Силабус “Foundation Level v4.0.1”

Неофіційний переклад

створений Certified Unicorns Community

за підтримки

USQB Ukrainian Software Quality Board

Передмова

За 7+ років підготовки тестувальників до сертифікації ISTQB Foundation Level ми неодноразово стикалися з питанням, чи існує україномовний силабус. Офіційна відповідь — ні, не існує.
Мало за те, ми завжди були проти перекладів матеріалів ISTQB будь-якою національною мовою, бо частиною місії ISTQB-спільноти є поширення універсальної професійної термінології тестування у всьому світі, а отже англійською.

У чому виклик перекладів?
В офіційних вимогах зазначено: перекладатися має повністю все — пояснення, терміни тощо. Проте більшість термінів на роботі ми з вами використовуємо лише англійською. Плюс в ISTQB багато термінів мають по декілька синонімів. Наприклад, для позначення сутності “bug” у різних контекстах існує 8 термінів: error, mistake, bug, defect, fault, failure, incident, anomaly. Придумувати 8 українських відповідників, якими ніхто не користується в роботі — на нашу думку, не має сенсу.

Нащо тоді ця ініціатива?
Проте під час підготовки чимало українських тестувальників стикаються зі специфічністю ISTQB-англійської. Текст в силабусі закручений, терміни незрозумілі, контекст губиться, і часто неможливо нічого запам’ятати, бо переклад забирає більше зусиль, ніж суть. Тож переклад самих пояснень точно має сенс.
Крім того, наявність національного перекладу є ознакою активності української спільноти тестувальників на світовій арені. Це — можливість підсвітити потужну культуру тестування у нашій країні.

Особливості українського силабусу

Ініціативу створення першого україномовного силабусу ISTQB Foundation Level v4.0 підтримали USQB — офіційне представництво ISTQB в Україні. Цей переклад ще не є офіційним, але в нього вже вкладений досвід та експертиза спільноти випускників Certified Unicorns. Ми викладаємо його в публічний доступ частинами по готовності, щоб будь-хто міг долучитися до його створення і використання.

Вам буде корисно, тому що:
- терміни залишаються англійською
- перекладаються тільки пояснення (і в цьому фіча, одразу видно терміни)
- всі пояснення правильно перекладені в контексті тестування

Для підготовки до іспиту НЕ можна:
- використовувати тільки український силабус
- завчати українські визначення для тестів (бо українського тесту не існує, лише англійський)
- не читати англійську версію силабусу зовсім — укр. силабус тільки інструмент для полегшення сприйняття інформації, що допоможе подружитись з оригіналом

Переклад здійснюється суто для Chapters 1 – 6 (без вступу та додатків) і публікується поступово у хронологічному порядку. За деталями слідкуйте на сторінці ініціативи https://certifiedunicorns.pro/ukrainian-istqb-syllabus.
Правки, зауваження та пропозиції про покращення надсилайте на: *istqb.community@gmail.com*

Chapter 1. Fundamentals of Testing

Keywords [терміни, які обов'язково вчити з Glossary]
coverage, debugging, defect, error, failure, quality, quality assurance, root cause, test analysis, test basis, test case, test completion, test condition, test control, test data, test design, test execution, test implementation, test monitoring, test object, test objective, test planning, test procedure, test process, testresult, testing, testware, traceability, validation, verification

Learning Objectives [пункти для самоперевірки від ISTQB]

1.1 What is Testing?
FL-1.1.1 (K1) Визначте типові цілі тестування
FL-1.1.2 (K2) Навчіться відрізняти testing від debugging

1.2 Why is Testing Necessary?
FL-1.2.1 (K2) Поясніть на прикладах, чому потрібне тестування
FL-1.2.2 (K1) Пригадайте, як тестування пов'язано із quality assurance
FL-1.2.3 (K2) Потренуйтесь розрізняти поняття root cause, error, defect і failure.

1.3 Testing Principles
FL-1.3.1 (K2) Поясніть 7 ключових принципів тестування

1.4 Test Activities, Testware and Test Roles
FL-1.4.1 (K2) Поясніть, які існують тестові активності та які задачі з ними пов'язані
FL-1.4.2 (K2) Розкажіть про вплив контексту проєкту на процес тестування
FL-1.4.3 (K2) Впевніться, що орієнтуєтесь в типах testware [тестових артефактів] в залежності від тестової активності
FL-1.4.4 (K2) Пояснити, чому важливо підтримувати traceability
FL-1.4.5 (K2) Порівняйте різні ролі в тестуванні

1.5 Essential Skills and Good Practices in Testing
FL-1.5.1 (K2) Наведіть приклади універсальних навичок, необхідних тестувальнику
FL-1.5.2 (K1) Пригадайте переваги whole team approach
FL-1.5.3 (K2) Назвіть переваги та недоліки independence of testing

1.1. What is Testing?

Програмні системи є невід'ємною частиною нашого повсякденного життя. Більшість людей мали досвід використання програмного забезпечення, яке працювало не так, як очікувалося. Некоректна робота програмного забезпечення може призвести до численних проблем, включно з втратою грошей, часу чи ділової репутації, а в крайніх випадках – навіть до травм або смерті. **Software testing** оцінює якість програмного забезпечення та допомагає зменшити ризик відмови програмного забезпечення під час експлуатації

Software testing – це набір дій, спрямованих на виявлення дефектів та оцінку якості software **work products** [зверніть увагу! коректний переклад виразу - результати роботи, а не "робочі продукти"]. Ті work products [результати роботи], що підлягають тестуванню, називаються **test objects**. Поширеним упередженням щодо тестування є те, що воно зводиться лише до виконання тестів (тобто запуску програмного забезпечення та перевірки **test results**). Однак, software testing також включає інші види діяльності та має бути узгодженим із життєвим циклом розробки програмного забезпечення (див. розділ 2).

Ще одне поширене хибне уявлення про тестування полягає в тому, що воно повністю зосереджується на перевірці test object. І хоча тестування дійсно передбачає **verification**, тобто перевірку відповідності системи визначеним вимогам, воно також включає **validation**, що означає перевірку відповідності системи потребам користувачів та інших зацікавлених сторін на її operational environment [робочому середовищі – тобто на продакшені].

Тестування може бути динамічним або статичним. **Dynamic testing** передбачає запуск програмного забезпечення, тоді як **static testing** – ні. Static testing включає в себе **reviews** та **static analysis** (див. розділ 3). Dynamic testing використовує різні типи **test techniques** [техніки тест дизайну] і **test approaches** для створення **test cases** (див. розділ 4).

Тестування – це не лише технічна діяльність. Воно також потребує належного планування, управління, оцінювання, моніторингу та контролю (див. розділ 5).

Тестувальники використовують інструменти (див. розділ 6), але важливо пам'ятати, що тестування є переважно інтелектуальною діяльністю. Вона вимагає від тестувальників спеціалізованих знань, застосування аналітичних навичок, а також критичного та системного мислення (Myers 2011, Roman 2018).

Стандарт **ISO/IEC/IEEE 29119-1** надає додаткову інформацію щодо концепцій тестування програмного забезпечення.

1.1.1. Test Objectives

Типові цілі тестування [**test objectives**]:

- оцінювання work products [результатів роботи], таких як вимоги, user stories, дизайни та код;
- спричинення **failures** і виявлення **defects**;
- забезпечення необхідного покриття об'єкту тестування [test object];
- зменшити ризик, пов'язаний із низькою якістю програмного забезпечення;
- перевірити виконання зазначених вимог;
- переконатися, що test object відповідає договірним, правовим і нормативним вимогам;
- надати інформацію зацікавленим сторонам, щоб вони могли ухвалювати зважені рішення;
- зміцнення довіри [впевненості] до якості об'єкта тестування;

- підтвердження [validating] того, що test object є завершеним і працює так, як очікують зацікавлені сторони.

Цілі тестування можуть відрізнятися залежно від контексту: який саме work product тестується, рівня тестування, ризиків, який життєвий цикл розробки (**SDLC**) використовується, а також чинників, пов'язаних з бізнес-контекстом. Наприклад, корпоративна структура, конкурентні міркування чи час виходу на ринок.

1.1.2. Testing and Debugging

Testing і **debugging** – це окремі види діяльності. Testing може спровокувати failures, викликані by defects [дефекти] у програмному забезпеченні (dynamic testing), або безпосередньо виявити defects в об'єкті тестування (static testing).

Якщо під час **dynamic testing** (див. розділ 4) вдається спровокувати failure, debugging зосереджується на пошуку його причин (дефектів), аналізі цих причин і їх усуненні. Зазвичай процес debugging складається з трьох кроків:

- відтворення failure;
- діагностика (пошук defect);
- виправлення дефекту.

Після виправлень [fixes] проводять **confirmation testing**, щоб переконатися, що проблема справді зникла. Найкраще, якщо його робить та сама людина, яка виконувала початковий тест. Додатково може проводитися **regression testing**, щоб перевірити, чи не спричинили виправлення failures в інших частинах об'єкта тестування (див. розділ 2.2.3 для деталей).

Коли **static testing** виявляє дефект, debugging зосереджується на його виправленні. Немає потреби у відтворенні чи діагностиці, оскільки static testing безпосередньо знаходить defects і не може спричиняти failures (див. розділ 3).

1.2. Why is Testing Necessary?

Testing, як форма **quality control**, допомагає досягати узгоджених цілей тестування в межах визначених обсягу [scope], часу, якості та бюджету. Внесок **testing** в успіх не повинен обмежуватися лише діяльністю команди тестування. Будь-який stakeholder [зацікавлена сторона] може використати свої навички тестування, щоб наблизити проєкт до успіху. Тестування компонентів, систем і пов'язаних work products (наприклад, документації) допомагає виявляти дефекти у програмному забезпеченні.

1.2.1. Testing's Contributions to Success

Testing забезпечує економічно ефективний спосіб виявлення дефектів. Ці дефекти згодом можуть бути усунені за допомогою debugging – діяльності, яка не є частиною тестування. Таким чином, testing опосередковано сприяє підвищенню якості об'єкта тестування.

Тестування дає змогу напряму оцінювати якість test object на різних фазах SDLC. Отримані показники використовуються як частина ширшої діяльності з управління проєктом, сприяючи ухваленню рішень щодо переходу до наступного етапу SDLC, наприклад до рішення про реліз.

Тестування також виступає способом представити інтереси користувачів у проєкті розробки. Тестувальники стежать, щоб їхнє розуміння потреб користувачів враховувалося на всіх етапах

життєвого циклу розробки. Альтернативою є безпосередня участь репрезентативної групи користувачів у проєкті розробки, що зазвичай неможливо через високі витрати та складність знайти відповідних учасників.

Іноді тестування потрібне не лише для перевірки якості, а й для того, щоб виконати умови договору, відповідати законодавчим вимогам чи нормативним стандартам [regulatory standards].

1.2.2. Testing and Quality Assurance (QA)

Хоча люди часто вживають терміни **"testing"** та **"quality assurance (QA)"** [забезпечення якості] як взаємозамінні, testing і QA не є тотожними поняттями.

Testing – це орієнтований на продукт, коригуючий підхід, що зосереджується на активностях, які підтримують досягнення належного рівня якості. Тестування є основною формою **quality control**, але існують й інші підходи: формальні методи (наприклад, перевірка моделей чи доведення правильності), симуляції та прототипування.

QA [забезпечення якості] – це орієнтований на процес, превентивний підхід, який зосереджується на впровадженні та удосконаленні процесів. Він ґрунтується на припущенні, що якщо належний процес виконується коректно, то він приведе до якісного продукту. QA застосовується як до процесів розробки, так і до процесів тестування та є відповідальністю кожного учасника проєкту.

Test results використовуються як в **QA**, так і в **testing**. В тестуванні вони застосовуються для виправлення дефектів, тоді як в QA вони надають зворотний зв'язок щодо того, наскільки ефективно виконуються процеси розробки та тестування.

1.2.3. Errors, Defects, Failures, and Root Causes

Люди припускаються **errors** (синонім - **mistakes**), які породжують **defects** (синоніми - **faults, bugs**), що, у свою чергу, можуть призводити до **failures**. Причини появи **errors** різноманітні: тиск часу, складність work products, процесів, інфраструктури чи взаємодій, а також звичайна втома або недостатня підготовка.

Defects можуть бути виявлені у документації, на кшталт специфікації вимог або test script, у вихідному коді чи у допоміжних work products [артефактах], наприклад, у файлі збірки. Якщо такі defects залишаються непоміченими на ранніх етапах SDLC, вони часто тягнуть за собою інші проблеми на пізніших етапах. Якщо код з дефектом виконується, система може або не виконати потрібну дію, або зробити щось зайве – і тоді виникає **failure** [збій]. Є **defects**, які завжди викликають **failure**, якщо їх виконати; є ті, що проявляються лише за певних умов; а деякі можуть взагалі ніколи не призвести до **failure**.

Errors та **defects** не є єдиною причиною **failures**. Вони також можуть бути спричинені умовами навколишнього середовища, наприклад, коли радіація чи електромагнітні поля викликають дефекти у прошивці (firmware).

Root cause [першопричина] – це фундаментальна причина виникнення проблеми (наприклад, ситуація, що призводить до **error**). Root causes визначаються за допомогою **root cause analysis**, який зазвичай виконується у випадку виникнення **failure** або виявлення **defect**. Вважається, що шляхом усунення **root cause** подібним відмовам чи дефектам можна запобігти в майбутньому або зменшити частоту їх виникнення.

1.3. Testing Principles

Протягом років сформувалася низка ключових принципів, які дають загальні рекомендації, універсальні для тестування. У цьому силабусі розглянуто сім таких принципів.

1. **Testing shows the presence, not the absence of defects.**
   Тестування демонструє наявність, а не відсутність дефектів.

   Тестування може показати, що в об'єкті тестування є дефекти, але воно не може довести, що дефектів немає (Buxton, 1970). Тестування знижує ймовірність залишення не виявлених дефектів у об'єкті тестування, проте навіть якщо жодного не знайдено, це ще не означає, що об'єкт працює абсолютно правильно.

2. **Exhaustive testing is impossible**.
   Вичерпне тестування неможливе.

   Перевірити абсолютно все – неможливо, за винятком тривіальних випадків (Manna, 1978). Замість спроб виконати вичерпне тестування, слід застосовувати test techniques (див. розділ 4), пріоритизацію test cases (див. розділ 5.1.5) та risk-based testing (див. розділ 5.2), щоб оптимально спрямовувати тестові зусилля [test efforts].

3. **Early testing saves time and money.**
   Раннє тестування заощаджує час і гроші.

   Defects, усунуті на ранніх етапах процесу, не спричинятимуть подальших дефектів у похідних артефактах.Завдяки цьому зменшується вартість забезпечення якості: на пізніших етапах SDLC буде менше failures (Boehm, 1981). Щоб виявляти дефекти на ранніх етапах, варто якомога швидше починати як статичне тестування (див. розділ 3), так і динамічне тестування (див. розділ 4).

4. **Defects cluster together**.
   Дефекти гуртуються [скупчуються разом]

   Невелика кількість компонентів системи зазвичай містить більшість виявлених дефектів або є причиною більшості відмов під час експлуатації (Enders, 1975). Це явище ілюструє принцип Парето. Прогнозовані кластери дефектів, а також фактичні кластери, виявлені під час тестування чи експлуатації, є важливими вводними для risk-based testing (див. розділ 5.2).

5. **Tests wear out**.
   Зношування тестів

   Якщо постійно запускати ті самі тести, вони з часом гірше знаходять нові дефекти (Beizer, 1990). Щоб уникнути цього, періодичної потрібно оновлювати вже існуючі тести й тестові дані, та писати нові. Але бувають ситуації, коли повторення тих самих тестів корисне – наприклад, під час автоматизованого regression testing (див. розділ 2.2.3).

6. **Testing is context dependent.**
   Тестування залежить від контексту.

   Не існує єдиного універсального підходу до тестування. Тестування виконується по-різному в різних контекстах (Kaner, 2011).

7. **Absence-of-defects fallacy.**
   Відсутність дефектів оманлива.

   Помилково вважати, що одна лише верифікація програмного забезпечення гарантує успіх системи. Навіть якщо ретельно перевірити всі вимоги й виправити всі знайдені дефекти, результатом може стати система, яка не відповідає очікуванням користувачів, не підтримує бізнес-цілі замовника та програє конкурентам. Тому поряд із **verification** [верифікацією] обов'язково треба проводити й **validation** [валідацію] (Boehm, 1981).

1.4. Test Activities, Testware and Test Roles

Тестування залежить від контексту, але на високому рівні існують типові набори тестових активностей,без яких складно досягти цілей тестування. Разом вони утворюють **test process** [процес тестування]. Цей процес можна підлаштовувати під конкретну ситуацію, враховуючи різні фактори. Які саме активності ввійдуть у test process, як саме вони виконуватимуться та коли проводитимуться – зазвичай вирішують під час **test planning** [планування тестування] (див. розділ 5.1).

У наступних розділах розглядаються загальні аспекти test process: які існують активності та задачі, як впливає контекст, що таке **testware** [тестові артефакти], як відстежується зв'язок між **test basis** [базою тестування] та testware, а також які ролі беруть участь у тестуванні.

Більше деталей про test processes можна знайти у стандарті **ISO/IEC/IEEE 29119-2**.

1.4.1. Test Activities and Tasks

Test process зазвичай включає основні групи активностей, які описані далі. Попри те, що вони можуть виглядати як послідовні кроки, насправді їх часто виконують паралельно чи у кілька ітерацій. Ці дії зазвичай потрібно підлаштовувати під конкретну систему та проєкт.

Test planning [планування тестування] полягає у визначенні цілей тестування та виборі підходу, що найкраще допоможе досягти цих цілей, з урахуванням обмежень, накладених загальним контекстом. Детальніше про **test planning** пояснюється в розділі 5.1.

Test monitoring [моніторинг тестування] і **test control** [керування тестуванням]. Під **test monitoring** мається на увазі постійне відстеження всіх активностей тестування та порівняння реального прогресу з планом. А **test control** означає виконання кроків, необхідних для досягнення цілей тестування. Детальніше про test monitoring та test control йдеться в розділі 5.3.

Test analysis включає аналіз **test basis** для визначення функціональностей [features], які придатні для тестування. Визначаються та пріоритизуються відповідні **test conditions** з урахуванням пов'язаних ризиків та **risk levels** (див. розділ 5.2). Test basis і test object також оцінюються, щоб виявити можливі дефекти та оцінити, наскільки їх можна ефективно тестувати. Test analysis часто підсилюється застосуванням test techniques (див. розділ 4). **Test analysis** відповідає на запитання «що тестувати?» з точки зору вимірюваних **coverage criteria**.

Test design [проєктування тестів] включає перетворення **test conditions** на конкретні **test cases** та інші **testware** [тестові артефакти], наприклад test charters Ця активність також включає ідентифікацію coverage items, що підказують, які саме вхідні дані слід використати в test cases. Для цього можуть застосовуватися різні техніки тестування (див. розділ 4).Test design також включає визначення вимог до тестових даних, проєктування test environment та визначення необхідної інфраструктури та інструментів. По суті, **test design** відповідає на запитання «як саме тестувати?».

Test implementation [виконання тестів] включає створення або отримання всіх потрібних **testware** [тестових артефактів] для запуску тестування (наприклад, тестових даних). Test cases об'єднують у **test procedures**, а ті вже формують **test suites**. Створюються як ручні, так і автоматизовані test scripts. Далі test procedures розставляють за пріоритетом і планують у **test execution schedule** [графіку виконання тестів], щоб тестування відбувалося максимально ефективно (див. розділ 5.1.5). Test environment створено та перевірено на коректність налаштувань.

Test execution включає в себе запуск тестів відповідно до test execution schedule (виконуються **test runs** – прогони тестів). **Test execution** може бути ручним або автоматизованим. Воно може набувати різних форм, зокрема **continuous testing** [безперервного тестування] або сесій парного тестування (**pair testing sessions**). Фактичні результати [actual results] тестування порівнюються з очікуваними результатами [expected results]. Результати тестів реєструються. Аномалії [anomalies] аналізуються для виявлення їхніх ймовірних причин. Такий аналіз дає змогу повідомляти про аномалії на основі спостережуваних **failures** (див. розділ 5.5).

Test completion [завершення тестування] зазвичай відбувається на віхах проєкту (наприклад, під час релізу, завершення ітерації чи завершення рівня тестування). Для будь-яких невирішених дефектів створюються **change requests** [запити на зміни] або елементи **product backlog**. Визначається будь-яке **testware** [тестові артефакти], яке може бути корисним у майбутньому, та архівується або передається відповідним командам. Тестове середовище переводиться у погоджений стан. Тестові активності аналізуються для виявлення отриманих уроків [lessons learned] та можливостей покращення у майбутніх ітераціях, релізах або проєктах (див. розділ 2.1.6). Створюється звіт про завершення тестування (test completion report) і доводиться до відома зацікавлених сторін.

1.4.2. Test Process in Context

Тестування ніколи не існує окремо. Це завжди частина процесів розробки в організації. Його фінансують зацікавлені сторони, а головна мета тестування – допомогти виконати їх бізнес-запити. Тому як саме проводиться тестування, залежить від багатьох факторів, зокрема:

- **Stakeholders** [зацікавлені сторони]: їх потреби, очікування, вимоги, готовність співпрацювати;
- **Team members** [члени команди]: навички, знання, рівень досвіду, доступність, потреби у навчанні тощо;
- **Business domain** [бізнес-домен]: наскільки критичним є об’єкт тестування, які ризики вже визначені, які ризики вже визначені, які є потреби ринку, чи існують специфічні законодавчі вимоги;
- **Technical factors** [технічні чинники]: тип програмного забезпечення, архітектура продукту, використані технології тощо;
- **Project constraints** [обмеження проєкту]: обсяг робіт, час, бюджет, ресурси;
- **Organizational factors** [організаційні чинники]: структура організації, політики, практики;
- **SDLC** [життєвий цикл розробки ПЗ]: використовувані інженерні практики та методи розробки;
- **Tools** [інструменти]: наявність, доступність, зручність, відповідність вимогам.

Ці чинники позначаються на багатьох питаннях, пов’язаних із тестуванням, серед яких: **test strategy** [стратегія тестування], які техніки використовуються, наскільки автоматизованим буде тестування, необхідний рівень покриття [coverage], рівень деталізації testware [тестових артефактів], звітність з тестування та інші аспекти.

1.4.3. Testware

Testware [тестові артефакти] створюються як результат роботи під час тестових активностей, описаних у розділі 1.4.1. У різних організаціях ці робочі продукти можуть суттєво відрізнятися: як саме їх створюють, оформлюють, називають, структурують і підтримують. Правильне **configuration management** [керування конфігураціями] (див. розділ 5.4) допомагає зберігати

узгодженість і цілісність таких продуктів. Нижче ви знайдете не вичерпний, але доволі повний перелік work products vs test activities:

- До **test planning work products** належать: **test plan**, **test schedule** [графік тестування], **risk register** [реєстр ризиків], **entry criteria** [критерії початку] і **exit criteria** [критерії завершення] (див. розділ 5.1). **Risk register** – це список ризиків із вказанням їхньої ймовірності [risk likelihood], впливу [risk impact] та заходів для зменшення ризиків [risk mitigation] (див. розділ 5.2). Часто test schedule, risk register, entry criteria та exit criteria входять до складу test plan.
- До **test monitoring and test control work products** належать: test progress reports (див. розділ 5.3.2), документи з контрольними вказівками (див. розділ 5.3) і дані про ризики (див. розділ 5.2).
- **Test analysis work products** включають: пріоритизовані **test conditions** (наприклад, **acceptance criteria**, див. розділ 4.5.2) та **defect reports**, що стосуються дефектів у **test basis** (якщо їх не виправили одразу).
- **Test design work products** включають: пріоритизовані **test cases**, **test charters**, **coverage items**, вимоги до тестових даних і вимоги до тестового середовища.
- **Test implementation work products** включають: **test procedures**, ручні та автоматизовані тестові скрипти, **test suites**, тестові дані, **test execution schedule** та елементи тестового середовища. Прикладами елементів тестового середовища є: **stubs** [заглушки], **drivers** [драйвери], **simulators** [симулятори] та **service virtualizations** [сервісні віртуалізації].
- **Test execution work products** включають: **test logs** [журнали тестування] та **defect reports** [звіти про дефекти] (див. розділ 5.5).
- **Test completion work products** включають: **test completion report** (див. розділ 5.3.2), **action items** [задачі] для покращення наступних проєктів чи ітерацій, задокументовані **lessons learned** [отримані уроки] та **change requests** [запити на зміни] (наприклад, як елементи product backlog).

1.4.4. Traceability between the Test Basis and Testware

Щоб test monitoring і test control були ефективними, потрібно підтримувати простежуваність [**traceability**] усього test process. Це означає зв'язок між елементами test basis, відповідними testware (наприклад, test conditions, ризики, test cases), результатами тестів і знайденими дефектами.

Точна простежуваність [traceability] підсилює оцінку покриття тестами, тому вона є особливо корисною, якщо у test basis є чітко визначені coverage criteria.Вони можуть слугувати своєрідними показниками ефективності [key performance indicators], які підказують, наскільки вдалося виконати test objectives (див. розділ 1.1.1). Наприклад:

- Завдяки простежуваності test cases до requirements можна переконатися, що всі вимоги перевіряються тестами.
- Простежуваність test results до risks допомагає оцінити, який рівень залишкового ризику [residual risk] у test object.

Якісна простежуваність не тільки допомагає оцінювати coverage, а й дозволяє зрозуміти вплив змін, спрощує аудити та підтримує відповідність критеріям IT governance. Вона робить test progress reports і test completion reports зрозумілішими, адже містить інформацію про статус елементів test basis [тобто вимог]. Це також допомагає пояснити технічні аспекти тестування stakeholder-ам у доступній формі. Простежуваність дає змогу оцінити якість продукту, ефективність процесу та прогрес проєкту у контексті бізнес-цілей.

1.4.5. Roles in Testing

У цьому силабусі описано дві головні ролі в тестуванні: управління тестуванням [test management role] і саме тестування [testing role]. Які саме активності та задачі вони виконують, залежить від контексту проєкту та продукту, від навичок людей у цих ролях та від особливостей організації.

Test management role відповідає за увесь test process, за test team та за керівництво тестовими активностями. Здебільшого вона охоплює такі напрямки: test planning, test monitoring, test control і test completion. Але як саме ця роль реалізується, залежить від контексту. Наприклад, в Agile-розробці частину завдань test management може виконувати сама Agile team. Якщо ж завдання стосуються кількох команд чи всієї організації, ними можуть займатися test managers, які не входять безпосередньо до development team.

Testing role відповідає за технічний бік тестування. Вона зосереджується переважно на таких завданнях: test analysis, test design, test implementation і test execution.

Ці ролі можуть виконувати різні люди в різний час. Наприклад, роль test management може бути у team leader, test manager чи навіть development manager. Також буває, що одна людина одночасно виконує і роль testing, і роль test management.

1.5. Essential Skills and Good Practices in Testing

Навичка [skill] – це здатність добре виконувати певну роботу завдяки знанням, практиці та здібностям. Гарні тестувальники мають володіти низкою ключових навичок, щоб якісно виконувати свою роботу. Вони мають уміти працювати в команді та проводити тестування на різних рівнях test independence.

1.5.1. Generic Skills Required for Testing

Хоча ці навички можна вважати універсальними, саме для тестувальників вони критично важливі:

- Ґрунтовні знання в тестуванні: допомагає підвищити ефективність роботи (наприклад, використання test techniques).
- Уважність, акуратність, допитливість, здатність помічати деталі й діяти методично (щоб знаходити defects, зокрема ті, що найскладніше виявити).
- Гарні комунікативні навички, активне слухання, вміння працювати в команді (для ефективної взаємодії зі всіма stakeholders, передавання інформації іншим, зрозумілості повідомлень, а також для звітування й обговорення дефектів).
- Аналітичне мислення, критичне мислення, креативність (щоб підвищити ефективність тестування).
- Технічні знання (для підвищення ефективності тестування, наприклад, за допомогою відповідних test tools).
- Доменні знання (щоб розумітися з end users та business representatives і ефективно з ними спілкуватися).

Тестувальникам часто доводиться повідомляти неприємні новини. Що природно – люди схильні звинувачувати тих, хто приносить погані новини. Тому навички комунікації для тестувальників надзвичайно важливі. Результати тестування можуть виглядати як критика продукту або його

автора. Confirmation bias [підтверджувальне упередження, див. “когнітивні викривлення”] може заважати людям приймати інформацію, яка суперечить їх переконанням. Дехто може вважати тестування руйнівною діяльністю, хоча насправді воно значно підвищує якість продукту й сприяє успіху проєкту. Щоб змінити таке ставлення, інформацію про дефекти і збої варто подавати у конструктивний спосіб.

1.5.2. Whole Team Approach

Однією з важливих навичок для тестувальника є здатність ефективно працювати в командному контексті та робити позитивний внесок у досягнення цілей команди. **Whole team approach** – підхід, що походить з **Extreme Programming** (див. розділ 2.1), — ґрунтується саме на цій навичці.

Згідно **whole team approach** будь-який учасник команди, маючи потрібні знання й навички, може взятися за будь-яке завдання, а відповідальність за якість несе вся команда. Учасники працюють у спільному просторі (фізичному чи віртуальному), бо спільне розташування полегшує комунікацію й співпрацю. Такий підхід покращує командну динаміку, підсилює взаємодію й комунікацію та створює синергію, адже дозволяє ефективно поєднувати різні навички команди на благо проєкту.

Тестувальники тісно співпрацюють з іншими членами команди, щоб забезпечити досягнення необхідного рівня якості. Це включає взаємодію з business representatives для допомоги у створенні відповідних acceptance tests та співпрацю з developers щодо узгодження test strategy й визначення підходів до test automation. Таким чином, тестувальники можуть передавати знання про тестування іншим членам команди та впливати на розробку продукту.

Залежно від контексту, whole team approach може бути не завжди доречним. Наприклад, у деяких ситуаціях, зокрема safety-critical, може знадобитися високий рівень test independence.

1.5.3. Independence of Testing

Певний рівень незалежності робить тестувальника більш ефективним у виявленні дефектів завдяки відмінностям між когнітивними упередженнями автора артефакту/коду та тестувальника (див. Salman, 1995). Водночас незалежність не замінює обізнаності: наприклад, програмісти можуть ефективно знаходити багато дефектів у власному коді.

Work products [артефакти] можуть тестуватися їх автором (no independence), колегами автора з тієї ж команди (some independence), тестувальниками з інших команд в межах організації (high independence) або тестувальниками ззовні організації (very high independence). Для більшості проєктів зазвичай найкраще виконувати тестування на кількох рівнях незалежності (наприклад, програмісти виконують component testing і component integration testing, а команда тестування проводить system та system integration testing, а business representatives виконують acceptance testing).

Основна перевага independence testing полягає в тому, що незалежні тестувальники здатні виявляти інші типи failures і defects, ніж розробники, завдяки своєму різному досвіду, технічним поглядам і упередженням. Крім того, незалежний тестувальник може перевіряти, ставити під сумнів або спростовувати припущення, зроблені stakeholders під час уточнення та реалізації системи.
Водночас існують і недоліки. Незалежні тестувальники можуть бути ізольованими від development team, що може призвести до нестачі співпраці, проблем комунікації або навіть конфліктних відносин із командою розробки. Програмісти можуть втратити відчуття відповідальності за якість. А ще незалежних тестувальників можуть вважати вузьким місцем [bottleneck] у процесі або звинувачувати у затримках із релізом.

Наступні розділи наразі в розробці.
TBD, як то кажуть**.**

Зверніть увагу!
Переклад здійснюється суто для Chapters 1 – 6 (без вступу та додатків) і публікується поступово у хронологічному порядку. За деталями слідкуйте на сторінці ініціативи https://certifiedunicorns.pro/ukrainian-istqb-syllabus.
Правки, зауваження та пропозиції про покращення надсилайте на: *istqb.community@gmail.com*

Сподіваємось, ви знайшли для себе корисне в українській версії сілабусу.
Лишаймось на хвилі ISTQB!