**Лабораторна робота № 09-4. Розподіл метрик тестування по рівнях тестування**

**Мета:** Навчитися формувати формальні специфікації функціональних елементів

**Методичні рекомендації до виконання лабораторної роботи.**

1. Опрацювати теоретичні відомості.
2. Розглянути надані в теоретичних відомостях метрики тестування та розподілити їх по рівнях тестування з урахуванням поділу на "підрівні", підходи та методи рівня. Заповнити таблицю:

|  |  |  |
| --- | --- | --- |
| №№ | Рівень тестування / підрівень / підхід / метод | Метрики тестування |
|  |  |  |
|  | … |  |
|  |  |  |

1. Робота повинна бути виконана згідно критеріїв оформлення документації та повинна містити

* Назва лабораторної роботи.
* Прізвище, група
* Назва проекту.
* Результати роботи оформлюються у вигляді таблиці:

По закінченню практичну роботу потрібно здати на перевірку викладачеві, надіславши електронною поштою на адресу [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com) . Якщо викладач знаходить помилки чи неточності, він може повернути роботу на доопрацювання.

Файл з роботою повинен мати назву в такому форматі:

**QA<Номер групи><Номер лекції / практичної / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвище англійською>**.. Наприклад, **QA4101Р**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

Тему в заголовку листа записати

**QA <Номер групи>-ЛР<Номер лабораторної>-<Прізвище >**

**Строк виконання цієї роботи ІПЗ-41 – 08.04.2024**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Пригадаємо минулу ЛР.

**Рівні тестування (Testing levels):**

* Компонентне або Модульне тестування (Component testing or Unit testing)
* Інтеграційне тестування (Integration testing).

Рівні інтеграційного тестування:

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

Два підходи до системного тестування:

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

Методи цього рівня.

* Тестування замовником самостійно.
* Тестування (Аудит) третьою стороною.
* Спільне тестування за сценаріями із замовником.

# **Базові метрики**

Метрики вводяться з метою отримати відповіді на запитання:

1. Які критерії оцінки?
2. З чим порівнювати?
3. Де узяти той еталон на який орієнтуватися?
4. Яким чином ідентифікувати проблеми і оперативно зреагувати, ще до того як вони стали невиправними?
5. Як зрозуміти, що саме слід покращувати у себе на проекті?

Відповідний **тестовий показник (test metric)** дозволит визначити його середнє значення та відстежувати прогрес згідно цього показника тестування чи набору показників протягом наступних тижнів, місяців, років.

**Метрики потрібні, щоб ефективно управляти проектом:**

* Інформування.
* Передбачати різноманітні моменти.
* Розуміти що саме потрібно вдосконалювати?
* Для оптимізації процесів.
* Діагностувати проблеми, локалізувати, виправляти і перевіряти їх.
* Приймати правильні рішення.

Метрики важливі на усьому циклі розробки програмного забезпечення **(SDLC)**. Проте потрібно аналізувати аспект ефективності застосування кожної метрики з точки зору, на якому етапі розробки проекту від отриманої інформації ми отримаємо максимальний результат.

**Метрика** — це міра, яка дозволяє отримати числове значення деяких властивостей ПЗ.

**Метрика** — іншими словами це показник поточного стану досягнення цілі. Оскільки керівництву, клієнтам є до вподоби гарна аналітика, красиві графіки, показники тощо, потрібно правильно враховувати при підборі метрик специфіку проекту **(потреби проекту), так званий контекст** і орієнтуватись на цільову аудиторію (кому маємо їх представляти).

**Які головні цілі метрик?**

* Зменшити кількість дефектів.
* Пришвидшити строки релізу.

**Test Metrics Life Cycle – життєвий цикл метрик тестування**

**Аналіз метрик** — на цьому етапі ми ідентифікуємо метрики і показники якості (QA), які допоможуть нам вирішити задачі і досягти поставлених цілей.

**Узгодження** — на цьому етапі тест-менеджер пояснює необхідність збору обраного набору метрик зацікавленим сторонам команді і замовнику відповідно.

**Оцінка** — перевіряємо отримані дані. Обчислюємо значення показників на основі зібраних даних.

**Звітування** — підсумовуємо наші метрики у звіт. Отримуємо відгуки про те наскільки вони виявились ефективними від зацікавлених сторін.

**Дії:**

* **Крок #1** Визначити ключові процеси тестування програмного забезпечення, що підлягають вимірюванню.
* **Крок #2** Визначення частоти відстеження та призначення відповідальної особи.
* **Крок #3** Збір Базових показників (дивись нижче ).
* **Крок #4** Обчислення, управління та інтерпретація визначених показників.
* **Крок #5** Визначення точок вдосконалення залежно від інтерпретації метрик.

**Типи Метрик**

Метрики поділяють на **Базові (Базові показники/Base Metrics)** і **Обчислені показники (Calculated metrics).**

**Базові показники** їх ще називають **абсолютними** — це необроблені дані, зібрані тестувальниками.

* Загальна кількість Тест-кейсів (Total number of test cases)
* Кількість пройдених Тест-кейсів (Number of test cases passed)
* Кількість зафейлених Тест-кейсів (Number of test cases failed)
* Кількість заблокованих (Number of test cases blocked)
* Кількість оброблених вимог
* Кількість знайдених багів (Number of defects found)
* Кількість прийнятих (Number of defects accepted)
* Кількість відхилених (Number of defects rejected)
* Кількість відкладених (Number of defects deferred)
* Кількість критичних (Number of critical defects)
* Кількість багів після доставки (Number of bugs found after shipping)
* Кількість запланованих годин (Number of planned test hours)
* Кількість реальних годин (Number of actual test hour) тощо.

**Обчислені показники або похідні** — виводяться з співвідношення даних, зібраних у базових показниках.

**Метрики, які вимірюють результат** — переважно це є абсолютні показники, які основною мірою відображають завершення активності / процесу. Приклад: час, необхідний пройти набір тест-кейсів.

**Прогнозовані** **метрики** — похідні показники, які виступають як ранні попереджувальні ознаки. Приклад: Діаграма, яка показує швидкість виправлення дефектів привертає увагу команди, якщо швидкість є нижча, аніж потрібна швидкість.

**Метрика процесу - Process Metrics: за її** допомогою можна підвищити ефективність процесу SDLC (життєвий цикл розробки програмного забезпечення)

**Метрика продукту - Product Metrics:** стосується якості програмного продукту

**Метрика проекту - Project Metrics: вона може бути використана для вимірювання ефективності проектної команди або будь-яких інструментів тестування, які використовуються членами групи**

Метрики ще поділяють **метрики ручного тестування, автоматизованого.**

Групують метрики ще за сутностями та задачами:

### ***Етап вимог***

Ця група метрик дозволить оцінити, нам якість документації, її складність. Наскільки пропрацьовані вимоги (user story).

**Аналіз вимог** — Співвідношення кількості вимог до днів за які вони будуть опрацьовані. Зазвичай формується графік, який візуально показує настільки швидко опрацьовуються вимоги. В такому графіку витрат можна відображати будь-які базові показники, візуалізуючі динаміку руху і надаючи можливість визначити чи не має "просідань".

**Вимірювання рейтингу вимог**, як кожної окремої вимоги так і набору вимог. Нам потрібно у табличці по кожній вимозі проставляти оцінки свої: повнота, характеристика можливості реалізувати ту чи іншу вимогу, характеристика, що ми зможемо вимогу перевірити і у нас є для цього усі необхідні інструменти. Наприклад: R1, R2, R3. Склавши рейтинг можна візуально бачити, які задачі можемо брати у роботу, а які вимоги слід доопрацювати. В процесі роботи над вимогами потрібно відслідковувати по цій табличці і бачити прогрес, які вимоги повністю опрацьовані, а з якими потрібно ще розбиратися.

**Метрика покриття вимог = (Кількість охоплених вимог / Загальна кількість вимог)х 100**

**Покриття вимог тестами** — метрика обраховується дуже просто загальну кількість тестів ділимо на загальну кількість вимог. ***Зверніть увагу*** деякий функціонал ми не можемо покрити тест-кейсами але можемо покрити тестами дослідницького тестування.

**Покриття вимог тестами = (Загальна кількість тестів/Загальна кількість Вимог)х100** для показника у відсотках.

**Наскільки вимоги є взаємопов’язані одна з одною** — дана метрика обчислюється як середня кількість зв’язків кожної вимоги з іншими вимогами. Призначення метрики: дати підставу для оцінки термінів тестування і врахування можливих ризиків. Знаючи ступінь взаємного впливу вимог одна на одну можна, наприклад, запланувати додатковий час і кейси для наскрізного тестування, опрацювати регресивні тести, подивитися в бік інтеграції тощо.

Значення цієї метрики буде коливатися від 0 до 1. 1 означає, що кожне вимога пов’язана, а 0 — що взаємозв’язків немає. Складно вводити якісь обмеження для значень даного коефіцієнта, багато що залежить від специфіки функціоналу, архітектури, технологій. Якщо показник високий, то значить доопрацювання однієї з вимог буде вести до ланцюжка змін, а значить і можливих помилок, в значній частині продукту.

**Коефіціент стабільності вимог** — призначення метрики показати, як багато вже реалізованих вимог доводитися переробляти від релізу до релізу при розробці нових аспектів (фіч). Кількість нових вимог повинно переважати над змінними а коефіцієнт бажано повинен бути менше 0,5. В цьому випадку ми впроваджуємо нові аспекти в 2 рази більше, ніж переробляємо існуючих. Якщо коефіцієнт вище 0,5, особливо якщо більше 1, то це швидше за все означає, що раніше ми зробили те, що виявилося непотрібним. Команда фокусується не на створенні нових цінностей для бізнесу, а на переробленні раніше випущених змін (фіч). Також метрика дає уявлення про те, наскільки легко масштабується функціонал системи, додаються нові можливості.

**Коефіціент Стабільності Вимог** **=** **Кількість змін у вимогах/Загальну кількість вимог реалізованих за ітерацію, включаючи нові**

### ***На етапі тест Дизайну***

Метрика складності Тест-Кейсів. Скільки кроків містить тест кейс. Ця метрика є надзвичайно важливою у Agile проектах. Суть її — прорахувати скільки Story-point робить команда за цикл.

Час виконання Тест-Кейсів . Показник може дати можливість порахувати яку кількість тестувальників і якого саме рівня нам потрібно щоб успішно випустити реліз і до визначеного часу.

### ***На етапі функціонального тестування***

На етапі функціонального тестування метриками ми перевіряємо саму якість розробки ПЗ.

* **Щільність Дефектів (Defect Density)** — це кількість дефектів на одиницю або весь продукт, спринт, функцію, об’єм вихідного коду тощо. Ціль знизити тут кількість дефектів. Обчислюється частка дефектів, яка припадає на окремий модуль протягом ітерації або релізу. Призначення метрики: показати нам, яка частина ПЗ є найбільш проблемною. Ця інформація допоможе при оцінці і плануванні робіт з даним модулем а також при аналізі ризиків. Причини великої кількості дефектів у якомусь одному конкретному модулі (коефіцієнт більше 0,3) можуть бути різні: неякісні вимоги, низька кваліфікація розробника, технічна складність тощо. У будь-якому випадку дана метрика відразу зверне увагу на проблемну зону.

**Defect Density = Defect Count/Size of the Release/Module**

* **Походження Дефектів (Defect Original)** потрібно позначати на якому етапі виник дефект. На етапі вимог, на етапі архітектури, на етапі кодування тощо.
* **Час життя багу (Defect Life** time ) скільки часу фіксували дефекти. Цей показник важливий для важливих і блокуючих дефектів.
* **Статус Тестового набору (Test Suite Status)** показує скільки тест-кейсів пройдено, скільки викликали проблеми (fail), скільки пройшло.
* **Коефіціент Ретесту дефектів (DefectTest Index)** показує, яку частину в тестуванні займає перепровірка дефектів. Допустима норма коли на кожні 10 нових дефектів повинно бути не більше 1 старого.
* **Час походження дефектів (TC Execution Time)** характеристика важлива для регресивного тестування, за якою можна виміряти скільки часу потрібно на протестовування старого функціоналу.

***Метрика запровадження і супроводу***

* **Ефективність усунення дефектів (Defect Removal Efficiency) —** співвідношення зафіксованих та виправлених дефектів до загальної кількості знайдених дефектів.
* **Коефіціент повторно відкритих багів** **—** метрика тестування, яка надає можливість оцінювати якість, професійність розробників, а також визначає у якісь мірі складність продукту або його певної частини.

**Кількість повторно виявлених багів/Загальну кількість помилок, включно із виправленими.**

Дану метрику можна розраховувати для усієї програми, окремого модуля або функціональності. У випадку отримане значення ближче до 0, тим менше при розробці повторюються старі помилки. Якщо коефіцієнт більше 0,2-0,3, це може свідчити або про технічну складність модуля і високу пов’язаність вимог у ньому, або суттєві недоліки в архітектурі, або попередній фікс був зроблений неякісно.

* **Метрика кількості пропущених багів (Defect Leakage)** небезпечний показник, показує скільки пропущених дефектів просочилось замовнику.

**Defect Leakage= (Загальна кількість пропущений дефектів/Загальна кількість виявлених дефектів) x 100**

***Ефективність і якість роботи команди тестування***

Завдання цього набору метрик оцінити наскільки якісно тестувальники виконують свої завдання, визначити рівень компетенцій і зрілості команди QA. Володіючи таким набором показників можна порівнювати команду з нею ж самою у різні моменти часу або з іншими, зовнішніми групами тестування.

**Швидкість роботи (velocity) команди QA** **—** розраховується як відношення реалізованих story points (або вимог, або user stories) за кілька, наприклад, 4-5 ітерацій (Sprint) до кількості обраних ітерацій.

**Velocity =** **Кількість story points за N ітерацій/N**

Призначення метрики виявити можливості роботи команди для подальшого планування обсягу робіт і аналізу трендів її розвитку. Також метрика дозволяє стежити за швидкістю роботи QA, спостерігати за тим, які внутрішні процеси чи зовнішні впливи на команду можуть на цю швидкість вплинути.

**Ефективність тестів і тестових наборів —**метрика є спірною. Відображає як багато помилок в середньому дозволяють виявити тест-кейси. Ця метрика каже про якість тест дизайну.

Найкраще розраховувати дану метрику для всіх наборів тестів, проте не заборонено і по окремому.

**Кількість виявлених помилок/Кількість тест-кейсів у тестовому наборі**

Коефіцієнт помилок, пропущених на продукті

**Реальний час роботи команди QA —** призначення метрики збільшити точність планування, а по-друге, відстежувати і керувати ефективністю роботи команди. Природно, даний коефіцієнт ніколи не буде дорівнює 1. Практика показує, що для ефективних команд він може становити 0,5-0,6.

**Кількість часу витраченого на цільові QA активності/Загальна кількість часу**

**Метрика додаткових витрат** - це витрати часу скажімо на зібрання/обговорення, налаштування середовища, пусте очікування якогось функціоналу, який є ще неготовий до загального часу.

**Точність оцінки часу по видам, типам робіт = Оціночний час роботи/Фактичний час роботи** **— це** коефіціент, який буде у нас коригувальний до наступних оцінок, щоб вони були точнішими при плануванні.

**Частка непідтверджених (відхилених) дефектів = Кількість неприйнятих дефектів/Загальна кількість заведених дефектів.**

Призначення метрики показати скільки дефектів були заведені «вхолосту». Якщо частка дефектів, які були відхилені перевищує 20%, то в команді може спостерігатися відсутність розуміння, що є дефектом, а що ні або зовсім низькою кваліфікацією QA.

***Метрики зворотного зв’язку***

Визначають наскільки користувачі задоволені тим що для них виробили.

Задоволеність користувачів ІТ сервісом

Регулярний опитування задоволеності користувачів сервісом ІТ з виставленням балів.

Призначення метрики: показати, чи довіряють користувачі продукту, наскільки його робота виправдовує їх очікування. Надає можливість дізнатись чи в тому напрямку рухаємося, чи правильно визначаємо важливість нових змін(фіч) і вибираємо варіанти рішень.

Показники тестування програмного забезпечення також допомагають оцінити, наскільки ефективні тестові зусилля та наскільки продуктивна тестова команда.

**Метрики покриття й глибини тестування**

***Метрики підрахунку дефектів***

Програма, що тестується, може оцінюватися на основі підрахунку та класифікації знайдених дефектів. Для кожного класу дефектів можна визначити відношення між кількістю відповідних дефектів і розміром програми (в термінах обраних метрик оцінки розміру).

Статистичні очікування щодо надійності програмної системи можуть використовуватися для ухвалення рішення про продовження або припинення (закінчення) тестування, виходячи із заданих параметрів прийнятної якості (наприклад, щільності дефектів заданого класу).

**Моделі зростання надійності** (Reliability growth models).

Дані моделі забезпечують можливості прогнозування надійності системи, базуючись на виявлені збоїв. Моделі такого роду розбиваються на дві групи:

* за кількістю збоїв (failure-count);
* часу між збоями (time-between-failure).

**Оцінка виконаних тестів** (Evaluation of the tests performed).

**Метрики покриття / глибини тестування** (Coverage / thoroughness measures).

Критерії "адекватності" тестування, у ряді випадків, вимагають систематичного виконання тестів для певного набору елементів програми, що задаються її архітектурою або специфікацією. Відповідні метрики дозволяють оцінити ступінь охоплення характеристик системи (наприклад, відсоток тестованих різних параметрів продуктивності) і глибину їх деталізації (наприклад, випадкове тестування параметрів продуктивності або з урахуванням граничних значень і т.п.). Такі метрики допомагають прогнозувати розподіл усіх досягнень заданих параметрів якості системи.

**Введення штучних дефектів** (Fault seeding).

На практиці, цей підхід допомагає класифікувати можливі помилки і наступні за ними збої, застосовуючи в подальшому отримані результати для моделювання можливих причин реальних збоїв, виявлених в процесі тестування.

Безумовно, ця техніка повинна використовуватися з максимальною обережністю досвідченими фахівцями.

**Критерії відбору тестів / критерії адекватності тестів**, правила припинення тестування (Test selection criteria / test adequacy criteria, or stopping rules).

Критерії відбору тестів можуть використовуватися як для створення набору тестів, так і для перевірки, наскільки вибрані тести адекватні вирішуваних завдань (тестування). При цьому, обговорювані критерії допомагають визначити, коли можна або необхідно припинити тестування.

***Ефективність тестування***

*Тестування* — це спостереження за виконанням програми, запущеної з метою тестування з заданими параметрами, за заданим сценарієм або з іншими заданими початковими умовами або цілями тестування. Ефективність тесту може бути визначена лише в контексті заданих умов.

**Тестування для ідентифікації дефектів** (Testing for defect identification).

Даний випадок тестування на увазі успішність процедури тестування, якщо дефект знайдено. Це відрізняється від підходу у тестуванні, коли тести запускаються для демонстрації того, що програмне забезпечення задовольняє пропонованим вимог і, відповідно, тест вважається успішним, якщо не знайдено дефектів.

***Проблема оракула***

"Оракул", в даному контексті, будь-який агент (людина або програма), що оцінює поведінку програми, формулюючи вердикт — тест пройдено ("pass") чи ні ("fail").

**Теоретичні і практичні обмеження тестування** (Theoretical and practical limitation of testing)

Теорія тестування виступає проти необгрунтованого рівня довіри до серії успішно пройдених тестів. На жаль, більшість встановлених результатів теорії тестування — негативні, означаючи, за словами Дейкстри (Dijkstra), те, що "тестування програми може використовуватися для демонстрації наявності дефектів, але ніколи не покаже їх відсутність". Основна причина цього в тому, що повне (всеосяжне) тестування недосяжне для реального програмного забезпечення.

**Проблема нездійсненних шляхів** (The problem of infeasible paths)

Ця складна проблема автоматизованого тестування пов’язана з тим, що шлях, по якому виконуються потоки робіт тестованої програмної системи, не можуть бути задані вхідними параметрами.

***Придатність до тестування*** (Testability)

Це поняття може мати на увазі дві різні ідеї:

* Перша описує ступінь легкості опису критеріїв покриття тестами для заданої програмної системи.
* Друга визначає можливість ймовірності, можливість статистичного вимірювання того, що під час тестування проявиться збій програмної системи. Обидві інтерпретації цього поняття однаково важливі для тестування.

***Зв’язок тестування з іншими видами діяльності*** (Relationships of testing with other activities)

Тестування програмного забезпечення відрізняється від статичних технік управління якістю і перевірки коректності, налагодження та програмування, але пов’язане з усіма цими роботами. Слід розглядати тестування з точки зору аналітиків і фахівців з сертифікації якості.

***Розробка, що управляється тестуванням*** (Test—driven development)

Це не стільки техніка тестування, скільки стиль організації процесу розробки, життєвого циклу, коли тести є невід’ємною частиною вимог (і відповідних специфікацій) замість того, щоб розглядатися незалежною діяльністю з перевірки задоволення вимог програмною системою.

За матеріалами https://medium.com/jonathans-musings/ab-testing-101-5576de6466b

**Питання для самоконтролю:**

**1.** Надати визначення метрикам покриття, глибині тестування (Coverage / thoroughness measures).

**2.** Пояснити, навіщо вводять штучні дефекти (Fault seeding) під час тестування програмних продуктів.

**3.** Навести критерії відбору тестів, критерії адекватності тестів, правила припинення тестування (Test selection criteria / test adequacy criteria, or stopping rules).

**4.** Описати тестування для ідентифікації дефектів (Testing for defect identification), обґрунтувати доцільність його використання.

**5.** Пояснити в чому полягає проблема оракула (The oracle problem), навести приклади.

**6.** Обґрунтувати теоретичні та практичні обмеження тестування.

**7.** Проаналізувати особливості розробки програмного забезпечення, керованої тестуванням (Test-driven development).

**8.** Зв’язок тестування з іншою діяльністю (Relationships of testing with other activities).