ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИШИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ» »

М А У П

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ЩОДО ОРГАНІЗАЦІЇ САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ З ДИСЦИПЛІНИ

" МЕТОДИ ТА ЗАСОБИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ "

(для магістрів)

Київ 2018

Підготовлено професором кафедри комп’ютерних інформаційних систем та технологій М.П.Дяченком.

Затверджено на засіданні кафедри комп’ютерних інформаційних систем та технологій
(Протокол № 1 від 23 серпня 2018 р.)

Схвалено Вченою радою Міжрегіональної Академії управління персоналом

Дяченко М.П. Методичні матеріали щодо забезпечення самостійної роботи студентів з дисципліни “Методи та засоби тестування програмного забезпечення” (для освітньо-кваліфікаційного рівня «магістр»). — К.: МАУП, 2018— 35 с.

Методична розробка містить пояснювальну записку, тематичний зміст, питання щодо самостійного вивчення студентами і самоконтролю та список літератури.

ПОЯСНЮВАЛЬНА ЗАПИСКА

Навчальний курс підпорядкований вивченню засобів контролю та забезпечення якості програмних виробів через наскрізне тестування всього процесу розробки систем в цілому і їх окремих компонент.

Метою курсу є формування у студента правильного розуміння завдання тестування у його зв'язку з надійністю та якістю програмного продукту, а також систематизація знань стосовно способів досягнення сприйнятного рівня надійності та якості продукту.

Завдання курсу полягає у розкритті змісту тестування як окремого зрізу процесу розробки програмного продукту, впливу тестування на показники якості продукту, а також у формуванні навичок з організації і проведення тестування програмного забезпечення довільного типу і рівня складності.

. **У результаті вивчення дисципліни:**

- *студент повинен знати* основні принципи, критерії, стратегію та етапи, тестування для досягнення необхідних характеристик якості програмного продукту, ознайомитись із формалізмом в організації тестування та засобами автоматизації тестування;
- *студент повинен вміти* створювати набори тестів для програм як на базі текстів програм та їх специфікацій, оцінювати якість тестового набору, проводити тестування програм, аналізувати результати тестування та оформляти відповідну документацію.

Даний методичний матеріал має на меті полегшити розв'язання завдань, які стоять перед студентом при вивченні курсу. Основна увага матеріалу концентрується на організаційних засадах тестування, тобто, налаштуванню процесу тестування, практичному здійсненню процесу тестування, створенню та веденню необхідної тестової документації.

ТЕМАТИЧНИЙ ПЛАН ДИСЦИПЛІНИ
« МЕТОДИ ТА ЗАСОБИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

№/п.п.	Назва змістового модуля та теми
Змістовий модуль І. Програмний продукт (ПП) та його властивості .	
Тема 1	Програмний продукт та технології його створення.
Тема 2	Поняття і характеристики зовнішньої якості ПП
Тема 3	Внутрішня якість та супроводжуваність ПП.
Змістовий модуль ІІ Контроль якості ПП через тестування	
Тема 4	Види тестування і проведення тестування.
Тема 5	Тестові сценарії.
Тема 6	Особливості тестування мобільних пристроїв та WEB застосунків
Змістовий модуль ІІІ. Автоматизоване тестування.	
Тема 7	Підходи та організація автоматизованого тестування
Тема 8	WEB-тестування та тестування програм мобільних пристроїв.
Тема 9	Модульне тестування і фреймворки модульного тестування
Змістовий модуль ІV. Забезпечення якості ПП	
Тема 10	Аудит ІТ-команди
Тема 11	Тестування вимог
Тема 12	Тестування процесу розробки ПП в цілому
	Разом годин : 150

1 Тестування програм і тест-план

1.1 Умови тестування

Тестування програмного забезпечення (ПЗ)– це процес дослідження ПЗ з метою отримання інформації про якість програмного продукту, а саме відповідність специфікації, технічному завданню, або вимогам замовника ПЗ.

Практичний підхід до тестування ПЗ особливу увагу приділяє процесам тестування на фоні стрімкого прискорення процесу розробки ПЗ.

Цей підхід орієнтований на використання спеціалістами з тестування ПЗ тестових робіт. Швидкість і ефективність розробки ПЗ залежить від того наскільки процес тестування вписується в загальний життєвий цикл розробки ПЗ і від ефективності використання технології тестування.

Тестування - це одна з технік контролю якості, що включає в себе діяльність з планування робіт (Test Management), проектуванню тестів (Test Design), виконанню тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Необхідними умовами для тестування є наявність :

- об'єкта тестування, доступного для проведення іспитів;
- виконавця(ів) (залежно від виду проведених іспитів їм може бути як людина, так і машина або комбінація людина + машина).

Достатніми умовами для тестування є наявність:

- об'єкта тестування, доступного для проведення іспитів;
- виконавця(ів) (залежно від виду діяльності на різних фазах їм може бути як людина, так і машина або комбінація людина + машина);
- плану тестування;

- тест кейсів / тестів;
- звіту, що підтверджує виконання задач і досягнення цілей, по тестуванню об'єкта.

В методичних вказівках викладені основні принципи, технології тестування, вимоги до документів з тестування згідно відповідних стандартів:

- тест план (тест план IEEE 829, тест план RUP, план приймально – здавальних випробувань RUP, план проведення навантажувального тестування);
- тест дизайн специфікації (тест дизайн специфікація MSF, тест дизайн специфікація IEEE 829-1998);
- тестовий випадок (test case);
- звіт про помилку (bug report).

1.2 Сутність тестування

Тестування програмного забезпечення (Software Testing) – це перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраних певним чином. (IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004) У більш широкому змісті, тестування - це одна з технік контролю якості, що включає в себе діяльність з планування робіт (Test Management), проектуванню тестів (Test Design), виконанню тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Верифікація (verification) програми і її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу (IEEE). Тобто чи виконуються цілі, терміни, задачі з розробки проекту, визначені на початку поточної фази.

Валідація (validation) - це визначення відповідності розроблювального ПЗ очікуванням і потребам користувача, вимогам до системи (BS7925-1).

План тестування (Test Plan) - це документ, що описує обсяг робіт з по тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи устаткування, спеціальних знань, а також оцінки ризиків з варіантами їхнього рішення.

Тест дизайн (Test Design) - це етап процесу тестування ПЗ, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критерій якості і цілей тестування.

Тестовий випадок (Test Case) - це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції або її частини, що тестуються.

Звіт про помилку/Дефект Репорт (Bug Report) - це документ, що описує ситуацію або послідовність дій, що призвели до некоректної роботи об'єкта тестування, із вказівкою причин і очікуваного результату.

Тестове Покриття (Test Coverage) - це одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується.

Деталізація Тест Кейсів (Test Case Detalization) - це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття.

Час Проходження Тест Кейса (Test Case Pass Time) - це час від початку проходження кроків тест кейса до одержання результату тесту.

Залежно від переслідуваних цілей види тестування можна умовно розділити на наступні типи: функціональне, не функціональне, пов'язані зі змінами.
Таблиця 1.1 – Класифікація видів тестування ПЗ

А Функціональне	Б Нефункціональне	В Пов'язані зі змінами
Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component/Unit testing), інтеграційному	Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, що можуть бути вимірювані різними величинами. У цілому, це тестування того, "Як" система працює. Основні види нефункціональних тестів:	Після проведення необхідних змін, таких як виправлення помилки - бага/дефекту, ПЗ має бути знов протестоване для підтвердження того факту, що проблема була дійсно вирішена. перераховані види тестування, які необхідно проводити після установки ПЗ, для підтвердження

(Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішнє поводження системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

працездатності додатка або вірності здійсненого виправлення дефекту:

А.1Функціональне тестування (Functional testing)

Б.1 Види тестування продуктивності:

Б.1.1 Навантажувальне тестування (Performance and Load Testing)

Б.1.2 Стресове тестування (Stress Testing)

Б.1.3 Тестування стабільності або надійності (Stability / Reliability Testing)

Б.1.4 Об'ємне тестування (Volume Testing)

В.1 Димове тестування (Smoke Testing)

А.2 Тестування безпеки (Security and Access Control Testing)

Б.2. Тестування установки (Installation testing

В.2 Регресійне тестування (Regression Testing)

А.3 Тестування взаємодії (Interoperability Testing):

А.3.1 Тестування сумісності

Б.3 Тестування зручності користування (Usability Testing)

В.3 Тестування зборки (Build Verification Test)

(compatibility testing);

А.3.2 Інтеграційне тестування (integration testing).

Б.4 Тестування на відмовлення і відновлення (Failover and Recovery Testing)

В.4 Санітарне тестування -перевірка погодженості/ справності (Sanity Testing)

Б.5 Конфігураційне тестування (Configuration Testing)

А.1 Функціональне тестування (Functional Testing) розглядає заздалегідь зазначену поведінку і ґрунтується на аналізі специфікацій функціональності компонента або системи в цілому.

Функціональні тести ґрунтуються на функціях, виконуваних системою, і можуть проводитися на всіх рівнях тестування (компонентному, інтеграційному, системному, приймальному). Як правило, ці функції описуються у вимогах, функціональних специфікаціях або у виді випадків використання системи (use cases).

Тестування функціональності може проводиться в двох аспектах:

- вимоги;
- бізнеси-процеси.

Тестування в перспективі «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити перелік того, що буде тестуватися, а що ні, визначають пріоритетність вимог на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього переліку пріоритетів формуються тестові сценарії (test cases). Це дозволяє сфокусуватися при тестуванні на важливішому функціоналі.

Тестування в перспективі «бізнеси-процеси» використовує знання цих самих бізнесів-процесів, що описують сценарії щоденного використання системи. У цій перспективі тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

А.2 Тестування безпеки (Security and Access Control Testing) – це перевірка безпеки системи, а також аналіз ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатка, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків. Ґрунтується на трьох основних принципах - *це конфіденційність, цілісність і доступність* (confidentiality, integrity, availability)

А.3 Тестування взаємодії (Interoperability Testing) – це функціональне тестування, що перевіряє здатність додатка взаємодіяти з одним і більш компонентами або системами, що включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).

Б.1 Види тестування продуктивності (Performance testing)

Задачею *тестування продуктивності (Performance testing)* є визначення масштабованості додатка під навантаженням, при цьому відбувається:

- вимір часу виконання обраних операцій при визначених інтенсивностях виконання цих операцій;
- визначення кількості користувачів, що одночасно працюють з додатком;
- визначення границь прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій);
- дослідження продуктивності на високих, граничних, стресових навантаженнях.

Б.1.1 Навантажувальне тестування (Load vs Performance Testing).
В англомовній термінології ви можете так само знайти ще один вид тестування - Load Testing - тестування реакції системи на зміну навантаження (у межах припустимого). Load і Performance переслідують одну й теж саму мету: перевірка продуктивності (часів відгуку) на різних навантаженнях. Власне тому їх не розділяють.

Б.1.2 Стресове тестування (Stress Testing) дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити

здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу.

Стресом у даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера.

Також однієї з задач при стресовому тестуванні може бути оцінка деградації продуктивності, у такий спосіб мети стресового тестування можуть перетинатися з цілями тестування продуктивності.

Б.1.3 Тестування стабільності або надійності (Stability / Reliability Testing) – це перевірка працездатності додатка при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження. Часи виконання операцій можуть грати в даному виді тестування другорядну роль.

При цьому на перше місце виходить відсутність витоків пам'яті, перезапусків серверів під навантаженням і інші аспекти, що впливають саме на стабільність роботи.

Б.1.4 Об'ємне тестування (Volume Testing) -це одержання оцінки продуктивності при збільшенні обсягів даних у базі дані додатки, при цьому відбувається:

- вимір часу виконання обраних операцій при визначених інтенсивностях виконання цих операцій;
- може вироблятися визначення кількості користувачів, що одночасно працюють з додатком

Б.2 Тестування установки (Installation Testing) направлено на перевірку успішної інсталяції і настроювання, а також відновлення або видалення ПЗ. Інсталяція відбувається автоматично вручну та за допомогою візардів.

Б.3 Тестування зручності користування (Usability Testing) - це метод тестування, спрямований на встановлення ступеня зручності використання, навчання, зрозумілості і привабливості для користувачів розроблювального продукту в контексті заданих умов.

В. Тестування, що пов'язане зі змінами

В.1 Димове тестування (Smoke Testing) спрямовано на поверхневу перевірку всіх модулів додатка на предмет працездатності і наявність швидкого знаходження критичних і дефектів, що блокують.

За результатами димового тестування робиться висновок про те, чи приймається чи ні установлена версія ПЗ в тестування, експлуатацію або на постачання замовникові. Для полегшення роботи, економії часу і людських ресурсів рекомендується автоматизувати димові тести.

B.2 Регресійне тестування (Regression Testing) спрямовано на перевірку змін, зроблених у додатку або навколишнім середовищі (налагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатка), для підтвердження того факту, що існуюча раніше функціональність працює як і колись. Регресійними можуть бути тести як функціональні, так і не функціональні.

Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки і тестування. Це дає гарантію того, що зміни в новій версії додатка не зашкодили вже існуючу функціональність. Рекомендується робити автоматизацію регресійних тестів, для прискорення наступного процесу тестування і виявлення дефектів на ранніх стадіях розробки програмного забезпечення.

Сам по собі термін "Регресійне тестування", залежно від контексту використання може мати різний сенс.

3 основних типи регресійного тестування:

- регресія помилок - багів (Bug regression) - спроба довести, що виправлена помилка насправді не виправлена
- регресія старих помилок - багів (Old bugs regression) - спроба довести, що недавня зміна коду або даних зламало виправлення старих помилок, тобто старі помилки - баги стали знову відтворюватися.
- регресія побічного ефекту (Side effect regression) - спроба довести, що недавня зміна коду або даних зламало інші частини розроблювального додатка

B.3 Тестування зборки (Build Verification Test)

Тестування спрямоване на визначення відповідності, випущеної версії, критеріям якості для початку тестування. По своїм цілям є аналогом Димового Тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглибину воно може проникати далі, залежно від вимог до якості випущеної версії.

B.4 Санітарне тестування або перевірка погодженості/справності (Sanity Testing). Вузькоспеціалізоване тестування достатнє для доказу того, що конкретна функція працює згідно заявленим у специфікації вимогам. Є підмножиною регресійного тестування.

Використовується для визначення працездатності визначеної частини додатка після змін зроблених у ньому або навколишньому середовищу. Звичайно виконується вручну.

1.3 Зміст тестового плану

Вступ (Introduction) подається в довільній формі

Мета (Purpose)

- Виявлення існуючих проектів і програмних компонентів, які необхідно перевірити.
- Перелік вимог для проведення випробування.
- Рекомендації щодо опису стратегії тестування.
- Визначення необхідних ресурсів і забезпечення оцінки випробувань.
- Перелік тестових елементів проекту.

Довідкова інформація (Background**)**

Опис елементів тестування (компоненти, додатки, системи тощо).
Інформація, щодо основних функцій і можливостей, архітектури, короткої історії проект

Галузь застосування (Scope), описують:

- Типи, наприклад, групи, інтеграцію, систему, етапи тестування (функціональність і п
- Перелік особливостей функцій, що будуть протестовані.
- Перелік пропозицій, що можуть вплинути на проектування, розробку тестування.
- Перелік усіх ризиків й непередбачуваних обставин, що можуть вплинути на розробку
- Перелік обмежень, які можуть вплинути на проектування, розробку тестування.

Визначення проекту (Project Identification)

1.4 Форма тест плану

Назва проекту

(Project Name)

План випробувань і тестування

(Test Plan)

Версія 1.0

(Version 1.0)

Історія змін (Revision History)

Дата (Date dd/mmm/yy)	Версія (Version x.x)	Детальний опис (Description details)	ПІБ Автора (Author name)

Примітка: видаляти або додавати елементи таблиці можна за необхідністю.

Документ і версія / дата) (Document and version / date)	Створено або Доступно (Created or Available)	Поступило або Відзив (Received or Reviewed)	Автор або ресурс (Author or Resource)	Примітка (Notes)
Специфікація вимог (Requirements Specification)	о Так о Ні	о Так о Ні		
Функціональна специфікація (Functional Specification)	о Так о Ні	о Так о Ні		
Звіти використання - випадок (Use-Case Reports)	о Так о Ні	о Так о Ні		
План проекту (Project Plan)	о Так о Ні	о Так о Ні		
Специфікація дизайну (Design Specifications)	о Так о Ні	о Так о Ні		

Прототип (Prototype)	o Так o Ні	o Так o Ні
Керівництво користувача (User’s Manuals)	o Так o Ні	o Так o Ні
Бізнес модель (Business Model or Flow)	o Так o Ні	o Так o Ні
Модель даних (Data Model or Flow)	o Так o Ні	o Так o Ні
Бізнес-функції (Business Functions and Rules)	o Так o Ні	o Так o Ні
Оцінка ризиків (Project or Business Risk Assessment)	o Так o Ні	o Так o Ні

Контрольні запитання та завдання

1. В чому полягає різниця між функціональним і не функціональним тестуванням?
2. Дайте характеристику тестуванню навантаження?
3. Що входить до тестування, що пов'язане зі змінами?
4. В чому відмінність димового та санітарного тестування?
5. В чому особливості тестування установки?
6. В чому особливості регресійного тестування?

2. Розробка тестових випадків (test case)

Тестування на різних рівнях виробляється протягом усього життєвого циклу розробки і супроводу ПЗ. Рівень тестування визначає те, над чим виробляються тести: над окремим модулем, групою модулів або системою, у цілому. Проведення тестування на всіх рівнях системи - це основа успішної реалізації і здачі проекту.

Рівні тестування:

- компонентне або модульне тестування (Component Testing or Unit Testing)
- інтеграційне тестування (Integration Testing)
- системне тестування (System Testing)
- приймальне тестування (Acceptance Testing)

Компонентне (модульне) тестування (Component or Unit Testing) перевіряє функціональність і шукає дефекти в частинах додатка, які доступні і можуть бути протестовані окремо (модулі програм, об'єкти, класи, функції тощо).

Звичайно компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити і за підтримкою середовищ розробки, таких як фреймворки (frameworks - каркаси) для модульного тестування або інструменти для налагодження.

Усі знайдені дефекти, як правило виправляються в коді без формального їхнього опису в системі менеджменту помилок/дефектів - багів (Bug Tracking System).

Один з найбільш ефективних підходів до компонентного (модульного) тестування - це підготовка автоматизованих тестів до початку основного кодування (розробки) програмного забезпечення.

Це називається «розробка від тестування» (test-driven development) або «підхід тестування спочатку» (test first approach). При цьому підході створюються й інтегруються невеликі частини коду, напроти яких запускаються тести, написані до початку кодування. Розробка ведеться доти поки всі тести не будуть успішними.

Інтеграційне тестування (Integration Testing) призначене для перевірки зв'язку між компонентами, а також взаємодії з різними частинами системи (операційною системою, устаткуванням або зв'язком між різними системами).

Рівні інтеграційного тестування:

- *компонентний інтеграційний рівень* (Component Integration testing). Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;

- *системний інтеграційний рівень* (System Integration Testing). Перевіряється взаємодія між різними системами після проведення системного тестування.

Підходи до інтеграційного тестування:

- *знизу нагору* (Bottom Up Integration). Усі низькорівневі модулі, процедури або функції збираються разом і потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування. Даний підхід вважається корисним, якщо всі або практично всі модулі, рівня, що розробляється готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності додатка;

- *зверху вниз* (Top Down Integration). Спочатку тестуються усі високорівневі модулі, і поступово один за іншим додаються низькорівневі. Усі модулі більш низького рівня симулюються заглушками з аналогічною функціональністю, потім в міру готовності вони заміняються реальними активними компонентами;

- *великий вибух* ("Big Bang" Integration). Всі або практично всі розроблені модулі збираються разом у вигляді закінченої системи або її основної частини, і потім проводиться інтеграційне тестування. Такий підхід дуже гарний для збереження часу. Однак якщо тест кейси і їхні результати записані не вірно, то сам процес інтеграції сильно ускладниться, що стане перешкодою для команди тестування при досягненні основної мети інтеграційного тестування.

Основною задачею *системного тестування* є перевірка як функціональних, так і не функціональних вимог у системі в цілому. При цьому виявляються дефекти, такі як невірне використання ресурсів системи, непередбачені комбінації даних користувальницького рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання тощо.

Для мінімізації ризиків, пов'язаних з особливостями поведінки в системі в будь-якому середовищі, під час тестування рекомендується використовувати оточення максимальне наближене до того, на яке буде встановлений продукт після видачі.

Можна виділити два підходи до системного тестування:

- на базі вимог (requirements based).Для кожної вимоги пишуться тестові випадки (test cases), що перевіряють виконання даної вимоги;
- на базі випадків використання (use case based). На основі представлення про способи використання продукту створюються випадки використання системи (Use Cases).
- По конкретному випадку використання можна визначити один або більш сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які мають бути протестовані.

Формальний процес *приймального тестування*, що перевіряє відповідність системи вимогам і проводиться з метою:

- визначення чи задовольняє система приймальним критеріям;
- винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні.

Приймальне тестування виконується на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до даного додатка.

Рішення про проведення приймального тестування приймається, коли:

- продукт досяг необхідного рівня якості;
- замовник ознайомлений із планом приймальних робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних із проведенням приймального тестування, дата проведення, відповідальні особи тощо.

Фаза приймального тестування триває доти, поки замовник не виносить рішення про відправлення додатка на доробку або видачі додатка.

Тестовий випадок (test case) **-** сукупність вхідних даних тесту, умови виконання і очікуваних результатів, які розроблені для конкретної мети. Тестовий випадок - це найменша одиниця тестування, яку можна самостійно виконати від початку до кінця. Шаблони тестового випадку і зразок їх заповнення представлені у додатку В.

Розглянемо особливості заповнення полів шаблону тестування.

Ідентифікатор тестового випадку - включає номер версії тесту.

Власник тесту – ПІБ особи, що експлуатує тест (воно може не співпадати з ПІБ автора тесту).

Дата останнього перегляду – ця інформація визначає актуальність тесту.

Назва тесту - опис назви тесту, що дозволяє його легко знайти і зрозуміти його призначення. Не рекомендується вживати назви, що не несуть ніякого сенсового навантаження, наприклад, "xxxLLL0123.tst".

Місцезнаходження тесту – повна назва шляху, розташування на диску ЕОМ.

Технічна вимога, що тестується - унікальний ідентифікатор, який відображається в документах технічних вимог.

Мета тестування - формулювання того, що має досягти тест.

Конфігурація засобів тестування - специфікація вводу / виводу, умови випробувань.

Налаштування на прогін тесту - процедура подібна методиці тестування. Вона передбачає опис дій тестувальника і очікуваних результатів. Якщо настроювання автоматизовані, це виглядає так: run setupSC03.pl.

Методика тестування - опис дій тестувальника і очікуваних результатів.

Взаємозалежність тестових випадків – ідентифікація будь-якого тестового випадку. Для того, щоб виконання даного тесту починалося при означених умовах, необхідно здійснити прогін попередніх тестів.

Очистка тесту – якщо система була переведена в нестійкий стан або дані були зруйнованими, очистка дозволяє усунути подібні ситуації.

Контрольні запитання та завдання

1. Назвіть рівні тестування і їх основні характеристики.
2. В чому різниця між компонентним і системним тестуванням?
3. Які підходи до системного тестування, ви знаєте?
4. Які підходи до інтеграційного тестування, ви знаєте?
5. Назвіть вимоги створення тестових випадків.

3. Розробка тест-кейсів і техніка техніка тест-дизайну

Тест дизайн – це етап процесу тестування ПЗ, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріїв якості і цілей тестування.

План роботи над тест дизайном:

- аналіз існуючих проектних артефактів: документація (специфікації, вимоги, плани), моделі, що виконується код, тощо;
- написання специфікації з тест дизайну (Test Design Specification);
- проектування і створення тестових випадків (Test Cases);

Техніки тест дизайну:

- *Еквівалентний поділ (Equivalence Partitioning - EP).* Як приклад, у вас є діапазон припустимих значень від 1 до 10, ви повинні вибрати одне вірне значення усередині інтервалу, скажемо, 5, і одне невірне значення поза інтервалом - 0.
- *Аналіз граничних значень (Boundary Value Analysis - BVA).* Якщо взяти приклад вище, як значення для позитивного тестування виберемо мінімальну і максимальну границі (1 і 10), і значення більше і менше границь (0 і 11). Аналіз Граничних значень може бути застосований до полів, записів, файлів, або до будь-якого роду сутностей, що мають обмеження.
- *Причина / Наслідок (Cause/Effect - CE).* Це, як правило, уведення комбінацій умов (причин), для одержання відповіді від системи (Наслідок). Наприклад, ви перевіряєте можливість додавати клієнта, використовуючи визначену екранну форму. Для цього вам необхідно буде ввести кілька полів, таких як "Ім'я", "Адреса", "Номер Телефону" а потім, натиснути кнопку "Додати" - ця "Причина". Після натискання кнопки "Додати", система додає клієнта в базу даних і показує його номер на екрані - це "Наслідок".
- *Передбачення помилки (Error Guessing - EG).* Це коли тест аналітик використовує свої знання системи і здатність до інтерпретації специфікації на предмет того, щоб "вгадати" при яких вхідних умовах система може видати помилку. Наприклад, специфікація говорить: "користувач має увести код". Тест аналітик, буде думати: "Що, якщо я не введу код?", "Що, якщо я введу неправильний код? ", і так далі. Це і є здогадування помилки.

- *Вичерпне тестування (Exhaustive Testing - ET)* - це крайній випадок. У межах цієї техніки ви повинні перевірити всі можливі комбінації вхідних значень, і в принципі, це повинно знайти всі проблеми. На практиці застосування цього методу не представляється можливим, через величезну кількість вхідних значень.

План розробки тест випадків - кейсів містить.

1. Аналіз вимог.

2. Визначення набору тестових даних на підставі EP, BVA, EG.

3. Розробка шаблона тесту на підставі CE.

4. Написання тест кейсів на підставі первинних, тестових даних і кроків тесту.

Далі на прикладі, розглянемо запропонований підхід.

Приклад: Протестувати функціональність форми прийому заявок, вимоги до якої надані в наступній таблиці 3.1.

Прийом заяв через форму графічно можна представити так:

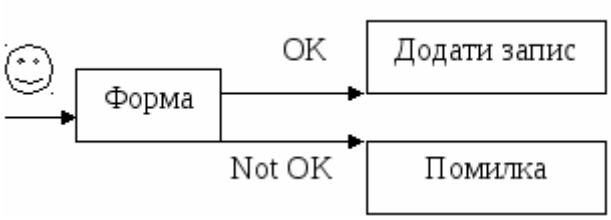

Рисунок 3.1 – Графічне зображення подання заяв через відповідну форму

Таблиця 3.1 – Вимоги до тестування функціональності форми прийому заявок

Елемент	**Тип елемента**	**Вимоги**
Тип звертання	combobox	Набір даних: Консультація Проведення тестування

		Розміщення реклами Помилка на сайте * - на процес виконання операції прийому заявок не впливає
Контактна особа	editbox	1. Обов'язкове для заповнення 2. Максимально 25 символів 3. Використання цифр і спец символів не допускається
Контактний телефон	editbox	Обов'язкове для заповнення Припустимі символи "+" і цифри "+" можна використовувати тільки на початку номера Припустимі формати: • починається з плюса - 11-15 цифр +31612361264 +375291438884 без плюса - 5-10 цифр, наприклад: 0613261264 або 2925167
Повідомлення	text area	1. Обов'язкове для заповнення 2. Максимальна довжина 1024 символу
Відправити	button	Стан: 1. За замовчуванням - не активна (Disabled) 2. Після заповнення обов'язкових полів стає активна (Enabled) Дії після натискання 1. Якщо введені дані коректні - відправлення

повідомлення

2. Якщо введені дані НЕ коректні - валідаційне повідомлення

План розробки тест випадків - кейсів:

1. Аналіз вимог.

Читаємо, аналізуємо вимоги і виділяємо для себе наступні нюанси:

- які з полів обов'язкові для заповнення?
- чи мають поля обмеження по довжині або по розмірності (границі)?
- які з полів мають спеціальні формати?

2. Визначення набору тестових даних.

Враховуючи вимоги до полів, використовуючи техніки тест дизайну визначаємо набір тестових даних:

- залежно від рівня обов'язковості поля, визначимо які поля необхідно перевірити на порожнє значення, тому що воно може викликати помилку (У результуючій таблиці +);
- необхідно визначити мінімальний набір даних, тому що вичерпне тестування не можливо через велику кількість всіляких комбінацій значень. Це можна зробити використовуючи такі техніки, як EP і BVA. (*У результуючій таблиці* **)
- у формі присутнє поле, що має складений тип (цифри використовуються разом із символами), має спеціальний формат даних і тому виділення тестових даних для нього - це досить трудомістка задача. У межах даного прикладу обмежимося тільки простою перевіркою форматів і основних вимог описаних у формі прийому заявок.
- по завершенню генерації даних використовуючи стандартні техніки, можна додати деяку кількість значень на підставі особистого досвіду (техніка EG) - це буде використання спеціальних символів, дуже довгих рядків, різних форматів даних, регістрів у рядках (Upper, Lowwer, Mixed cases), негативні і нульові значення, кейворды Null - Na – Infinity, тощо. Сюди можна включити все, що на гадку тестувальника, може вивести додаток з ладу (*У результуючій таблиці* ^)

Примітка: Кількість тестових даних після остаточної генерації буде досить великим, навіть при використанні спеціальних технік тест дизайну. Тому обмежимося лише декількома значеннями для кожного поля, тому що ціль прикладу показати саме процес створення тест кейсів, а не процес одержання конкретних тестових даних.

2.1 Вибір тестових даних для кожного окремо узятого поля

Поле *Тип звертання*. Беремо будь-яку (1-у) позицію в листі з очікуваним результатом, тому що всі дані входять у 1 клас еквівалентності, тобто не змінюють сам процес виконання прийому заявки, ОК. Необхідно розглянути і граничні умови (техніка BVA), тобто беремо перший і останній елементи, але тому що реалізовано поле як лист, має також сенс. Разом: 1-а й остання позиції в листі. Очікуваний результат при використанні - ОК.

Поле *Контактна особа*. Це обов'язкове поле розміром від 1 до 25 символів (включаючи границі). Перевірка на обов'язковість додає до тестових даних порожнє значення. Проведемо аналіз граничних умов (BVA), одержимо набір: 0, 1, 2, 24, 25 і 26 символів. Порожнє значення (0 символів) уже було додано при аналізі обов'язковості поля для введення, тому при BVA ми не будемо додавати його ще раз.

(Якщо його додати другий раз, відбудеться дублювання тестових даних, що не приведе до дублювання нових дефектів, а значить повторне додавання в домен не має сенсу).

У зв'язку з тим, що значення 2 і 24 символи є, на наш погляд, некритичними, їх можна не додавати. У підсумку одержуємо, що мінімальний набір даних для тестування поля - це рядки 1 і 25 - ОК, і 0 (порожнє значення), 26 символів - NOK.

Поле *Контактний телефон* складається з декількох частин: код країни, код оператора, номер телефону (який може бути складеними і розділений дефісами). Для визначення правильного набору тестових даних необхідно розглядати кожну складову частину окремо. Застосовуючи BVA і EP, одержимо:

для номерів з плюсом	для номерів без плюса
По BVA одержимо номера з 10, 11, 12 і 14, 15, 16 цифрами, де 10 і 16 - NOK, а 11, 12, 14, 15 –	По BVA одержимо номера з 4, 5, 6 і 9, 10, 11 цифрами.

OK

Розглядаючи отримані дані з позиції ЕР виділимо, що 11, 12, 14, 15 входять в один клас еквівалентності. Тому при тестуванні ми можемо використовувати кожне з них, але тому що 11 і 15 - це границі інтервалу, то на наш погляд їх пропускати не можна. Отже ми можемо зменшити набір значень до двох, виключивши 12 і 14, а залишивши 11 і 15 для перевірки граничних умов.

Разом маємо: 11 і 15 цифр - OK, (+12345678901, +123456789012345)

10 і 16 цифр - NOK; (+1234567890, +1234567890123456);

Діючи аналогічно прикладу для номерів телефонів з плюсом, виключимо значення 6 і 9, залишивши 5 і 10.

Разом маємо: 5 і 10 цифр - OK, (12345, 1234567890) 4 і 11 цифр-NOK;(1234, 12345678901).

Поле *Повідомлення*. Підбор даних проводимо за аналогією з полем *Контактна особа*. На виході одержуємо значення: рядка 1 і 1024 - OK, і 1025 символів - NOK.

Таблиця 3.2 - Результуюча таблиця даних, для використання при наступному складанні тест кейсів

Поле	**OK**	**Значення**	**Коментар**
Тип звертання	OK	**Консультація**	**перший у списку****
		Помилка на сайті	**останній у списку****
	NOK		
Контактна особа	OK	**Йцукенгшщзйцукенгшщзйцуке**	**25 символів нижній регістр****
		а	**1 символ****
		ЙЦУКЕНГШЩЗФЫВАПРОЛДЖЯЧ	**25 символів ВЕРХНІЙ регістр****

		СМИ	
		ЙЦУКЕНГШЩЗфывапролджЯЧСМИ	**25 символів Змішаний регістр****
	NOK		порожнє значення +
		Йцукенгшщзйцукенгшщзйцукей	**довжина більш максимальної 26 сим.**
		@#$%^&;.?,>|\/№"!()_{}[<~	*спец. символи (ASCII)^*
		12345678901234567 89012345	*тільки цифри ^*
		adsadasdasdas dasdasd asasdsads(...)sas	*дуже довгий рядок (~1Mb)^*
Контактний телефон	OK	+12345678901	з плюсом - мінімальна довжина
		+123456789012345	з плюсом – максимальна довжина
		12345	без плюса - мінімальна довжина
		1234567890	без плюса - максимальна довжина
	NOK		порожнє значення +
		+1234567890	з плюсом - < мінімальної довжини
		+1234567890123456	з плюсом - > максимальної довжини

		1234	без плюса - < мінімальної довжини
		12345678901	без плюса - > максимальної довжини
		+YYYXXXyyyxxzz	с плюсом - букви замість цифр
		yyyxxxxzz	без плюса - букви замість цифр
		+###-$$$-%^-&^-&!	*спец. символи (ASCII) ^*
		12323123231232132 31232(...)99	*дуже довгий рядок (~1Mb) ^*
Повідомлення	OK	**йццуйцуйц(...)йцу**	**Максимальна довжина (1024 симв.)****
	NOK	*	порожнє значення +
		йццуйцуйц(...)йцуц	**Довжина більше максимальної (1025)**
		adsadasdasdas dasdasd asasdsads(...)sas	очень довгий рядок (~1Mb)
		@##$$$%^&^&	тільки спец. символи (ASCII)

3. Розробляємо шаблон тесту

На підставі техніки CE і варіантів використання (Use case)(якщо є) створимо шаблон планованого тесту. Даний документ містить кроки й очікувані результати тесту, але без конкретних даних, що заповнюються на наступному етапі розробки тест кейсів.
Таблиця 3.3 – Приклад шаблона тест кейса

Дія	**Очікуваний результат**
•	

1. Відкриваємо форму відправлення повідомлення	Форма відкрита • Усі поля за замовчуванням порожні • Обов'язкові поля позначені - * • Кнопка «Відправити» не активна
2. Заповнюємо поля форми: • Тип звертання • Контактна особа • Контактний телефон • Повідомлення	• Поля заповнені • Кнопка «Відправити» - активна (Enabled)
3. Натискаємо кнопку «Відправити»	Якщо введені дані коректні: • Повідомлення «Заявка відправлена» виведене на екран. • Нова заявка з'явилася в списку на сторінці «Заявки». Якщо введені дані НЕ коректні: • Валідаційне повідомлення з усіма помилками виведено на екран. • Заявка НЕ з'явилася в списку на сторінці «Заявки».

4. Написання тест кейсів на підставі первісних вимог, тестових даних і шаблона тесту

Після того, як тестові дані і кроки тесту готові приступаємо безпосередньо до розробки тест кейсів. Тут нам допоможуть такі методи комбінування як:

- послідовний перебір - перебір усіх можливих комбінацій значень,що маємо. Таким чином виходить, що кількість тест кейсів буде дорівнювати добутку кількості варіантів тестових даних для кожного поля. Для нашого конкретного приклада ми одержимо 1404 тест кейсів;
- попарний перебір (Pairwise Testing).Найчастіше, збої викликають не складна комбінація всіх параметрів, а комбінація лише пари параметрів. Техніка попарного перебору, дозволяє створити тестові набори, що комбінують дані з двох полів. Завдяки цьому, кількість отриманих на виході тест кейсів у рази менше, ніж при комбінуванні того ж самого набору даних при послідовному переборі.
- Відзначимо також, що в даний момент існує кілька алгоритмів генерації комбінацій для попарного тестування: Orthogonal Arrays Testing, All pairs, IPO (In-Parameter Order). Так наприклад, при використанні техніки All Pairs у нашому конкретному випадку ми одержимо 118 тест кейса.

По завершенню підготовки комбінацій даних, підставляємо їх у шаблон тест кейса, і в результаті маємо набір тестових випадків, що покриває вимоги, що тестуються, до форми прийому заявок.

Примітка: Тест кейси розділяються по очікуваному результаті на позитивні й негативні тест кейси. У позитивному тест кейсі усі поля ОК:
Таблиця 3.4 – Приклад позитивного і негативного тест кейсу

Позитивний тест кейс		Негативний тест кейс поле Контактна особа NOK):
Дія	**Очікуваний результат**	**Дія**
1. Відкриваємо	• Форма відкрита • Усі поля за	1. Відкриваємо форму

форму відправлення повідомлення	замовчуванням порожні • Обов'язкові поля позначені - * • Кнопка «Відправити» не активна	відправлення повідомлення
2. Заповнюємо поля форми: • Тип звертання **Консультація** • Контактна особа **йцукенгшщзйцуке нгшщзйцуке** • Контактний телефон **+7-916-111-11-11** • Повідомлення	• Поля заповнені • Кнопка «Відправити» - активна (Enabled)	2. Заповнюємо поля форми: • Тип звертання **Консультація** • Контактна особа **@#$%^&;.?,>|\/№"!()_{}[** • Контактний телефон **(916)333-33-33** • Повідомлення **йццуйцуйц(...)йцу - 1024 символу**

Контрольні запитання та завдання

1. Що таке тест дизайн?

2. Назвіть техніки тест дизайну і їх основні характеристики.
3. Які основні розділи включає план розробки тестових випадків?

4. Розробка звітів про помилки/ дефекти (bug report)

Звіт про помилки/дефекти (Bug Report, Баг репорт) - це технічний документ і мова опису має бути технічною. Повинна використовуватися правильна термінологія при використанні назв елементів інтерфейсу користувача (editbox, listbox, combobox, link, text area, button, menu, popup menu, title bar, system tray, тощо.), дій користувача (click link, press the button, select menu item,тощо) і отриманих результатах (window is opened, error message is displayed, system crashed, тощо).

Обов'язковими полями баг репорту є: короткий опис (Bug Summary), серйозність (Severity), кроки до відтворення (Steps to reproduce), результат (Actual Result), очікуваний результат (Expected Result). Нижче приведені вимоги і приклади по заповненню цих полів.

Короткий опис. Зміст усього баг репорту. Наприклад:

1. Додаток зависає, при спробі збереження текстового файлу розміром більше 50Мб.

2. Дані на формі «Профайл» не зберігаються після натискання кнопки «Зберегти».
При написанні баг репорту використовується *принцип "Де? Що? Коли?":*

Де?: У якому місці інтерфейсу користувача або архітектури програмного продукту знаходиться проблема. Причому, починайте пропозицію з іменника, а не з прийменника (предлог).

Що?: Що відбувається або не відбувається відповідно до специфікації або вашій уяві про нормальну роботу програмного продукту. При цьому вказуйте на наявність або відсутність об'єкта проблеми, а не на його зміст (його вказують в описі). Якщо зміст проблеми варіюється, усі відомі варіанти вказуються в описі.

Коли?: У який момент роботи програмного продукту, або при якій події або при яких умовах проблема виявляється.

Серйозність. Якщо проблема знайдена в ключовій функціональності додатку і після її виникнення додаток стає цілком недоступний, і подальша робота з ним неможлива, то вона є помилкою, що блокує. Звичайно всі проблеми, що блокують, знаходяться під час первинної перевірки нової версії продукту (Build Verification Test, Smoke Test), тому що їхня наявність не дозволяє повноцінно проводити тестування. Якщо ж тестування може бути продовжено, то серйозність даного дефекту буде критична.

Кроки до відтворення / Результат / Очікуваний результат. Дуже важливо чітко описати всі кроки, зі згадуємо усіх вводи даних (імені користувача, даних для заповнення форми) і проміжних результатів.

Наприклад:

Кроки до відтворення

1. Увійдіть у систему: Користувач Тестер1, пароль xxxXXX > Вхід у систему здійснений.

2. Кликніть линк Профайл > Сторінка Профайл відкрилася.

3. Уведіть Нове ім'я користувача: Тестер2.

4. Натисніть кнопку Зберегти.

Результат.

На екрані з'явилася помилка. Нове ім'я користувача не був збережено.

Очікуваний результат.

Сторінка профайл перевантажилася. Нове значення імені користувача збережено.
Серйозність і пріоритет дефекту.

Серйозність (Severity) - це атрибут, що характеризує вплив дефекту на працездатність додатка.

Пріоритет (Priority) - це атрибут, що вказує на черговість виконання задачі або усунення дефекту. Можна сказати, що це інструмент менеджера по плануванню робіт. Чим вище пріоритет, тим швидше потрібно виправити дефект.

Градація серйозності дефекту (Severity)

- *S1 Що Блокує (Blocker)* – ця помилка, приводить додаток у неробочий стан, у результаті якого подальша робота з системою, що тестується або її ключовими функціями стає неможлива. Рішення проблеми необхідно для подальшого функціонування системи.
- *S2 Критична (Critical)* – неправильно працює ключова бізнес логіка, діра в системі безпеки, проблема, що призвела до тимчасового падіння сервера або, що приводить у неробочий стан деяку частину системи, без можливості рішення проблеми, використовуючи інші вхідні крапки. Рішення проблеми необхідно для подальшої роботи з ключовими функціями системи, що тестуються.
- *S3 Значна (Major)* - частина основний бізнес логіки працює некоректно. Помилка не критична або є можливість для роботи з функцією, що тестується, використовуючи інші вхідні крапки.
- *S4 Незначна (Minor)* - не порушує бізнес логіку частини додатка, що тестується, очевидна проблема інтерфейсу користувача.
- *S5 Тривіальна (Trivial)* - не стосується бізнес логіки додатка, погано відтворена проблема, малопомітна по засобах інтерфейсу користувача, проблема сторонніх бібліотек або сервісів, проблема, що не робить ніякого впливу на загальну якість продукту.

Градація пріоритету дефекту (Priority).

- *P1 Високий (High).* Помилка повинна бути виправлена якнайшвидше, тому що її наявність є критичною для проекту.
- *P2 Середній (Medium).* Помилка повинна бути виправлена, її наявність не є критичної, але вимагає обов'язкового рішення.
- *P3 Низький (Low).* Помилка повинна бути виправлена, її наявність не є критичної, і не вимагає термінового рішення.

Порядок виправлення помилок за пріоритетами: High > Medium > Low

Наявність відкритих дефектів P1, P2 і S1, S2, вважається неприйнятним для проекту. Усі подібні ситуації вимагають термінового рішення і йдуть під контроль до менеджерів проекту.

Наявність строго обмеженої кількості відкритих помилок P3 і S3, S4, S5 не є критичним для проекту і допускається у додатку. Кількість же відкритих помилок залежить від розміру проекту і встановлених критеріїв якості.

Контрольні питання

1. Назвіть причини створення звіту про помилки/дефекти (Bug Report).
2. Назвіть особливості створення звіту про помилки/дефекти.
3. Назвіть градацію серйозності дефекту.
4. Назвіть градацію пріоритету дефекту.

СПИСОК ЛІТЕРАТУРИ

1. Про затвердження загальних вимог до програмних продуктів. Постанова Кабінету Міністрів України від 12 серпня 2009 р. N 869
2. Процеси життєвого циклу програмного забезпечення. ДКА України СОУ-Н ДКА 0061:2012
3. Марголин Л. Н. Компьютерные методы обработки информации. Интернет издание. http://hi-edu.ru/e-books/xbook709/01/title.htm
4. Майерс Г. Надежность программного обеспечения. М:, Мир, 1980
5. Куликов С. С. Тестирование программного обеспечения. Базовый курс. Интернет-издание 2017
6. Канер С., и др. Тестирования программного обеспечения. К:, Диасофт, 2001.
7. Майерс Г. Искусство тестирования программ. М:, Финансы и статистика, 1982.
8. Савин Р. Тестирование Дот Ком. М:, «Дело», 2007.
9. Бахтизин В.В. Автоматизация тестирования программного обеспечения. Минск, БГУИР 2012
10. Особенности тестирования WEB-приложений. http://quality-lab.ru/key-principles-of-web-testing/
11. Мобильное тестирование: полное руководство. *http://www.cmsmagazine.ru/library/items/moblile/testing-mobile-apps/*
12. Тестирование по-пайтоновски. https://dou.ua/lenta/articles/nose-tests-intro/
13. Тестирование в Python [unittest]. https://devpractice.ru/unit-testing-in-python-part-1/
14. Python. Генерация юнит-тестов. https://habrahabr.ru/post/192512/
15. Python уроки: Введение в тестирование на Python. https://pynsk.ru/blog/2016/02/10/intro-test/

16. Автоматизация тестирования веб-приложения с использованием Selenium WebDriver, Python, и Behave/ https://habrahabr.ru/post/262929/
17. doctest — Тестирование интерактивных примеров на Python. http://python-lab.ru/documentation/27/stdlib/doctest.html
18. Памятка по написанию тестов припомощи PyTest. https://eax.me/pytest/
19. Маглинец Ю.А. Анализ требований к автоматизированным информационным системам. https://www.intuit.ru/studies/courses/2188/174/info
20. Книберг Х. Скарин М. Scrum и Kanban: выжимаем максимум. Киев: InfoQ, 2011.
21. Мутилин В. С. Паттерны проектирования тестовых сценариев. Труды Института системного программирования РАН. http://www.citforum.ru/SE/testing/
22. Уиттакер Д. и др. Как тестируют в Гугл. Спб: Питерб 2014.