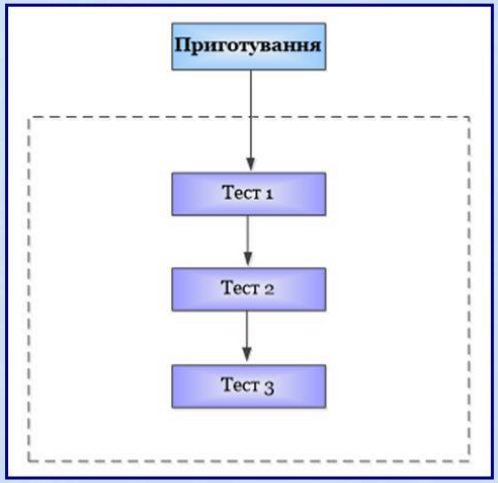
Набори тест-кейсів

Детальна класифікація наборів тест-кейсів

Набір ізольованих послідовних тест-кейсів: дії з розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейси потрібно виконувати у строго визначеному порядку.



Набір узагальнених послідовних тест-кейсів: дії з розділу «приготування» потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси потрібно виконувати в певному порядку.



Головна перевага ізольованості: кожен тест-кейс виконується в «чистому середовищі», на нього не впливають результати роботи попередніх тест-кейсів.

Головна перевага узагальненості: приготування не потрібно повторювати.

Головна перевага послідовності: відчутне скорочення кроків у кожному тест-кейсі, оскільки результат виконання попереднього тест-кейсу є початковою ситуацією для наступного.

Головна перевага свободи: можливість виконувати тест-кейси в будь-якому порядку, а також те, що при провалі якогось тест-кейсу, інші тест-кейси, як і раніше, можна виконувати.

@ М.В.Добролюбова

28

Набори тест-кейсів

Принципи побудови наборів тест-кейсів

Єдине завдання наборів – підвищити ефективність тестування за рахунок прискорення і спрощення виконання тест-кейсів, збільшення глибини дослідження якоїсь області додатку або функціональності, дотримання типових сценаріїв користувача або зручної послідовності виконання тест-кейсів тощо.

Найбільш типові підходи до складання наборів тест-кейсів:

- на основі чек-листів
- на основі розбиття програми на модулі та під модулі
- за принципом перевірки найважливіших, менш важливих та інших функцій додатку
- за принципом групування тест-кейсів для перевірки певного рівня вимог або типу вимог, групи вимог або окремої вимоги
- за принципом частоти виявлення тест-кейсами дефектів у додатку
- за архітектурним принципом
- за областю внутрішньої роботи програми
- за видами тестування

@ М.В.Добролюбова 29

Логіка створення ефективних перевірок

Якість (quality) – це ступінь, в якій компонент, система чи процес відповідають визначеним вимогам та/або потребам та очікуванням користувача/замовника.

Завдання тестувальника – знайти максимум ВАЖЛИВИХ помилок за наявний час. Під важливими помилками розуміють такі, які призводять до порушення важливих для користувача функцій або властивостей продукту.

Питання, які тестувальнику необхідно задати собі та отримати чіткі відповіді, приступаючи до продумування чек-листа, тест-кейс або набору тест-кейсів:

1. Що перед вами?
2. Кому і навіщо воно потрібне (і наскільки це важливо)?
3. Як воно зазвичай використовується?
4. Як воно може зламатися, тобто почати працювати неправильно?

@ М.В.Добролюбова

30

Логіка створення ефективних перевірок

Перелік універсальних рекомендацій, які дозволять проводити тестування краще:

- починати якомога раніше
- якщо потрібно тестувати щось велике і складне, доцільно розбити його на модулі та підмодулі, функціональність піддати функціональній декомпозиції
- обов'язково писати чек-листи
- по мірі створення чек-листів, тест-кейсів тощо прямо в текст вписувати питання, що виникають; коли питань накопичиться достатньо, зібрати їх окремо, уточнити формулювання та звернутися до того, хто може дати відповіді
- якщо використовуваний інструментальний засіб дозволяє застосовувати косметичне оформлення тексту – використати
- використати техніку швидкого перегляду для отримання відгуку від колег та покращення створеного документа
- планувати час на покращення тест-кейсів
- починати опрацювання (і виконання) тест-кейсів із простих позитивних перевірок найважливішої функціональності; потім поступово підвищувати складність перевірок, пам'ятаючи не тільки про позитивні, а й про негативні перевірки
- пам'ятати, якщо тестувальник не може швидко і просто сформулювати ціль створеного ним тест-кейсу, він створив поганий тест-кейс
- уникати надлишкових тест-кейсів, що дублюють один одного
- якщо показник тест-кейсу можна збільшити, при цьому не сильно змінивши його складність і не відхилившись від вихідної мети, зробити це
- пам'ятати, що багато тест-кейсів потребують окремої підготовки, яку потрібно описати у відповідному полі тест-кейсу
- кілька позитивних тест-кейсів можна безбоязно об'єднувати, але об'єднання негативних тест-кейсів майже завжди заборонено
- подумати, як можна оптимізувати створений тест-кейс (набір тест-кейсів тощо) так, щоб знизити трудовитрати на його виконання
- перед тим, як надсилати фінальну версію створеного документа, ще раз перечитати написане

@ М.В.Добролюбова 31

Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів

Перелік типових помилок оформлення та формулювань:

- відсутність назви тест-кейсу або погано написана назва
- відсутність нумерації кроків та/або очікуваних результатів
- посилання на багато вимог
- використання особової форми дієслів
- використання минулого чи майбутнього часу в очікуваних результатах
- постійне використання слів «перевірити» (і йому подібних) у чек-листах
- опис стандартних елементів інтерфейсу замість використання їх усталених назв
- пунктуаційні, орфографічні, синтаксичні та подібні до них помилки

@ М.В.Добролюбова

32

Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів

Перелік типових логічних помилок:

- посилання на інші тест-кейс або кроки інших тест-кейсів
- деталізація, що не відповідає рівню функціонального тестування
- розпливчасті двозначні описи дій та очікуваних результатів
- опис дій в якості найменувань модуля/під модуля
- опис подій або процесів в якості кроків або очікуваних результатів
- «вигадування» особливостей поведінки програми
- відсутність опису приготування для виконання тест-кейсу
- повне дублювання (копіювання) одного і того ж тест-кейсу на рівнях димового тестування, тестування критичного шляху, розширеного тестування
- занадто довгий перелік кроків, що не належать до суті (мети) тест-кейсу
- некоректне найменування елементів інтерфейсу чи його властивостей
- нерозуміння принципів роботи додатку та викликана цим некоректність тест-кейсів
- перевірка типової «системної» функціональності
- неправильна поведінка додатку як очікуваний результат
- загальна некоректність тест-кейсів
- неправильне розбиття наборів даних на класи еквівалентності
- тест-кейси, що не належать до додатку, який тестується
- формальні та/або суб'єктивні перевірки

@ М.В.Добролюбова 33


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 3. ДЕФЕКТИ
Лекція 6
«Звіти про дефекти»
@ М.В.Добролюбова

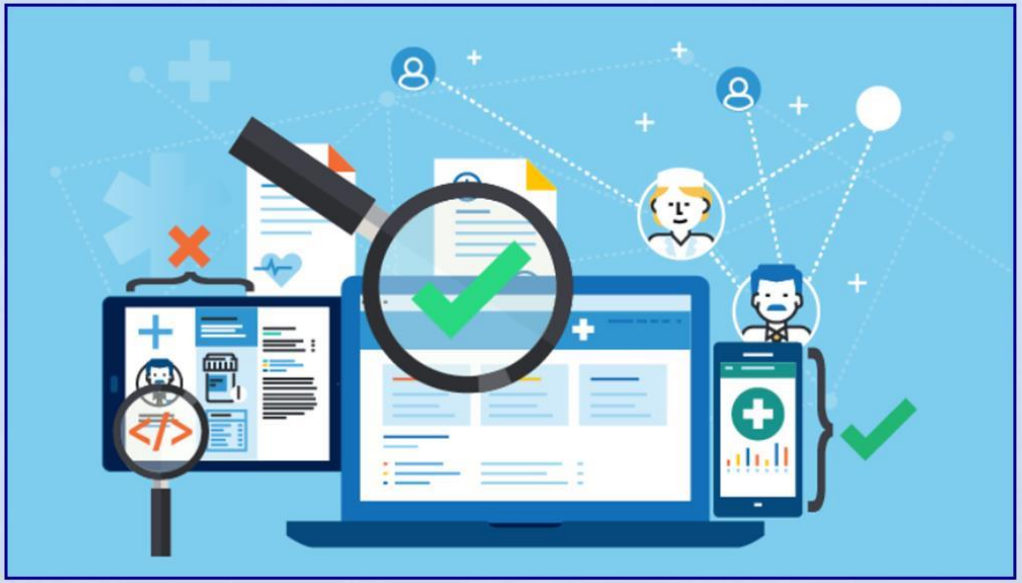
Помилки, дефекти, збої, відмови

Спрощений погляд на поняття дефекту

Дефект – розбіжність очікуваного та фактичного результату.

Очікуваний результат – поведінка системи, описана у вимогах.

Фактичний результат – поведінка системи, що спостерігається в процесі тестування.

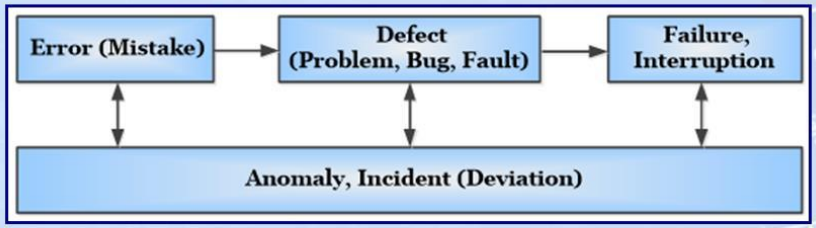


@ М.В.Добролюбова

2

Помилки, дефекти, збої, відмови

Розширений погляд на термінологію, що описує проблеми



Взаємозв'язок проблем розробки програмних продуктів

Помилка (error, mistake) – це дія людини, яка дає неправильний результат.

Дефект, помилка, проблема, несправність (defect, bug, problem, fault) – це несправність компонента або системи, яка може призвести до того, що компонент або система не зможуть виконувати свої необхідні функції.

Переривання, збій, відмова (interruption) – це призупинення процесу, наприклад виконання комп'ютерної програми, викликане подією, зовнішньою по відношенню до цього процесу, і виконується таким чином, що процес можна відновити.

Відмова (failure) – це відхилення компонента або системи від очікуваної доставки, обслуговування або результату.

Аномалія (anomaly) – це будь-яка умова, що відхиляється від очікувань, заснована на специфікаціях вимог, проєктної документації, документації користувача, стандартів тощо або з чийогось сприйняття чи досвіду.

Інцидент, відхилення (deviation) – це будь-яка подія, яка потребує розслідування.

Фактичний результат (actual result) – це поведінка, яка виникає/спостерігається під час тестування компонента або системи.

Очікуваний результат, прогнозований результат (expected result) – це подія, передбачена специфікацією або іншим джерелом компонента або системи за певних умов.

Rudolph Frederick Stapelberg, «Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design»
Стандарт «IEEE 1044:2009 Standard Classification For Software Anomalies»

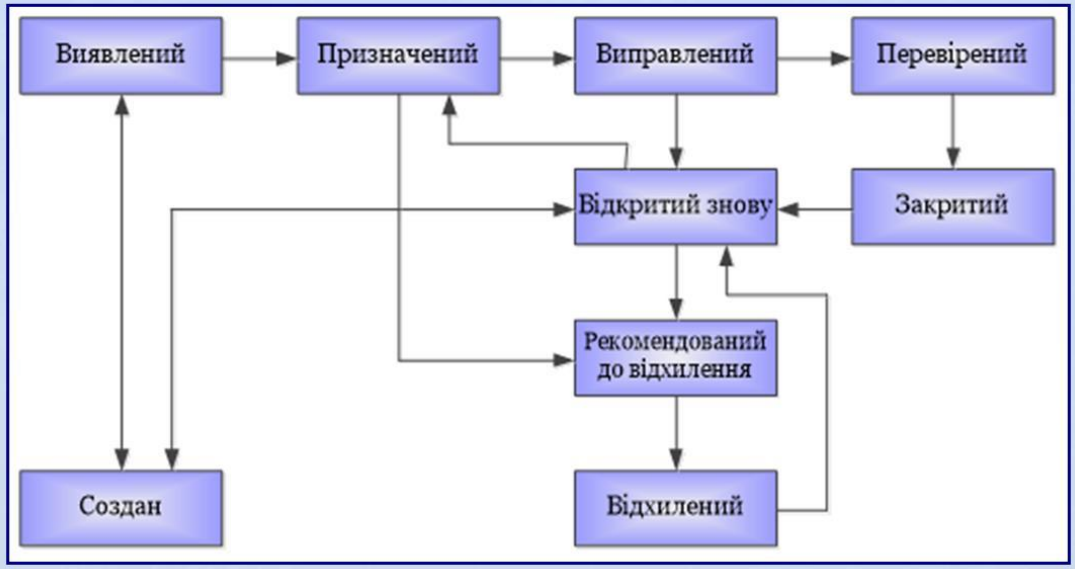
@ М.В.Добролюбова 3

Звіт про дефект та його життєвий цикл

Звіт про дефект, звіт про помилку (defect report) – це документ, у якому повідомляється про будь-який недолік у компоненті або системі, який може призвести до того, що компонент або система не зможуть виконувати свої необхідні функції.

Основні цілі написання звіту про дефект:

- надати інформацію про проблему
- пріоритизувати проблему
- сприяти усуненню

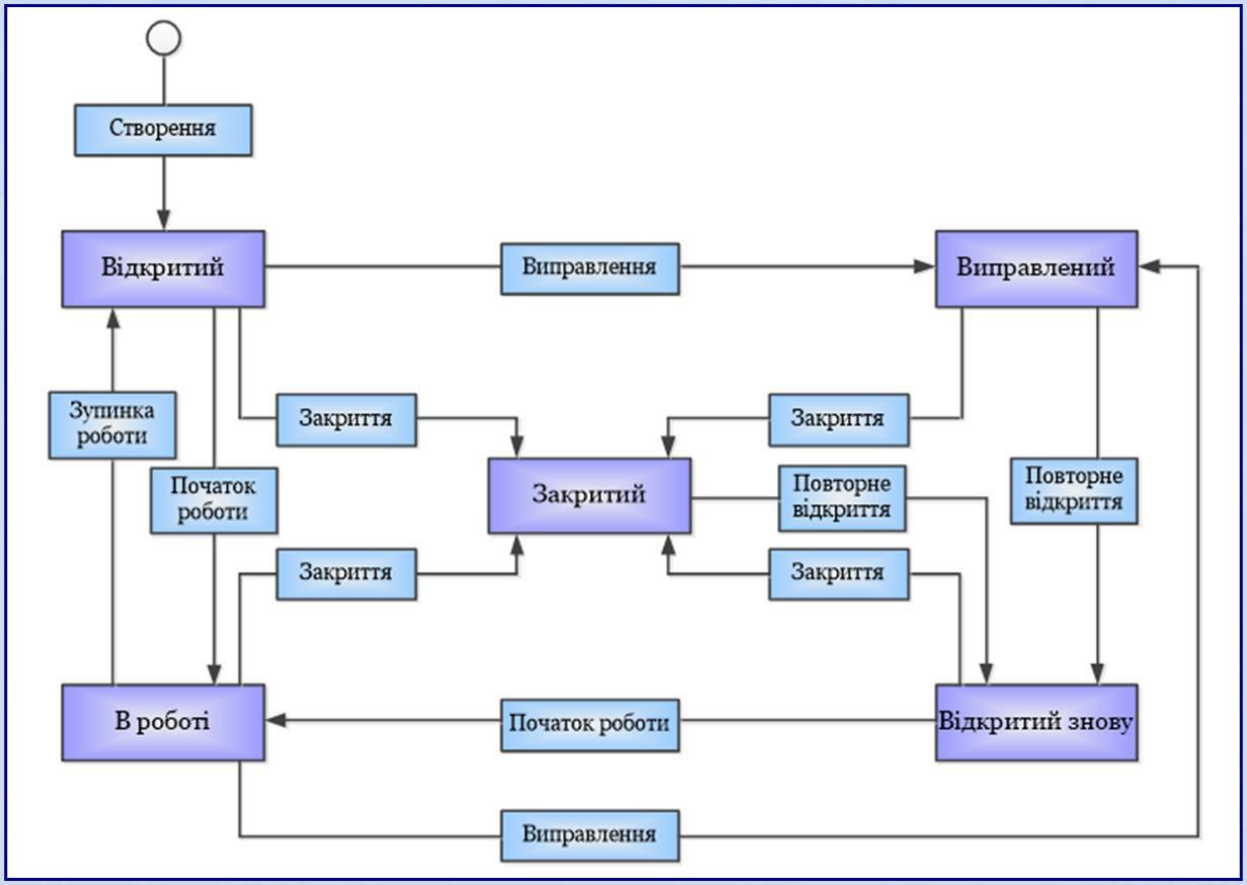


Життєвий цикл звіту про дефект з найбільш типовими переходами між станами

@ М.В.Добролюбова

4

Звіт про дефект та його життєвий цикл



Життєвий цикл звіту про дефект JIRA

"What is Workflow" [https://confluence.atlassian.com/jira063/what-is-workflow-683542483.html]

@ М.В.Добролюбова

5

Атрибути (поля) звіту про дефект

Ідентифікатор	Короткий опис	Детальний опис	Кроки по відтворенню
19	Нескінченний цикл обробки вхідного файлу з атрибутом «тільки для читання».	Якщо у вхідного файлу виставлено атрибут «тільки для читання», після обробки додатку не вдається перемістити його в каталог-приймач: створюється копія файлу, але оригінал не видаляється, і додаток знову і знову обробляє цей файл і безуспішно намагається перемістити його до каталогу. **Очікуваний результат:** після обробки файл переміщений з каталогу-джерела в каталог-приймач. **Фактичний результат:** оброблений файл копіюється до каталогу-приймача, але його оригінал залишається в каталозі-джерелі. **Вимога:** ДС-2.1.	1. Помістити в каталог-джерело файл допустимого типу та розміру. 2. Встановити даному файлу атрибут «тільки для читання». 3. Запустити програму. Дефект: оброблений файл з'являється в каталозі-приймачі, але не видаляється з каталогу-джерела, файл у каталозі-приймачі безперервно оновлюється (видно за значенням часу останньої зміни).

Відтворюваність	Важливість	Терміновість	Симптом	Можливість обійти	Коментар	Додатки
Завжди	Середня	Звичайна	Некоректна операція	Ні	Якщо замовник не планує використовувати установку атрибута «тільки для читання» файлів у каталозі-джерелі для досягнення деяких своїх цілей, можна просто знімати цей атрибут та спокійно переміщати файл.	-

Загальний вигляд звіту про дефект

@ М.В.Добролюбова

6

Атрибути (поля) звіту про дефект

Ідентифікатор (identifier) – це унікальне значення, що дозволяє однозначно відрізнити один звіт про дефект від іншого і використовуване в у різних посиланнях.

Короткий опис (summary) має у гранично лаконічній формі давати вичерпну відповідь на запитання «Що сталося?», «Де це сталося?», «За яких умов це сталося?».

Приклад

«Відсутній логотип на сторінці привітання, якщо користувач є адміністратором»:

- Що сталося? Немає логотипу.
- Де це сталося? На сторінці вітання.
- За яких умов це сталося? Якщо користувач є адміністратором.

Алгоритм для створення хороших коротких описів дефектів:

- повноцінно зрозуміти суть проблеми
- сформулювати докладний опис (description) дефекту – спочатку без огляду на довжину тексту, що вийшов
- прибрати з детального опису все зайве, уточнити важливі деталі
- виділити у докладному описі слова (словосполучення, фрагменти фраз), які відповідають на питання, «що, де і за яких умовах сталося»
- оформити отримане у пункті 4 у вигляді закінченого граматично правильного речення
- якщо речення вийшло занадто довгим, переформулювати його, скоротивши довжину

@ М.В.Добролюбова 7

Атрибути (поля) звіту про дефект

Приклади застосування алгоритму

Ситуація 1. **Тестуванню піддається деякий веб-додаток, поле опису товару має допускати введення максимум 250 символів; у процесі тестування виявилося, що цього обмеження немає.**

1. *Суть проблеми:* дослідження показало, що ні на клієнтській, ні на серверній частині немає жодних механізмів, що перевіряють та/або обмежують довжину введених у полі «Про товар» даних.

2. *Вихідний варіант докладного опису:* у клієнтській і серверній частині додатку відсутні перевірка та обмеження довжини даних, що вводяться в поле «Про товар» на сторінці http://testapplication/admin/goods/edit/.

3. *Кінцевий варіант докладного опису:*

• Фактичний результат: в описі товару (поле «Про товар», http://testapplication/admin/goods/edit/) відсутні перевірка та обмеження довжини тексту (MAX=250 символів).

• Очікуваний результат: у разі спроби вводу 251+ символів відображається повідомлення про помилку.

4. *Визначення «що, де і за яких умов сталося»:*

• Що: Перевірка та обмеження довжини тексту, що вводиться.

• Де: опис товару, поле «Про товар», http://testapplication/admin/goods/edit/.

• За яких умов: – (в даному випадку дефект присутній завжди, незалежно від будь-яких особливих умов).

5. *Первинне формулювання:* відсутні перевірка та обмеження максимальної довжини тексту, що вводиться в полі «Про товар» опису товару.

6. *Скорочення (підсумковий короткий опис):* немає обмеження максимальної довжини поля «Про товар». Англійський варіант: no check for «Про товар» max length.

Ситуація 2. **Спроба відкрити в додатку порожній файл призводить до краху клієнтської частини додатку та втрати не збережених даних користувача на сервері.**

1. *Суть проблеми:* клієнтська частина додатку починає «наосліп» читати заголовок файлу, не перевіряючи ні розміру, ні коректності формату, нічого; виникає якась внутрішня помилка, і клієнтська частина програми некоректно припиняє роботу, не закривши сесію із сервером; сервер закриває сесію за таймаутом (повторний запуск клієнтської частини запускає нову сесію, так що стара сесія та всі дані в ній у будь-якому випадку втрачені).

2. *Вихідний варіант докладного опису:* некоректний аналіз файлу, що відкривається клієнтом, призводить до краху клієнта і необоротної втрати поточної сесії з сервером.

3. *Кінцевий варіант докладного опису:*

• Фактичний результат: відсутність перевірки коректності файлу, що відкривається клієнтською частиною (у тому числі порожнього), призводить до краху клієнтської частини та незворотної втрати поточної сесії із сервером (див. BR852345).

• Очікуваний результат: виконується аналіз структури файлу, який відкривається; у разі виявлення проблем з'являється повідомлення про неможливість відкриття файлу.

4. *Визначення «що, де і за яких умов сталося»:*

• Що: крах клієнтської частини додатку.

• Де: – (конкретне місце у додатку визначити навряд чи можливо).

• За яких умов: при відкритті порожнього або пошкодженого файлу.

5. *Первинне формулювання:* відсутність перевірки коректності відкритого файлу призводить до краху клієнтської частини програми та втрати даних користувача.

6. *Скорочення (підсумковий короткий опис):* крах клієнта та втрата даних при відкритті пошкоджених файлів. Англійський варіант: client crash and data loss on damaged/empty files opening.

@ М.В.Добролюбова

8

Атрибути (поля) звіту про дефект

Приклади застосування алгоритму

Ситуація 3. **Вкрай рідко з абсолютно незрозумілих причин на сайті порушується відображення всього українського тексту (як статичних написів, так і даних з бази даних, що генеруються динамічно тощо – все «стає питаннями»).**

1. *Суть проблеми:* фреймворк, у якому побудований сайт, підвантажує специфічні шрифти з віддаленого сервера; якщо з'єднання обривається, потрібні шрифти не підвантажуються, і використовуються стандартні шрифти, в яких немає українських знаків.

2. *Вихідний варіант докладного опису:* помилка завантаження шрифтів із віддаленого сервера призводить до використання локальних несумісних із потрібним кодуванням шрифтів.

3. *Кінцевий варіант докладного опису:*

• Фактичний результат: періодична неможливість завантажити шрифти з віддаленого сервера призводить до використання локальних шрифтів, несумісних із потрібним кодуванням.

• Очікуваний результат: необхідні шрифти завжди завантажуються (або використовується локальне джерело необхідних шрифтів).

4. *Визначення «що, де і за яких умов сталося»:*

• Що: використовуються несумісні з потрібним кодуванням шрифти.

• Де: – (по всьому сайту).

• За яких умов: у разі помилки з'єднання із сервером, з якого підвантажуються шрифти.

5. *Первинне формулювання:* періодичні збої зовнішнього джерела шрифтів призводять до збою відображення українського тексту.

6. *Скорочення (підсумковий короткий опис):* неправильний показ українського тексту при відсутності зовнішніх шрифтів. Англійський варіант: wrong presentation of Ukrainian text in case of external fonts inaccessibility.

Ситуація	Український варіант короткого опису	Англійський варіант короткого опису
Тестуванню піддається деякий веб-додаток, поле опису товару повинно допускати введення максимум 250 символів; в процесі тестування виявилося, що цього обмеження немає.	Немає обмеження максимальної довжини поля «Про товар».	No check for «Про товар» max length.
Спроба відкрити у додатку порожній файл призводить до краху клієнтської частини програми та втрати не збережених даних користувача на сервері.	Крах клієнта та втрата даних при відкритті пошкоджених/порожніх файлів.	Client crash and data loss on damaged/empty files opening.
Вкрай рідко з абсолютно незрозумілих причин на сайті порушується відображення всього українського тексту (як статичних написів, так і даних з бази даних, що генеруються динамічно тощо – все «стає питаннями»).	Невірний показ українського тексту за недоступності зовнішніх шрифтів.	Wrong presentation of Ukrainian text in case of external fonts inaccessibility.

Проблемні ситуації та формулювання коротких описів дефектів

@ М.В.Добролюбова

9

Атрибути (поля) звіту про дефект

Детальний опис (description) представляє у розгорнутому вигляді необхідну інформацію про дефект, а також (обов'язково!) опис фактичного результату, очікуваного результату та посилання на вимогу.

Якщо в систему входить адміністратор, на сторінці привітання відсутній логотип.

Фактичний результат: логотип відсутній у верхньому лівому куті сторінки.

Очікуваний результат: логотип відображається у верхньому лівому куті сторінки.

Вимога: R245.3.23b.

Приклад детального опису

Кроки з відтворення (steps to reproduce, STR) описують дії, які необхідно виконати для відтворення дефекту.

1. Відкрити http://testapplication/admin/login/.
2. Авторизуватися з ім'ям «defaultadmin» та паролем «dapassword».

Дефект: у верхньому лівому куті сторінки відсутній логотип (замість нього відображається порожній простір з написом «logo»).

Приклад кроків відтворення

@ М.В.Добролюбова 10

Атрибути (поля) звіту про дефект

Відтворюваність (reproducibility) показує, чи при кожному проходженні по кроках відтворення дефекту вдається викликати його прояв. Це поле приймає лише два значення: завжди (always) або іноді (sometimes). Відтворюваність «іноді» означає, що тестувальник не знайшов справжню причину виникнення дефекту.

Важливість (severity) показує ступінь шкоди, що завдається проєкту існуванням дефекту.

Градації важливості:

- *критична* (critical) – існування дефекту призводить до масштабних наслідків катастрофічного характеру
- *висока* (major) – існування дефекту приносить відчутні незручності багатьом користувачам у межах їх типової діяльності
- *середня* (medium) – існування дефекту слабко впливає на типові сценарії роботи користувачів та/або існує обхідний шлях досягнення
- *низька* (minor) – існування дефекту рідко виявляється незначним відсотком користувачів і (майже) не впливає на їх роботу

@ М.В.Добролюбова 11

Атрибути (поля) звіту про дефект

Терміновість (priority) показує, як швидко дефект має бути усунений.

Градації терміновості:

- *найвища* (ASAP, as soon as possible) терміновість вказує на необхідність усунути дефект настільки швидко, наскільки це можливо
- *висока* (high) терміновість означає, що дефект слід виправити позачергово, оскільки його існування або вже об'єктивно заважає роботі, або почне створювати такі перешкоди у найближчому майбутньому
- *звичайна* (normal) терміновість означає, що дефект слід виправити у порядку загальної черги
- *низька* (low) терміновість означає, що у найближчому майбутньому виправлення даного дефекту не вплине на підвищення якості продукту

		Важливість (Severity)	
		Критична (Critical)	**Низька (Minor)**
Терміновість (Priority)	**Найвища (ASAP)**	Проблеми з безпекою у введеному в експлуатацію банківському ПЗ.	На корпоративному сайті пошкодилася картинка із фірмовим логотип.
	Низька (Low)	На самому початку розробки проекту виявлено ситуацію, за якої можуть бути пошкоджені або зовсім втрачені дані користувача.	У керівництві користувача виявлено кілька помилок, що не впливають на зміст тексту.

Приклади поєднання важливості та терміновості дефектів

@ М.В.Добролюбова

12

Атрибути (поля) звіту про дефект

Симптом (symptom) – дозволяє класифікувати дефекти за їх типовим проявом. Не існує загальноприйнятого списку симптомів. В одного дефекту може бути відразу кілька симптомів.

Приклади значень симптомів дефекту:

- косметичний дефект (cosmetic flaw)
- пошкодження/втрата даних (data corruption/loss)
- проблема в документації (documentation issue)
- некоректна операція (incorrect operation)
- проблема інсталяції (installation problem)
- помилка локалізації (localization issue)
- нереалізована функціональність (missing feature)
- проблема масштабованості (scalability)
- низька продуктивність (low performance)
- крах системи (system crash)
- несподівана поведінка (unexpected behavior)
- недружня поведінка (unfriendly behavior)
- розбіжність із вимогами (variance from specs)
- пропозиція щодо покращення (enhancement)

Можливість обійти (workaround) – показує, чи існує альтернативна послідовність дій, виконання якої дозволило б користувачеві досягти поставленої мети.

Коментар (comments, additional info) – може містити будь-які корисні для розуміння та виправлення дефекту дані.

Вкладення (attachments) – це не так поле, як список прикріплених до звіту про дефект додатків.

@ М.В.Добролюбова 13

Інструментальні засоби управління звітами про дефекти

Інструмент управління дефектами, інструмент управління інцидентами – це інструмент, який полегшує запис і відстеження стану дефектів і змін. Вони часто мають засоби, орієнтовані на робочий процес, для відстеження та контролю розподілу, виправлення та повторного тестування дефектів і надання засобів звітності. Найчастіше такі інструментальні засоби є частинами інструментальних засобів управління тестуванням.

Загальний набір функцій, які зазвичай реалізуються інструментальними засобами управління звітами про дефекти:

- створення звітів про дефекти, керування їх життєвим циклом з урахуванням контролю версій, прав доступу та дозволених переходів зі стану до стану
- збір, аналіз та надання статистики у зручній для сприйняття людиною формі
- розсилка повідомлень, нагадувань та інших артефактів відповідним співробітникам
- організація взаємозв'язків між звітами про дефекти, тест-кейсами, вимогами та аналіз таких зв'язків із можливістю формування рекомендацій
- підготовка інформації для включення до звіту про результати тестування
- інтеграція із системами управління проєктами

@ М.В.Добролюбова 14

Приклади інструментальних засобів

Створення звіту про дефект в JIRA

"JIRA - Issue & Project Tracking Software"
[https://www.atlassian.com/software/jira/]

"What is an Issue"
[https://confluence.atlassian.com/jira063/what-is-an-issue-683542485.html]

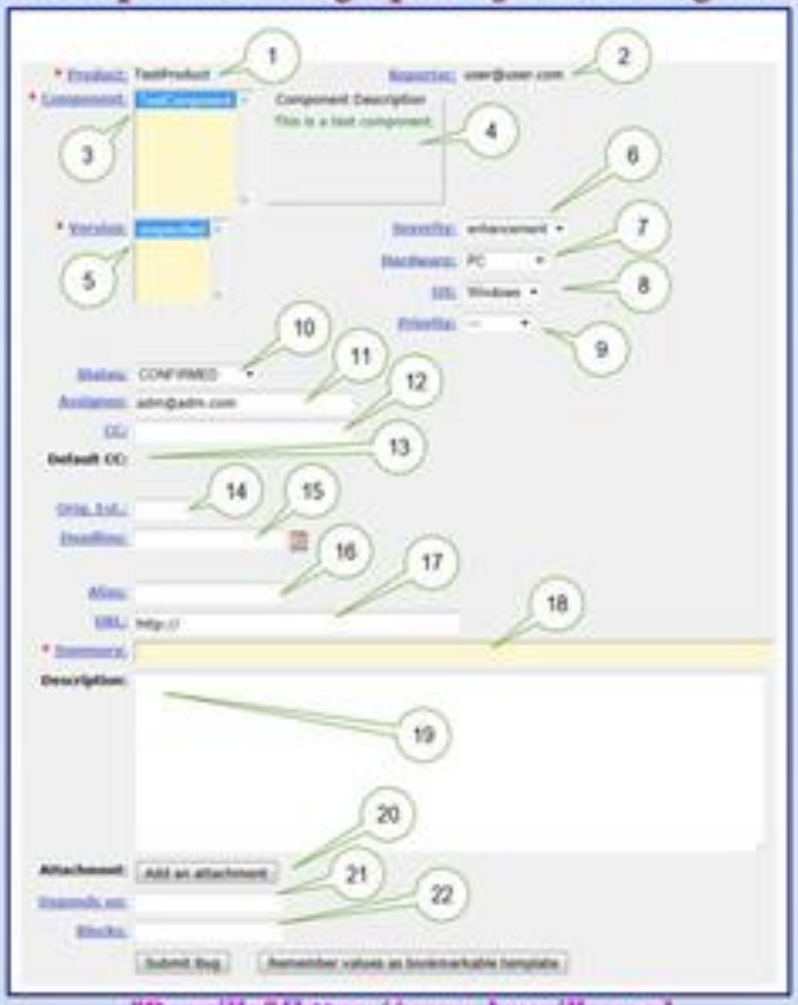
" Defining Issue Type Field Values "
[https://confluence.atlassian.com/display/JIRA/Defining+Issue+Type+Field+Values]

@ М.В.Добролюбова

15

Приклади інструментальних засобів

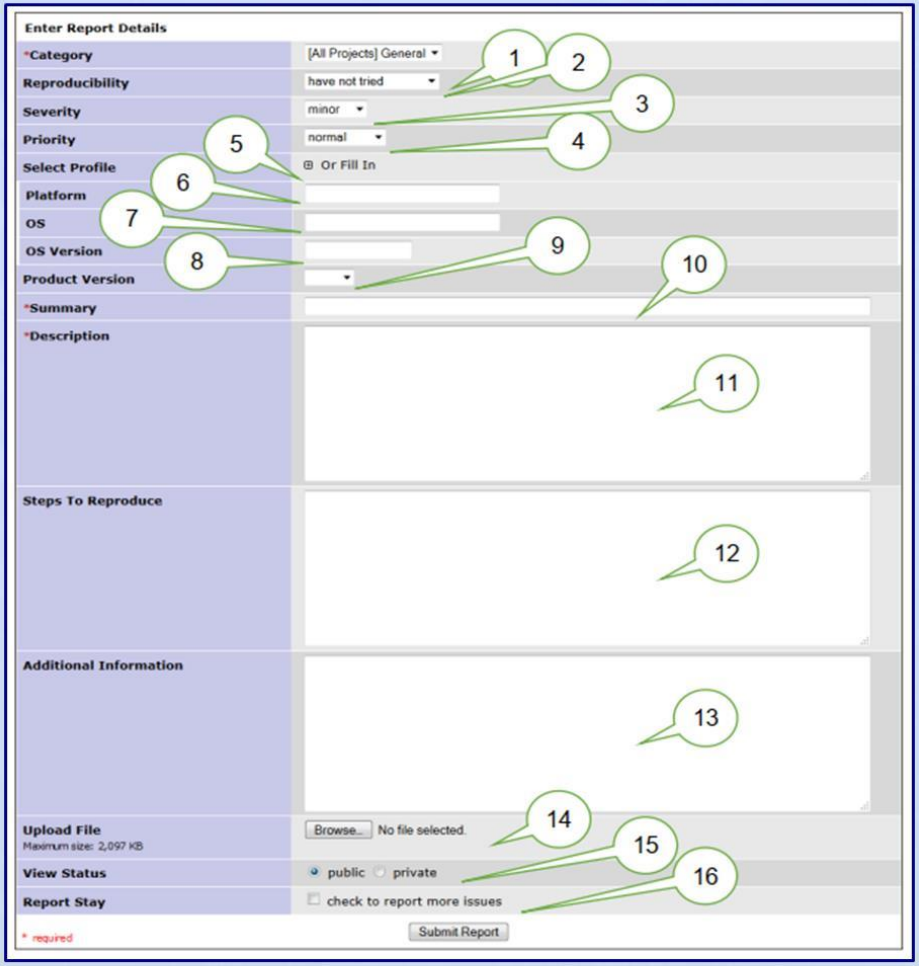
Створення звіту про дефект в Bugzilla



"Bugzilla" [https://www.bugzilla.org]

Приклади інструментальних засобів

Створення звіту про дефект в Mantis



"Mantis Bug Tracker" [https://www.mantisbt.org]

@ М.В.Добролюбова

17

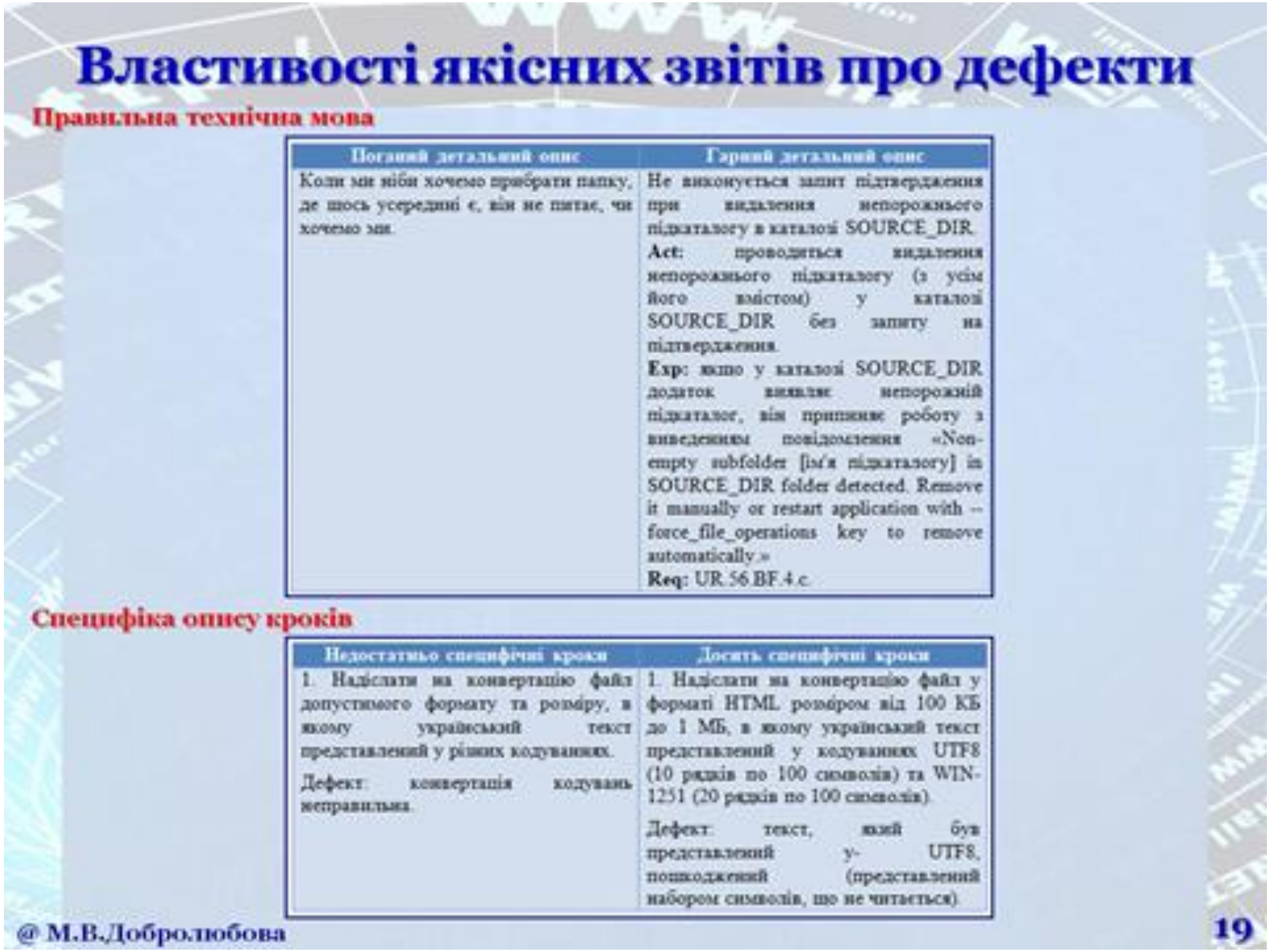
Властивості якісних звітів про дефекти

Ретельне заповнення всіх полів точною та коректною інформацією

Найяскравіші проявами такої проблеми:

- частина важливих для розуміння проблеми полів не заповнена
- наданої інформації недостатньо для розуміння проблеми
- надана інформація є некоректною
- «дефект» знайдений у функціональності, яка ще не була оголошена як готова до тестування
- у звіті присутня жаргонна лексика
- звіт замість опису проблеми з програмою критикує роботу когось із учасників проєктної команди
- у звіті втрачено якусь незначну на перший погляд, але за фактом критичну для відтворення дефекту проблему
- звіту виставлені неправильні (зазвичай занижені) важливість або терміновість
- до звіту не додаються необхідні копії екрана (особливо важливі для косметичних дефектів) чи інші файли
- звіт написаний безграмотно з погляду людської мови

@ М.В.Добролюбова 18


Властивості якісних звітів про дефекти
Правильна технічна мова
Поганий детальний опис
Коли ми ніби хочемо прибрати папку, де щось усередині є, він не питає, чи хочемо ми.
Гарний детальний опис
Не виконується запит підтвердження при видаленні непорожнього підкаталогу в каталозі SOURCE_DIR.
Act: проводиться видалення непорожнього підкаталогу (з усім його вмістом) у каталозі SOURCE_DIR без запиту на підтвердження.
Exp: якщо у каталозі SOURCE_DIR додаток виявляє непорожній підкаталог, він припиняє роботу з виведенням повідомлення «Non-empty subfolder [ім'я підкаталогу] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.»
Req: UR.56.BF.4.c
Специфіка опису кроків
Недостатньо специфічні кроки
1. Надіслати на конвертацію файл допустимого формату та розміру, в якому український текст представлений у різних кодуваннях.
Дефект: конвертація кодувань неправильна.
Досить специфічні кроки
1. Надіслати на конвертацію файл у форматі HTML розміром від 100 КБ до 1 МБ, в якому український текст представлений у кодуваннях UTF8 (10 рядків по 100 символів) та WIN-1251 (20 рядків по 100 символів).
Дефект: текст, який був представлений у- UTF8, пошкоджений (представлений набором символів, що не читається).
@ М.В.Добролюбова
19

Властивості якісних звітів про дефекти

Відсутність зайвих дій та/або їх довгих описів

Погано	Добре
1. Вказати як перший параметр програми шлях до папки з вихідними файлами. 2. Вказати як другий параметр програми шлях до папки з кінцевими файлами. 3. Вказати як третій параметр програми шлях до файлу журналу. 4. Запустити програму. Дефект: додаток використовує перший параметр командного рядка і як шлях до папки з вихідними файлами, і як шлях до папки з кінцевими файлами.	1. Запустити додаток з усіма трьома коректними параметрами (особливо звернути увагу, щоб SOURCE_DIR та DESTINATION_DIR не збігалися та не були вкладені один в одного у будь-якому поєднанні). Дефект: програма припиняє роботу з повідомленням «SOURCE_DIR and DESTINATION_DIR may not be the same!» (судячи по всьому, перший параметр командного рядка використовується для ініціалізації імен обох каталогів).

Погано	Добре
1. Запустити додаток з усіма вірними параметрами. 2. Зачекати більше 30 хвилин. 3. Надіслати на конвертацію файл допустимого формату та розміру. Дефект: програма не обробляє файл.	Передмова: додаток запущений та пропрацював більше 30 хвилин. 1. Надіслати на конвертацію файл допустимого формату та розміру. Дефект: програма не обробляє файл.

@ М.В.Добролюбова

20

Властивості якісних звітів про дефекти

Відсутність дублікатів

Набір рекомендацій:

- якщо ви не впевнені, що дефект не було описано раніше, здійсніть пошук за допомогою інструментального засобу управління життєвим циклом звітів про дефекти
- пишіть максимально інформативні короткі описи (бо пошук в першу чергу проводять за ними)
- використовуйте максимум можливості вашого інструментального засобу: вказуйте у звіті про дефект компоненти програми, посилання на вимоги, розставляйте теги тощо
- вказуйте у докладному описі дефекту текст повідомлень від додатку, якщо це можливо
- намагайтеся брати участь у так званих «нарадах по проясненню»
- якщо ви виявили якусь додаткову інформацію, внесіть її до вже існуючого звіту про дефект (або попросіть зробити це його автора), але не створюйте окремий звіт

Уточнююча нарада – обговорення, яке допомагає клієнтам досягти «попередньої ясності» – консенсусу щодо бажаної поведінки кожної історії – шляхом постановки питань та отримання прикладів.

"Agile Testing", Lisa Crispin, Janet Gregory

@ М.В.Добролюбова 21

Властивості якісних звітів про дефекти

Очевидність та зрозумілість

Поганий докладний опис	Добрий докладний опис
Додаток не повідомляє про виявлені підкаталоги у каталозі SOURCE_DIR.	Додаток не повідомляє користувача про виявлені у каталозі SOURCE_DIR підкаталоги, що призводить до необґрунтованих очікувань користувачами оброки файлів у таких підкаталогах. **Act:** додаток починає (продовжує) роботу, якщо в каталозі SOURCE_DIR знаходяться підкаталоги. **Exp:** якщо у каталозі SOURCE_DIR додаток при запуску або в процесі роботи виявляє порожній підкаталог, він автоматично його видаляє (чи це логічно?), якщо ж виявлено непорожній підкаталог, додаток припиняє роботу з виведенням повідомлення «Non-empty subfolder [ім'я підкаталогу] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.c.

@ М.В.Добролюбова

22

Властивості якісних звітів про дефекти

Простежуваність

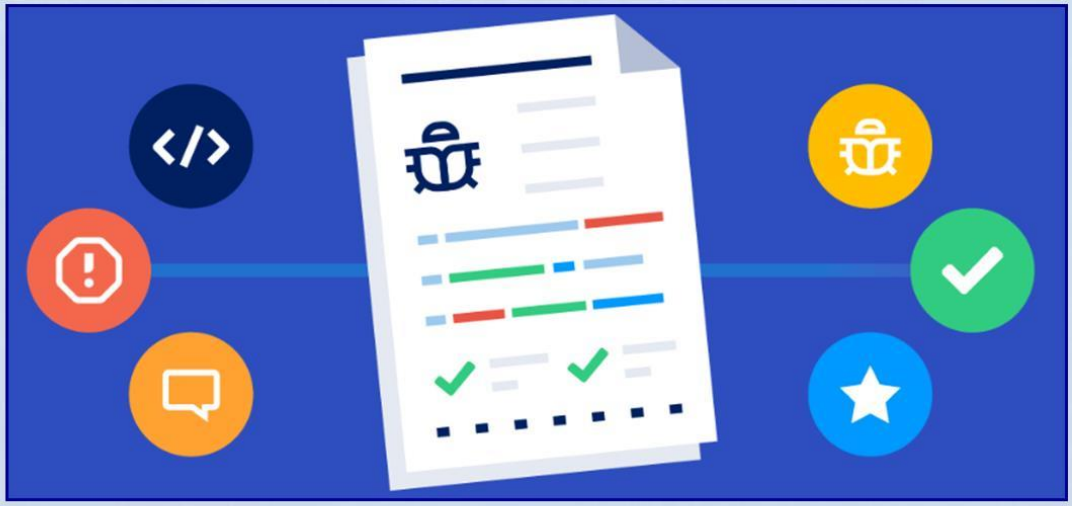
Вказуйте у звіті про дефект компоненти програми, посилання на вимоги, тест-кейси, суміжні звіти про дефекти (схожі дефекти; залежні та ті, що залежать від даного, дефекти), розставляйте теги тощо.

Окремі звіти кожного нового дефекту

Правила:

1. У кожному звіті описується рівно один дефект.
2. При виявленні нового дефекту створюється новий звіт.

Відповідність прийнятим шаблонам оформлення та традиціям



@ М.В.Добролюбова 23

Логіка створення ефективних звітів про дефекти

Алгоритм створення ефективного звіту про дефект:

Виявити дефект

Зрозуміти суть проблеми

Короткий опис	Детальний опис	Кроки відтворення
Обробляються файли поза SOURCE_DIR.	Іноді з незрозумілих причин додаток обробляє випадкові файли поза каталогом SOURCE_DIR. **Act:** обробляються окремі файли поза SOURCE_DIR. **Exp:** обробляються тільки файли, що знаходяться в SOURCE_DIR. **Req:** ДС-2.1.	На жаль, не вдалося виявити послідовність кроків, що призводять до появи цього дефекту.

Відтворюваність	Важливість	Терміновість	Симптом	Можливість обійти	Коментар
Іноді	Висока	Висока	Некоректна операція	Ні	

Поганий звіт, при написанні якого суть проблеми не зрозуміла

@ М.В.Добролюбова

24

Логіка створення ефективних звітів про дефекти

Алгоритм створення ефективного звіту про дефект:

Зрозуміти суть проблеми

Короткий опис	Детальний опис	Кроки відтворення
Додаток не розрізняє файли та символічні посилання на файли.	Якщо в каталог SOURCE_DIR помістити символічне посилання на файл, виникає така помилкова поведінка: а) Якщо символічне посилання вказує на файл усередині SOURCE_DIR, файл обробляється двічі, а в DESTINATION_DIR переміщається як сам файл, так і символічне посилання на нього. б) Якщо символічне посилання вказує на файл поза SOURCE_DIR, програма обробляє цей файл, переміщує символічне посилання та сам файл у DESTINATION_DIR, а потім продовжує обробляти файли в каталозі, в якому спочатку був оброблений файл. **Act:** програма вважає символічні посилання на файли самими файлами (див. подробиці вище). **Exp:** якщо в каталозі SOURCE_DIR програма виявляє символічне посилання, вона припиняє роботу з виведенням повідомлення «Symbolic link [ім'я символічного посилання] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.e.	1. У довільному місці створити таку структуру каталогів: /SRC/ /DST/ /X/ 2. Помістити в каталоги SRC та X кілька довільних файлів допустимого формату та розміру. 3. Створити у каталозі SRC два символічні посилання: а) на будь-який із файлів усередині каталогу SRC; б) на будь-який із файлів усередині каталогу X. 4. Запустити програму. Дефект: у каталог DST переміщені як файли, так і символічні посилання; вміст каталогу X оброблено та переміщено до каталогу DST.

Відтворюваність	Важливість	Терміновість	Симптом	Можливість обійти	Коментар
Завжди	Висока	Звичайна	Некоректна операція	Ні	Швидкий погляд на код показав, що замість is_file() використовується file_exists(). Схоже, проблема в цьому. Також цей дефект спричиняє спробу обробити каталоги як файли (див. BR-999.99). В алгоритмі обробки SOURCE_DIR явно є логічна помилка: за жодних умов програма не повинна обробляти файли, що знаходяться поза SOURCE_DIR, тобто щось не так з генерацією чи перевіркою повних імен файлів.

Хороший звіт, при написанні якого суть проблеми зрозуміла

@ М.В.Добролюбова

25

Логіка створення ефективних звітів про дефекти

Алгоритм створення ефективного звіту про дефект:

Відтворити дефект

Перевірити наявність опису знайденого вами дефекту у системі управління дефектами

Сформулювати суть проблеми у вигляді «що зробили, що отримали, що очікували отримати»

Переваги застосування формули «що зробили, що отримали, що очікували отримати»:

- прозорість і зрозумілість
- легкість верифікації дефекту
- очевидність для розробників
- позбавлення зайвої безглуздої комунікації
- простота

Заповнити поля звіту, починаючи з детального опису

Після заповнення всіх полів уважно перечитати звіт, виправивши неточності та додавши подробиці

Ще раз перечитати звіт, оскільки у пункті 6 ви точно щось пропустили

@ М.В.Добролюбова 26

Типові помилки при написанні звітів про дефекти

Перелік типових помилок оформлення та формулювань:

- погані короткі описи (summary)
- ідентичні короткий та детальний опис (summary і description)
- відсутність у докладному описі явного надання фактичного результату, очікуваного результату та посилання на вимогу
- ігнорування лапок, що призводить до спотворення сенсу
- загальні проблеми з формулюваннями фраз українською та англійською мовами
- зайві пункти на кроках відтворення
- копії екрана як «копії всього екрана повністю»
- копії екрана, на яких не позначено проблем
- копії екрана та інші артефакти, розміщені на сторонніх серверах
- відкладання написання звіту «на потім»
- пунктуаційні, орфографічні, синтаксичні та подібні до них помилки

@ М.В.Добролюбова 27

Типові помилки при написанні звітів про дефекти

Приклади типових помилок оформлення та формулювань

- *«Несподіване переривання».*
- *«Знайдено 19 елементів».*
- *«Пошук по всіх типах файлів».*
- *«Неінформативна помилка».*
- *«У програмі червоний шрифт».*
- *«Error when entering just the name of computer disk or folder».*
- *«No reaction to 'Enter' key».*

«Довідка в імені пункту головного меню «File» (зараз «Fille»)»

«додаток показує вміст OpenXML-файлів»

«запис зникає при наведенні миші»

- *«Пошук не працює натисканням кнопки Enter».*
- *«За замовчуванням у полі, де шукати стоїть +».*
- *«При пошуку файлів у великому каталозі програма ненадовго зависає».*
- *«При закритті помилки програма закривається».*
- *«The application doesn't work with the from keyboard by the user in the field "What to search"».*

Погано	Добре
1. Запустити програму. 2. Відкрити пункт меню «Файл». 3. Вибрати пункт меню «Новий». 4. Заповнити текстом не менше трьох сторінок. 5. Відкрити пункт меню «Файл». 6. Відкрити підпункт меню «Друк». 7. Відкрити вкладку «Параметри друку». 8. Вибрати у списку «Двосторонній друк» значення «Ні». 9. Роздрукувати документ на принтері, який підтримує дуплексний друк. Дефект: друк, як і раніше, двосторонній.	1. Створити або відкрити файл із трьома і більше непорожніми сторінками. 2. Вибрати «Файл» → «Друк» → «Параметри» → «Двосторонній друк» → «Ні». 3. Роздрукувати документ на принтері, який підтримує дуплексний друк. Дефект: друк, як і раніше, двосторонній.

@ М.В.Добролюбова

28

Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів

Перелік типових логічних помилок:

- вигадані дефекти
- віднесення розширених можливостей застосування до дефектів
- невірно вказані симптоми
- надмірно занижені (або завищені) важливість та терміновість
- концентрація на дрібницях на шкоду головному
- технічна безграмотність
- вказування на кроки відтворення інформації, неважливої для відтворення помилки
- відсутність у кроках відтворення інформації, що є важливою для відтворення дефекту
- ігнорування так званих «послідовних дефектів»

@ М.В.Добролюбова

29

Типові помилки при написанні звітів про дефекти

Приклади типових логічних помилок

• *«Кількість знайдених файлів не відповідає реальній глибині вкладеності каталогу».*

• *Короткий опис: «За замовчуванням вибрано каталог аудіо запису».*

• *Пояснення у докладному описі: «Каталог не може мати дати та часу створення».*

• *Очікуваний результат: «Додаток вірно розпізнав не підтримувану файлову систему та показав список файлів».*

• *Додаток не зберігав налаштування користувача у випадку, якщо в шляху до каталогу їх збереження були пробіли* (два і більше поспіль пробілів).

• *Додаток некоректно завершував роботу при відкритті файлів,* розмір яких не дозволяв прочитати їх повністю в оперативну пам'ять, доступний об'єм якої, своєю чергою, визначається значенням параметра memory_limit у налаштуваннях середовища виконання.

• *Додаток відображав неправильну статистику роботи користувачів,* якщо у системі був хоча б один користувач, роль якого не була явно вказана (NULL-значення в таблиці БД, неправильна робота підзапиту).

Погано	Добре
1. Створити на диску «J:» каталог «Data». 2. Розмістити у створеному каталозі «Data» файли, що додаються «song1.mp3» розміром 999.99 Kb і «song2.mp3» розміром 888.88 Kb. 3. У полі «Де шукати» вказати «J:\Data». 4. У списку «Що шукати» вибрати «Аудіофайли». 5. Натиснути кнопку «Шукати». Дефект: вказані у пункті 2 файли не знайдено.	1. У довільному місці на локальному диску розмістити один (або більше) файл із розширенням «.mp3». 2. Встановити параметри пошуку («Що шукати» → «Аудіофайли», «Де шукати» → місце розташування файлу(ів) з пункту 1). 3. Здійснити пошук. Дефект: програма не виявляє файли з розширенням «.mp3».

@ М.В.Добролюбова 30


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 7
«Види додатків»
@ М.В.Добролюбова

Мобільні додатки

Мобільний додаток (mobile application) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних додатків передвстановлені на самому пристрої або можуть бути завантажені на нього з онлайнових магазинів додатків, таких як App Store, BlackBerry App World, Google Play, 1mobile market, Windows Phone Store тощо, безкоштовно або за плату.

Різновиди мобільних додатків:

- кросплатформені
- нативні
- гібридні

Основні принципи тестування мобільних додатків:

- постійна мобільність
- знаходження Wi-Fi в екстремальних умовах
- переривання
- особливості операційних систем і апаратного забезпечення
- людський фактор

@ М.В.Добролюбова 2


Мобільні додатки
Мобільні операційні системи
ANDROID
iOS
BlackBerry
Windows Phone
@ М.В.Добролюбова
3

Мобільні додатки

Переваги та недоліки операційної системи Android

Переваги	Недоліки
Різноманіття додатків та ігор	Пристрій під управлінням ОС Android необхідно часто перезаряджати
Android – операційна система з відкритим вихідним кодом (розробка додатків, ігор, різноманітних поправок та оновлень спрощено; стрімко набирає обертів нова професія – програміст додатків для Android)	Проблеми сумісності. Нові версії операційної системи часто конфліктують зі знятими з продажу застарілими пристроями або пристроями, які випущені «невідомими китайськими» виробниками
Багатозадачність (без проблем працюють одночасно декілька додатків)	Звичайні користувачі, у яких в пріоритеті практичність і швидкість роботи, можуть бути незадоволені великою кількістю налаштувань
Оперативні оновлення (ведеться безперервна робота над покращенням функціоналу операційної системи, виправляються дефекти, вносяться зміни до інтерфейсу)	
Абсолютна незалежність від апаратного вмісту мобільного пристрою	
Широкі можливості індивідуалізації	
Можливість заміни/видалення додатків, встановлених за замовчуванням	

@ М.В.Добролюбова 4

Мобільні додатки

Переваги та недоліки операційної системи iOS

Переваги	Недоліки
Постійні оновлення та тривала підтримка застарілих пристроїв	Закрита файлова система
Оптимізація і великий вибір додатків магазин додатків App Store	Відсутнє пряме копіювання файлів (переміщення можливе лише за допомогою iTunes)
Магазин додатків App Store	Вартість додатків (користувачеві доводиться переплачувати)
Дизайн	В iOS все зав'язано на Internet (якщо у користувача відсутнє підключення, то він позбавляється багатьох функцій)
Багатозадачність	Працює лише на пристроях Apple
Акцент на надійність і якість	

@ М.В.Добролюбова 5

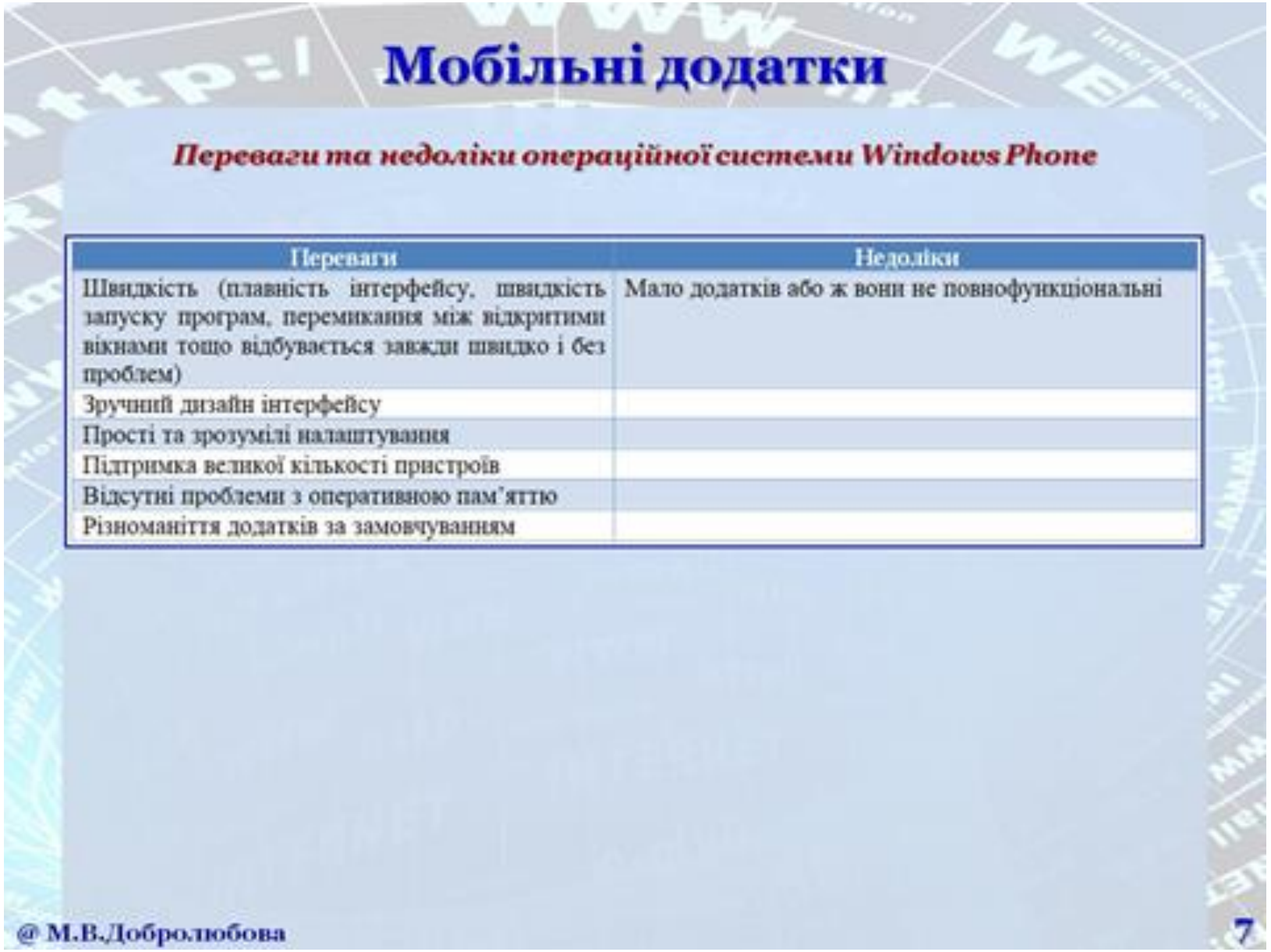
Мобільні додатки

Переваги та недоліки операційної системи BlackBerry

Переваги	Недоліки
Підтримка всіх сервісів компанії BlackBerry	Недостатньо розвинені мультимедійні функції
Шифрування даних, що забезпечує повну безпеку та конфіденційність при роботі в Internet, а також миттєвий обмін електронною поштою через сервер компанії	
Зручний перегляд електронних документів всіх популярних форматів	
Налаштування меню під кожного окремого користувача	

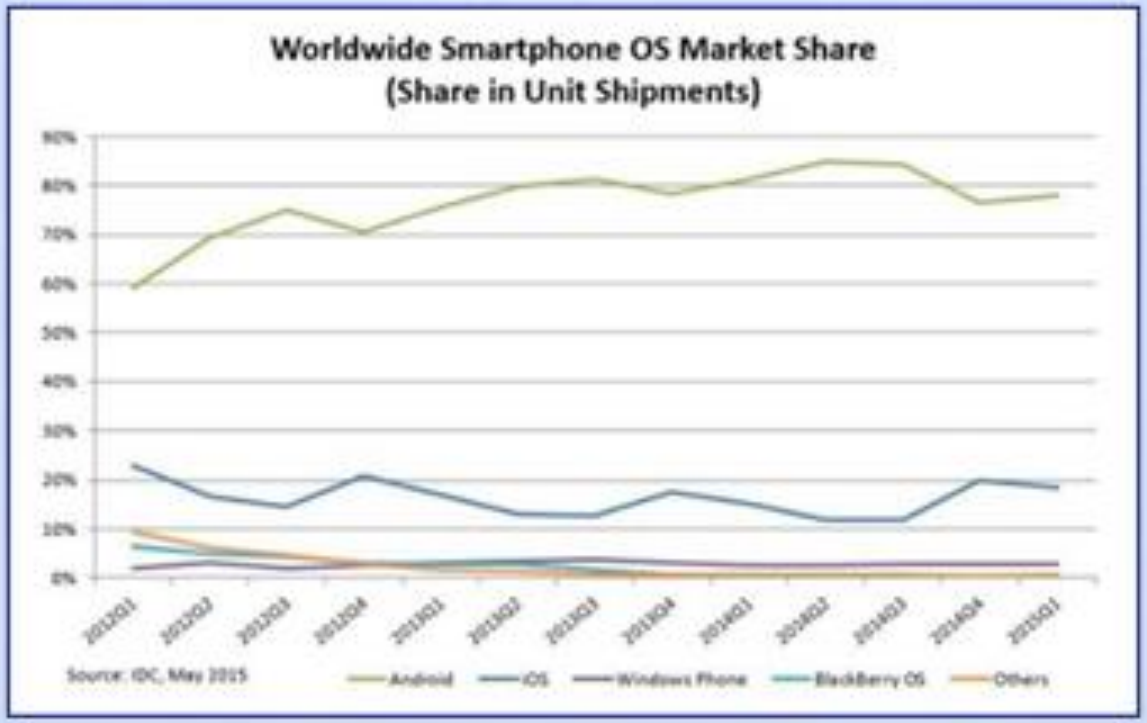
@ М.В.Добролюбова

6


Мобільні додатки
Переваги та недоліки операційної системи Windows Phone
Переваги
Недоліки
Швидкість (плавність інтерфейсу, швидкість запуску програм, перемикання між відкритими вікнами тощо відбувається завжди швидко і без проблем)
Мало додатків або ж вони не повнофункціональні
Зручний дизайн інтерфейсу
Прості та зрозумілі налаштування
Підтримка великої кількості пристроїв
Відсутні проблеми з оперативною пам'яттю
Різноманіття додатків за замовчуванням
@ М.В.Добролюбова
7

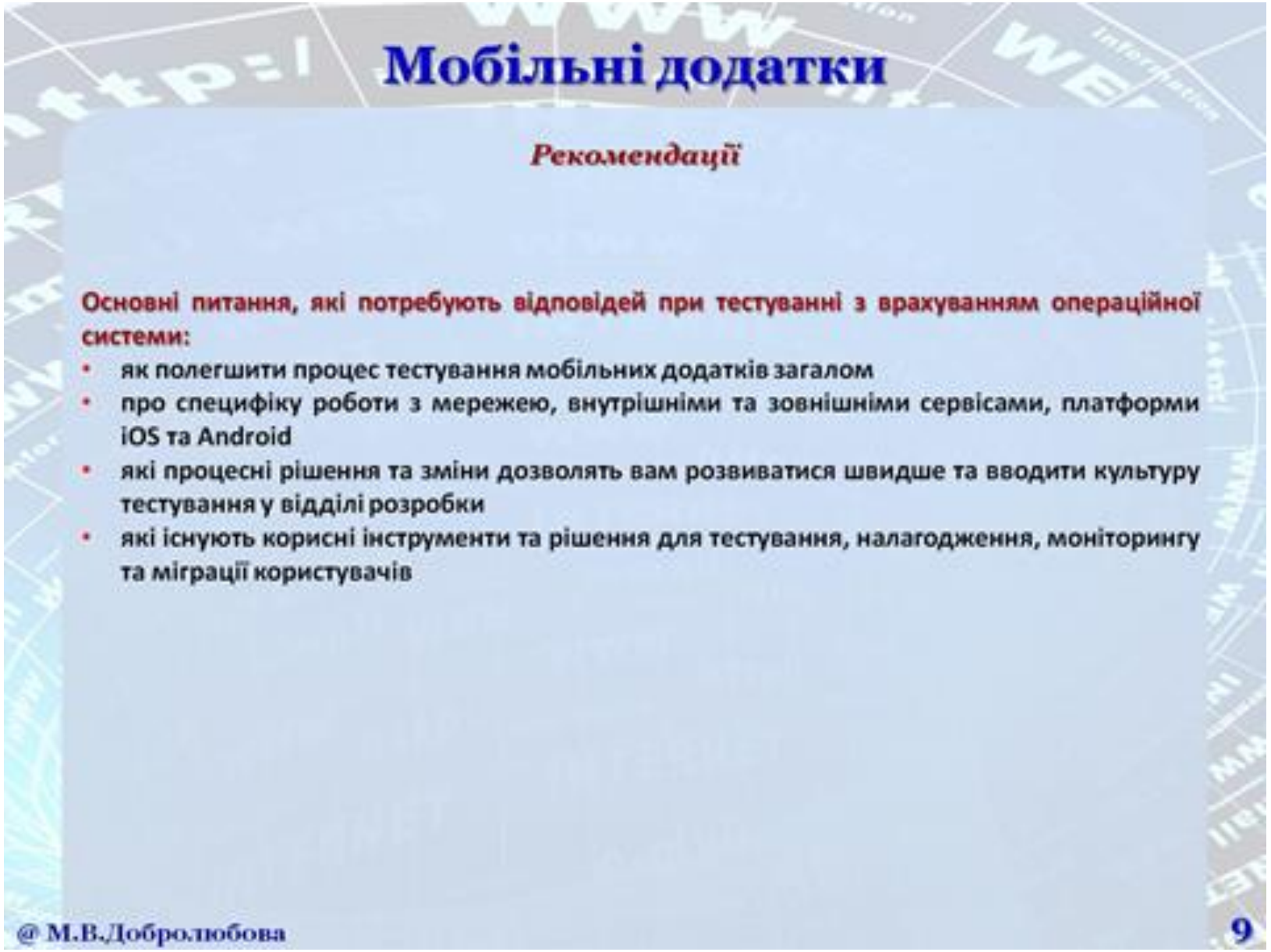
Мобільні додатки

Популярність мобільних операційних систем



@ М.В.Добролюбова

8


Мобільні додатки
Рекомендації
Основні питання, які потребують відповідей при тестуванні з врахуванням операційної системи:
• як полегшити процес тестування мобільних додатків загалом
• про специфіку роботи з мережею, внутрішніми та зовнішніми сервісами, платформи iOS та Android
• які процесні рішення та зміни дозволять вам розвиватися швидше та вводити культуру тестування у відділі розробки
• які існують корисні інструменти та рішення для тестування, налагодження, моніторингу та міграції користувачів
@ М.В.Добролюбова
9

Десктопні додатки

Десктопні додатки – це повнофункціональні програми, які працюють незалежно від інших додатків та вимагають наявності оператора. Для їх роботи необхідні достатні апаратні ресурси комп'ютера, сам додаток та набір функцій для роботи з додатком.

Інсталятор – це «звичайна» програма, основні функції якої – встановлення (інсталяція), оновлення та видалення (деінсталяція) програмного забезпечення.

Види тестування, що необхідно проводити на десктопних додатках окрім основних:

- тестування інсталяції
- тестування оновлення
- тестування деінсталяції

@ М.В.Добролюбова 10


Десктопні додатки
Порівняння основних особливостей тестування десктопних та Web-додіткв
Параметр
Desktop-додаток
Web-додаток
Доступ до мережі Internet
Не потрібно
Необхідний, виняток: деякі додатки можуть тимчасово працювати автономно
Встановлення / оновлення
Повинний бути розгорнутим або встановленим
Одноразове налаштування. Одна установка для всіх користувачів
Інтерфейс взаємодії
Стандартні інтерфейси, стандартна взаємодія
Різноманітний інтерфейс взаємодії. Плюси – різноманітність реалізації. Мінуси, складності – кросбраузерна сумісність. Вирішується застосуванням бібліотек JavaScitp, впровадженням стандартів
Сумісність з пристроями
Залежність від платформи. Виключення – кросплатформені додатки
Найчастіше – платформено незалежне
Анімація, графіка
Швидка, швидкий відгук
Відносно повільний відгук, пов'язаний із передачею даних через мережу
Медіа
Незначні проблеми з аудіо та відео
Є проблема – реалізація через Flash. Вирішується стандартом, який передбачає підтримку аудіо та відео на рівні браузера
Шрифти
В наявності лише ті шрифти, які встановлені у користувача
Будь-які шрифти – є можливість підвантаження необхідного шрифту через Internet
Пошук за контентом
Відсутній, якщо тільки не реалізований на рівні додатку
Так є. До того ж можна організувати свій пошук або скористатися сторонніми сервісами, наприклад, запитувати дані у Google
Надання доступу іншим комп'ютерам
Якщо додатково налаштувати
Більшість Веб-додатків одразу налаштовані на спільний доступ
Розробка
Наявність власних інструментів під кожну платформу. Часто під кожну платформу необхідно писати свою версію
Все виконується на сервері, не зачіпаючи користувача. Кросплатформенно, потрібен лише браузер. Інструменти, софт на сервері часто кросплатформений.
@ М.В.Добролюбова
11

Десктопні додатки

Порівняння основних особливостей тестування десктопних та Web-додіткв

Параметр	Desktop-додаток	Web-додаток
Масштаби	Повсюдне використання	Поки що web-додатки не такі популярні. Але темпи зростання популярності (у купі з «хмарами») великі. Вже зараз багато хто переходить до зберігання документів на Google Docs та інші сервіси
Тестування	Виконується QA, групою QA	По суті так само. Тільки відкритість (розташування в мережі) таких додатків дозволяє залучити більшу кількість QA. Сотні, тисячі, мільйони. В результаті більше покриття тестами і більш швидке виявлення вразливостей і некоректної роботи софту

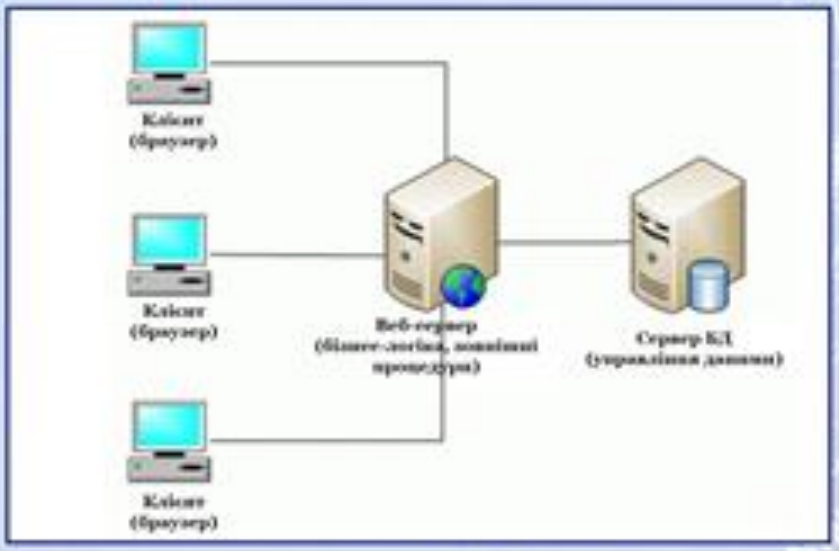
@ М.В.Добролюбова 12

Web-додатки

Web-додаток – це клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером web-сервер, що є по суті двома різними програмами, які необхідно тестувати як окремо, так і у зв'язці.

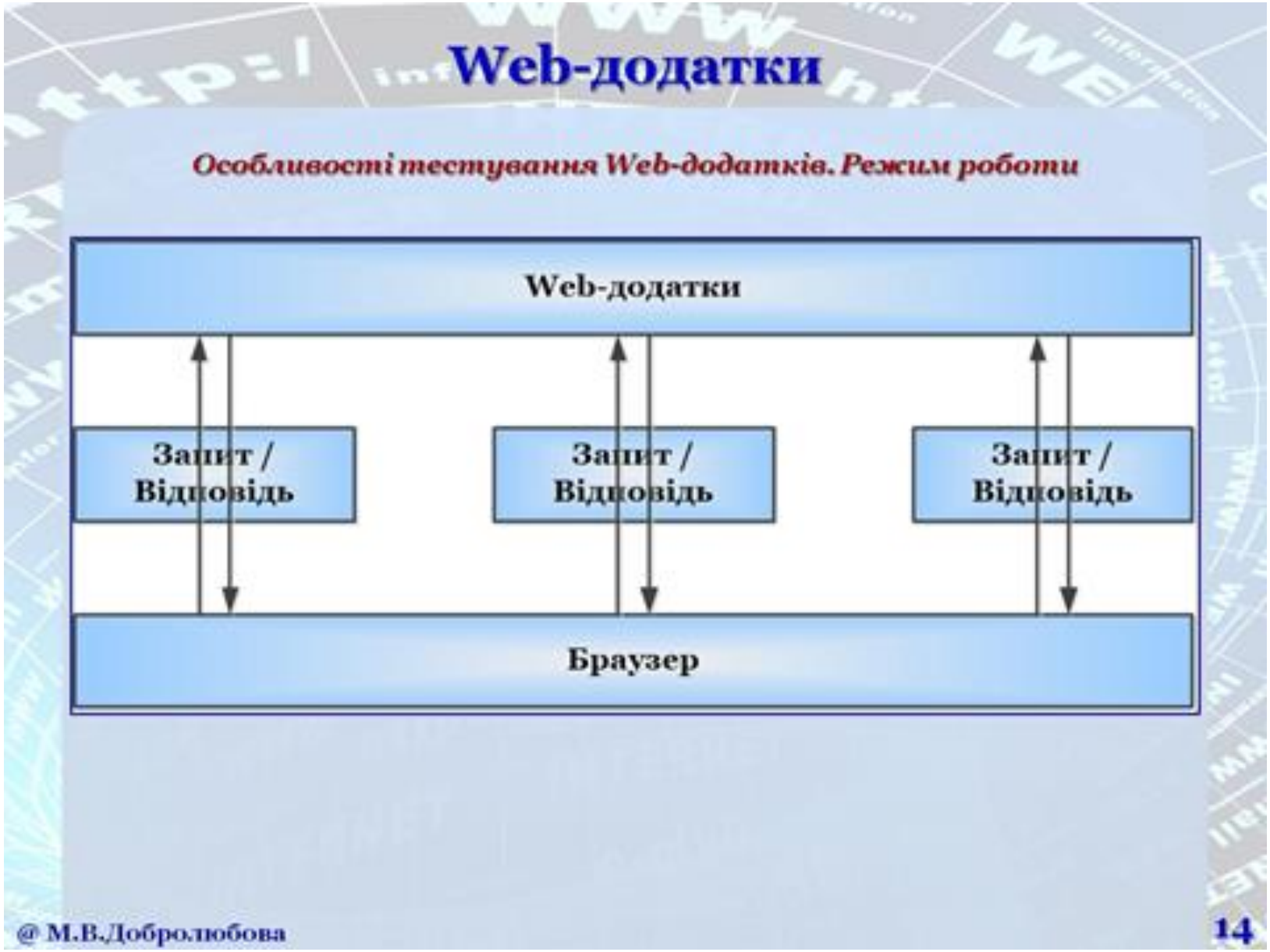
Особливості тестування Web-додатків:

- технологічні відмінності
- структурні відмінності
- відмінності режимів роботи
- відмінності формування інтерфейсу
- відмінності роботи з мережею
- відмінності запуску та зупинки
- різниця у кількості користувачів
- особливості збоїв та відмов
- відмінності в інсталяції
- відмінності в деінсталяції
- особливості середовища функціонування



Структура Web-додатку

@ М.В.Добролюбова 13


Web-додатки
Особливості тестування Web-додатків. Режим роботи
Web-додатки
Запит / Відповідь
Запит / Відповідь
Запит / Відповідь
Браузер
@ М.В.Добролюбова
14


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 8
«Клієнт-серверна архітектура»
@ М.В.Добролюбова

Загальні відомості

Веб-додаток – це клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер.

Клієнт-серверна архітектура – архітектура, яка визначає загальні принципи взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) і клієнти (споживачі цих функцій).

Клієнт-серверна технологія – практична реалізація клієнт-серверної архітектури.

Функції, орієнтовані на розв'язок різних підзадач:

- функції введення та відображення даних
- прикладні функції
- функції управління ресурсами

Компоненти мережевого додатку

@ М.В.Добролюбова

2

Загальні відомості. Дволанкова архітектура

Дволанкова клієнт-серверна архітектура – розподіл трьох базових компонентів між двома вузлами (клієнтом та сервером).

Розподіл компонентів в дволанковій архітектурі:

- сервер терміналів – розподілене надання даних
- файл-сервер – доступ до віддаленої бази даних та файлових ресурсів
- сервер БД – віддалене надання даних
- сервер додатків – віддалена програма

Клієнт – це браузер.
Веб-сервер – це сервер, який приймає HTTP-запити від клієнтів і видає HTTP-відповіді.
База даних – це інформаційна модель, що дозволяє впорядковано зберігати дані про об'єкт або групу об'єктів, що мають набір властивостей, які можна категоризувати.

Переваги дволанкової архітектури:

- проста, оскільки всі запити обслуговуються одним сервером

Недоліки дволанкової архітектури:

- менш надійна
- висуває підвищені вимоги до продуктивності сервера

@ М.В.Добролюбова

3

Загальні відомості

Моделі клієнт-серверної взаємодії

@ М.В.Добролюбова

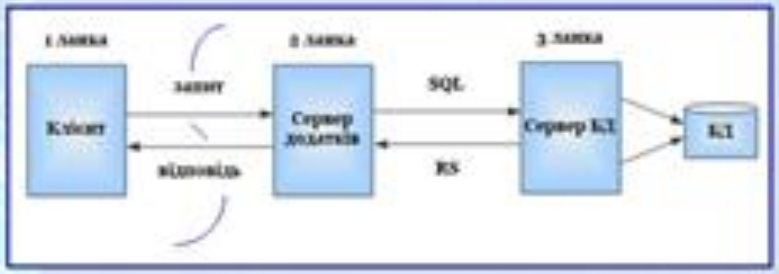
4

Загальні відомості. Триланкова архітектура

Триланкова архітектура – мережевий додаток розділений на дві або більше частин, кожна з яких може виконуватися на окремому комп'ютері. Третьою ланкою в триланковій архітектурі стає сервер додатків.

Розподіл компонентів в триланковій архітектурі:

- подання даних – на стороні клієнта
- прикладний компонент – на виділеному сервері програм
- управління ресурсами – на сервері БД, який і представляє дані, що запитуються



Переваги триланкової архітектури:

- високий ступінь гнучкості та масштабованості;
- висока безпека (захист можна визначити для кожного сервісу або рівня);
- висока продуктивність (завдання розподілені між серверами).

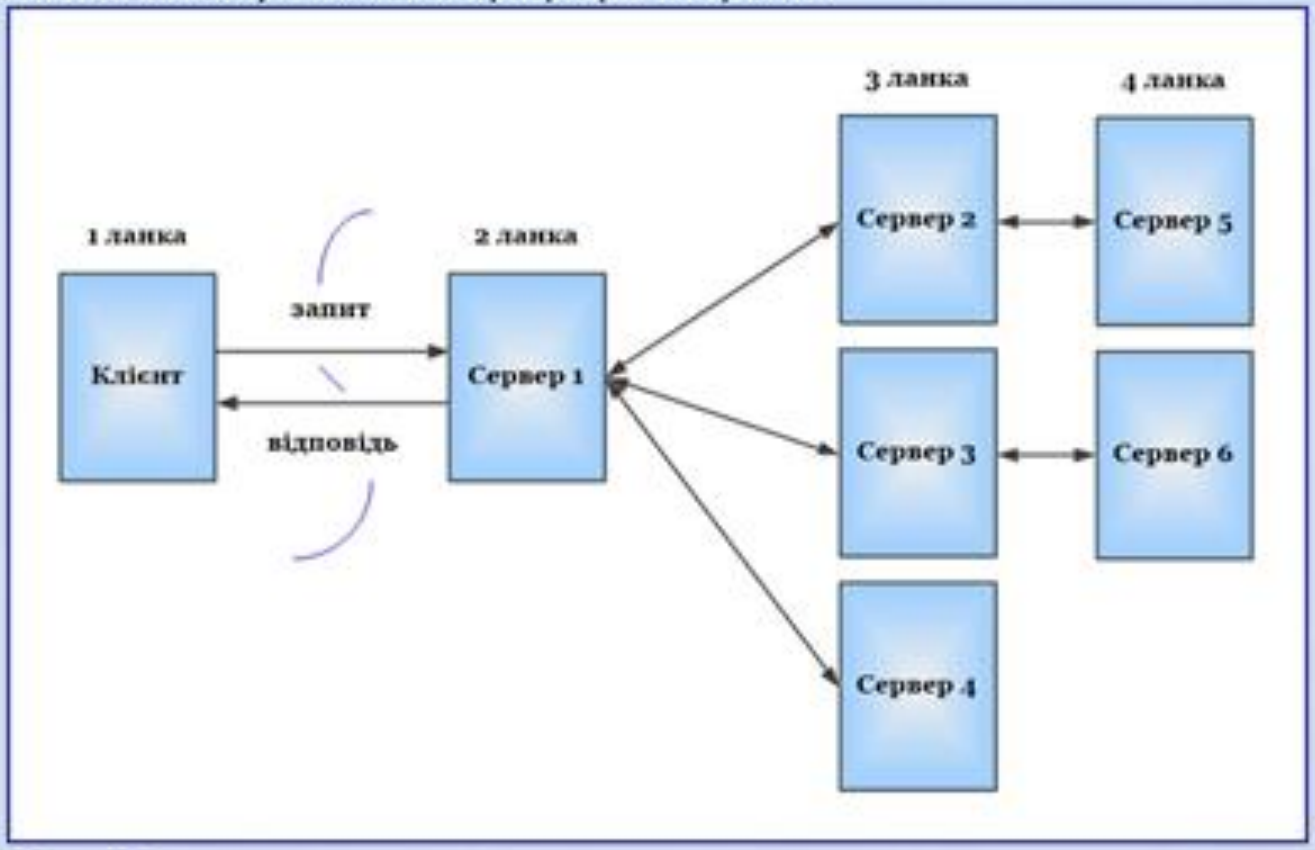
Недоліки триланкової архітектури:

- складна

@ М.В.Добролюбова 5

Загальні відомості. Багатоланкова (N-tier) клієнт-серверна архітектура

Триланкова архітектура може бути розширена до **багатоланкової** (N-tier, Multi-tier) шляхом виділення додаткових серверів, кожен з яких представлятиме власні сервіси та користуватиметься послугами інших серверів різного рівня.



@ М.В.Добролюбова

6

Загальні відомості

Клієнт-серверні технології

Web-сервери

сервери додатків

сервери баз даних

файл-сервери

проксі-сервер

файрволи (брандмауери)

поштові сервери

@ М.В.Добролюбова

7

Загальні відомості

Основні протоколи «rich»-клієнтів на базі XML

XAML (eXtensible Application Markup Language) – розроблений Microsoft, використовується в додатках на платформі .NET

XUL (XML User Interface Language) – стандарт, розроблений в рамках проєкту Mozilla, використовується, наприклад, у поштовому клієнті Mozilla Thunderbird або браузері Mozilla Firefox

Flex – мультимедійна технологія на основі XML, розроблена Macromedia/Adobe

@ М.В.Добролюбова

8

Шаблони Web-архітектури

Простий Web-клієнт – найчастіше застосовується з додатками Internet, де неможливо керувати конфігурацією клієнта. Для клієнта потрібен лише звичайний Web-браузер, здатний обробляти форми. Уся бізнес-логіка виконується на сервері.

Розширений Web-клієнт – значна частина бізнес-логіки виконується в системі клієнта. Зазвичай клієнт застосовує для роботи з бізнес-логікою динамічний HTML, аплети Java або елементи ActiveX. Обмін даними з сервером, як і раніше, здійснюється за протоколом HTTP.

Web-доставка – поряд із протоколом HTTP для підтримки розподіленої системи об'єктів, що включає і клієнт, і сервер, можуть застосовуватись такі протоколи як IIOP та DCOM. Web-браузер, головним чином, виступає як агент зберігання та доставки об'єктів у розподіленій системі.

@ М.В.Добролюбова 9

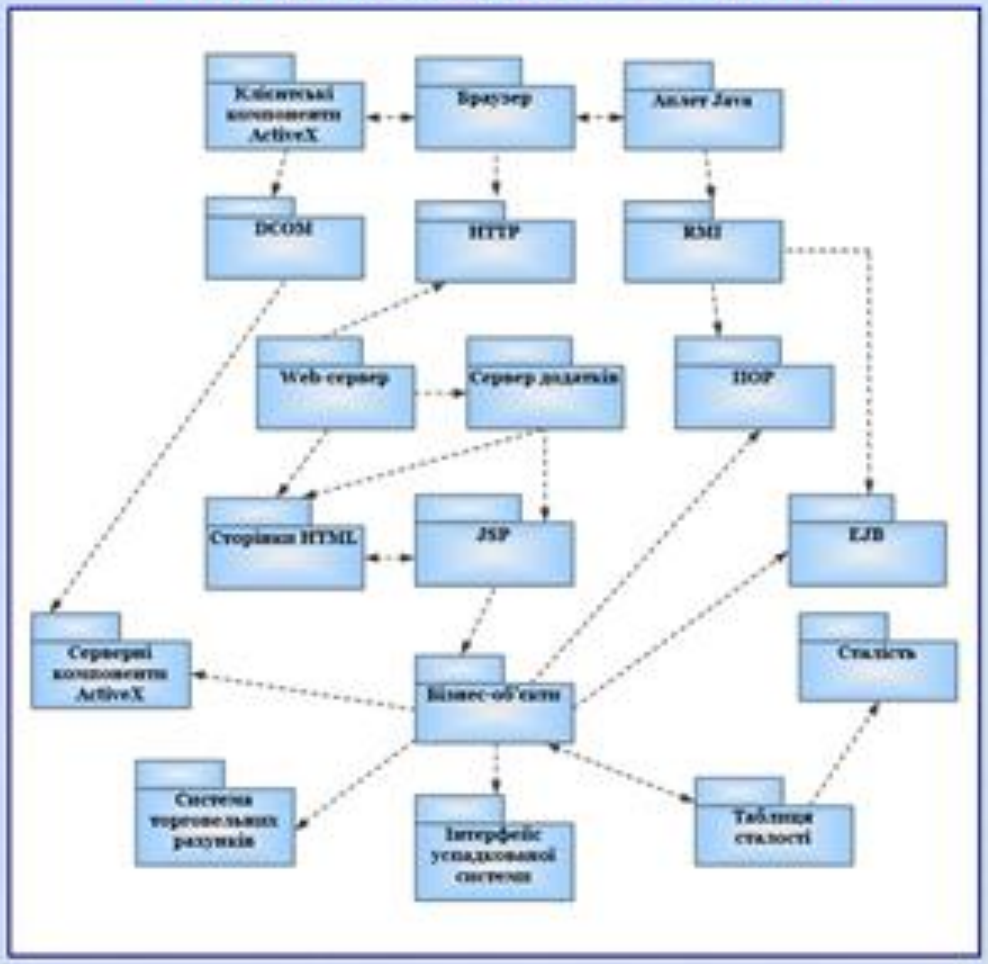

Шаблони Web-архітектури
Мінімальна архітектура простого Web-клієнта
HTTP
Web-сервер
+Ідентифікація
+Управління Cookie
Сервер додатків
+Стан сеансу
Браузер
+Cookie
Сторінки HTML
+Форма
+Елементи вводу
Сторінки сервера
+ASP
+Сервлети Java
+ISAPI
+NSAPI
+CQI
+JSP
@ М.В.Добролюбова
10


Шаблони Web-архітектури
Логічна схема простого Web-клієнта
HTTP
Web-сервер
+Ідентифікація
+Управління Cookie
Сервер додатків
+Стан сеансу
Система торговельних рахунків
Бізнес-об'єкти
Браузер
+Cookie
+DOM
Сторінки сервера
+ASP
+Сервлети Java
+ISAPI
+NSAPI
+CQI
+JSP
Сторінки HTML
+Форма
+Елементи вводу
Таблиця сталості
Інтерфейс успадкованої системи
Сталість
@ М.В.Добролюбова
11


Шаблони Web-архітектури
Схема архітектури розширеного Web-клієнта
Web-сервер
HTTP
Браузер
Сценарій кл. JavaScript VBScript
Сервер додатків
Елемент ActiveX
Аплет Java
Бізнес-об'єкти
Сторінки HTML
Документи XML
Сторінки сервера
Таблиця сталості
Інтерфейс успадкованої системи
Сталість
Система торговельних рахунків
@ М.В.Добролюбова
12

Шаблони Web-архітектури

Схема архітектури Web-доставки



@ М.В.Добролюбова

13


Структура мобільного додатку
Загальна структура додатку
Application
Environment
Core
Recycling Libraries
GUI
Resources
Шар локального кешування
View Controller
Network Command
Network Layer
Network Request
Local loading
Callback
Ядро
Core
Start Layer
Network layer
API layer
Network Cache Layer
Validation Items layer
Network Items layer
Data model
Local cache layer
Workflows layer
Local storage
@ М.В.Добролюбова
14


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 9
«Веб-клієнт»
@ М.В.Добролюбова

Загальні відомості

Браузер (веб-браузер, Web browser) – спеціальна програма, призначена для перегляду веб-сайтів.

Основна мета Інтернет-браузера – переведення коду, за допомогою якого комп'ютери створюють веб-сайти, у текст, графіку та інші функції веб-сторінок.

@ М.В.Добролюбова

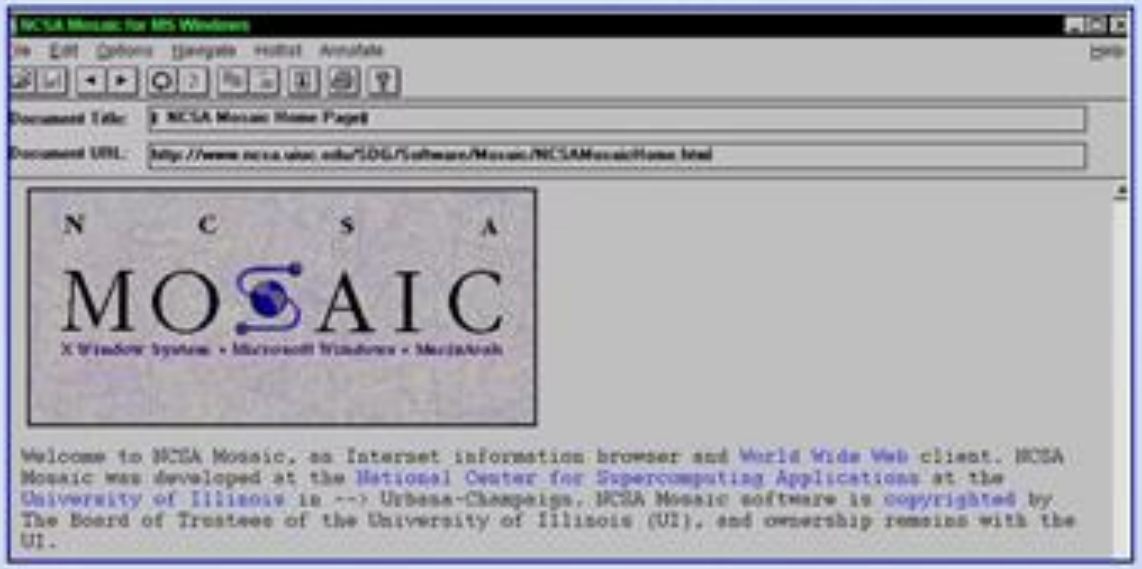
2

Загальні відомості

Перші веб-браузери. Історія виникнення

Nexus – перший веб-браузер (попередня назва **WorldWideWeb**), створений сером Тімом Бернерсом-Лі та випущений в 1990 році. Nexus дав людям можливість перегляду веб-сторінок. Через той факт, що Інтернет був тоді текстовим, малим і його не можна було використовувати без наявності технічних ноу-хау, коло людей, які мали можливість та інтерес користуватися ним, було обмеженим.

Mosaic – браузер з графічним інтерфейсом, випущений у 1992 році.



@ М.В.Добролюбова 3

Загальні відомості

Типи браузерів:

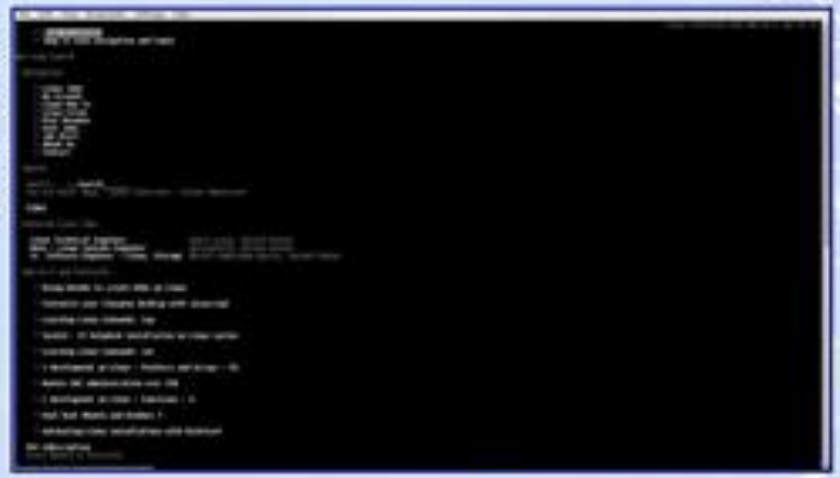
- браузер режиму командного рядка
- повноекранний браузер
- браузер із підтримкою мультимедіа
- браузери-доповнення
- офлайн браузери

@ М.В.Добролюбова

4

Загальні відомості

Браузери режиму командного рядка – найраніші браузери – не дають можливості переглядати текст та графіку, підтримують переміщення тільки з використанням цифрових адрес (IP), на даний час практично не використовуються.



Повноекранний браузер – текстовий браузер без підтримки мультимедійних ресурсів мережі Інтернет для перегляду лише тексту та посилань, використовуються досить рідко, але швидкість завантаження сторінок в них вражає. Один із найпопулярніших повноекранних браузерів – Lynx.

@ М.В.Добролюбова

5

Загальні відомості

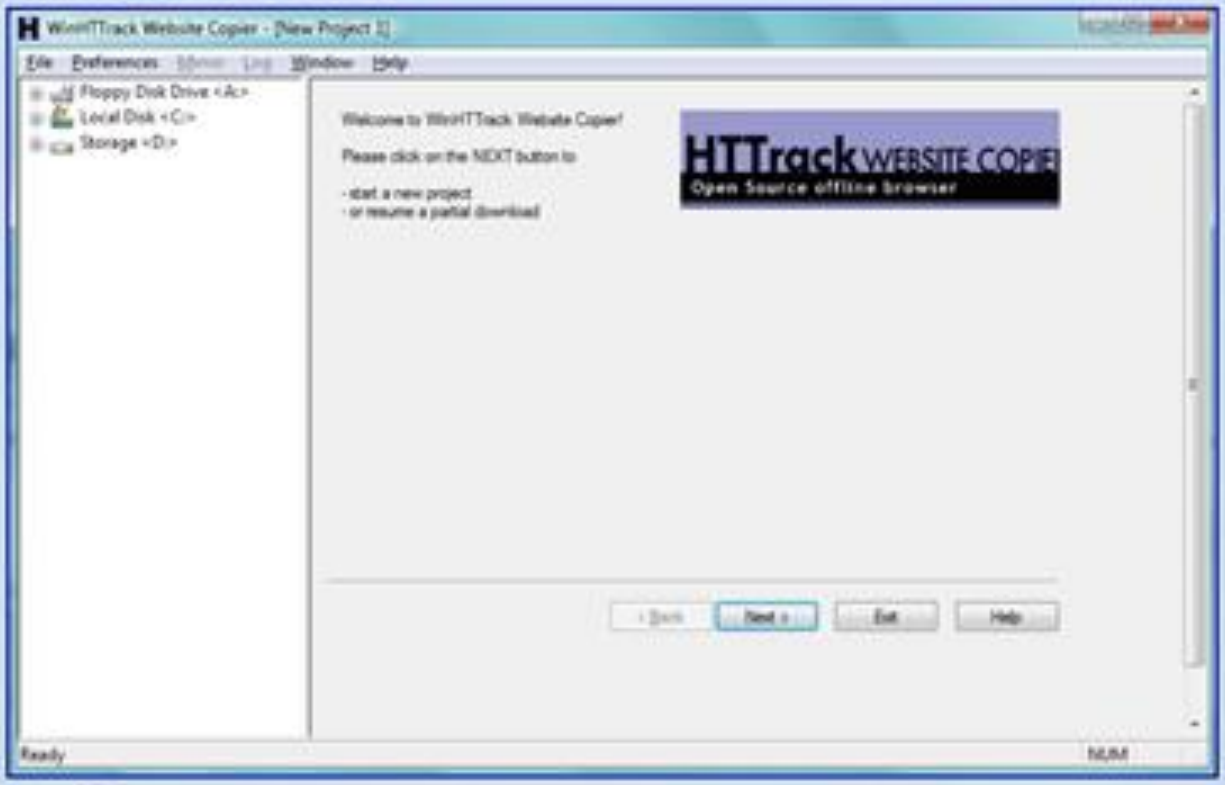
Браузер з підтримкою мультимедіа – найпоширеніші та найпопулярніші браузери на теперішній час – дозволяють працювати практично з усіма видами інформації, представленої в Інтернеті. Найчастіше використовувані: Internet Explorer, Opera, Mozilla, Google Chrome.

Браузери-доповнення – надбудови над повнофункціональними браузерами. Найчастіше розробниками доповнень використовується Internet Explorer. Надбудови використовують для відображення сайтів «движок» цього браузера. Тому їх можливості у цій галузі повністю ідентичні з Internet Explorer. Доповнення лише змінюють інтерфейс і додають деякі функції, які розробники з Microsoft обійшли своєю увагою.

@ М.В.Добролюбова 6

Загальні відомості

Офлайн браузери – це програми, які автоматично завантажують інформацію з Інтернету та зберігають її на локальному диску комп'ютера для подальшого перегляду і аналізу. Ідея, що лежить в основі роботи офлайн браузерів: користувач вказує адресу Web-сайту, що його цікавить, а програма у відповідь завантажує на його комп'ютер всі файли, які необхідні для автономного (тобто відключившись від Internet) перегляду цього сайту.



@ М.В.Добролюбова

7


Загальні відомості
Основи роботи веб-браузера
Google
Request
Response
Client
Server
CSS
HTML
JAVASCRIPT
@ М.В.Добролюбова
8


Загальні відомості
Компоненти браузера
User Interface
Browser Engine
Rendering Engine
Networking
Javascript Interpreter
UI Backend
Data Persistence
High-level Components of Browser
@ М.В.Добролюбова
9

Найбільш розповсюджені різновиди браузерів

Браузери з підтримкою мультимедіа:

- Google Chrome
- Mozilla Firefox
- Internet Explorer
- Opera
- Apple Safari
- Tor

@ М.В.Добролюбова 10

Найбільш розповсюджені різновиди браузерів

Google Chrome

Переваги:

- простий інтерфейс з обмеженою кількістю інструментів на головній сторінці
- висока швидкість завантаження веб-сторінок
- наявність корисних розширень для підвищення швидкості перегляду і безпеки
- нові вбудовані інструменти, такі як PDF Viewer, Language Translator тощо
- легка інтеграція з іншими програмами Google, такими як Gmail, Диск, AdSense, Реклама Google

Недоліки:

- використовує багато системної пам'яті
- порівняно менше можливостей для налаштування кнопок та меню

@ М.В.Добролюбова 11

Найбільш розповсюджені різновиди браузерів

Mozilla Firefox

Переваги:

- швидкий та якісно зроблений браузер
- підтримує всі стандарти W3C
- дозволяє працювати як з вікнами, так і з вкладками всередині кожного вікна
- підтримує плагіни
- має клієнтів для роботи з поштою, у news конференціях та IRC
- постійно оновлюється

Недоліки:

- вибагливий до ресурсів
- тривалий час завантаження програми
- помічені в закладках веб-сторінки нелегко знайти
- цільова сторінка заповнена безліччю рекомендованих посилань та оголошень

@ М.В.Добролюбова 12

Найбільш розповсюджені різновиди браузерів

Internet Explorer та Microsoft Edge

Переваги:

- найлегший браузер для Windows 10
- надійний та простий у використанні
- один із найшвидших браузерів для Windows 10 у порівнянні з іншими варіантами
- один із найбезпечніших браузерів для ПК з Windows завдяки попередженням користувачів про небезпечні веб-сайти під час перегляду
- найкращий браузер для завантаження веб-сайтів у вигляді програм

Недоліки:

- несумісний з комп'ютерами під управлінням попередніх версій Windows
- доступно менше розширень для перегляду, ніж в інших популярних браузерах

@ М.В.Добролюбова

13

Найбільш розповсюджені різновиди браузерів

Opera

Переваги:

- дозволяє прикріплювати ярлики до улюблених сайтів та найчастіше використовуваних налаштувань для підвищення ефективності
- дозволяє переглядати веб-сторінки за допомогою голосових команд
- дозволяє синхронізувати всі пристрої користувача, на яких він його використовує
- постачається із вбудованою функцією безпеки, яка дозволяє користувачам перевіряти веб-сайти на наявність шкідливого контенту та інших інфекцій

Недоліки:

- повільніший у порівнянні з іншими найпопулярнішими браузерами

@ М.В.Добролюбова

14

Найбільш розповсюджені різновиди браузерів

Apple Safari

Переваги:

- дозволяє користувачам організовувати закладки
- має перевірку правопису, щоб полегшити введення тексту та уникнути помилок друку
- блокує спливаючі вікна
- суворий з точки зору управління паролями та безпеки
- додає програвач мультимедійних файлів
- має яскраву 3D-«зовнішність» і відмінні функції для неспішного читання веб-сторінок та зручні інструменти на кшталт RSS або вбудованого рідера

Недоліки:

- безнадійно застарів у питаннях безпеки та функціональності
- велика кількість соціальних мереж і мультимедійних сервісів (YouTube, Google Map тощо) відмовляються працювати в браузері Safari
- не варто користуватися цим браузером там, де потрібна авторизація і, тим більше, є можливість онлайн-оплати послуг

@ М.В.Добролюбова 15

Найбільш розповсюджені різновиди браузерів

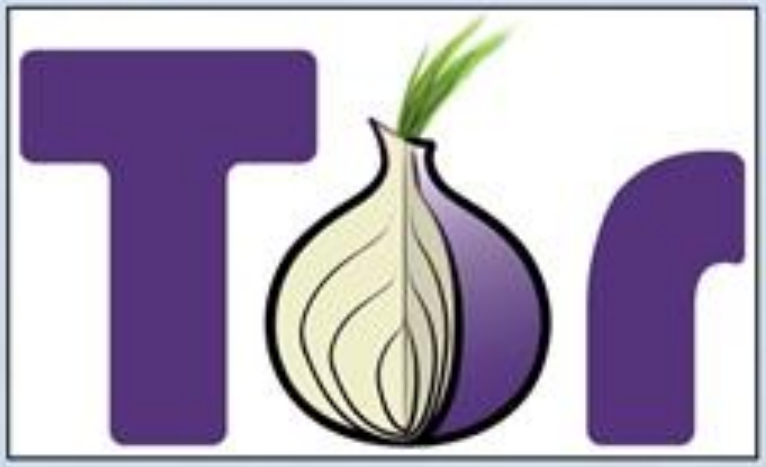
Tor

Переваги:

- відкритий вихідний код
- найкраща пошукова система, яка дбає про конфіденційність – Surf Deep, Dark і навіть заблоковані веб-сайти доступні без жодних обмежень із цим настільним браузером
- повна анонімність, можливість приховати вихідну IP-адресу
- можливість перегляду непроіндексованих сторінок в Яндекс, Bing та Google
- є браузери для iPhone та Android

Недоліки:

- не рекомендується використовувати з BitTorrent або будь-якими іншими торент-клієнтами
- знижена швидкість смуги пропускання
- застарілий інтерфейс

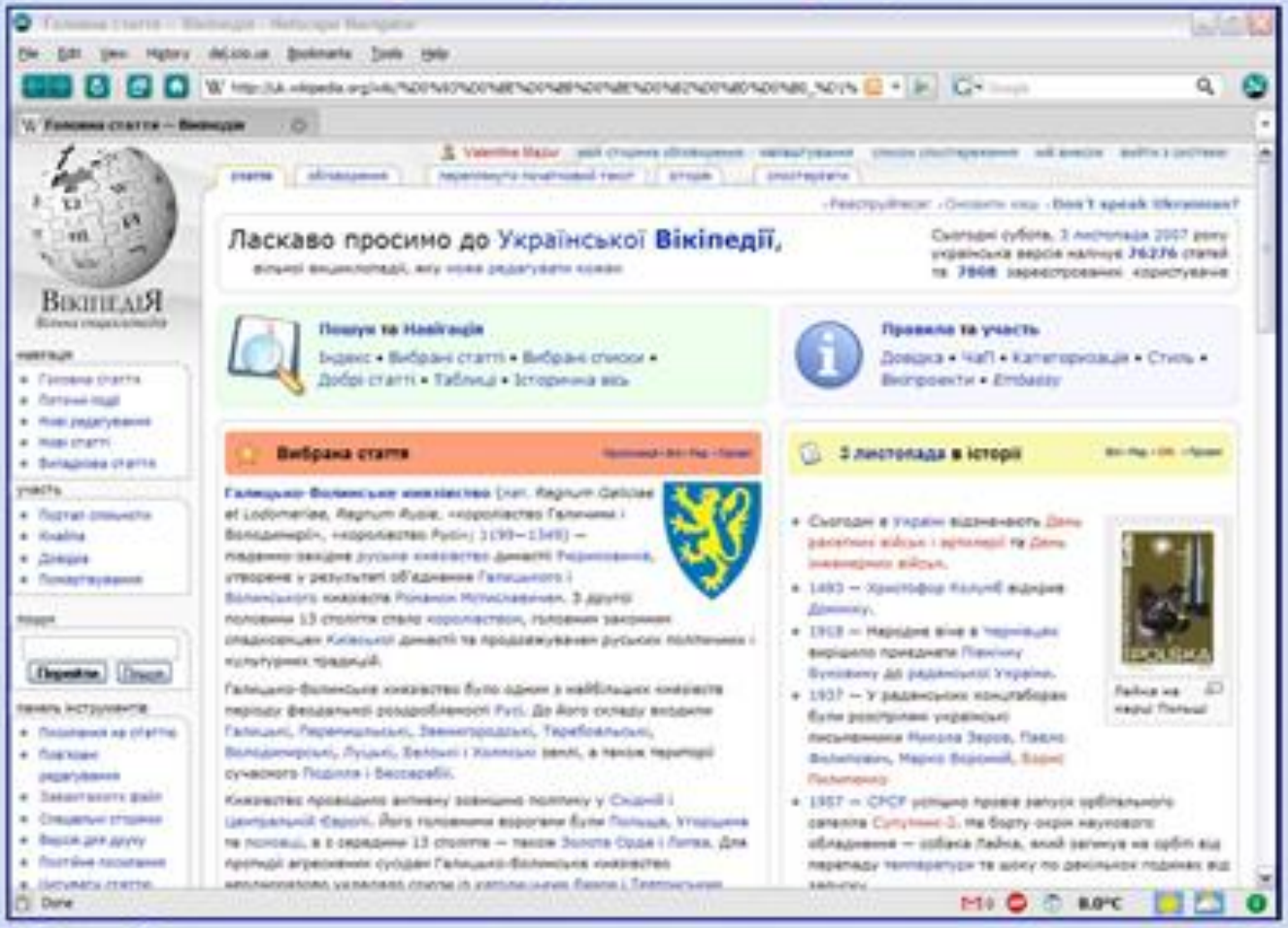


@ М.В.Добролюбова

16

Найбільш розповсюджені різновиди браузерів

Netscape Navigator. Час прощатися



@ М.В.Добролюбова

17

Найбільш розповсюджені різновиди браузерів

Додаткові браузери:

- Netsurf
- Konqueror
- Neoplanet
- MyIE2 (Maxthon)
- Internet Surfer
- Vivaldi
- Brave
- Chromium
- Comodo IceDragon







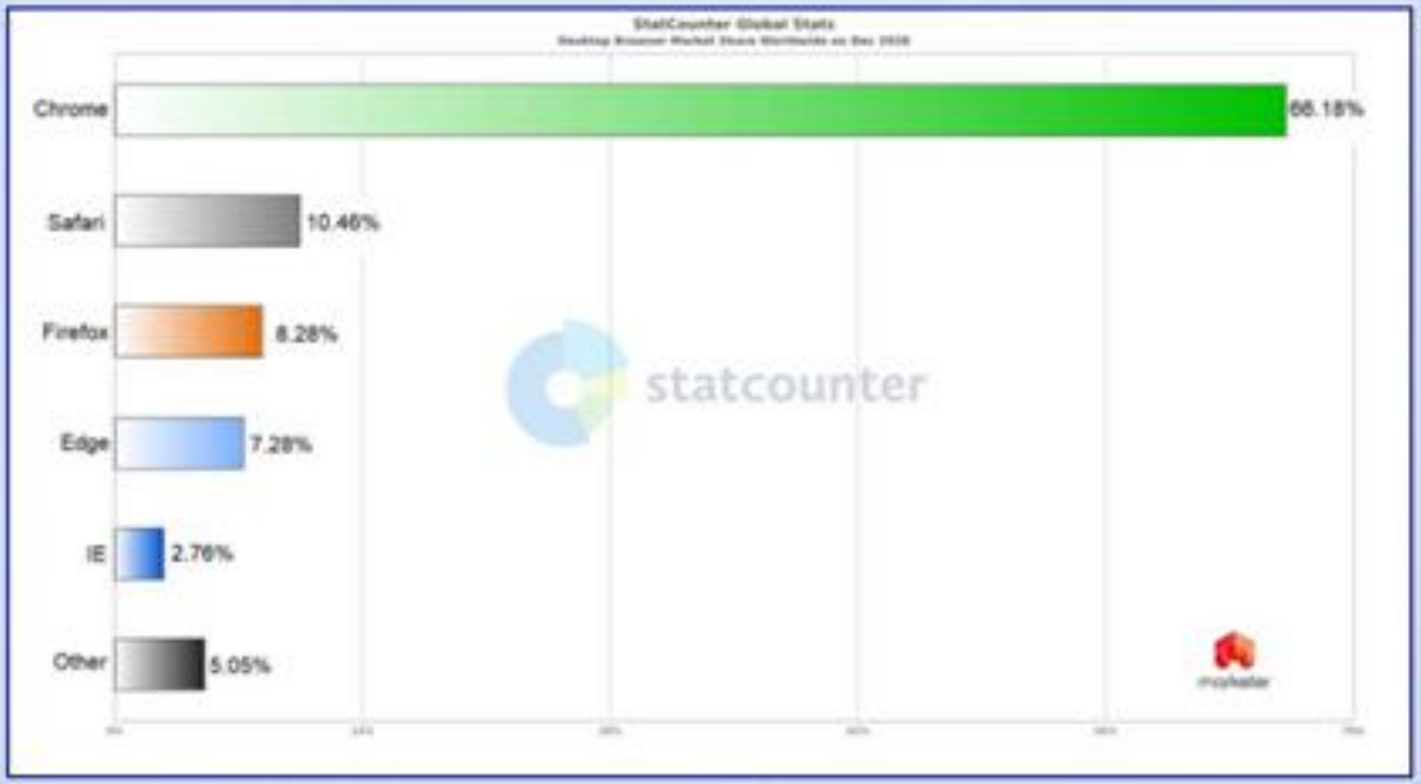


@ М.В.Добролюбова

18

Найбільш розповсюджені різновиди браузерів

Статистка браузерів, встановлених на ПК

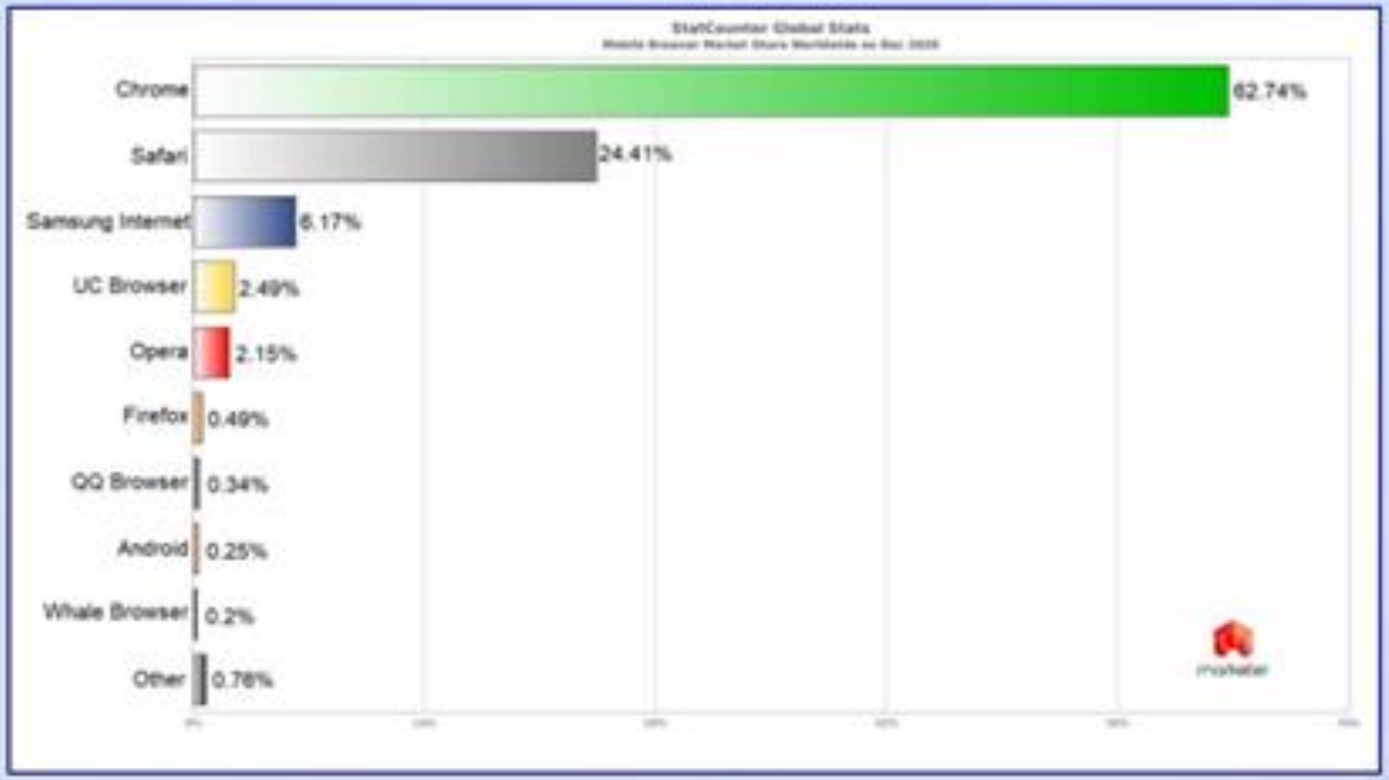


@ М.В.Добролюбова

19

Найбільш розповсюджені різновиди браузерів

Статистка браузерів, що використовуються на Android, iOS і інших мобільних

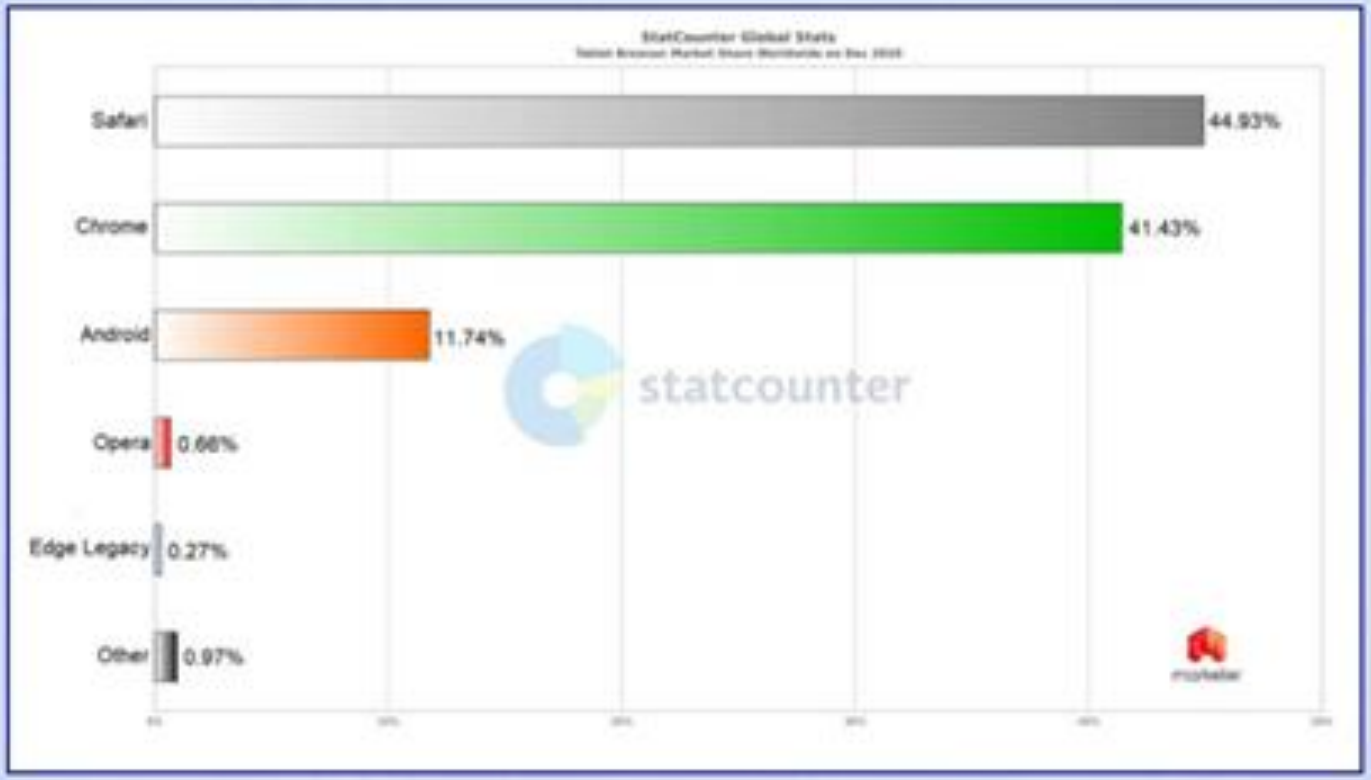


@ М.В.Добролюбова

20

Найбільш розповсюджені різновиди браузерів

Статистка браузерів, що використовуються на планшетах

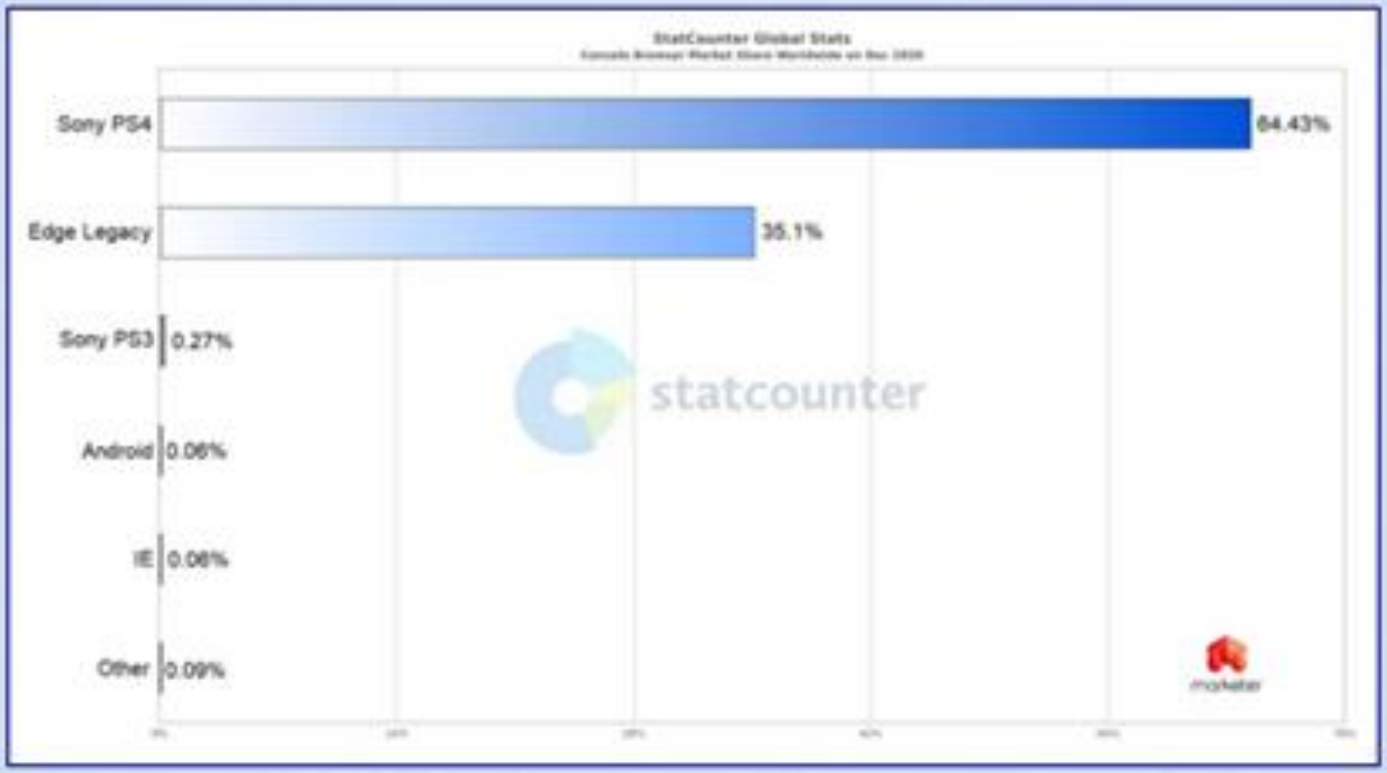


@ М.В.Добролюбова

21

Найбільш розповсюджені різновиди браузерів

Статистка браузерів, що використовуються на ігрових приставках

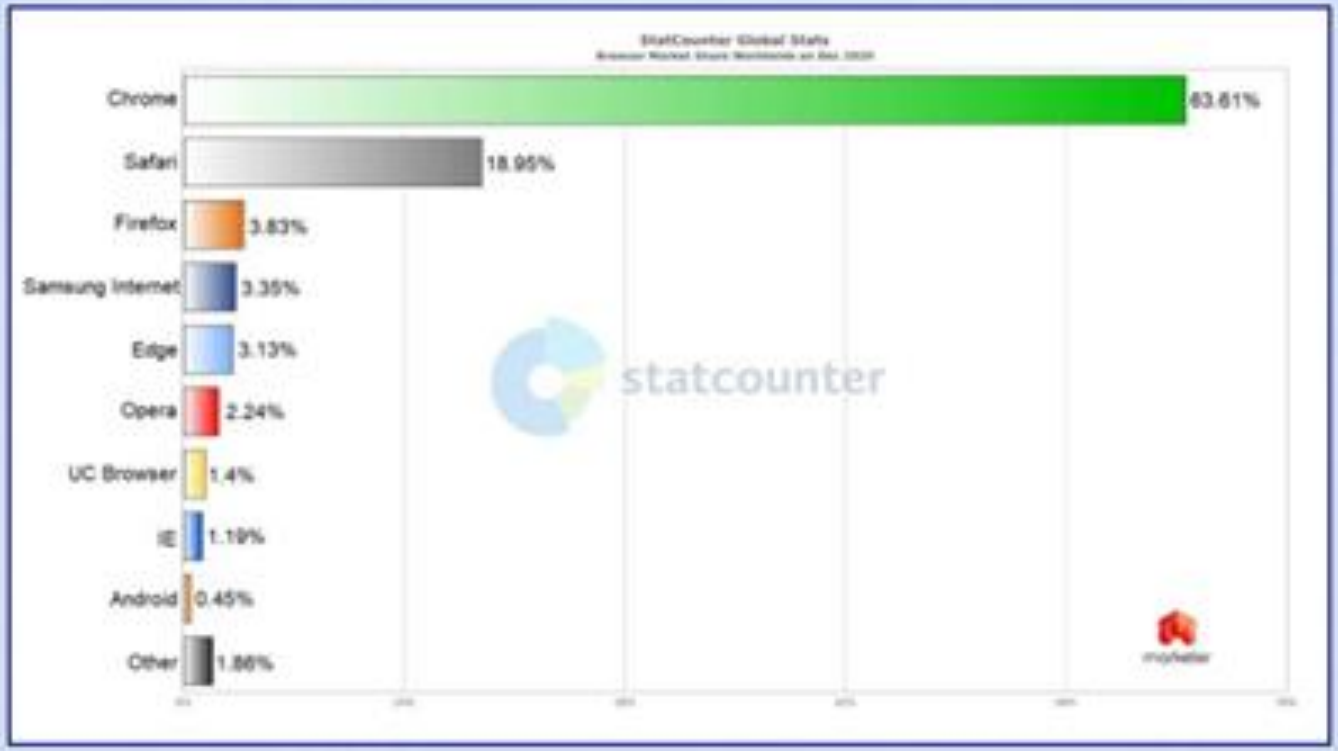


@ М.В.Добролюбова

22

Найбільш розповсюджені різновиди браузерів

Загальна статистика браузерів в світі



@ М.В.Добролюбова

23

Найбільш розповсюджені різновиди браузерів

Різновиди браузерів для мобільних пристроїв:

- для Android
- для iOS
- без ОС

@ М.В.Добролюбова

24


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 10
«Дані та їх вплив на програму»
@ М.В.Добролюбова

Управління доступом на основі ролей

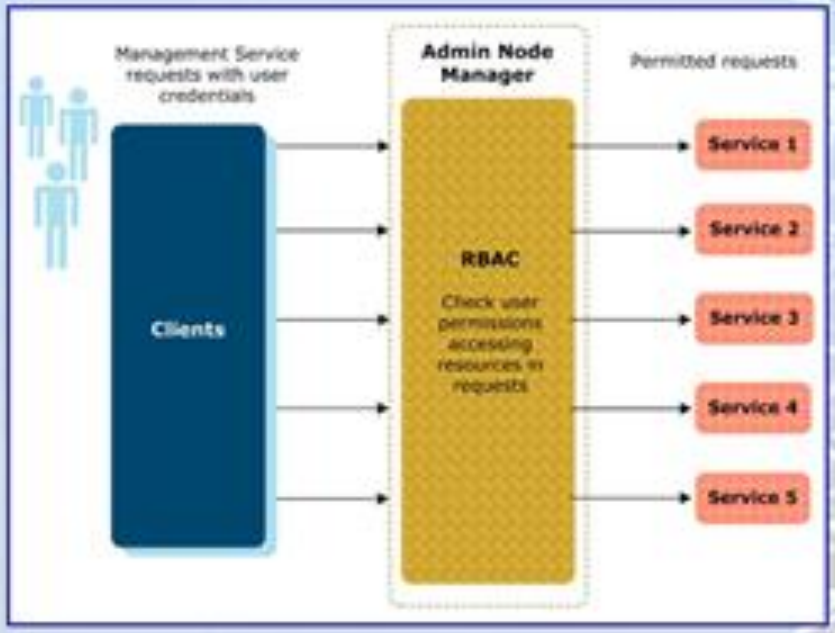
Role Based Access Control (RBAC) – це модель дозволів, що використовується в Microsoft Exchange Server. При використанні RBAC відпадає потреба у зміні та підтримці списків керування доступом (ACL).

Проблеми, пов'язані зі списками ACL:

- важко змінювати списки ACL без непередбачених наслідків
- важко зберігати зміни ACL після оновлення
- важко діагностувати проблеми, що виникають через нестандартне використання

Модель RBAC дозволяє:

- контролювати, які дії можуть здійснювати адміністратори та кінцеві користувачі
- більш точно зіставляти ролі, призначені користувачам та адміністраторам, зі своїми фактичними ролями в організації



@ М.В.Добролюбова

2

Управління доступом на основі ролей

Базова модель RBAC

S = Суб'єкт (Subject) = Людина або автоматизований агент (множина користувачів)

R = Роль (Role) = Робоча функція або назва, яка визначається на рівні авторизації (множина ролей)

P = Дозволи (Permissions) = Затвердження режиму доступу до ресурсу (множина прав доступу до об'єктів системи)

SE = Сесія (Session) = Відповідність між S, R та/або P

SA = Призначення суб'єкта (Subject Assignment)

PA (Permission Assignment): $R \rightarrow 2^P$ – функція, що визначає для кожної ролі множину прав доступу; при цьому для кожного $p \in P$ існує $r \in R$ така, що $p \in PA(r)$

RH = Частково впорядкована ієрархія ролей (Role Hierarchy). Ще один запис RH: ≥

- Один суб'єкт може мати кілька ролей
- Одну роль можуть мати кілька суб'єктів
- Одна роль може мати кілька дозволів
- Один дозвіл може належати кільком ролям

@ М.В.Добролюбова 3

Управління доступом на основі ролей

Можливі типи користувачів:

- Viewer
- Editor
- Worker
- Creator
- Professional
- Insights Analyst
- Storyteller

Різновиди ролей користувача:

- вьювер
- редактор даних
- користувач
- видавець
- адміністратор

@ М.В.Добролюбова

4

Поняття конфігурацій

Конфігурація – сукупність значень параметрів, які визначають роботу пристрою.

Конфігурація програмного забезпечення – це сукупність налаштувань програми, що задається користувачем, а також процес зміни цих налаштувань відповідно до потреб користувача.

Конфігурація додатку – це все, що може змінюватися між розгортаннями.

Різновиди збереження конфігурацій:

- константи в коді
- конфігураційні файли
- змінні оточення
- групування
- спеціальна база даних

@ М.В.Добролюбова

5

Часові пояси

Часовий пояс – це географічний регіон, у якому використовується єдиний місцевий час, встановлений урядом країни.

Адміністративний часовий пояс – ділянка земної поверхні, на якій відповідно до певного закону встановлено певний офіційний час.

Time zone – поняття, еквівалентне адміністративному часовому поясу.

GMT (Greenwich Mean Time) – середній час за Грінвічем.

Система UTC – це система, яка компенсує ефект обертання Землі.

DST (Daylight Saving Time) – це переведення годинника в літній період на одну годину вперед відносного стандартного часу.

IANA Time Zone Database – стандартна база даних, яка містить історичні дані про зміни стандартного часу та DST по всій земній кулі.

@ М.В.Добролюбова

6

Локалізації

Локалізація програмного продукту – приведення програмного продукту у відповідність із законами та іншими нормативно-правовими актами, стандартами, нормами і правилами, що діють в країні, для якої проводиться локалізація.

Локалізація (localization) – переклад і адаптація елементів інтерфейсу, допоміжних файлів та документації.

Локалізація бага – пошук першопричини виникнення помилки.

Локалізація програмного забезпечення – переклад на різні мови.

Те, що тестується в багатомовних програмах:

- адекватність перекладу
- пункти меню
- кнопки
- рисунки
- довгі фрази

Те, що важливо тестувати при локалізації:

- всі взаємодії з користувачем
- взаємодії з обладнанням
- переходи по сторінках і назад зі зміною мови
- правильний вибір мови за замовчуванням

@ М.В.Добролюбова

7


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 11
«Бази даних»
@ М.В.Добролюбова

Поняття бази даних

База даних – це сукупність взаємозалежних даних певної предметної галузі, збережених у пам'яті комп'ютера і організованих таким чином, що ці дані можуть бути використані для розв'язання різних завдань багатьма користувачами.

База даних – набір логічно пов'язаних даних спільного використання (і опис цих даних), призначений для задоволення інформаційних потреб.

База даних – сукупність зв'язаних даних, організована за певними правилами, які передбачають загальні принципи опису, збереження та маніпулювання даними, та незалежна від прикладних програм.

База даних – сукупність даних, організованих відповідно до концептуальної структури, що описує характеристики цих даних та взаємовідносини між ними, причому таке накопичення даних, яке підтримує одну або більше областей застосування.

@ М.В.Добролюбова

2

Поняття бази даних

Характерні ознаки БД:

- єдине сховище даних, яке визначається одноразово, а потім одночасно використовується багатьма користувачами
- загальний корпоративний ресурс
- зберігає не лише дані, а і їх опис

Вимоги до бази даних:

- ненадлишковість даних
- спільне використання даних
- можливість розширення бази даних
- простота роботи з базою даних
- ефективність доступу до бази даних
- цілісність бази даних
- захист даних

@ М.В.Добролюбова

3

Поняття бази даних

Реляційна база даних – це сукупність відношень, що містять всю інформацію, яка повинна зберігатися в БД.

Вимоги до реляційної бази даних:

- кожна таблиця складається з однотипних рядків і має унікальне ім'я
- рядки мають фіксоване число полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі)
- рядки таблиці обов'язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці
- стовпцям таблиці однозначно присвоюються імена і в кожному з них розміщуються однорідні значення даних
- повний інформаційний зміст бази даних представляється у вигляді явних значень даних і такий метод представлення є єдиним
- при виконанні операцій з таблицею її рядки і стовпці можна обробляти в будь-якому порядку, не зважаючи на їх інформаційний зміст

Недоліки реляційних баз даних:

- модель не надає достатніх засобів для представлення змісту даних
- для багатьох програм важко моделювати предметну галузь на основі плоских таблиць
- реляційна модель даних не пропонує апарату для поділу сутностей і зв'язків

@ М.В.Добролюбова 4

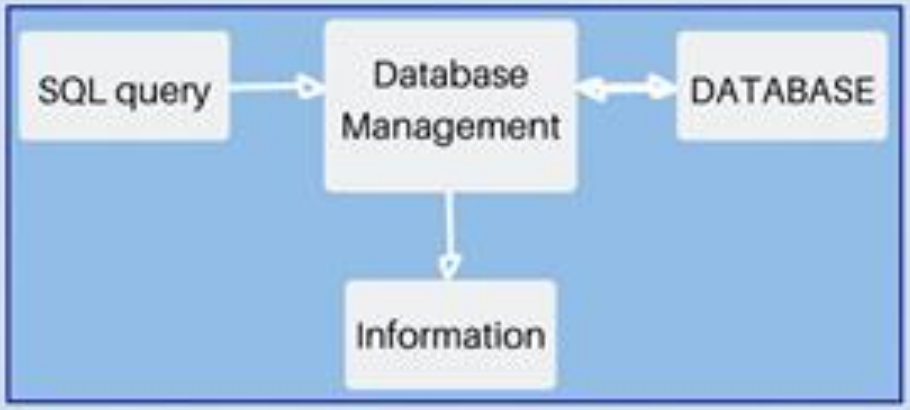
Необхідність знання SQL для тестувальника

Знання баз даних та SQL, необхідні тестувальнику:

- вміння розпізнати різні типи баз даних
- вміння підключитися до бази даних за допомогою різних клієнтів SQL-з'єднань
- розуміння відносин між таблицями бази даних, ключами та індексами
- вміння написання простого оператору вибору або SQL запиту на з'єднання
- вміння інтерпретувати складніші запити

Обгрунтування важливості SQL для тестування програмного забезпечення:

- SQL дає розробникам більш адекватний, так би мовити, зворотний зв'язок
- SQL допомагає зрозуміти, чи додаються на бекенд дані, які додаються до форми (на frontend)
- SQL допомагає отримати тестові дані
- SQL допомагає в автоматизації тестування



@ М.В.Добролюбова

5

Мови опису та маніпулювання даним

SQL – це стандартна комп'ютерна мова для управління базами даних та для обробки даних.

SQL – це засіб зв'язку між користувачем та системою управління базою даних.

SQL – це мова програмування, за допомогою якої можна звертатися до бази даних.

SQL – це приклад мови з трансформуючою орієнтацією, або ж мови, призначеної для роботи з таблицями з метою перетворення вхідних даних до необхідного вихідного виду.

Основні компоненти мови SQL:

- мова опису даних (DDL, Data Definition Language) – призначена для визначення структур бази даних
- мова маніпулювання даними (DML, Data Manipu1ation Langllage) – призначена для вибірки і поновлення даних

Способи використання мови SQL:

- інтерактивна робота – полягає у введенні користувачем окремих SQL-операторів
- впровадження SQL-операторів в програми на інших мовах

@ М.В.Добролюбова 6

Мови опису та маніпулювання даним

Характеристики мови SQL:

- не вимагає вказування методів доступу до даних (непроцедурна мова)
- підтримує вільний формат запису операторів (при введенні окремі елементи операторів не пов'язані фіксованими позиціями екрану)
- структура команд задається набором ключових слів, які представляють собою звичайні слова англійської мови (CREATE TABLE (створити таблицю), INSERT (вставити), SELECT (вибрати), тощо)
- може використовуватися широким колом користувачів

Типи даних мови SQL:

- текстовий (CHAR або TEXT)
- числовий (BYTE, INT, LONG, FLOAT, REAL)
- часовий (DATE, TIME, DATETIME)
- грошовий (MONEY)
- лічильник (COUNTER)
- логічний (LOGICAL)
- поле MEMO
- об'єкт (IMAGE)

@ М.В.Добролюбова

7

Мови опису та маніпулювання даним

Приклад таблиці з даними про співробітників

Табельний номер	ПІБ	Дата народження	E-mail	Посада	Відділ
1000	Петренко П.П.	19.05.1977	p.petrenko@test.it	Директор	Адміністрація
1001	Сидоренко С.С.	03.12.1987	s.sydorenko@test.it	Старший програміст	IT
1002	Іваненко І.І.	07.07.1971	i.ivanenko@test.it	Бухгалтер	Бухгалтерія
1003	Павленко П.П.	09.08.1994	p.pavlenko@test.it	Програміст	IT

Тип стовпця – характеристика, яка говорить про те, якого роду дані може зберігати даний стовпець.

Стовпці таблиці та типи даних:

- Табельний номер – ціле число
- ПІБ -- рядок
- Дата народження --дата
- E-mail -- рядок
- Посада -- рядок
- Відділ -- рядок

Коментарі мови SQL:

- однорядковий (--)
- багаторядковий(/* … */)

@ М.В.Добролюбова

8

Мови опису та маніпулювання даним

Оператори мови опису даних

Оператори мови опису даних – це оператори, призначені для створення опису, зміни опису і знищення об'єктів бази даних.

Види об'єктів в SQL:

- база даних (database)
- таблиця (table)
- стовпець (column)
- індекс (index)
- представлення (view)
- синонім (synonym)

Основні оператори мови SQL, призначені для опису даних:

- створення об'єктів: CREATE SCHEMA, CREATE DOMAIN, CREATE DATABASE, CREATE TABLE, CREATE VIEW, CREATE INDEX
- зміна існуючого об'єкта: ALTER DOMAIN, ALTER TABLE
- видалення об'єкта: DROP SCHEMA, DROP DOMAIN, DROP DATABASE, DROP TABLE, DROP VIEW, DROP INDEX

@ М.В.Добролюбова 9

Мови опису та маніпулювання даним

Оператори мови опису даних. Приклади

Створення простої бази даних (без додаткових параметрів): CREATE DATABASE Test

Видалення бази даних : DROP DATABASE Test

Перехід на потрібну базу даних: USE Test

Створення таблиць в БД з використанням пробілів та символів кирилиці:

```
CREATE TABLE [Співробітники](
[Табельний номер] int,
[ПІБ] nvarchar(30),
[Дата народження] date,
[E-mail] nvarchar(30),
[Посада] nvarchar(30),
[Відділ] nvarchar(30)
)
```

```
CREATE TABLE Employees(
ID int, Name nvarchar(30),
Birthday date,
Email nvarchar(30),
Position nvarchar(30),
Department nvarchar(30)
)
```

Перевизначення полів існуючої таблиці:

```
-- оновлення поля ID ALTER TABLE Employees ALTER COLUMN ID int NOT NULL
-- оновлення поля Name ALTER TABLE Employees ALTER COLUMN Name nvarchar(30) NOT NULL
```

@ М.В.Добролюбова 10

Мови опису та маніпулювання даним

Оператори мови опису даних. Приклади

Створення таблиці з обов'язковими для заповнення стовпцями ID та Name:

```
CREATE TABLE Employees(
ID int NOT NULL,
Name nvarchar(30) NOT NULL,
Birthday date,
Email nvarchar(30),
Position nvarchar(30),
Department nvarchar(30)
)
```

Зміна існуючого обов'язкового для заповнення стовпця необов'язковим:

```
ALTER TABLE Employees ALTER COLUMN Name nvarchar(30) NULL
або
ALTER TABLE Employees ALTER COLUMN Name nvarchar(30)
```

@ М.В.Добролюбова 11

Мови опису та маніпулювання даним

Оператори мови маніпулювання даними

Оператори мови маніпулювання даними – це оператори, призначені для заповнення таблиць даними, вибірки з них інформації за допомогою запитів, внесення змін в дані, які зберігаються в БД, видалення даних з таблиць.

Основні оператори мови SQL, призначені для маніпулювання даними:

- SELECT – вибірка даних з бази
- INSERT – вставка даних в таблицю
- UPDATE – оновлення (зміна) даних таблиці
- DELETE – видалення даних з таблиці

@ М.В.Добролюбова 12

Мови опису та маніпулювання даним

Оператори мови опису даних. Приклади

Вставка в поля таблиці ID, Position таDepartment даних :

```
INSERT Employees(ID,Position,Department) VALUES
(1000,N'Директор',N'Адміністрація'),
(1001,N' Старший програміст',N'IT'),
(1002,N'Бухгалтер',N'Бухгалтерія'),
(1003,N'Програміст',N'IT'
)
```

Найпростіші SQL запити

1. Виведе список ВСІХ баз

```
SHOW databases;
```

2. Виведе список ВСІХ таблиць у базі даних base_name

```
SHOW tables in base_name;
```

Прості SELECT (вибрати) запити до бази даних MySQL

1. Вибирає ВСІ дані у таблиці tbl_name

```
SELECT * FROM tbl_name;
```

2. Виведе кількість записів у таблиці tbl_name

```
SELECT count(*) FROM tbl_name;
```

3. Вибирає (SELECT) з (FROM) таблиці tbl_name ліміт (LIMIT) 3 записи, починаючи з 2

```
SELECT * FROM tbl_name LIMIT 2,3;
```

4. Вибирає (SELECT) ВСІ (*) записи з (FROM) таблиці tbl_name і сортує їх (ORDER BY) за полем ID по порядку

```
SELECT * FROM tbl_name ORDER BY id;
```

5. Вибирає (SELECT) ВСІ записи з (FROM) таблиці tbl_name і сортує їх (ORDER BY) за полем ID у зворотному порядку

```
SELECT * FROM tbl_name ORDER BY id DESC;
```

@ М.В.Добролюбова 13

Мови опису та маніпулювання даним

Оператори мови опису даних. Приклади

Прості SELECT (вибрати) запити до бази даних MySQL

6. Вибирає (SELECT) ВСІ (*) записи з (FROM) таблиці users і сортує їх (ORDER BY) за полем ID у порядку зростання, ліміт (LIMIT) перші 5 записів

```
SELECT * FROM users ORDER BY id LIMIT 5;
```

7. Вибирає всі записи з таблиці users, де поле fname відповідає значенню Gena

```
SELECT * FROM users WHERE fname='Gena';
```

8. Вибирає всі записи з таблиці users, де значення поля fname починається з Ge

```
SELECT * FROM users WHERE fname LIKE 'Ge%';
```

9. Вибирає всі записи з таблиці users, де fname закінчується на na і впорядковує записи у порядку зростання значення id

```
SELECT * FROM users WHERE fname LIKE '%na' ORDER BY id;
```

10. Вибирає всі дані із стовпців fname, lname таблиці users

```
SELECT fname, lname FROM users;
```

11. Якщо необхідно вивести ТІЛЬКИ список значень, що зустрічаються (щоб, наприклад, одна країна не виводилося декілька разів, а лише один), використовується DISTINCT. Виведе, із маси повторюваних значень Україна, США, Польща. Таким чином, з таблиці users колонки country будуть виведені УСЕ УНІКАЛЬНІ значення

```
SELECT DISTINCT country FROM users;
```

12. Вибирає всі дані рядків з таблиці users, де age = 18, 19 і 21

```
SELECT * FROM users WHERE age IN (18,19,21);
```

13. Вибирає МАКСИМАЛЬНЕ значення age в таблиці users

```
SELECT max(age) FROM users;
```

14. Вибере дані з таблиці users по полях name та age, ДЕ age приймає найменше значення

```
SELECT name, min(age) FROM users;
```

15. Вибере дані з таблиці users по полю name, ДЕ id НЕ РІВНИЙ 2

```
SELECT name FROM users WHERE id!='2';
```

@ М.В.Добролюбова 14

Мови опису та маніпулювання даним

Оператори мови опису даних. Приклади

Прості INSERT (новий запис) запити до бази даних MySQL

Робить новий запис в таблиці users, в поле name вставляє Сергій, а в поле age вставляє 25. Таким чином, до таблиці дописується новий рядок з даними значеннями. Якщо стовпців більше, то вони залишаться або порожніми, або з встановленими за замовчуванням значеннями

```
INSERT INTO users (name, age) VALUES ('Сергій', '25');
```

Прості UPDATE запити до бази даних MySQL

Є готовий рядок, але в ньому потрібно перезаписати параметр віку, оскільки він змінився з часом.

1. У таблиці users, ДЕ id дорівнює 3, значення поля age стає 18

```
UPDATE users SET age = '18' WHERE id = '3';
```

2. У таблиці users, ДЕ id дорівнює 3 значення поля age стає 18, а country Україна (перезапис 2 та більше полів)

```
UPDATE users SET age = '18', country = 'Україна' WHERE id = '3';
```

Прості DELETE (видалити запис) та DROP (видалити таблицю) запити до бази даних MySQL

1. Видаляє рядок з таблиці users, ДЕ id дорівнює 10

```
DELETE FROM users WHERE id = '10';
```

2. Видаляє всю таблицю tbl_name

```
DROP TABLE tbl_name;
```

@ М.В.Добролюбова 15

Мови опису та маніпулювання даним

Найбільш вживані в тестуванні оператори SQL

Оператори SQL, які найчастіше використовуються у тестуванні:

SELECT
UPDATE
INSERT
CREATE
ALTER
DROP
COMMIT
ROLLBACK
INNER JOIN
DISTINCT
IN

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary
                        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS Average
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	Average
Mark	Male	1000	1500
John	Male	2000	2000
Pam	Female	3000	3000
Sara	Female	4000	4000
Todd	Male	5000	5000
Mary	Female	6000	6000
Ben	Male	7000	7000
Jodi	Female	8000	8000
Tom	Male	9000	8833
Ron	Male	9500	9250

BETWEEN
WHERE
LIKE
ORDER BY
GROUP BY
HAVING
AVG
MIN
MAX
SUM
COUNT

Id	Name	Gender	Salary
1	Mark	Male	6000
2	John	Male	8000
3	Pam	Female	4000
4	Sara	Female	5000
5	Todd	Male	3000

Name	Salary	Gender	RowNumber	Rank	DenseRank
John	8000	Male	1	1	1
Mark	6000	Male	2	2	2
Sara	5000	Female	3	3	3
Pam	4000	Female	4	4	4
Todd	3000	Male	5	5	5

```
SELECT Name, Salary, Gender,
ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

@ М.В.Добролюбова

16


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 4. БАЗОВІ ПОНЯТТЯ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ТА БАЗИ ДАНИХ
Лекція 12
«Тест-дизайн»
@ М.В.Добролюбова

Поняття, цілі, завдання та навички, необхідні для тест-дизайну

Тест-дизайн – це етап процесу тестування програмного забезпечення, на якому проєктуються та створюються тестові випадки (тест-кейси), у відповідності до визначених раніше критеріїв якості та цілей тестування.

Тест-дизайн – процес перетворення загальних цілей тестування на реальні тестові умови та тестові приклади.

Тестове покриття – це одна з метрик оцінки якості тестування, що представляє собою щільність покриття тестами вимог, або виконуваного коду.

Тест-дизайнер – це фахівець, який повинен побудувати процес тестування всіх найважливіших елементів програмного продукту, використовуючи мінімально можливу кількість перевірок.

@ М.В.Добролюбова

2

Поняття, цілі, завдання та навички, необхідні для тест-дизайну

Цілі тест-дизайну:

- вигадати тести, які виявлять найбільш серйозні помилки продукту
- мінімізувати кількість тестів, необхідних для знаходження більшості серйозних помилок

Задачі тест-дизайну:

- проаналізувати вимоги до продукту
- оцінити ризики, можливі під час використання продукту
- написати достатню мінімальну кількість тестів
- розмежувати тести на приймальні, критичні, розширені

Необхідні навички:

- вміння розділяти систему на складові (робити декомпозицію)
- вміння збирати та аналізувати вимоги до продукту
- вміння розставляти пріоритети
- вміння формулювати свої думки (письмово та усно)
- знання технік тест-дизайну
- вміння застосовувати техніки тест-дизайну на практиці

@ М.В.Добролюбова 3

Техніки тест-дизайну

1. Розбиття на класи еквівалентності – це техніка, яка полягає в розбитті всього набору тестів на класи еквівалентності з подальшим скороченням числа тестів.

Мета техніки – не лише скорочення кількості тестів, а й збереження прийнятного тестового покриття.

Приблизний алгоритм використання техніки:

1. Визначити класи еквівалентності.
2. Вибрати одного представника кожного класу. На цьому кроці з кожного еквівалентного набору тестів обирається один тест.
3. Виконати тести. На цьому кроці виконуються тести від кожного класу еквівалентності.

Задача

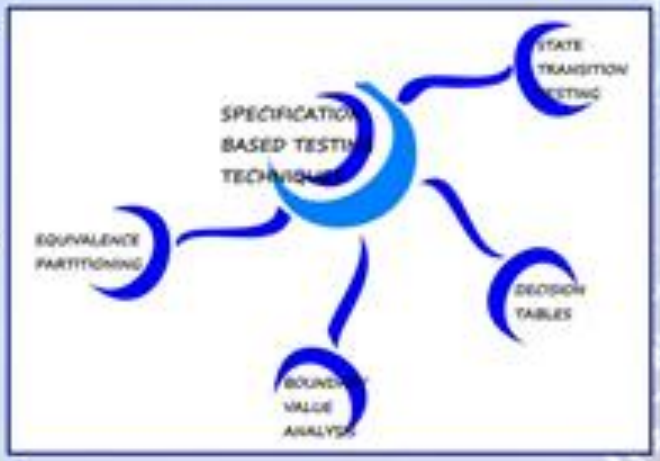
Розмір знижки в інтернет-магазині розраховується залежно від витраченої суми наступним чином:

Знижка = 10% при замовленні на суму більше 5000 грн.

Знижка = 5% при замовленні на суму менше 5000 грн.

Сума замовлення не може бути рівною 0.

На які класи еквівалентності можна розбити суму для перевірки правильності знижки, що надається?



Відповідь

Клас 1 = 0<сума<5000

Клас2 = сума > = 5000

@ М.В.Добролюбова

4

Техніки тест-дизайну

2. Аналіз граничних значень – це перевірка значеннями, що знаходяться на границях класів еквівалентності. Необхідно перевірити кожну границю класу еквівалентності з обох боків.

Задача

Поле "пароль" приймає кількість символів від 6 до 20. Якими будуть граничні значення?

Відповідь

Для нижнього – 5, 6

Для верхнього – 20, 21

3. Передбачення помилки – це коли тест-аналітик використовує свої знання системи та здатність інтерпретувати специфікацію на предмет того, щоб «передбачити» за яких вхідних умов система може видати помилку.

Приклад

Специфікація каже: "користувач повинен ввестикод".

Тест-аналітик буде думати: "Що, якщо я не введу код?", "Що, якщо я введу неправильний код?" і так далі. Це і є передбаченням помилки.

@ М.В.Добролюбова 5

Техніки тест-дизайну

4. Причина/Наслідок – це, як правило, введення комбінацій умов (причин) для отримання відповіді від системи (наслідок).

Приклад

Тестувальник перевіряє можливість додавати клієнта, використовуючи певну екранну форму. Для цього йому необхідно буде ввести кілька полів, таких як "Ім'я", "Адреса", "Номер Телефону", а потім, натиснути кнопку "Додати" – це "Причина".
Після натискання кнопки "Додати" система додає клієнта в базу даних і показує його номер на екрані – це «Наслідок".

5. Вичерпне тестування – це крайній випадок. В межах даної техніки необхідно перевірити всі можливі комбінації вхідних значень, і, в принципі, це дозволить знайти всі проблеми. На практиці застосування цього методу не представляється можливим, через величезну кількість вхідних значень.

6. Попарне тестування – техніка тестування, в якій замість перевірки всіх можливих комбінацій значень всіх параметрів перевіряються лише комбінації значень кожної пари параметрів. Полягає вона в наступному: формуються такі набори даних, в яких кожне тестоване значення кожного з параметрів, що перевіряються, хоча б один раз поєднується з кожним тестованим значенням всіх інших параметрів, що перевіряються.

@ М.В.Добролюбова 6

Техніки тест-дизайну

Приклад

Марка	Категорія замовлення	Місцезнаходження	ОС	Розрахунок	Доставка
HP	купівля	Київ	+	готівковий	поштою
HP	продаж	Харків	-	безготівковий	зустріч
~HP~	купівля	~Харків~	~~~	~безготівковий~	зустріч
Lenovo	купівля	Харків	+	безготівковий	поштою
Lenovo	продаж	Київ	-	готівковий	зустріч
~Lenovo~	продаж	~Київ~	~+~	~готівковий~	поштою
Asus	купівля	Київ	-	безготівковий	поштою
Asus	продаж	Харків	+	готівковий	зустріч

Утиліти для автоматизації pairwise testing:

- Pairwise Independent Combinatorial Testing, provided by Microsoft Corp
- IBM FoCuS – "Functional Coverage Unified Solution", provided by IBM
- ACTS – "Advanced Combinatorial Testing System",an agency of the US Government
- Hexawise
- Jenny
- Pairwise by Inductive AS
- VPTag free All-Pair Testing Tool

@ М.В.Добролюбова 7

Техніки тест-дизайну

7. Таблиця прийняття рішень – це зручний інструмент для фіксації вимог та опису функціональності програми. Таблицею зручно описувати бізнес-логіку програми і вони можуть служити чудовою основою для тест-кейсів.

Таблиці прийняття рішень описують логіку додатку на основі умов системи, що характеризують її стани. Кожна таблиця повинна описувати один стан системи.

Приклад

Умови	Тест 1	Тест 2	Тест 3	Тест 4
Старіше 18 років	так	так	ні	ні
Перебуває у шлюбі	ні	так	так	ні
Пенсіонер	так	так	так	ні
Має картку Visa	ні	ні	так	так
Дати знижку	60	40	30	10

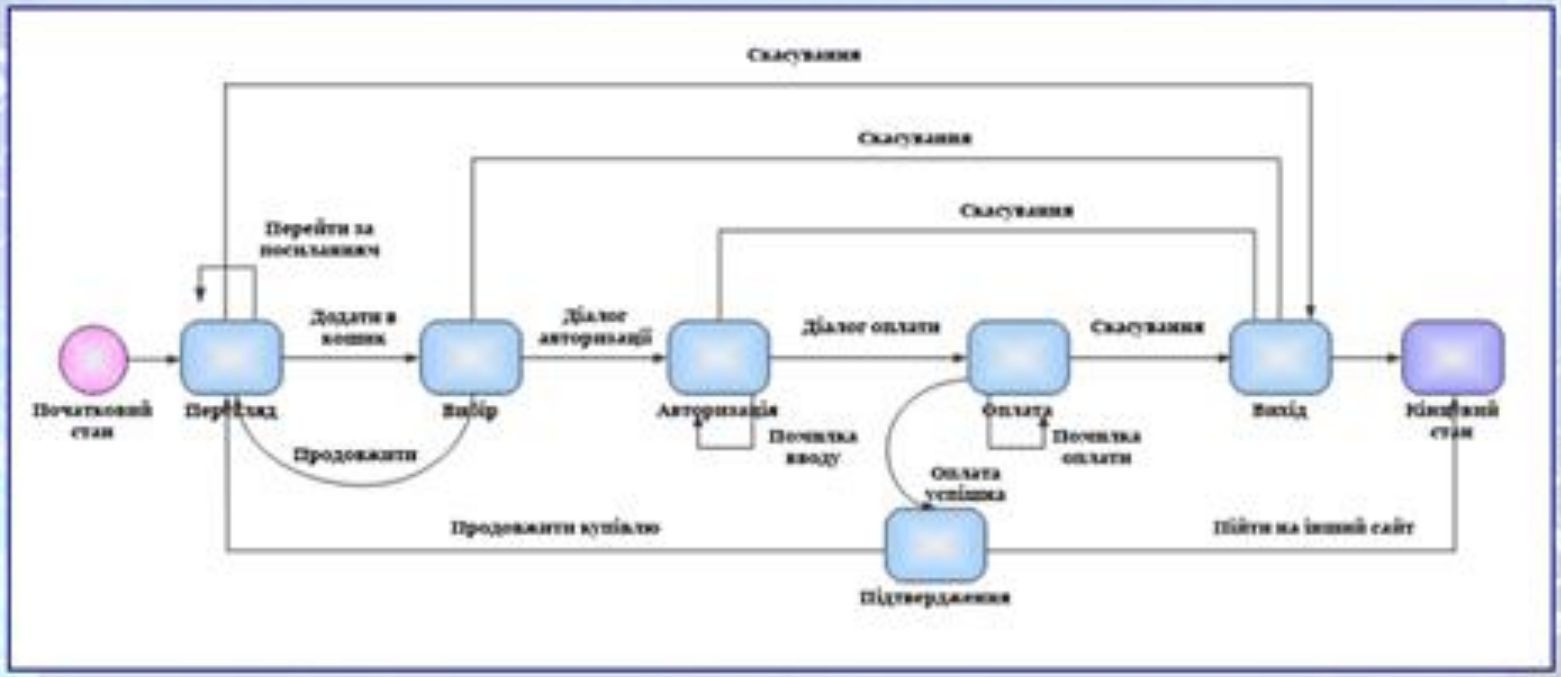
@ М.В.Добролюбова

8

Техніки тест-дизайну

8. Тестування переходів станів (state transition testing) – це розробка тестів методом чорної скриньки, в якій сценарії тестування будуються на основі виконання коректних та некоректних переходів станів.

Приклад



@ М.В.Добролюбова 9


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 5. ТРУДОВИТРАТИ
Лекція 13
«Планування та звітність»
© М.В.Добролюбова

Поняття планування та звітності

Планування (planning) – це безперервний процес прийняття управлінських рішень з орієнтацією на майбутнє та методичної організації зусиль, необхідних для виконання цих рішень.

Високорівневі завдання планування:

- зниження невизначеності
- підвищення ефективності
- покращення розуміння цілей
- створення основи для управління процесами

Основні причини планування проекту:

- усунення або зменшення невизначеності
- підвищення ефективності операції
- краще розуміння цілі
- створення основи для моніторингу та контролю роботи

Звітність (reporting) – збір та розповсюдження інформації про ефективність (включаючи звіти про стан, оцінку прогресу та прогнозування).

Високорівневі завдання звітності:

- збір, агрегація та надання у зручній для сприйняття формі об'єктивної інформації про результати роботи
- формування оцінки поточного статусу та прогресу (порівняно з планом)
- позначення існуючих та можливих проблем
- формування прогнозу розвитку ситуації та фіксація рекомендацій щодо усунення проблем та підвищення ефективності роботи

«Project Management: A Systems Approach to Planning, Scheduling, and Controlling, Harold Kerzner

© М.Б.Добролюбова *PMBOK (Project Management Body of Knowledge)* 2

Тест-план та звіт про результати тестування

Тест-план

Тест-план (test plan) – це документ, що описує обсяг, підхід, ресурси та графік запланованих заходів з тестування. Він визначає серед іншого тестові елементи, функції, які підлягають тестуванню, завдання тестування, хто виконуватиме кожне завдання, ступінь незалежності тестувальника, тестове середовище, методи розробки тесту та критерії входу та виходу, які будуть використані, а також обґрунтування їх вибору та будь-які ризики, що вимагають планування на випадок непередбачених обставин.

Низькорівневі завдання планування у тестуванні:

- оцінка обсягу та складності робіт
- визначення необхідних ресурсів та джерел їх отримання
- визначення розкладу, термінів та ключових точок
- оцінка ризиків та підготовка превентивних контрзаходів
- розподіл обов'язків та відповідальності
- узгодження робіт із тестування з діяльністю учасників проектної команди, що займаються іншими завданнями

Розширення до вимог якісного тест-плану:

- реалістичність
- узгодженість із загальним проектним планом та іншими окремими планами
- гнучкість

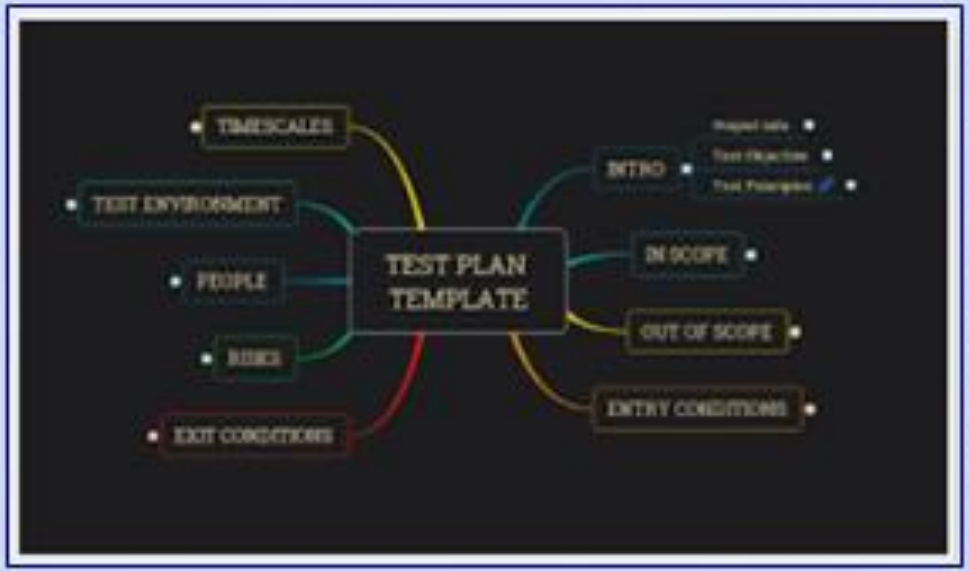
© М.Е.Добролюбова 3

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

- ціль
- області, що піддаються тестуванню
- області, які не піддаються тестуванню
- тестова стратегія
- тестовий підхід
- критерії
- ресурси
- розклад
- ролі та відповідальність
- оцінка ризиків
- документація
- метрики
- покриття



© М.В.Добролюбова

4

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

1. **Ціль (purpose)** –це граничнокороткий опис мети розробки додатку.
2. **Області, що піддаються тестуванню (features to be tested)** – це список функцій та/або нефункціональних особливостей додатку, які будуть піддаватися тестуванню.
3. **Області, які не піддаються тестуванню (features not to be tested)** – це перелік функцій та/або нефункціональних особливостей додатку, які не будуть піддаватися тестуванню.
4. **Тестова стратегія (test strategy)** – це високорівневий опис тестових рівнів, які необхідно виконати, і тестування на цих рівнях для організації або додатку.
5. **Тестовий підхід (test approach)** – це реалізація тестової стратегії для конкретного проєкту. Зазвичай він включає в себе рішення, прийняті на основі мети (тестового) проєкту та проведеної оцінки ризику, вихідні точки щодо процесу тестування, методи проєктування тестів, які потрібно застосувати, критерії виходу та типи тестів, які необхідно виконати.
6. **Критерії (criteria)** включають в себе декілька різновидів.

Види критеріїв:

- приймальні критерії, критерії якості
- критерії початку тестування
- критерії призупинення тестування
- критерії відновлення тестування
- критерії завершення тестування

© М.Е.Добролюбова

5

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

6. Критерії (criteria) включають в себе декілька різновидів.

Прийнятні критерії, критерії якості (acceptance criteria) – це критерії виходу, яким повинен задовольняти компонент або система, щоб бути прийнятими користувачем, клієнтом або іншим уповноваженим органом.

Критерії початку тестування (entry criteria) – це набір загальних і специфічних умов, що дозволяють процесу йти вперед із визначеним завданням. Мета критеріїв початку тестування – запобігти запуску завдання, яке спричинило б більше (марних) зусиль у порівнянні з зусиллями, необхідними для видалення невдалих критеріїв входу.

Критерії призупинення (suspension criteria) – це критерії, які використовуються для (тимчасової) зупинки всіх або частини тестових заходів для елементів тестування.

Критерії відновлення тестування (resumption criteria) – це критерії, які використовуються для перезапуску всіх або частини тестових заходів, що були призупинені раніше.

Критерії завершення тестування (exit criteria) – набір загальних і специфічних умов, узгоджених із зацікавленими сторонами для дозволу офіційного завершення процесу. Метою критеріїв завершення тестування є запобігання тому, щоб завдання вважалося виконаним, якщо є ще невиконані частини. Критерії завершення використовуються для звітування та планування часу припинення тестування.

© М.Е.Добролюбова 6

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

7. Ресурси (resources) поділяються на:

- програмні ресурси
- апаратні ресурси
- людські ресурси
- часові ресурси
- фінансові ресурси

8. Розклад тесту (test schedule) – це список дій, завдань або подій процесу тестування з визначенням передбачуваних дат і/або часу їх початку та завершення, а також взаємозалежностей.

9. Ролі та відповідальність (roles and responsibility) – це перелік необхідних ролей та область відповідальності фахівців, які виконують ці ролі.

10. Оцінка ризиків (risk evaluation) – це перелік ризиків, які з високою ймовірністю можуть виникнути в процесі роботи над проєктом.

11. Документація (documentation) – це перелік використовуваної тестової документації із зазначенням, хто і коли повинен її готувати та кому передавати.

@ М.В.Добролюбова 7

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

12. Метрика (metric) – числова характеристика показника якості. Може включати опис способів оцінки та аналізу результату.

Метрика – це шкала вимірювання та метод вимірювання.

Види метрик:

- прямі (не вимагають обчислень)
- розрахункові (обчислюються за формулою)

Проста розрахункова метрика	Складна розрахункова метрика
$T^{\%} = \frac{T^{i_{success}}}{T^{i_{total}}} \cdot 100\%$, $T^{\%}$ – відсотковий показник успішного проходження тест-кейсів, $T^{i_{success}}$ – кількість успішно виконаних тест-кейсів, $T^{i_{total}}$ – загальна кількість виконаних тест-кейсів. Мінімальні границі значень: • Початкова фаза проекту: 10 %. • Основна фаза проекту: 40 %. • Фінальна фаза проекту: 85 %.	$T^{\Sigma} = \sum_{i=1}^{i_{total}} \frac{(T_{i_{case}} \cdot I)^{i_{case}}}{B_{i_{case}}}$, T^{Σ} – інтегральна метрика проходження тест-кейсів у взаємозв'язку з вимогами та дефектами, $T_{i_{case}}$ – ступінь важливості тест-кейсу, I – кількість виконань тест-кейсу, $R_{i_{case}}$ – ступінь важливості вимоги, що перевіряється тест-кейсом, $B_{i_{case}}$ – кількість дефектів, виявлених тест-кейсом. Спосіб аналізу: • Ідеальним станом є безперервне зростання значення T^{Σ}. • У разі негативної динаміки зменшення значення T^{Σ} на 15 % і більше за останні три спринти може трактуватися як неприпустиме і бути достатнім приводом для припинення тестування.

© М.В.Добролюбова

8

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

12. Метрика (metric)

Приклади метрик, які можна зібрати автоматично з використанням інструментальних засобів управління проєктами:

- відсоткове відношення (не) виконаних тест-кейсів до всіх наявних
- відсотковий показник успішного проходження тест-кейсів
- відсотковий показник заблокованих тест-кейсів
- густина розподілу дефектів
- ефективність усунення дефектів
- розподіл дефектів за важливістю та терміновістю

$$ScheduleSlippage = \frac{DaysToDeadline}{NeededDays} - 1,$$

де

ScheduleSlippage – значення зсуву розкладу,

DaysToDeadline – кількість днів до запланованого завершення роботи,

NeededDays – кількість днів, необхідна для завершення роботи.

Значення *ScheduleSlippage* не повинно ставати від'ємним.

«Important Software Test Metrics and Measurements – Explained with Examples and Graphs» [http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/]

@ М.В.Добролюбова

9

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

13. Покриття, тестове покриття (coverage) – це ступінь, виражена у відсотках, в якій до певного елементу покриття був використаний набір тестів.

Елемент покриття (coverage item) – це сутність або властивість, які використовуються як основа для тестового покриття.

Найпростіші представники метрик покриття:

- метрика покриття вимог (вимога вважається «покритою», якщо на неї посилається хоча б один тест-кейс)

$$R^{SimpleCoverage} = \frac{R^{Covered}}{R^{Total}} \cdot 100\%,$$

де

$R^{SimpleCoverage}$ – метрика покриття вимог,

$R^{Covered}$ – кількість вимог, покритих хоча б одним тест-кейсом,

R^{Total} – загальна кількість вимог.

@ М.В.Добролюбова 10

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

13. Покриття, тестове покриття (coverage)

Найпростіші представлення метрик покриття:

- метрика щільності покриття вимог (враховується, скільки тест-кейсів посилається на кілька вимог)

$$R^{Density Coverage} = \frac{\sum T_i}{T^{Total} \cdot R^{Total}} \cdot 100\%,$$

де

$R^{Density Coverage}$ – щільність покриття вимог,

T_i – кількість тест-кейсів, що покривають і-е вимога,

T^{Total} – загальна кількість тест-кейсів,

R^{Total} – загальна кількість вимог.

- метрика покриття класів еквівалентності (аналізується, скільки класів еквівалентності зачіпалося тест-кейсами)

$$E^{Coverage} = \frac{E^{Covered}}{E^{Total}} \cdot 100\%,$$

де

$E^{Coverage}$ – метрика покриття класів еквівалентності,

$E^{Covered}$ – кількість класів еквівалентності, покритих хоча б одним тест-кейсом,

E^{Total} – загальна кількість класів еквівалентності.

© М.В.Добролюбова

11

Тест-план та звіт про результати тестування

Тест-план

Розділи тест-плану:

13. Покриття, тестове покриття (coverage)

Найпростіші представники метрик покриття:

- метрика покриття граничних умов (аналізується, скільки значень з групи граничних умов зачіпалося тест-кейсами)

$$B^{Coverage} = \frac{B^{Covered}}{B^{Total}} \cdot 100\%,$$

де

$B^{Coverage}$ – метрика покриття граничних умов,

$B^{Covered}$ – кількість граничних умов, покритих хоча б одним тест-кейсом,

B^{Total} – загальна кількість граничних умов.

- метрики покриття коду модульними тест-кейсами (виявляється певна характеристика коду та визначається, який відсоток представників цієї характеристики покритий тест-кейсами)

© М.В.Добролюбова 12

Тест-план та звіт про результати тестування

Приклад тест-плану

Ціль

Кількісне вимірювання перетворення вмісту документів до єдиного подання з продуктивністю, що значно перевищує продуктивність людини при виконанні аналогічного завдання.

Області, що підлягають тестуванню

(Див. відповідні розділи вимог.)

- ПТ-1 *: димовий тест.
- ПТ-2 *: димовий тест, тест критичного шляху.
- ПТ-3: тест критичного шляху.
- БП-1: димовий тест, тест критичного шляху.
- АК-2*: димовий тест, тест критичного шляху.
- О-4: димовий тест.
- О-5: димовий тест.
- ДС-*: димовий тест, тест критичного шляху.

Області, що не підлягають тестуванню

- СХ-1: додаток розробляється як консольний.
- СХ-2, О-1, О-2: додаток розробляється на PHP зазначеної версії.
- АК-1.1: зазначена характеристика накладається поверх кожної групи продуктивності операцій, характерних для додатку, що розробляється.
- О-3: не потребує реалізації.
- О-6: не потребує реалізації.

Тестова стратегія та підходи

Загальний підхід

Специфіка роботи додатку полягає в одноразовому конфігуруванні досліджених фізичних та віртуальних використання кінцевими користувачами, для яких доступна лише одна операція – розміщення файлу в каталозі-приймачі. Тому питання зручності використання, безпеки тощо не досліджуються в процесі тестування.

Типові функціональні тестування:

- Димовий тест: автоматизований з використанням консольних файлів ОС Windows та Linux.
- Тест критичного шляху: виконується вручну.
- Розширений тест: не виконується, оскільки для даної програми ймовірність виявлення дефектів на цьому рівні дуже мала.

Через кросплатформеність основним значного вкладу у підвищення якості можна очікувати від аудиту коду, покладеного з ручних тестувальних методів більш скрупульозних. Автоматизація тестування коду не буде застосовуватись через вкрай обмежений час.

Критерії

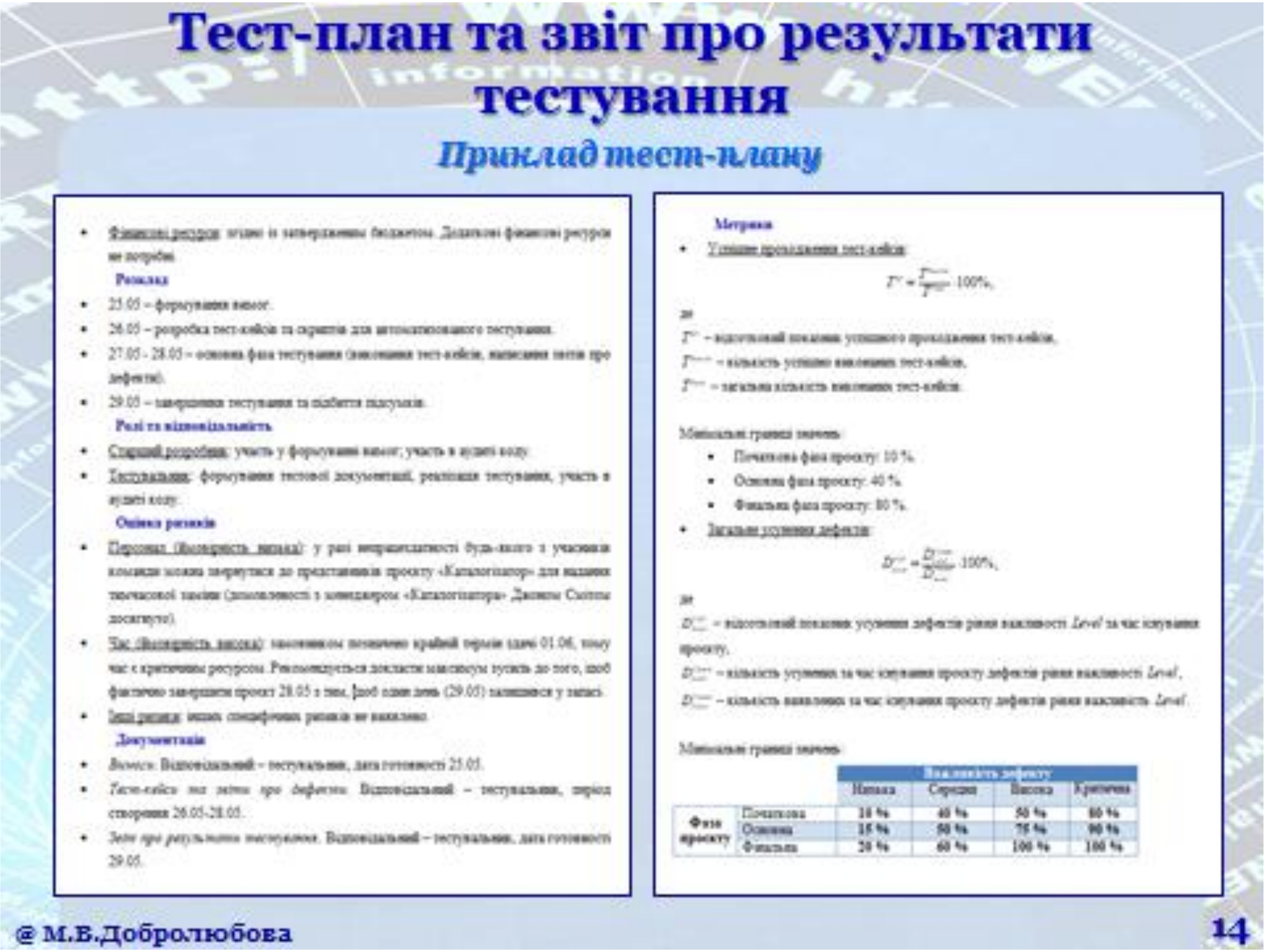
- Прийнятні критерії: успішне проходження 100 % тест-кейсів рівня димового тестування та 90 % тест-кейсів рівня критичного шляху (див. матрицю «Успішне проходження тест-кейсів») та умови усунення 100 % дефектів критичної та високої важливості (див. матрицю «Загальне усунення дефектів»). Пасувальне покриття вимог тест-кейсами (див. матрицю «Покриття вимог тест-кейсами») має становити мінімальних 80 %.
- Критерії початку тестування: вимог немає.
- Критерії призупинення тестування: перехід до тесту критичного шляху припустимий лише при успішному проходженні 100 % тест-кейсів димового тесту (Див. матрицю «Успішне проходження тест-кейсів»); тестування може бути призупинене у разі, якщо при виконанні не менше 25 % запланованих тест-кейсів більше 50 % з них завершилися виявленням дефекту (див. матрицю «Стоп-фактор»).
- Критерії відновлення тестування: виправлення понад 50 % виявлених на попередній ітерації дефектів (див. матрицю «Поточне усунення дефектів»).
- Критерії завершення тестування: виконання більше 80 % запланованих на ітерацію тест-кейсів (див. матрицю «Виконання тест-кейсів»).

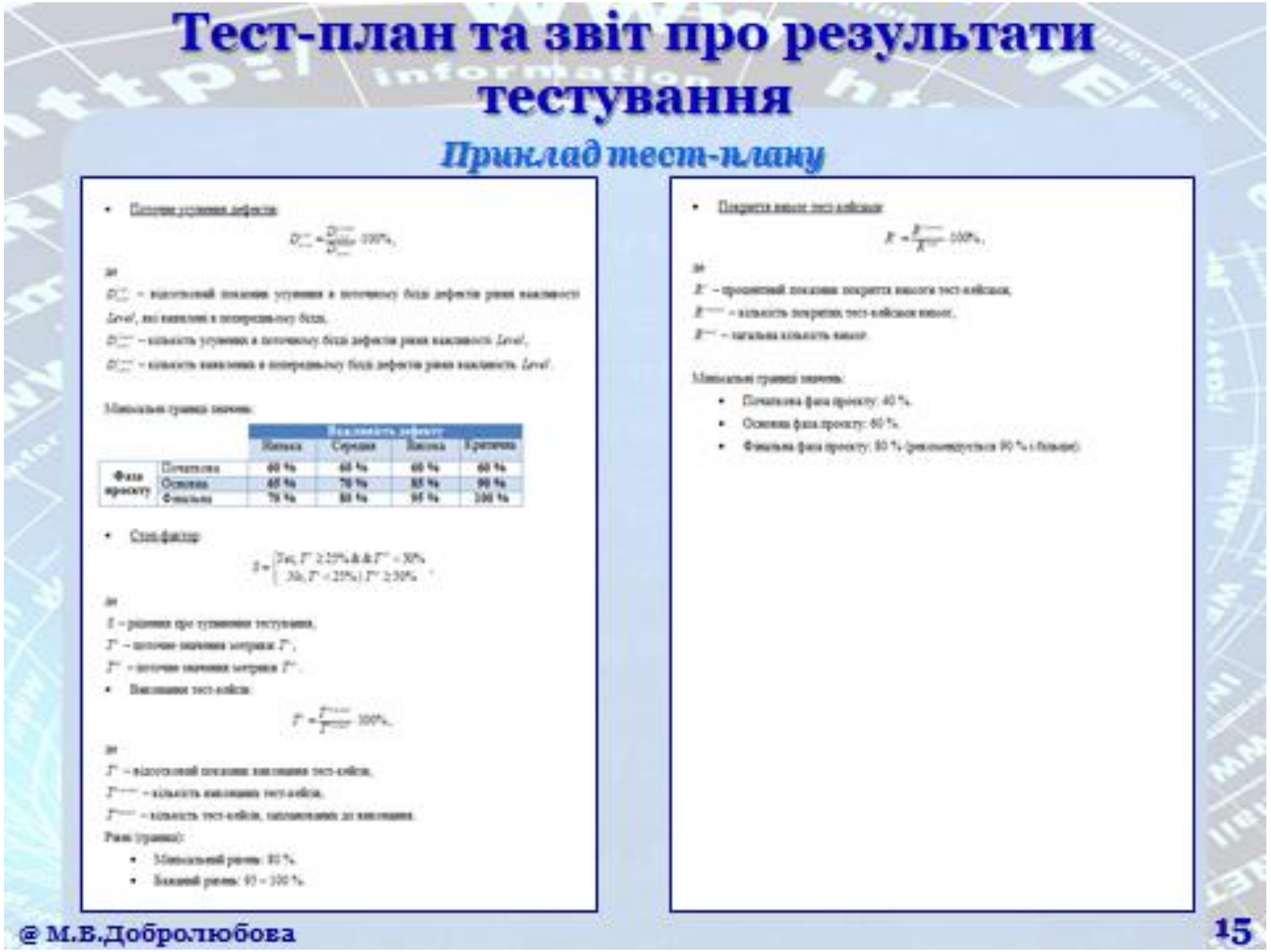
Ресурси

- Програмні ресурси: чотири віртуальні машини (дві з ОС Windows 10 Ent x64, дві з ОС Linux (Ubuntu 14 LTS x64), два комп PHP Storm 8.
- Апаратні ресурси: два стандартні робочі станції (8GB RAM, i7 3GHz).
- Людські ресурси:
  - Старший розробник з досвідом тестування (100 % зайнятість протягом проекту). Роль на проекті: лідер команди, старший розробник.
  - Тестувальник зі знанням PHP (100 % зайнятість протягом проекту). Роль на проекті: тестувальник.
- Часові ресурси: один робочий тиждень (40 годин).

@ М.В.Добролюбова

13


Тест-план та звіт про результати тестування
Приклад тест-плану
@ М.В.Добролюбова
14


Тест-план та звіт про результати тестування
Приклад тест-плану
Покриття умовних дефектів
де
Покриття вимог тест-кейсами
де
Мінімальні граничні значення:
Фази проекту
Початкова
Основна
Фінальна
Стоп-фактор
де
Виконання тест-кейсів
де
Рівні тяжкості:
Мінімальний рівень: 80 %
@ М.В.Добролюбова
15

Тест-план та звіт про результати тестування

Звіт про результати тестування

Звіт про хід тестування (test progress report) – це документ, що підсумовує діяльність з тестування, підготований на регулярних умовах, для звіту про хід діяльності з тестування у порівнянні з вихідним планом та для інформування керівництва про ризики і альтернативи, що потребують прийняття рішення.

Звіт про підсумки тестування (test summary report) – документ, що підсумовує результати тестування. Він також містить оцінку відповідних тестових завдань до критеріїв виходу.

Низькорівневі завдання звітності у тестуванні:

- оцінка обсягу та якості виконаних робіт
- порівняння поточного прогресу із тест-планом
- опис наявних складнощів та формування рекомендацій щодо їх усунення
- надання особам, зацікавленим у проєкті, повної та об'єктивної інформації про поточний стан якості проєкту, вираженої у конкретних фактах та числах

Розширення до вимог якісного звіту про результати тестування:

- інформативність
- точність та об'єктивність

@ М.В.Добролюбова 16

Тест-план та звіт про результати тестування

Звіт про результати тестування

Особи, яким потрібний звіт про результати тестування:

- менеджер проекту
- керівник команди розробників («дев-ліду»)
- керівник команди тестувальників («тест-лід»)
- замовник

Розділи звіту про результати тестування:

- короткий опис
- команда тестувальників
- опис процесу тестування
- розклад
- статистика щодо нових дефектів
- список нових дефектів
- статистика з усіх дефектів
- рекомендації
- програми

© М.В.Добролюбова

17

Тест-план та звіт про результати тестування

Звіт про результати тестування

Розділи звіту про результати тестування:

1. **Короткий опис (summary)** – це розділ, який відображає у гранично короткій формі основні досягнення, проблеми, висновки та рекомендації.
2. **Команда тестувальників (test team)** – це список учасників проєктної команди, задіяних у забезпеченні якості, із зазначенням їх посад та ролей у підзвітний період.
3. **Опис процесу тестування (testing process description)** – це послідовний опис того, які роботи було виконано за підзвітний період.
4. **Розклад (timetable)** – це детальний розклад роботи команди тестувальників та/або особисті розклади учасників команди.
5. **Статистика щодо нових дефектів (new defects statistics)** – це таблиця, в якій представлені дані щодо виявлених за підзвітний період дефектів.
6. **Список нових дефектів (new defects list)** – це список виявлених за підзвітний період дефектів з їх короткими описами та важливістю.
7. **Статистика з усіх дефектів (overall defects statistics)** – це таблиця, в якій представлені дані щодо виявлених за весь час існування проєкту дефектів.
8. **Рекомендації (recommendations)** – це обґрунтовані висновки та рекомендації щодо прийняття тих чи інших управлінських рішень.
9. **Програми (appendixes)** – це фактичні дані (як правило, значення метрик та графічне надання їх зміни у часі).

© М.Е.Добролюбова 18

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Універсальна логіка звітності:

- висновки будуються на основі цілей
- висновки доповнюються рекомендаціями
- як висновки, так і рекомендації суворо обґрунтовуються
- обґрунтування спирається на об'єктивні факти

© М.В.Добролюбова 19

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Вимоги до висновків:

- короткі

Погано	Добре
1.17. Як показав глибокий аналіз протоколів виконання тестування, можна зробити досить впевнені висновки про те, що основна частина функцій, відзначених замовником як найважливіші, функціонує в рамках допустимих відхилень від узгоджених на останньому обговоренні із замовником метрик якості.	1.11. Базова функціональність є повністю працездатною (див. 2.1–2.2). 1.23. Існують некритичні проблеми з деталізацією повідомлень у файлі журналу (див. 2.3-2.4). 1.28. Тестування програми під ОС Linux не вдалося провести через відсутність сервера SR-85 (див. 2.5).

- інформативні

Погано	Добре
1.8. Результати обробки файлів з множинними кодуваннями, представленими в порівняльних пропорціях, залишають бажати кращого. 1.9. Програма не запускається за певних значень параметрів командного рядка. 1.10. 1.10. Незрозуміло, що відбувається з аналізом зміни вмісту вхідного каталогу.	1.8. Виявлено серйозні проблеми з бібліотекою розпізнавання кодувань (див. BR 834). 1.9. Порушено функціональність аналізу параметрів командного рядка (див. BR 745, BR 877, BR 878). 1.10. Виявлено нестабільність у роботі модуля "Сканер", проводяться додаткові дослідження.

@ М.В.Добролюбова

20

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Вимоги до висновків:

- корисні для читача звіту

Погано	Добре
1.18. Деякі тести пройшли напрочуд добре. 1.19. В процесі тестування ми не мали труднощів з налаштуванням середовища автоматизації. 1.20. Порівняно з результатами, які були отримані вчора, ситуація трохи покращилась. 1.21. З якістю, як і раніше, є деякі проблеми. 1.22. Частина команди була у відпустці, але ми все одно впоралися.	**Представленого у колонці «Погано» просто не повинно бути у звіті!**

© М.В.Добролюбова

21

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Вимоги до рекомендацій:

- **короткі**

Погано	Добре
2.98. Ми рекомендуємо розглянути можливі варіанти виправлення цієї ситуації в контексті пошуку оптимального рішення за умови мінімізації зусиль розробників та максимального підвищення відповідності додатка заявленим критеріям якості, а саме: дослідити можливість заміни деяких бібліотек їх якіснішими аналогами.	2.98. Необхідно змінити спосіб визначення кодування тексту у документі. Можливі рішення: • [складно, надійно, але дуже довго] написати власне рішення; • [вимагає додаткового дослідження та узгодження] замінити проблемну бібліотеку «cflk_n_r_coding» аналогом (можливо, комерційним).

- **реально виконувані**

Погано	Добре
2.107. Використовувати механізм обробки слів, аналогічний використовуваному в Google.	2.107. Реалізувати алгоритм приведення слів української мови до називного відмінка (див. опис за посиланням …)
2.304. Не завантажувати до оперативної пам'яті інформацію про файли у вхідному каталозі.	2.304. Збільшити розмір доступної скрипту оперативної пам'яті на 40-50 % (в ідеалі - до 512 МБ).
2.402. Повністю переписати проект без використання зовнішніх бібліотек.	2.402. Замінити власними рішеннями функції аналізу вмісту каталогу та параметрів файлів бібліотеки «cflk_n_r_flstm».

© М.В.Добролюбова

22

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Вимоги до рекомендацій:

- такі, що дають як розуміння того, що треба зробити, так і деякий простір для прийняття власних рішень

Погано	Добре
2.212. Рекомендуємо пошукати варіанти вирішення цього питання. 2.245. Використовувати лише дискове сортування. 2.278. Виключити можливість передачі некоректних імен файлу журналу через параметр командного рядка.	2.212. Можливі варіанти рішення: а) … б) [рекомендуємо!] … в) … 2.245. Додати функціональність визначення оптимального методу сортування залежно від кількості доступної оперативної пам'яті. 2.278. Додати фільтрацію імені файлу журналу, який отримується через параметр командного рядка, за допомогою регулярного виразу.

Тест-план та звіт про результати тестування

Логіка побудови звіту про результати тестування

Обґрунтування висновків та рекомендацій – це проміжна ланка між гранично стислими результатами аналізу та величезною кількістю фактичних даних.

Погано	Добре
4.107. Покриття вимог тест-кейсами достатньо. 4.304. Потрібно більше зусиль направити на регресійне тестування. 4.402. Від скорочення термінів розробки варто відмовитись.	4.107. Покриття вимог тест-кейсами вийшло на достатній рівень (значення склало 63 % при заявленому мінімумі 60 % для поточної стадії проекту). 4.304. Потрібно більше зусиль направити на регресійне тестування, оскільки дві попередні ітерації виявили 21 дефект високої важливості (див. список 5.43) у функціональності, в якій раніше не виявлялося проблем. 4.402. Від скорочення термінів розробки варто відмовитись, оскільки поточне випередження графіка на 30 людино-годин може бути легко поглинене на стадії реалізації вимог R84.* та R89.*.

Фактичний матеріал – це матеріал, який містить найрізноманітніші дані, отримані в процесі тестування.

24


Тест-план та звіт про результати тестування
Приклад звіту про результати тестування
Короткий опис
Команда тестувальників
Опис процесу тестування
Розклад
Статистика щодо нових дефектів
Список нових дефектів
@ М.В.Добролюбова
25

Тест-план та звіт про результати тестування

Приклад звіту про результати тестування

Статистика з усіх дефектів

Статус	Кількість	Важливість			
		Низька	Середня	Висока	Критична
Знайдено	34	7	18	8	1
Виправлено	25	5	12	7	1
Перевірено	17	3	7	7	0
Відкрито знову	3	0	0	3	0
Відхилено	4	1	3	1	0

Рекомендації

На даний момент додаткових тестів не потрібно.

Додатки

Графік зміни значень метрики


Інформаційні технології оцінювання якості
РОЗДІЛ 1
Основні відомості про інформаційні технології оцінювання якості
Тема 5. ТРУДОВИТРАТИ
Лекція 14
«Оцінка трудовитрат»
© М.В.Добролюбова

Вимоги до оцінки трудовитрат

Трудовитрати – кількість робочого часу, необхідного для виконання роботи (виражається в людино-годинах).

Людино-година (man-hours) – одиниця виміру роботи в промисловості, що дорівнює роботі, виконаній однією людиною за одну годину.

Правила виконання оцінки трудовитрат:

- будь-яка оцінка краща за її відсутність
- недооцінювання складності незнайомих завдань
- оцінка має бути аргументована
- простий спосіб навчитися оцінювати – оцінювати

Алгоритм навчання формувати оцінку:

- сформувати оцінку
- запишіть отриману оцінку
- виконати роботу
- звірити реальні результати з раніше сформованою оцінкою
- зважити на помилки при формуванні нових оцінок
- повторювати цей алгоритм якнайчастіше для різних областей життя

© М.В.Добролюбова 2

Вимоги до оцінки трудовитрат

Корисні ідеї щодо формування оцінки трудовитрат:

- додати невеликий «буфер» (за часом, бюджетом чи іншими критичними ресурсами) на непередбачені обставини
- з'ясувати власний «коефіцієнт спотворення»
- зважати на обставини, що не залежать від вас
- заздалегідь замислюватись про необхідні ресурси
- шукати способи організувати паралельне виконання завдань
- періодично звірятись з планом, вносити коригування в оцінку та повідомляти зацікавлених осіб про внесені зміни завчасно
- використовувати інструментальні засоби

«The Mythical Man Month», Frederick Brooks
«Controlling Software Projects», Tom De Marco
«Software engineering metrics and models», Samuel Conte

Оцінка з використанням структурної декомпозиції

Структурна декомпозиція (work breakdown structure, WBS) – ієрархічна декомпозиція об'ємних завдань на дедалі більші підзадачі з метою спрощення оцінки, планування та моніторингу виконання роботи.

WBS – це орієнтована на результат ієрархічна декомпозиція роботи, яка має бути виконана проектною командою для досягнення цілей проекту та створення необхідних результатів. WBS організовує та визначає загальний обсяг проекту. WBS підрозділяє роботу над проектом на менші, більш керовані частини роботи, причому кожен спадний рівень WBS представляє все більш детальне визначення роботи над проектом. Запланована робота, що міститься в компонентах WBS найнижчого рівня, які називаються робочими пакетами, може плануватися, оцінюватися за вартістю, відстежуватися та контролюватися.

В процесі виконання структурної декомпозиції великі завдання поділяються на все більш дрібні підзадачі, що дозволяє:

- описати весь обсяг робіт з точністю, достатньою для чіткого розуміння суті завдань, формування досить точної оцінки трудовитрат та вироблення показників досягнення результатів
- визначити весь обсяг трудовитрат як суму трудових витрат за окремими завданнями (з урахуванням необхідних поправок)
- перейти від інтуїтивного уявлення до конкретного переліку окремих дій, що спрощує побудову плану, прийняття рішень про розпаралелювання робіт тощо

«Essential Scrum», Kenneth Rubin; «Agile Estimating and Planning», Mike Cohn; «Extreme programming explained: Embrace change», Kent Beck; PMBOK (Project Management Body of Knowledge); «Software Estimation Techniques – Common Test Estimation Techniques used in SDLC»

© М.Е.Добролюбова

4

Оцінка з використанням структурної декомпозиції

Кроки при застосуванні структурної декомпозиції у поєднанні із спрощеним поглядом на оцінку трудовитрат на основі вимог та тест-кейсів:

- декомпозиція вимог до рівня, на якому з'являється можливість створення якісних чек-листів
- декомпозиція завдань із тестування кожного пункту чек-листа до рівня «тестування дій» (створення тест-кейсів, виконання тест-кейсів, створення звітів про дефекти тощо)
- виконання оцінки з урахуванням своєї продуктивності

«Test Effort Estimation Using Use Case Points», Suresh Nageswaran
[http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.597.6800&rep=rep1&type=pdf]

«Test Case Point Analysis», Nirav Patel
[http://www.stickyminds.com/sites/default/files/article/file/2013/XUS373692File1_0.pdf]

@ М.В.Добролюбова 5

Оцінка з використанням структурної декомпозиції

Приклад

«При вказуванні невірного значення будь-якого з параметрів командного рядка програма має завершити роботу, видавши повідомлення про використання (ДС-3.1), а також повідомивши ім'я невірно зазначеного параметра, його значення та суть помилки (див. ДС-3.2)».

Поділ вимоги на складові:

- Якщо всі три параметри командного рядка вказані правильно, повідомлення про помилку не видається
- Якщо вказано неправильно від одного до трьох параметрів, видається повідомлення про використання, ім'я (або імена) неправильно вказаного параметра та неправильне значення, а також повідомлення про помилку:
  - якщо неправильно вказано SOURCE_DIR або DESTINATION_DIR: «Directory not exists or inaccessible»
  - якщо DESTINATION_DIR знаходиться в SOURCE_DIR: «Destination dir may not reside within source dir tree»
  - якщо неправильно вказано LOG_FILE_NAME: «Wrong file name or inaccessible path»

@ М.В.Добролюбова 6

Оцінка з використанням структурної декомпозиції

Приклад

Створення чек-листа та приблизна кількість тест-кейсів на кожен пункт:

- Всі параметри коректні {1 тест-кейс}
- Неіснуючий/некоректний шлях для:
  - SOURCE_DIR {3 тест-кейси}
  - DESTINATION_DIR {3 тест-кейси}
- Неприпустиме ім'я файлу LOG_FILE_NAME {3 тест-кейси}
- Значення SOURCE_DIR та DESTINATION_DIR є коректними іменами існуючих каталогів, але DESTINATION_DIR знаходиться всередині SOURCE_DIR {3 тест-кейси}
- Неприпустимі/неіснуючі імена об'єктів ФС вказані в більш ніж одному параметрі {6 тест-кейсів}
- Значення SOURCE_DIR та DESTINATION_DIR не є коректними/існуючими іменами каталогів, і при цьому DESTINATION_DIR знаходиться всередині SOURCE_DIR {3 тест-кейси}

@ М.В.Добролюбова 7

Оцінка з використанням структурної декомпозиції

Приклад

Оцінка кількості проходів для тестування нової функціональності:

- проста функціональність: 1-1.5 (не всі тести повторюються)
- функціональність середньої складності: 2
- складна функціональність: 3-5

Оцінка кількості створюваних та виконуваних тест-кейсів

	Створення	Виконання
Кількість	12	22
Повторення (проходи)	1	1.2
Загальна кількість	12	26.4
Час на один тест-кейс		
Підсумковий час		

Коефіцієнти, що уточнюють оцінку часу, необхідного на розробку та виконання одного тест-кейсу:

- професіоналізм та досвід тестувальника
- складність та об'ємність тест-кейсів
- продуктивність додатка, що тестується, та тестового оточення
- вид тестування
- наявність та зручність засобів автоматизації
- стадія розробки проєкту

@ М.В.Добролюбова

8

Оцінка з використанням структурної декомпозиції

Приклад

Припустимо, що деякому вигаданому тестувальнику протягом місяця (28 робочих днів) вдається:

- створити 300 тест-кейсів (приблизно 11 тест-кейсів на день або 1.4 на годину)
- виконати 1000 тест-кейсів (приблизно 36 тест-кейсів на день або 4.5 на годину)

Підставимо отримані значення таблицю 1 і отримаємо таблицю 2

Оцінка трудовитрат

	Створення	Виконання
Кількість	12	22
Повторення (проходи)	1	1.2
Загальна кількість	12	26.4
Час на один тест-кейс, год	0.7	0.2
Підсумковий час, год	8.4	5.2
РАЗОМ	**13.6 годин**	

@ М.В.Добролюбова

9


Інформаційні технології оцінювання якості
РОЗДІЛ 2
Автоматизація тестування
Тема 1. ВИГОДИ ТА РИЗИКИ АВТОМАТИЗАЦІЇ
Лекція 15
«Області застосування автоматизації»
© М.В.Добролюбова

Переваги та недоліки автоматизації

Автоматизоване тестування – це набір технік, підходів та інструментальних засобів, що дозволяє виключити людину з виконання деяких завдань у процесі тестування.

Переваги автоматизації:

- швидкість виконання тест-кейсів може в рази та на порядки перевершувати можливості людини
- відсутній вплив людського фактора в процесі виконання тест-кейсів
- засоби автоматизації здатні виконати тест-кейси, непосильні для людини через свою складність, швидкість або інші фактори
- засоби автоматизації здатні збирати, зберігати, аналізувати, агрегувати та представляти у зручній для сприйняття людиною формі колосальні обсяги даних
- засоби автоматизації здатні виконувати низькорівневі дії з додатком, операційною системою, каналами передачі тощо

Використання автоматизації збільшує тестове покриття за рахунок:

- виконання тест-кейсів, про які раніше не варто було й думати
- багаторазового повторення тест-кейсів із різними вхідними даними
- вивільнення часу створення нових тест-кейсів

@ М.В.Добролюбова

2

Переваги та недоліки автоматизації

Співвідношення часу розробки та виконання тест-кейсів ручному та автоматизованому тестуванні

Недоліки та ризики автоматизації:

- необхідність наявності висококваліфікованого персоналу
- багато часу на розробку та супровід як самих автоматизованих тест-кейсів, так і всієї необхідної інфраструктури
- ретельніше планування та управління ризиками
- висока вартість комерційних засобів автоматизації
- різноманіття засобів автоматизації

@ М.В.Добролюбова 3

Переваги та недоліки автоматизації

Параметри та фактори спеціальних підходів з оцінки застосування та ефективності автоматизованого тестування, що враховуються в першу чергу:

- витрати часу на ручне виконання тест-кейсів та на виконання цих самих тест-кейсів, але вже автоматизованих
- кількість повторень виконання тих самих тест-кейсів
- витрати часу на налагодження, оновлення та підтримку автоматизованих тест-кейсів
- наявність у команді відповідних фахівців та їх робоче завантаження
- автоматизацією займаються найкваліфікованіші співробітники, які в цей час не можуть вирішувати інших задач

«Implementing Automated Software Testing – Continuously Track Progress and Adjust Accordingly», Thom Garrett

The ROI of Test Automation, Michael Kelly

«Cost Benefits Analysis of Test Automation», Douglas Hoffman

@ М.В.Добролюбова

4

Переваги та недоліки автоматизації

Приклад

Швидка оцінка ефективності автоматизації

$$A_{effect} = \frac{N \cdot T^{overall}_{manual}}{N \cdot T^{run\ and\ analysis}_{automated} + N \cdot T^{development\ and\ support}_{automated}}$$

A_{effect} – коефіцієнт вигоди від використання автоматизації

N – запланована кількість білдів програми

$T^{overall}_{manual}$ – розрахунковий час розробки, виконання та аналізу результатів ручного тестування

$T^{run\ and\ analysis}_{automated}$ – розрахунковий час виконання та аналізу результатів автоматизованого тестування

$T^{development\ and\ support}_{automated}$ – розрахунковий час розробки та супроводу автоматизованого тестування

«Introduction to automation», Vitaliy Zhyrytskyy

@ М.В.Добролюбова

5

Переваги та недоліки автоматизації

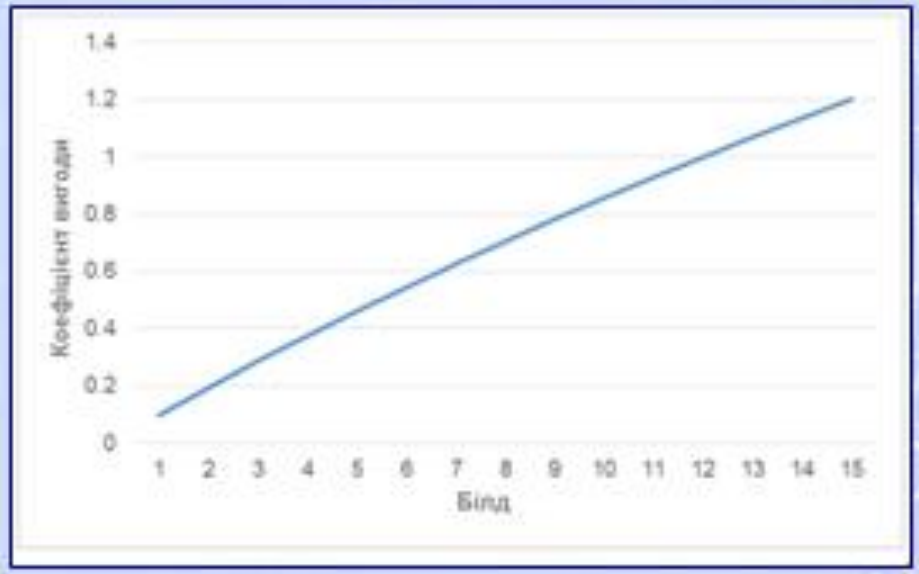
Приклад

Швидка оцінка ефективності автоматизації

$T_{manual}^{one\ build}$ = 30 годин на кожен білд

$T_{automated}^{run\ and\ analyze}$ = 5 годин на кожний білд

$T_{automated}^{develop\ and\ support}$ = 300 годин на весь проект



автоматизації залежно від кількості білдів

@ М.В.Добролюбова

6

Області застосування автоматизації

Завдання, які дозволяє вирішити автоматизація:

- виконання тест-кейсів, непідсильних людині
- вирішення рутинних завдань
- прискорення тестування
- вивільнення людських ресурсів для інтелектуальної роботи
- підвищення тестового покриття
- покращення коду за рахунок збільшення тестового покриття та застосування спеціальних технік автоматизації

© М.В.Добролюбова 7

Області застосування автоматизації

Випадки найбільшої застосовності автоматизації

Випадок / завдання	Яку проблему вирішує автоматизація
Регресійне тестування	Необхідність виконувати вручну тести, кількість яких невпинно росте з кожним білдом, але вся суть яких зводиться до перевірки того факту, що функціональність, яка раніше працювала, продовжує працювати коректно.
Інсталяційне тестування та налаштування тестового оточення	Багато часто повторюваних рутинних операцій з перевірки роботи інсталятора, розміщення файлів у файловій системі, вмісту конфігураційних файлів, реєстру тощо. Підготовка програми у заданому середовищі та із заданими налаштуваннями для проведення основного тестування.
Конфігураційне тестування та тестування сумісності	Виконання одних і тих самих тест-кейсів на великій кількості вхідних даних, під різними платформами та в різних умовах. Класичний приклад: є файл налаштувань, у ньому сто параметрів, кожен може набувати сто значень; існує 100100 варіантів конфігураційного файлу – всі їх потрібно перевірити.
Використання комбінаторних технік тестування (в тому числі доменного тестування)	Генерація комбінацій значень та багаторазове виконання тест-кейсів з використанням цих згенерованих комбінацій в якості вхідних даних.
Модульне тестування	Перевірка коректності роботи атомарних ділянок коду та елементарних взаємодій таких ділянок коду – практично нездійсненне для людини завдання за умови, що потрібно виконати тисячі таких перевірок і ніде не помилитись.
Інтеграційне тестування	Глибока перевірка взаємодії компонентів у ситуації, коли людині майже нічого спостерігати, оскільки всі цікаві процеси, що піддаються тестуванню, проходять на рівнях більш глибоких, ніж інтерфейс користувача.
Тестування безпеки	Необхідність перевірки прав доступу, паролів за замовчуванням, відкритих портів, уразливостей поточних версій ПЗ тощо, тобто швидке виконання дуже великої кількості перевірок, в процесі якого не можна щось пропустити, забути чи «неправильно зрозуміти».
Тестування продуктивності	Створення навантаження з інтенсивністю та точністю, недоступною людині. Збір із високою швидкістю великого набору параметрів роботи програми. Аналіз великого обсягу даних із журналів роботи системи автоматизації.
Димовий тест для великих систем	Виконання при отриманні кожного білду великої кількості є досить простими для автоматизації тест-кейсів.
Додатки (або їх частини) без графічного інтерфейсу.	Перевірка консольних додатків на великих наборах значень параметрів командного рядка (та їх комбінацій). Перевірка додатків та їх компонентів, взагалі не призначених для взаємодії з людиною (веб-сервіси, сервери, бібліотеки тощо).
Довгі, рутинні, стомлюючі для людини операції та/або операцій, що вимагають підвищеної уваги.	Перевірки, що вимагають порівняння великих обсягів даних, високої точності обчислень, обробки великої кількості розміщених по всьому дереву каталогів файлів, відчутно великого часу виконання тощо. Особливо коли такі перевірки повторюються дуже часто.
Перевірка «внутрішньої функціональності» веб-додатків (посилань, доступності сторінок тощо).	Автоматизація дуже рутинних дій (наприклад, перевірити всі 30'000+ посилань на предмет того, що всі вони ведуть на реально існуючі сторінки). Автоматизація тут спрощується в силу стандартності завдання – існує багато готових рішень.
Стандартна, однотипна для багатьох проектів функціональність	Навіть висока складність при первинній автоматизації в такому разі окупиться за рахунок простоти багаторазового використання отриманих рішень у різних проектах.
«Технічні задачі»	Перевірки коректності протоколювання, роботи з базами даних, коректності пошуку, файлових операцій, коректності форматів та вмісту генерованих документів тощо.

Області застосування автоматизації

Випадки найменшої застосовності автоматизації

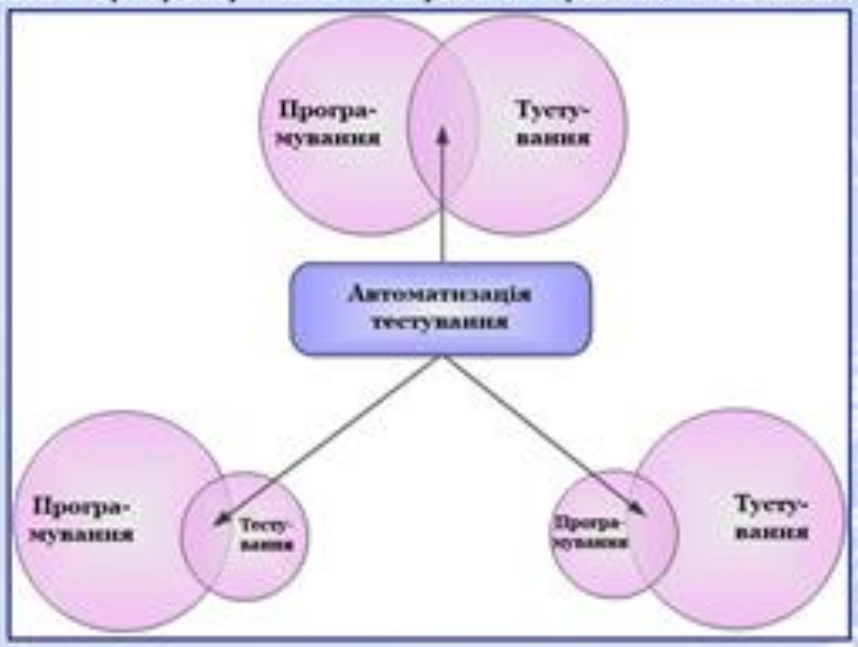
Випадок / завдання	У чому проблема автоматизації
Планування	Комп'ютер поки що не навчився думати.
Розробка тест-кейсів	
Написання звітів про дефекти	
Аналіз результатів тестування та звітність	
Функціональність, яку потрібно (достатньо) перевірити лише кілька разів	Витрати на автоматизацію не окупляться
Тест-кейси, які потрібно виконати всього кілька разів (якщо людина може їх виконати)	
Низький рівень абстракції у наявних інструментах автоматизації	Доведеться писати дуже багато коду, що не тільки складно і довго, але і призводить до появи безлічі помилок у самих тест-кейсах.
Слабкі можливості засобу автоматизації з протоколювання процесу тестування та збору технічних даних про додаток та оточення	Є ризик отримати дані у вигляді «щось десь зламалося», що не допомагає у діагностиці проблеми.
Низька стабільність вимог	Доведеться дуже багато переробляти, що у випадку автоматизації обходиться дорожче, ніж у випадку ручного тестування.
Складні комбінації великої кількості технологій	Висока складність автоматизації, низька надійність тест-кейсів, висока складність оцінки трудовитрат і прогнозування ризиків.
Проблеми з плануванням та ручним тестуванням	Автоматизація хаосу призводить до появи автоматизованого хаосу, але ще й вимагає трудовитрат. Спочатку варто вирішити наявні проблеми, а потім включатися в автоматизацію.
Нестача часу та загроза зриву термінів	Автоматизація не дає миттєвих результатів. Спочатку вона лише споживає ресурси команди (зокрема час). Також є універсальний афоризм: «краще протестувати руками хоч щось, ніж автоматизовано протестувати нічого».
Області тестування, що вимагають оцінки ситуації людиною (тестування зручності використання, тестування доступності тощо)	В принципі, можна розробити деякі алгоритми, що оцінюють ситуацію так, як її могла б оцінити людина. Але на практиці жива людина може зробити це швидше, простіше, надійніше та дешевше.

@ М.В.Добролюбова **9**


Інформаційні технології оцінювання якості
РОЗДІЛ 2
Автоматизація тестування
Тема 1. ВИГОДИ ТА РИЗИКИ АВТОМАТИЗАЦІЇ
Лекція 16
«Особливості автоматизованого тестування»
@ М.В.Добролюбова

Необхідні знання та навички

Автоматизація тестування – це поєднання програмування і тестування в різних масштабах.



Навички та вміння спеціаліста з автоматизації тестування:

- вміння і програмувати, і тестувати
- вміння адмініструвати операційні системи, мережі, різні сервери
- вміння працювати з базами даних; розуміння мобільних платформ
- вкрай висока здатність навчання
- здатність в гранично стислий термін самостійно знайти, вивчити, зрозуміти та почати застосовувати на практиці абсолютно нову інформацію з, можливо, раніше абсолютно незнайомої області

@ М.В.Добролюбова 2

Особливості тест-кейсів в автоматизації

Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації:

- **очікуваний результат в автоматизованих тест-кейсах має бути описаний максимально чітко із зазначенням конкретних ознак його коректності**

Погано	Добре
… 7. Завантажується стандартна сторінка пошуку.	7. Завантажується сторінка пошуку: title = «Search page», є форма з полями «input type="text"», «input type="submit" value="Go!"», присутній логотип «logo.jpg» та відсутні інші графічні елементи («img»).

- **опис тест-кейсу без специфічних для того чи іншого інструментального засобу рішень**

Погано	Добре
1. Натисніть на посилання «Search». 2. Використовуйте clickAndWait для синхронізації таймінгу.	1. Натисніть на посилання «Search». 2. Дочекайтеся завершення завантаження сторінки.

@ М.В.Добролюбова 3

Особливості тест-кейсів в автоматизації

Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації:

- **не варто прописувати в тест-кейс щось, характерне лише для однієї платформи**

Погано	Добре
… 8. Надіслати додатку повідомлення WM_CLICK у будь-яке з видимих вікон.	… 8. Передати фокус введення будь-якому із незгорнутих вікон програми (якщо таких немає – розгорнути будь-яке з вікон). 9. Проемулювати подію «клік лівою кнопкою миші» для активного вікна.

- **синхронізація засобу автоматизації та додатку, що тестується, за часом**

Погано	Добре
1. Натиснути на посилання «Expand data». 2. Вибрати зі списку значення «Unknown».	1. Натиснути на посилання «Expand data». 2. Дочекатися завершення завантаження даних до списку Extended data (select id="extended_data"): список перейде в стан enabled. 3. Вибрати в списку Extended data значення Unknown.

@ М.В.Добролюбова

4

Особливості тест-кейсів в автоматизації

Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації:

- **опис словами значення та/або сенсу якоїсь змінної**

Погано	Добре
1. Відкрити http://application/.	1. Відкрити головну сторінку додатку.

- **використовувати найбільш універсальні способи взаємодії з додатком, що тестується**

Погано	Добре
… 8. Передати в полі "Search" набір подій WM_KEY_DOWN, {знак}, WM_KEY_UP, в результаті в полі має бути введений пошуковий запит.	… 8. Проемулювати введення значення поля "Search" з клавіатури (не підходить вставка значення з буфера або пряме надання значення!)

@ М.В.Добролюбова

5

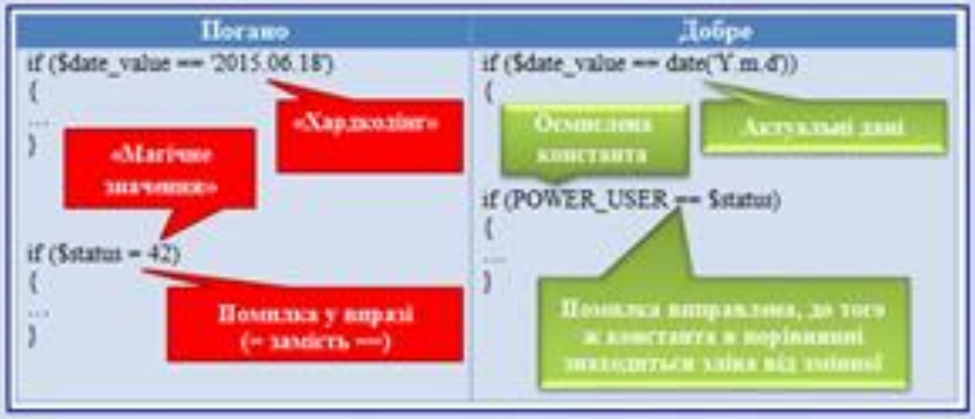
Особливості тест-кейсів в автоматизації

Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації:

- **автоматизовані тест-кейси мають бути незалежними**

Погано	Добре
1. З файлу, створеного попереднім тестом.	1. Перевести чек-бокс «Use stream buffer file» у стан checked. 2. Активувати процес передачі (Клікнути по кнопці «Start»). 3. З файлу буферизації прочитати…

- **автоматизований тест-кейс – це програма і варто враховувати хороші практики програмування хоча б лише на рівні відсутності так званих «магічних значень», «хардкодінгу» тощо**



@ М.В.Добролюбова

6

Особливості тест-кейсів в автоматизації

Список рекомендацій щодо підготовки тест-кейсів до автоматизації та безпосередньо самої автоматизації:

- уважно вивчати документацію по використовуваному засобу автоматизації, щоб уникнути ситуації, коли через невірно обрану команду тест-кейс стає хибно позитивним

Погано			Добре		
...	...	...	...	...	...
verifyEditable	id=cb		verifyChecked	id=cb	
...	...	...	...	...	...

- уважно слідкувати, щоб не відбулася заміна перевірки дією та навпаки

Хибно позитивний тест-кейс – це тест-кейс, який успішно проходить в ситуації, коли програма працює неправильно.

Джерела даних та способи їх генерації в автоматизованих тест-кейсів:

- випадкові величини
- генерація (випадкових) даних щодо алгоритму
- отримання даних із зовнішніх джерел
- зібрані робочі дані, тобто дані, створені реальними користувачами в процесі їх реальної роботи
- ручна генерація

@ М.В.Добролюбова

7

Автоматизація поза межами прямих задач тестування

Типові рішень автоматизації поза межами прямих задач тестування:

- використання командних файлів для виконання послідовностей операцій
- генерація та оформлення даних з використанням офісних можливостей додатків, баз даних, невеликих програм на високорівневих мовах програмування
- підготовка та оформлення технічних розділів для звітів
- керування своїм робочим місцем
- сортування та обробка пошти
- віртуалізація як спосіб позбавлення необхідності щоразу встановлювати та налаштовувати необхідний набір програм

@ М.В.Добролюбова

8