МЕТОДИ ТЕСТУВАННЯ ТА ОЦІНКИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ ЗАСТОСУВАННЯМ PAIRWISE ТЕСТУВАННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

Полтава ▪ 2016

Міністерство освіти і науки України

Полтавський національний технічний університет
імені Юрія Кондратюка

Провідна українська компанія
з тестування програмного забезпечення QATestLab

Ukrainian HI-Tech Initiatiave

Кафедра комп'ютерних та інформаційних технологій та систем

МЕТОДИ ТЕСТУВАННЯ ТА ОЦІНКИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ ЗАСТОСУВАННЯМ PAIRWISE ТЕСТУВАННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

Полтава ▪ 2016

Навчальний посібник «Методи тестування та оцінки якості програмного забезпечення із застосуванням Pairwise тестування» для студентів денної та заочної форми навчання – Полтава: ПолтНТУ, 2016. – 391 с.

Укладачі: Гаврилей Н. В., керівник Training Center компанії QATestLab,
Маянська Ю. В., менеджер по партнерству компанії Ukrainian HI-Tech Initiative,
Ляхов О.Л., завідувач кафедри, доктор технічних наук, професор,
Бородіна О.О., асистент кафедри.

Відповідальний за випуск: доцент кафедри комп'ютерних та інформаційних технологій і систем А.М. Гафіяк, к. е. н., доцент

Рецензент: О.В. Скакаліна, кандидат техн. наук., доцент

Затверджено та рекомендовано до друку науково-методичною радою університету

Протокол № ___ від _____

Коректор _____

40.40.01.01

ЗМІСТ

ВСТУП

Теоретичний та практичний матеріал навчального посібника є суттевою складовою підготовки тестувальника та спрямований на вивчення існуючих дефектів ПЗ, в якому проводиться тестування, процесів розробки ПЗ, методів та засобів тестування, технік проектування тестів та додатків на мобільних пристроях.

Мета даної роботи: створення підґрунтя для оволодіння усіма концепціями тестування програмного забезпечення: тестування WEB проектів, функціональне тестування, тестування мобільних додатків, тестування ігор та ін.

Навчальний посібник «Методи тестування та оцінки якості програмного забезпечення із застосуванням Pairwise тестування» включає 19 тем, в яких розглянуто основні методи та засоби тестування програмного забезпечення, підходи проведення різноманітних видів тестування.

Навчальний посібник розроблено у співпраці з Українською HI-Tech-Ініціативою і компанією QATestLab.

"Ukrainian HI-Tech Initiative" – лідируючий альянс в індустрії програмного забезпечення та IT-аутсорсингу в Україні. Ініціатива об`єднує 77 провідних компаній і ставить перед собою завдання розвитку галузі розробки програмного забезпечення, просування українських IT-компаній (ukrainian software development companies) на міжнародні ринки аутсорсингових розробок програмного забезпечення (outsourcing software development) та їх інтеграцію у світове співтовариство software development community. Українська HI-Tech-Ініціатива є автором освітньої програми для вищих навчальних закладів "Зробимо цифровий світ кращим".

QATetLab – провідна міжнародна компанія, яка надає повний спектр послуг по тестуванню програмного забезпечення.

Заснована в 2005 році. За ці роки реалізовано велику кількість економічно ефективних і високоякісних проектів в різноманітних галузях, таких як e-commerce, телекомунікації, туризм, урядові проекти та ін. Замовниками послуг QATestLab є компанії зі світовим ім`ям, відомі

ТЕМА 1

ВСТУП

ДО ТЕСТУВАННЯ

1 ВСТУП: ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (ПЗ)

У цій вступній лекції даються базові поняття та позначення, які будуть використовуватися протягом усього курсу.

1.1 Етимологія терміну "баг"

За однією з версій, у 1946 році вчені Гарвардського університету, які тестували обчислювальну машину Mark II Aiken Relay Calculator, знайшли метелика, що застряг між контактами електромеханічного реле, і Грейс Хоппер промовила цей термін. Витягнута комаха була вклеєна скотчем в технічний щоденник з супровідним написом: "First actual case of bug being found" ("перший реальний випадок, коли було знайдено жука").

Слово "bug" у значенні технічної помилки вживалося задовго до цього персоналом телеграфних і телефонних компаній щодо несправностей в електрообладнанні та радіотехніці. Під час Другої світової війни словом "bugs" називалися проблеми з радарною електронікою. У 1878 році Томас Едісон писав:

"Так було з усіма моїми винаходами. Перший крок - інтуїція, що приходить як спалах, потім виникають труднощі - пристрій відмовляється працювати, і саме тоді виявляються "жучки" - як називають ці дрібні помилки і труднощі - і необхідні місяці пильного спостереження, досліджень і зусиль, перш ніж справа дійде до комерційного успіху або невдачі".

1.2 Історія розвитку тестування програмного забезпечення

Перші програмні системи розроблялися в рамках програм наукових досліджень або програм для потреб міністерств оборони. Тестування таких продуктів проводилося суворо формалізовано із записом усіх тестових процедур, тестових даних та отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувалося тим самим персоналом.

У 1960-х багато уваги приділялося "вичерпному" тестуванню, яке мало проводитися з використанням усіх шляхів у коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування ПЗ неможливе, тому що, по-перше, кількість можливих вхідних даних дуже велика, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі та специфікаціях. З цих причин "вичерпне" тестування було відхилене і визнане теоретично неможливим.

На початку 1970-х тестування ПЗ позначалося як "процес, спрямований на демонстрацію коректності продукту" або як "діяльність по підтвердженню правильності роботи ПЗ". У програмній інженерії, що лише починала зароджуватись, верифікація ПЗ значилася як "доказ правильності". Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу і була недостатньо всеохоплюючою. Було вирішено, що доказ правильності - неефективний метод тестування ПЗ. Проте, в деяких випадках, демонстрація правильної роботи використовується і в наші дні, наприклад, в правилах приймання. У другій половині 1970-х тестування уявлялося, як виконання програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест - це тест, який виявляє раніше невідомі проблеми. Даний підхід є прямо протилежним попередньому. Зазначені два визначення являють собою "парадокс тестування", в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого - виявляє помилки в ПЗ, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, тому що не дозволяє ігнорувати недоліки ПЗ.

У 1980-х до тестування додалося таке поняття, як попередження дефектів. Проектування тестів - найбільш ефективний з відомих методів попередження помилок. У цей же час почали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування має включати перевірки протягом усього циклу розробки, і це повинен бути керований процес. У ході тестування треба перевірити не тільки зібрану програму, а й вимоги, код, архітектуру, самі тести. "Традиційне" тестування, яке існувало до початку 1980-х, відносилося тільки до компільованої, готової системи (зараз це зазвичай називається системним тестуванням), але надалі тестувальники стали залучатися до усіх аспектів життєвого циклу розробки. Це дозволяло раніше знаходити проблеми у вимогах та архітектурі і тим самим скорочувати терміни і бюджет розробки. У середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, і зробить це більш надійно. Спочатку ці інструменти були вкрай простими і не мали можливості написання сценаріїв на скриптових мовах.

На початку 1990-х в поняття "тестування" стали включати планування, проектування, створення, підтримку і виконання тестів і тестових оточень, і це означало перехід від тестування до забезпечення якості, що охоплює весь цикл розробки ПЗ. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПЗ для проведення навантажувального тестування. У середині 1990-х з розвитком інтернету і

розробкою великої кількості веб-додатків особливу популярність набуває "гнучке тестування" (за аналогією з гнучкими методологіями програмування).

У 2000-х з'явилося ще більш широке визначення тестування, коли до нього було додано поняття "оптимізація бізнес-технологій" (business technology optimization, BTO). BTO направляє розвиток інформаційних технологій у відповідності з цілями бізнесу. Основний підхід полягає в оцінці і максимізації значущості всіх етапів життєвого циклу розробки ПЗ для досягнення необхідного рівня якості, продуктивності, доступності [6].

1.3 Особливості та вимоги до професії тестувальника *(tester)*

Тестувальник є особою, відповідальною за якісне і своєчасне виконання дорученої йому роботи в проекті розробки інформаційно-програмної системи. Тестувальник виконує завдання з підготовки та проведення тестування системи відповідно до встановленого в компанії регламенту. Точний обсяг завдань, що виконуються в проекті тестувальником, визначається його роллю в проекті.

Професія тестувальника в першу чергу затребувана в IT-компаніях, які займаються розробкою ПЗ, комп'ютерних ігор та інтернет-сайтів.

Обов'язки тестувальника:

- ✓ контроль якості розроблюваних продуктів;
- ✓ планування тестування та всіх необхідних видів робіт;
- ✓ виявлення та аналіз помилок і проблем, що виникають у користувачів при роботі з програмними продуктами;
- ✓ розробка сценаріїв тестування;
- ✓ документування знайдених дефектів;
- ✓ складання технічної документації російською та іноземною *(найчастіше англійською на рівні Upper Intermediate)* мовами;
- ✓ розробка автотестів та їх регулярний прогін *(при необхідності і наявності відповідних знань)*.

Вимоги до тестувальника:

- ✓ вища освіта;
- ✓ аналітичні здібності;
- ✓ знання англійської мови на рівні, достатньому для читання і написання технічних текстів [7].

Додаткові вимоги:

- ✓ досвід в організації та проведенні різних видів тестування;
- ✓ вміння тестувати веб-додатки;

- ✓ знання Windows, UNIX, OS X;
- ✓ знання мобільних платформ *(iOS, Android);*
- ✓ базові знання HTML, CSS, SQL;
- ✓ знання основ мов програмування.

1.4 Основні поняття і терміни

1.4.1 Баг і атрибути бага

Існує кілька визначень терміна "***Баг***" *(bug):*

- дефект**,** помилка в програмі або в системі, яка видає несподіваний або невірний результат;
- жаргонне слово, яке зазвичай означає помилку в програмі або системі, яка видає несподіваний або невірний результат;
- відхилення фактичного результату *(actual result)* від очікуваного результату *(expected result).*

До атрибутів бага відносять:

- номер бага в системі (bug number);
- серйозність (severity) – це технічна категорія, яка визначає критичність багу з точки зору тестувальника: особливість, помилка в тексті, дрібна проблема, значна проблема, падіння продукту, проблема блокуючого характеру:
  - ~ критичний *(critical)*:
    - ○ критичний системний збій *(crash);*
    - ○ втрата даних *(data loss);*
    - ○ проблема з безпекою *(security issue);*
  - ~ значний *(major):*
    - ○ сайт "зависає" *(site hangs);*
    - ○ баг блокує кодування, тестування або використання веб-сайту *(blocker);*
  - ~ помірний *(minor):*
    - ◦ функціональні проблеми *(functional bugs);*
  - ~ косметичний *(cosmetic):*
    - ○ косметична проблема *(cosmetic problem)*
    - ○ normal;
    - ○ trivial;
- пріоритет *(priority)* – пріоритет, з яким проблема повинна бути виправлена – також є показником важливості бага для бізнесу компанії*:*

  - immediate;
  - urgent;
  - high;
  - normal;
  - low;
- короткий опис *(summary)* – це максимально інформативний і стислий опис проблеми;
- опис (description) – корисна інформація про баг: опис, коментарі, нюанси;
- кроки відтворення (steps to reproduce) – конкретні кроки для відтворення проблеми;
- прикріплення *(attachment)* – будь-яка інформація, яка допоможе відтворити ситуацію *(скріншоти, відео, лог-файл);*
- додаткова інформація *(операційна система, браузер + версія, мобільний пристрій).*

Атрибути бага можуть змінюватись в залежності від системи відсліджування помилок *(Bug / Defect tracking system),* яка використовується, а також правил, прийнятих у конкретній організації.

1.4.2 Тестування та мета тестування

Загалом, випадки будь-якого тестування зводяться до пошуку багів.

Тестування програмного забезпечення *(Software Testing)* - це перевірка відповідності між реальною та очікуваною поведінкою програми, здійснюваною на кінцевому наборі тестів, обраному певним чином.

У більш широкому сенсі, ***тестування*** - це одна з технік контролю якості, що включає в себе активності з планування робіт *(Test Management)*, проектування тестів *(Test Design)*, виконання тестування *(Test Execution)* та аналіз отриманих результатів *(Test Analysis)*.

Верифікація *(verification)* – це процес оцінки системи або її компонентів, метою якого є визначення того, чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу [IEEE]. Тобто, чи виконуються наші цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази.

Валідація *(validation)* – це визначення відповідності ПЗ, що розробляється, очікуванням і потребам користувача, вимогам до системи [BS7925-1].

Інцидент *(Test Incident)* – будь-яка подія, спостереження, знайдене в рамках тестування, що вимагає дослідження.

Звіт щодо інциденту *(Incident Report)* – документ, що описує подію, яка відбулася під час тестування, і яку необхідно досліджувати.

Звіт про запит на зміну *(Improvement Report)* – документ, що описує пропозицію про вдосконалення продукту. Включає в себе детальний опис пропозиції та обґрунтування внесення змін до програмного забезпечення.

План Тестування *(Test Plan)* – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладів, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх дозволу.

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

Баг / Дефект Репорт *(Bug Report)* – це документ, що описує ситуацію або послідовність дій, яка призвела до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату.

Тестове Покриття *(Test Coverage)* – це набір тестів для перевірки тестованої функції. Розрахунок тестового покриття проводиться за формулою: відношення кількості рядків коду, покритих тестами, до загальної кількості рядків коду тестованої функції, помножене на 100%.

Деталізація Тест-Кейсів *(Test Case Detalization)* – це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується раціональне співвідношення часу проходження до тестового покриття [1].

Час Проходження Тест-Кейсу *(Test Case Pass Time)* – це час від початку проходження кроків тест-кейсу до отримання результату тесту.

Мета тестування – це знаходження багів до того, як їх знайдуть користувачі. Іншими словами, внесок тестувальника в щастя користувача – це пріоритет у знаходженні багів [2].

1.4.3 Система відслідковування помилок та життєвий цикл дефекту

Система відслідковування помилок *(bug tracking system)* – це прикладна програма, розроблена, щоб допомогти розробникам програмного забезпечення (програмістам, тестувальникам та ін.) враховувати й контролювати помилки і неполадки, знайдені в програмах, побажання користувачів, а також стежити за процесом усунення цих помилок і виконання або невиконання побажань.

Багтреккер *(bugtracker)* – система обліку та відстеження помилок, яка дозволяє:

- створювати;
- зберігати;
- переглядати;
- модифікувати інформацію про баги.

Як правило, система відслідковування помилок використовує той чи інший варіант "життєвого циклу" помилки *(рис.1.1.),* стадія якого визначається поточним станом, або статусом, в якому знаходиться помилка.

Система може надавати адміністраторові можливість налаштувати, які користувачі можуть переглядати й редагувати помилки в залежності від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відслідковування помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів, через те що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми не має прямого відношення до складності усунення помилки [3].

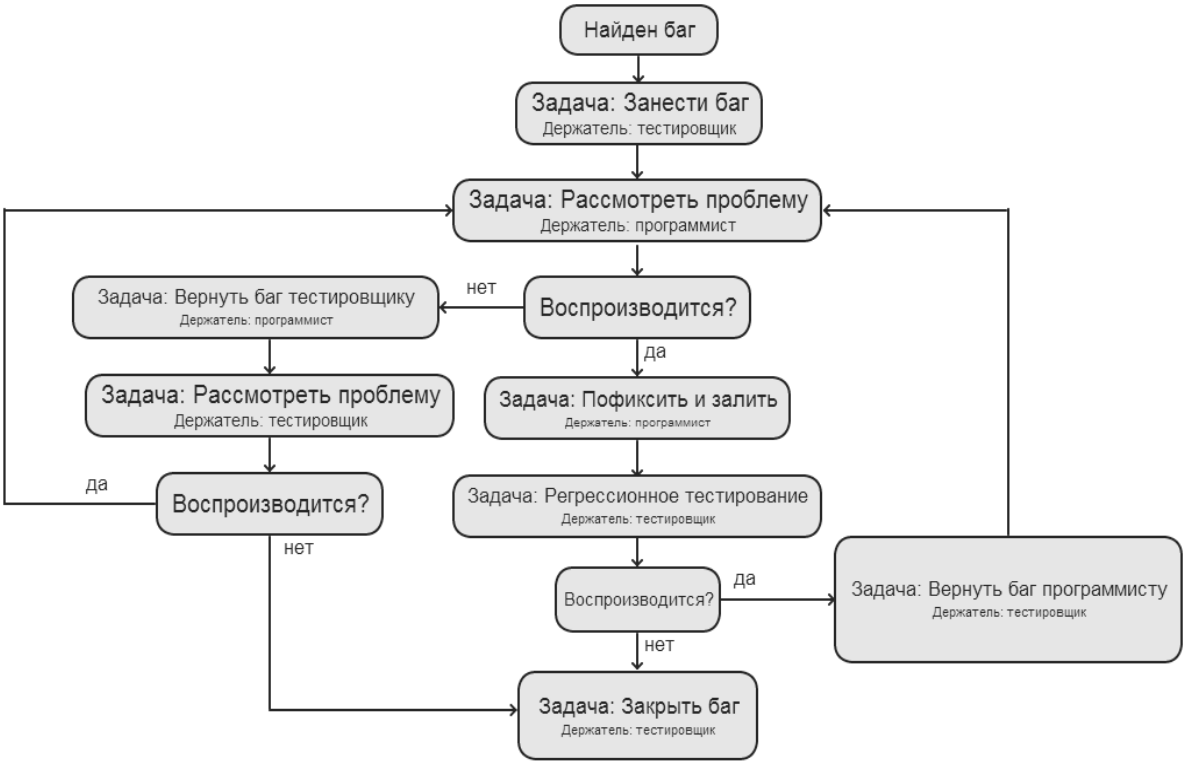


Рис. 1.1. Життєвий цикл дефекту

Сучасні системи відслідковування помилок:

1. ***Mantis bug tracking system*** – вільно розповсюджувана система відслідковування помилок у програмних продуктах *(bugtracker)*. Забезпечує взаємодію розробників з користувачами (тестувальниками). Дозволяє користувачам заводити повідомлення про помилки й відстежувати подальший процес роботи над ними з боку розробників. Система має гнучкі можливості конфігурації, що дозволяє настроювати її не тільки для роботи над програмними продуктами, але і в якості системи обліку заявок для технічної підтримки *(helpdesk)*.

2. ***Redmine*** – відкритий серверний веб-додаток для управління проектами і завданнями (у тому числі, для відстеження помилок). Redmine написаний мовою Ruby і являє собою додаток на основі широко відомого веб-фреймворку Ruby on Rails.

Redmine надає наступні можливості:

- ведення декількох проектів;
- гнучка система доступу, заснована на ролях;

- система відслідковування помилок;
- діаграми Ганта і календар;
- ведення новин проекту, документів і управління файлами;
- оповіщення про зміни за допомогою RSS-потоків і електронної пошти;
- вікі (wiki) для кожного проекту;
- форуми для кожного проекту;
- облік тимчасових витрат;
- настроюються довільні поля для інцидентів, тимчасових витрат, проектів і користувачів;
- легка інтеграція з репозиторіями *(SVN, CVS, Git, Mercurial, Bazaar і Darcs);*
- створення записів про помилки на основі отриманих листів;
- підтримка множинної аутентифікації LDAP;
- можливість самостійної реєстрації нових користувачів;
- багатомовний інтерфейс *(у тому числі російська);*
- підтримка СУБД MySQL, PostgreSQL, SQLite, Oracle.

3. ***Atlassian JIRA*** – комерційна система відслідковування помилок, призначена для організації спілкування з користувачами, хоча в деяких випадках систему можна використовувати для управління проектами. Розроблено компанією Atlassian Software Systems. Назву системи (JIRA) отримано шляхом усічення слова "Gojira", японського імені монстра Годзилла, що, в свою чергу, є відсиланням до назви конкуруючого продукту - Bugzilla. JIRA створювалася для заміни Bugzilla і багато в чому повторює її архітектуру. Система дозволяє працювати з декількома проектами. Для кожного з проектів створює і веде схеми безпеки і схеми оповіщення.

1.5 Приклади та техніка написання якісного звіту про помилки (bug report)

Основним результатом тестування є надання інформації про стан системи розробникам, менеджерам проекту та іншим зацікавленим учасникам проекту. Інженери з тестування надають зворотній зв'язок у вигляді звітів про тестові інциденти. Звіти про інциденти є головним артефактом та індикатором якості проекту.

Правила, яких необхідно дотримуватися при складанні звіту про помилки (bug report):

1. *Одна помилка - один звіт про помилку (bug report):* кожну виявлену проблему необхідно описати в окремому звіті *(bug report)*, так як

ефективніше буде сфокусуватися на одній проблемі, ніж розглядати велику їх кількість, так само, як і обговорення різних багів не повинне накладатися одне на інше.
2. *Вказуйте URL або розділ сайту*.
3. *Опишіть послідовність дій, яка призвела до даної помилки:* програміст при перевірці може упустити ту послідовність дій, яка призвела до виникнення тих чи інших недоліків.
4. *Вкажіть, що отримали після виконання дій, що призвели до помилки і що очікували отримати.*
5. *Зробіть скріншот:* бажано супроводжувати повідомлення про помилки скріншотом з додавання прямокутника *(виділяючи проблемне місце)* та червоної стрілки, а також показати сторінку повністю, з адресним рядком браузера.

Кожен якісний опис помилки повинен містити рівно три речі:

1. Які кроки привели до помилки?
2. Що Ви очікували побачити?
3. Що Ви насправді побачили?

Поле "короткого опису" (summary) у звіті про помилки (bug report) має [4]:

- містити гранично коротку, але в той же час достатню для розуміння суті проблеми інформацію про баг;

- **відповідати на "три вічних питання":**
  - ***"Що?"*** – що відбувається або не відбувається згідно специфікації або уявленням тестувальника про нормальну роботу програмного продукту. При цьому вказується наявність або відсутність об'єкта проблеми, зміст вказується в описі *(description)*. Якщо зміст проблеми варіюється - всі відомі варіанти вказуються в описі;
  - ***"Де?"*** – в якому місці інтерфейсу користувача або архітектури програмного продукту знаходиться проблема;
  - ***"Коли?"*** – в який момент роботи програмного продукту, по настанню якої події або за яких умов проблема проявляється [5].

- бути достатньо коротким, щоб повністю поміщатися на екрані (у системах відслідковування помилок в програмних продуктах *(bugtracker)*, де кінець пропозиції в полі короткого опису *(summary)* обрізається або призводить до появи скролінгу);

- (можливо) містити інформацію про оточення, в якому був виявлений баг (залежить від проектних традицій);

- бути закінченим пропозицією російською чи англійською (або іншою) мовою, побудованою за відповідними правилами граматики.

Таким чином, при написанні короткого опису *(summary)* необхідно в одній пропозиції вмістити сенс всього звіту про помилки *(bug report)*, а саме: коротко та ясно, використовуючи правильну термінологію вказати: "що", "де" *("що", "де "," за яких умов ")* не працює.

Наприклад:

1. *Додаток зависає при спробі збереження текстового файлу розміром більше 50Мб.*
2. *Дані на формі "Профайл" не зберігаються після натискання кнопки "Зберегти".*
3. *Поле "Підтвердіть пароль" не є обов'язковим на формі Реєстрації.*
4. *Текст випадаючого списку головного меню (вкладки "COURSES") відображається за межами виділеної області.*

Поле "опис" (description) у звіті про помилки (bug report) має:

- містити опис у розмірі одного речення за принципом: *"що?", "де?", "коли?"* згідно з темою;

- містити кілька пропозицій за умови, що зазначена інформація вказує на важливі нюанси, опис яких не помістився в темі через обмеження по кількості символів.

Не менш важливим для заповнення полем у звіті про помилки *(bug report)* є поле "кроки відтворення" *(steps to reproduce)*.

Кроки відтворення *(steps to reproduce)* є найціннішою інформацією у звіті, оскільки представляють собою керівництво до дії для тих, хто буде виправляти проблему, при цьому кроків має бути достатньо для відтворення проблеми, а також повинні **задовольнятися такі вимоги:**

- у першому кроці необхідно використати посилання на головний домен:
  - *Відкрити сайт http://....(Open the site http://...., Go to the site http://...);*
- в кроках необхідно відповідати на запитання: "що необхідно зробити?":

- *Натиснути на кнопку "Знайти" (Press the "Знайти" button, Click the "Знайти" button);*

- *Ввести e-mail і пароль (Enter the e-mail and the password);*
- *Заповнити необхідні поля (Fill the necessary fields);*

- в кроках необхідно коротко писати, що зробити, куди натискати, не уточнюючи назви сторінки;
- описується як мінімум 2 кроки, але не більше 7-8 кроків;
- в кроках необхідно уточнювати, на що саме необхідно натискати (посилання, кнопку, логотип ...);
- кроки необхідно писати з великої літери;
- в останньому кроці необхідно описати, на яку область дивитися, на що звернути увагу або що оглянути:
  - *Оглянути текст в випадаючому списку (підменю) (Look at the drop-down list (the submenu));*
  - *Звернути увагу на "Profile" форму (Take a look at the "Profile" form, Pay attention to the "Profile" form).*

Актуальний / Фактичний результат (actual result) і Очікуваний результат (expected result) повинні задовольняти такі вимоги:

- результати необхідно описувати інформативно *(згідно темі за принципом: "що?", "де?", "коли?", іноді можна використовувати "що?", "де?");*
- очікуваний результат слід вказувати після актуального / фактичного результату;
- результати ***НЕ*** потрібно нумерувати і ***НЕ*** варто писати кілька результатів в одному дефекті;
- результати в системі відслідковування помилок *(bug tracking system)* необхідно описувати відразу після кроків у полі "кроки відтворення" *(steps to reproduce)*;
- результати слід описувати повними реченнями з підметом і присудком;
- результати необхідно писати з великої літери.

Приклад оформлення звіту про помилки *(bug report)* у системі відслідковування помилок ***Mantis bug tracking system*** (рис.1.2 – 1.9).

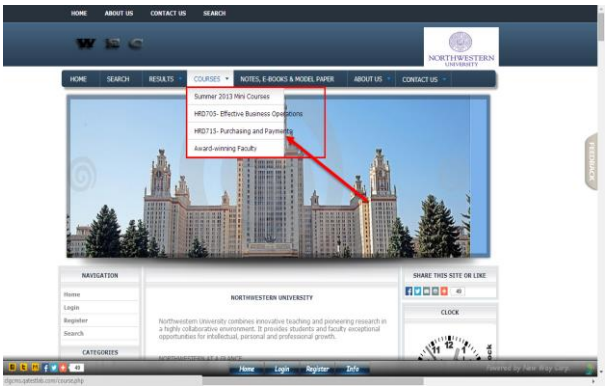
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063549		Текстовый	private	2014-05-16 15:26	2015-01-09 00:47

Reporter					
Assigned To	Training Center				
Priority	high	**Severity**	major	**Reproducibility**	always
Status	information	**Resolution**	open		
Platform	FF 21.0.9	**OS**	Windows 7	**OS Version**	2002

Summary:	0063549: Текст выпадающего списка главного меню (вкладки "COURSES") отображается за пределами выделенной области
Description	Текст выпадающего списка главного меню (вкладки "COURSES") отображается за пределами выделенной области
Steps to reproduce	1. Открыть сайт http://clgcms.qatestlab.com [^] 2. В главном меню навести курсор на "COURSES" 3. Осмотреть текст в выпадающем списке (подменю) Актуальный результат: Текст выпадающего списка главного меню (вкладки "COURSES") отображается за пределами выделенной области Ожидаемый результат: Текст подменю отображается в пределах выделенной области.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit Assign To: [Myself] Change Status To: information Monitor Stick Clone Move Delete

Рис.1.2. Звіт про помилки *(bug report)* у системі відслідковування помилок

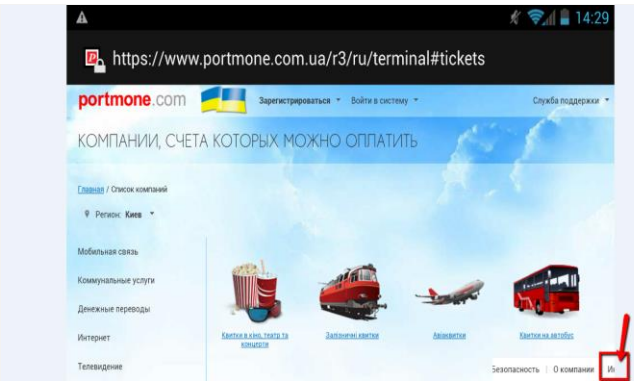
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063989		Дизайн	private	2014-05-16 15:26	2015-01-09 00:47

Reporter					
Assigned To	Training Center				
Priority	normal	**Severity**	minor	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	FF 21.0.9	**OS**	Windows 7	**OS Version**	2002

Summary:	0063989: Панель нижнего меню во вкладке “Электронные билеты” не помещается на экране полностью в горизонтальном положении.
Description	Панель нижнего меню во вкладке “Электронные билеты” не помещается на экране полностью в горизонтальном положении.
Steps to reproduce	1. Зайти в мобильное приложение https://www.portmone.com.ua/ . 2. Повернуть телефон в горизонтальное положение. 3. Перейти во вкладку “Электронные билеты”. 4. Обратить внимание на панель нижнего меню. Ожидаемый результат: Панель нижнего меню во вкладке “Электронные билеты” помещается на экране полностью в горизонтальном положении. Фактический результат: Панель нижнего меню во вкладке “Электронные билеты” не помещается на экране полностью в горизонтальном положении.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit Assign To: [Myself] Change Status To: information Monitor Stick Clone Move Delete

Рис.1.3. Звіт про помилки *(bug report):*
Панель нижнього меню у вкладці "Електронні квитки" не вміщується на екрані повністю в горизонтальному положенні

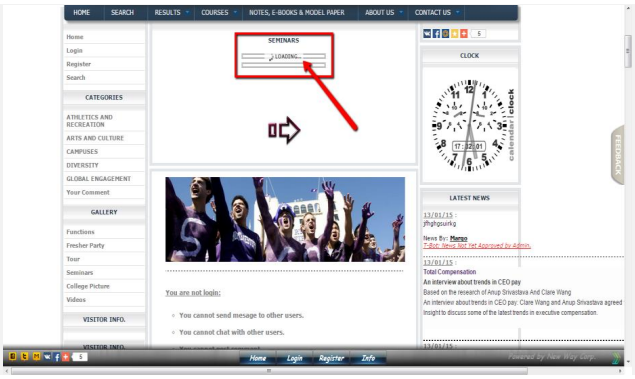
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063641		Дизайн	public	2015-01-13 17:57	2015-01-20 13:24

Reporter					
Assigned To	Training Center				
Priority	normal	**Severity**	minor	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	IE 7.0	**OS**	WindowsXP	**OS Version**	2002

Summary:	0063641: Не загружается изображение в разделе "Function" после перехода на страницу "Seminars" в левом блоке "Gallery".
Description	Не загружается изображение в разделе "Function" после перехода на страницу "Seminars" в левом блоке "Gallery".
Steps to reproduce	1. Открыть сайт http://clgcms.qatestlab.com [^] 2. Нажать на кнопку "Function". Фактический результат: Не загружается изображение в разделе "Function" после перехода на страницу "Seminars" в левом блоке "Gallery'. 'Ожидаемый результат: Загружается изображение в разделе "Function" после перехода на страницу "Seminars" в левом блоке "Gallery".
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit Assign To: [Myself] Change Status To: information Monitor Stick Clone Move Delete

Рис.1.4. Звіт про помилки *(bug report):*
Не завантажується зображення в розділі "Function" після переходу на сторінку "Seminars" в лівому блоці "Gallery"

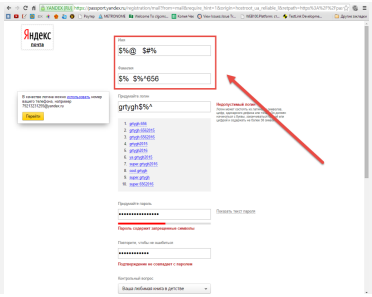
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063802		Пользовате-льский интерфейс	public	2015-01-13 17:57	2015-01-20 13:24

Reporter					
Assigned To	Training Center				
Priority	normal	**Severity**	minor	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	Google Chrome 39.0.2171.95 m	**OS**	Windows 8.1	**OS Version**	8.1

Summary:	0063802: Не отображается ошибка формата ввода в полях "Имя" и "Фамилия" после вводе недопустимых символов.
Description	Не отображается ошибка формата ввода в полях "Имя" и "Фамилия" после вводе недопустимых символов.
Steps to reproduce	1. Открыть сайт http://ya.ru. 2. Нажать на кнопку "Завести почту". 3. В поле "Имя" ввести недопустимые символы ("№;%: ,…). 4. В поле "Фамилия" ввести недопустимые символы ("№;%: ,…). Актуальный результат: Не отображается ошибка формата ввода в полях "Имя" и "Фамилия" после вводе недопустимых символов. Ожидаемый результат: Отображается ошибка формата ввода в полях "Имя" и "Фамилия" после вводе недопустимых символов.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit | Assign To: [Myself] | Change Status To: information | Monitor | Stick | Clone | Move | Delete

Рис.1.5. Звіт про помилки *(bug report):*
Не відображається помилка формату введення в полях "Ім'я" та "Прізвище" після введення неприпустимих символів

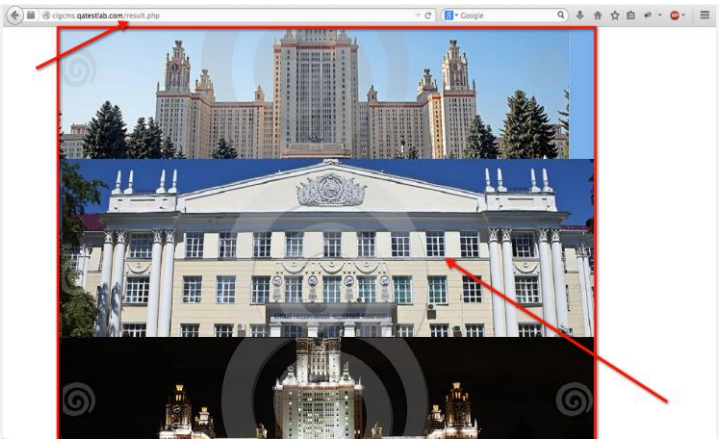
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063549		Функциональный	private	2014-05-16 15:26	2015-01-09 00:47

Reporter					
Assigned To	Training Center				
Priority	high	**Severity**	major	**Reproducibility**	always
Status	information	**Resolution**	open		
Platform	FF 21.0.9	**OS**	Windows 7	**OS Version**	2002

Summary:	0063549:The slider pictures are not switched automatically after clicking the "Results" tab of the main menu.
Description	The slider pictures are not switched automatically after clicking the "Results" tab of the main menu.
Steps to reproduce	1. Open the site http://clgcms.qatestlab.com [^] 2. Click the "Results" tab of the main menu. 3. Take a look at the slider pictures. Actual result: The slider pictures are not switched automatically after clicking the "Results" tab of the main menu. Expected result: The slider pictures are switched automatically at the site pages.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit | Assign To: [Myself] | Change Status To: information | Monitor | Stick | Clone | Move | Delete

Рис 1.6. Звіт про помилки *(bug report):*
The slider pictures are not switched automatically after clicking the "Results" tab of the main menu.

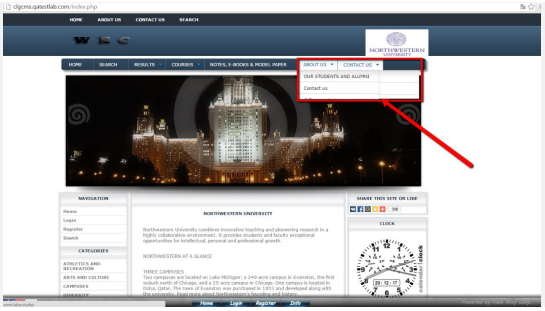
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063502		Дизайн	public	2015-01-13 02:58	2015-01-14 13:04

Reporter					
Assigned To					
Priority	urgent	**Severity**	block	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	Opera 26.0	**OS**	Windows 7 (64 bit)	**OS Version**	Windows 7

Summary:	0063502:The dropdown menu in the “About us” headline is superimposed on the dropdown menu in the headline “Contact us”.
Description	The dropdown menu in the “Contact us” headline is blocked after guiding the cursor at the dropdown menu in the “About us" headline.
Steps to reproduce	1. Go to the site http://clgcms.qatestlab.com. [^] 2. Take a look at the main menu. 3. Put the cursor on the “Contact us” headline. 4. Transfer the cursor on the “About us" headline. Actual result: The dropdown menu of the “Contact us” and “About us" headlines are superimposed. Expected result: The dropdown menu of the “Contact us” and “About us" headlines are not superimposed.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit Assign To: [Myself] Change Status To: information Monitor Stick Clone Move Delete

Рис 1.7. Звіт про помилки *(bug report):*
“The dropdown menu in the “About us” headline is superimposed on the dropdown menu in the headline “Contact us”.”

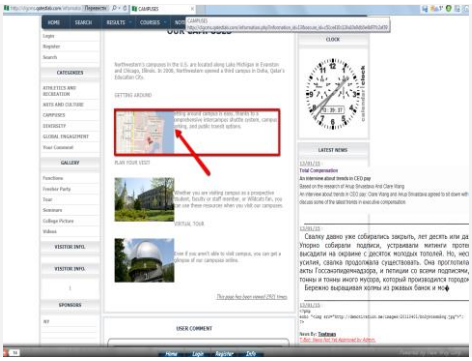
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0063549		Дизайн	public	2015-01-13 02:58	2015-01-14 13:04

Reporter					
Assigned To					
Priority	normal	**Severity**	text	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	Opera 26.0	**OS**	Windows 7 (64 bit)	**OS Version**	Windows 7

Summary:	0063549:The “Getting around” image is superimposed on the text on the “Our Campuses” web-page
Description	The “Getting around” image is superimposed on the text on the “Our Campuses” web-page.
Steps to reproduce	1. Open the site http://clgcms.qatestlab.com [^] 2.Go to the “Categories” headline in the left menu. 3. Click the “Campuses” subhead. 4. Take a look at the “Getting around” image. Actual result: The “Getting around” image is superimposed on the text at the “Our Campuses” web-page. Expected result: The “Getting around” image is not superimposed on the text on the “Our Campuses” web-page.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit | Assign To: [Myself] | Change Status To: information | Monitor | Stick | Clone | Move | Delete

Рис 1.8. Звіт про помилки *(bug report):*
The “Getting around” image is superimposed on the text on the “Our Campuses” web-page.

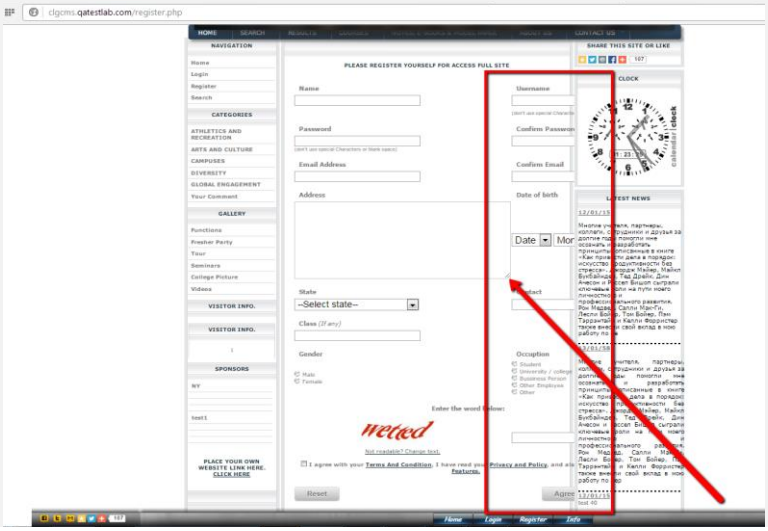
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Manage | My Account | Logout Issue # Jump

Recently Visited: 0063621, 0063572, 0063568, 0077531, 0077776

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

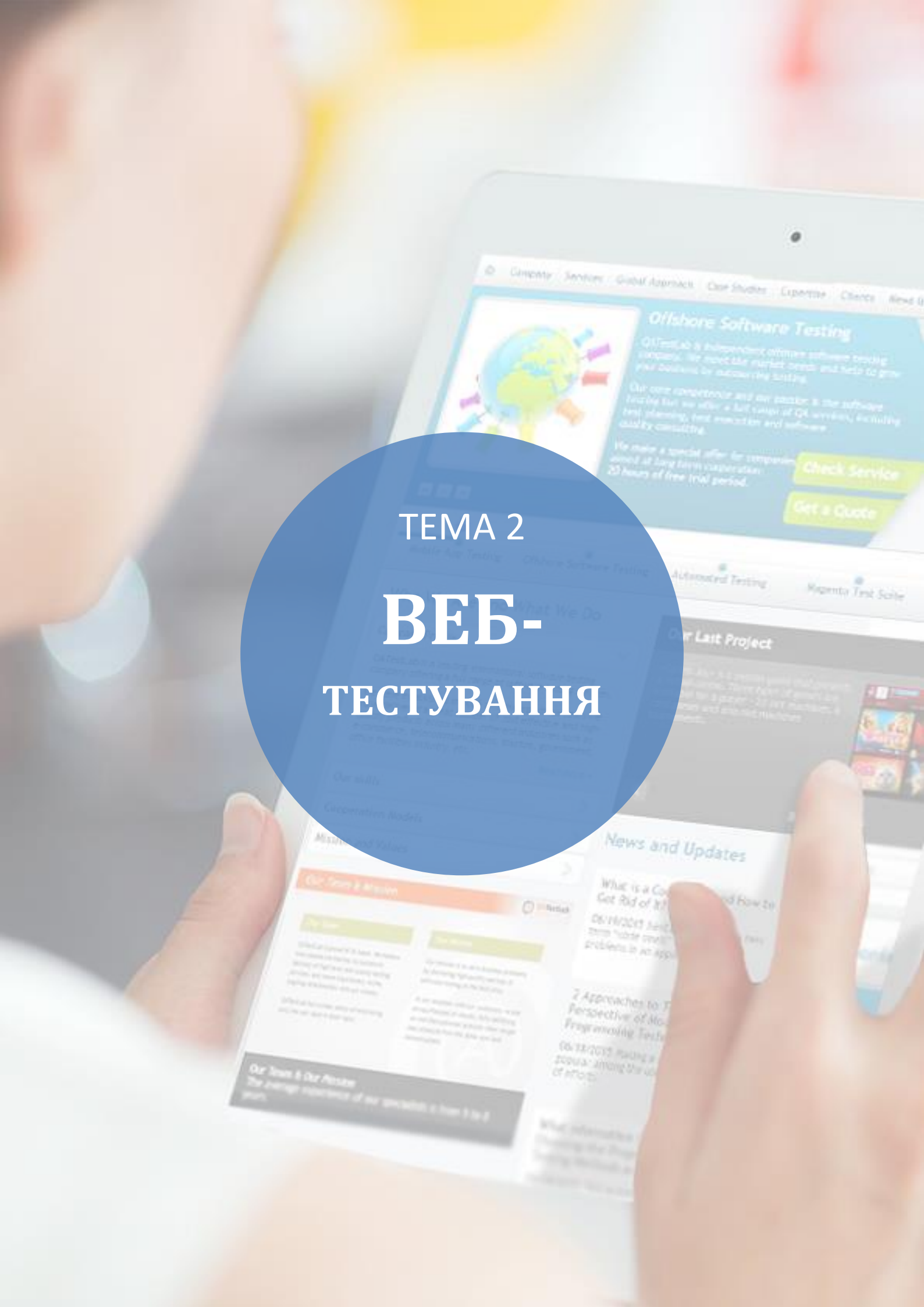
ID	Project	Category	View Status	Date Submitted	Last Update
0063549		Дизайн	public	2015-01-13 02:58	2015-01-14 13:04

Reporter					
Assigned To					
Priority	urgent	**Severity**	crash	**Reproducibility**	always
Status	confirmed	**Resolution**	open		
Platform	Opera 26.0	**OS**	Windows 7 (64 bit)	**OS Version**	Windows 7

Summary:	0063549: The layout is crashed on the "Register" form after changing the "Address" field.
Description	The layout is crashed on the "Register" form after changing the "Address" field.
Steps to reproduce	1. Open the site http://clgcms.qatestlab.com/index.php/. 2. Go to the footer of the web-page. 3. Click the "Register" button. 4. Pull at the right bottom corner "Address" field at the "Register" form. Actual result: The fields are superimposed on the other elements on the right side of the "Register" form. Expected result: The "Address" field has the fixed scale on the "Register" form.
Tags	No tags attached.
Attach Tags	(Separate by ",") Existing tags Attach
Attached Files	

Edit | Assign To: [Myself] | Change Status To: information | Monitor | Stick | Clone | Move | Delete

Рис 1.9. Звіт про помилки *(bug report):*
The layout is crashed on the "Register" form after changing the "Address" field.



ТЕМА 2

ВЕБ-ТЕСТУВАННЯ

2 ВЕБ –ТЕСТУВАННЯ

Компанії та окремі користувачі все більше залежать в своїй роботі від веб-додатків *(web-applications)*. Веб-додатки дуже динамічні, а їх функціональні можливості безперервно ростуть.

Безперервно зростає потоковий трафік засобів інформації і запитів, які формуються переносними та вбудованими пристроями. Веб-додатки стають все більш поширеними і більш складними, граючи, таким чином, основну роль в більшості онлайнових проектів. Як і у всіх системах, що ґрунтуються на взаємодії між клієнтом і сервером, вразливості веб-додатків зазвичай виникають із-за некоректної обробки запитів клієнта чи недостатньої перевірки вхідної інформації зі сторони розробника.

Для усунення всякого роду неполадок і недоліків виконують тестування під час створення та використання веб-додатків [9].

2.1 Поняття веб-тестування (web-testing) та основні підходи до тестування веб-додатків (web-applications)

Важливою особливістю веб-тестування *(web-testing)* являються часові рамки. Якщо при тестуванні програмного забезпечення *(software)* в тестувальника є місяці (а інколи і роки), то у веб-тестувальника є лише дні і неділі (в кращому випадку) на тестування запропонованого сайту. Отже, якщо при тестуванні програм дозволяється і потрібно формувати детальні плани тестування, описувати тестові випадки *(test cases)* на основі отриманої від розробника документації, то при тестуванні веб-сторінок це может значно продовжити строки публікації матеріалів в мережі Інтернет. Таким чином, на веб-тестування *(web-testing)* може виділятися до 50% загального бюджету і часових ресурсів.

Веб-тестування (web-testing) – тестування програмного забезпечення, сфокусоване на веб-додатках *(web*-applications), яке здійснюється з застосуванням систем, що використовують веб-технології *(web-basedsystem),* та вирішують такі питання як: безпека веб-додатків, базова функціональність сайту, доступність користувачам і т.д. [8].

Веб-додатки (web-applications) – клієнт серверний додаток, в якому клієнтом виступає браузер, а сервером - веб-сервер.

Веб-сайт (Web-site) – система з декількох сторінок, що мають одну адресу в мережі Інтернет.

Верстка (layout) – етап дизайну сторінки сайту, що представляє собою просторове розміщення на ній текстових елементів і графічних зображень відповідно до концепції оформлення ресурсу.

Веб-дизайн (web-design) - галузь веб-розробки, а також різновид дизайну, до завдань якої входить проектування веб-інтерфейсів *(Web UI)* для сайтів або веб-додатків.

Розрізняють такі етапи тестування (stages of testing):

1. ***Вивчення документації (documentation)*** – вивчається отримана документація, аналізується функціонал за технічним завданням, кінцеві макети сайту і складається план тесту для подальшого тестування. Необхідно визначити цілі сайту і споживачів сайту.

2. ***Тестування верстки (Layout testing)*** – перевіряється розташування елементів, відповідність їх позицій наданим макетам, а також перевіряється оптимізація зображень і графіки. Завершивши перевірку на валідність, фахівець приступає до перевірки на кроссбраузерність, тобто перевіряє працездатність сайту в різних браузерах та при різних параметрах налаштувань екрану.

3. ***Функціональне тестування (functional testing)*** – процес верифікації відповідності функціонування продукту його початковим специфікаціям і представляє найбільш тривалий етап перевірки ресурсу. ***Суть цього процесу полягає у перевірці всього описаного функціоналу:***
   - посилання (працездатність, відкриття в тому ж або новому вікні і т.п.);
   - форми (введення тексту, чисел, використання маски, робота з незаповненими полями, довжина символів, що вводяться, коректна робота чек боксів, комбо боксів, radio кнопок, логічність установок "за замовчуванням" і т.д.);
   - бази даних (пошук, додавання інформації, редагування, видалення, перевірка на дублювання інформації).
   - секретність (робота з паролями, передача даних, захист і т.д.);
   - кешування (перевірка на установку кешування та оновлення файлів);
   - перевірка роботи з браузером (оновлення сторінки, посиляння / кнопка “Назад”, зміна розмірів вікна, вибір кодування, скролінг);
   - фрейми (завантаження сторінок, скролінг і т.п.);
   - анімація (наявність, зміна розмірів, завантаження і т.д.);

- аудіо і відео (наявність, розміщення, якість та ін.);
- ActiveX;
- друк (чи коректно друкуються сторінки);
- тестування додатка шляхом включення або відключення куків *(cookie)* в настройках браузера [10,11,12,13].

4. ***Тестування практичності / перевірки на простоту використання (usability testing)*** – проводиться для оцінки зручності продукту у використанні:
   - ***тестування навігації (test for navigation)*** – веб-сайт повинен бути простим у використанні, інструкції повинні бути зрозумілими і головне меню повинно бути передбаченим на кожній сторінці і бути послідовним;
   - ***перевірка вмісту (content checking)*** – зміст має бути логічним і простим для розуміння без орфографічних помилок, без використання кольорової гами, яка дратує користувачів.

Тестування практичності одне з найдорожчих, тому що найбільш цінну інформацію можна отримати тільки від реальних користувачів, спостерігаючи за їх роботою з сайтом. Такі дослідження вимагають дорогої інфраструктури і тимчасових витрат, їх складно автоматизувати. Тестування практичності / перевірки на простоту використання *(usability testing)* частково перетинається з тестуванням якості виконання сайту і зручності інтерфейсу.

5. ***Тестування безпеки (Security testing)*** – перевіряється доступ користувача до службових / закритих сторінок, а також проводиться перевірка захисту всіх критично важливих сторінок (наприклад, розділу адміністрування сайту) від зовнішнього впливу і включає наступні дії:
   - виявлення помилок в коді сайту та програмному забезпеченні сервера;
   - виявлення наявності таких вразливостей, як SQL Injection або ін'єкція SQL, XSS (Cross SiteScripting), PHP includes, биті посилання *(broken links)* [14];
   - тестування на стійкість до комбінованих методів атак і нестандартним технікам взлому.

На відміну від інших тестів, тестування безпеки *(security testing)* слід проводити регулярно. Крім того, тестуванню піддається не тільки сам сайт або веб-додаток, а весь сервер повністю - і веб-сервер, і операційна система, і всі мережеві сервіси. Як і у випадку інших тестів, програма "прикидається" реальним користувачем-хакером і намагається застосувати до сервера всі відомі методи атаки і перевіряє всі вразливості. Результатом

роботи буде звіт про знайдені вразливості та рекомендації щодо їх усунення. Зазвичай такі сканери безпеки продаються як самостійний продукт. Для прикладу, одним із кращих сканерів на даний момент являється XSpider [15].

6. ***Тестування продуктивності сайту (performance testing)*** – проводиться з метою визначення швидкодії сайту або його частини під певним навантаженням.

Тестування продуктивності включає в себе такі види тестування:

- ***Навантажувальне тестування (Load-testing)*** - найпростіша форма тестування продуктивності. Тестування навантаження зазвичай проводиться для того, щоб оцінити поведінку сайту (або програми) під заданим очікуваним навантаженням. Цим навантаженням може бути, наприклад, очікувана кількість одночасно працюючих користувачів на сайті, що виконують задане число транзакцій за інтервал часу. Такий тип тестування зазвичай дозволяє отримати час відгуку всіх найважливіших бізнес-функцій. ***Основні цілі:***
  - оцінка продуктивності і працездатності програми на етапі розробки і передачі в експлуатацію;
  - оцінка продуктивності і працездатності програми на етапі випуску нових релізів, інформації, призначеної для автоматизованого внесення певних змін в комп'ютерні файли *(patch);*
  - оптимізація продуктивності додатку, включаючи налаштування серверів та оптимізацію коду;
  - підбір відповідної для цього додатку апаратної (програмної платформи) і конфігурації сервера.

- ***Тестування швидкодії*** – перевірка швидкості завантаження сайту для визначення швидкості відпрацювання скриптів, завантаження зображень і контенту. Цей тест проводиться з метою оптимізації процесу завантаження сайту, а також визначення оптимальності налаштувань сервера [13].

У такому тесті перевіряється не тільки і не стільки сам сайт, скільки спільна злагоджена робота всього комплексу - апаратної частини сервера, веб-серверу, програмного ядра *(engine)* та інших компонентів сайту. До подібних рішень можна віднести і описану утиліту WAPT, а також ряд інших – Microsoft Web Application Stress Tool (WAS), SilkPerformer, Webserver Stress Tool та інші.

2.2 Анатомія веб-сторінки

Елементи веб-сторінки *(web-page)* забезпечують життєво важливу і естетичну функцію сторінки.

До основних елементів анатомії веб - сторінки (web-page) відносять (рис. 2.5, 2.6) [16]:

- ***заголовок (header)*** – використовується у верхній частині сторінки, де міститься логотип *(logo)*,навігацію *(navigation)*,рекламний слоган *(tagline)*, назву сайту та сервісні форми, такі як пошук *(search)*, зв'язок або авторизація (authorization).
- Заголовок можна зробити фіксованим, тобто задати певну ширину, яка не змінюється від розміру вікна браузера, або нефіксованим, який може розтягуватися на всю ширину вікна браузера (рис.2.1);

Рис. 2.1. Заголовок *(header)*

- ***логотип (logo)*** – форма товарного знаку, елемент фірмового стилю, що представляє собою оригінальний обрис рекламодавця.
- Найчастіш використовуване місце для логотипу *(logo)* - всередині заголовка (вирівнювання по лівому краю), тому традиційно користувач читає зліва направо і зверху вниз, логотип *(logo)* буде першим елементом, на який користувач може звернуто увагу (рис. 2.2);

- ***навігація (navigation)*** – важливий елемент веб - сторінки *(web-page)*, що дозволяє легко орієнтуватися на сторінці.
- Навігація повинна бути простою у використанні, майже завжди зосереджена всередині заголовка або, щонайменше знаходитися зверху сторінки. Іноді для сайту з об'ємним наповненням використовується ***два типи навігації:***

  - ***горизонтальна:*** серія посилань, розташованих в рядок, зазвичай іменується «навігацією»;

- ***вертикальна:*** серія посилань, розташованих у вертикальній області, зазвичай іменується «меню» (рис.2.2).

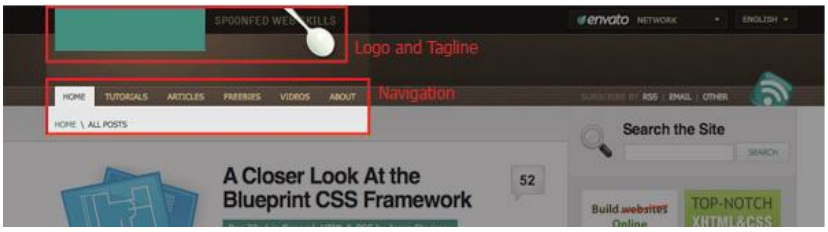


Рис. 2.2. Логотип *(logo)* і навігація *(navigation)*

- ***основний контент (main content)*** – текстове наповнення веб-сайту, яке за правилами практичності використання *(usability)* має займати не менше 70% сторінки, а також включати в себе заголовки, текстовий і графічний контент (рис.2.3).

Зліва від поля контенту розташовуються додаткові навігаційні меню, що дозволяють користувачеві перейти на другорядні по важливості сторінки, наприклад на статті - рекомендації або відгуки.

Праворуч від поля контенту знаходиться поле для реклами, додаткових елементів інтерфейсу *(widgets)*, таких як коментатори або хмара міток *(tags cloud)*.

- ***бокова панель (sidebar)*** – блок, в якому розміщується додатковий контент: оголошення, реклама, пошук, посилання *(links)* на підписку, способи зв'язку.
  Бокова панель *(sidebar)* не є обов'язковою, проте багато сайтів містять такі панелі.

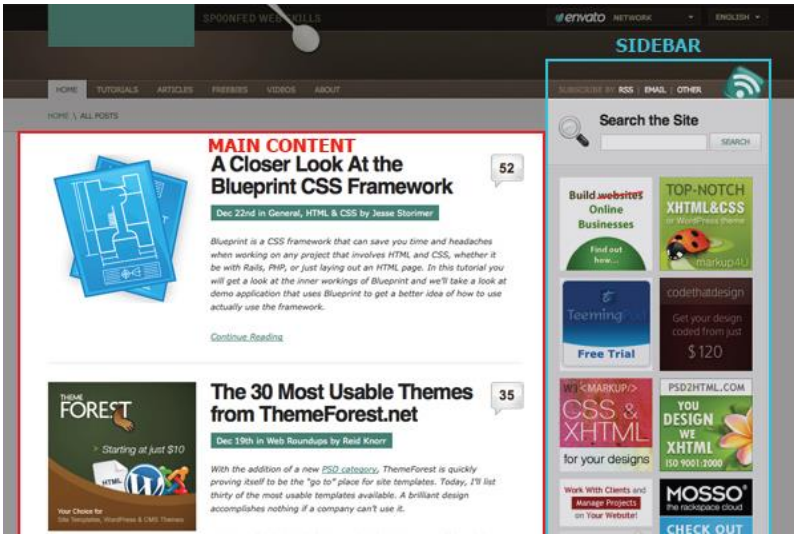


Рис.2.3. Основний контент *(main content)* та бокова панель *(sidebar)*

- ***підвал сайту (footer)*** – елемент сайту, який зазвичай поміщається внизу веб-сторінки і виконує допоміжну роль, де найчастіше перебувають посилання на основні розділи сторінки; знак охорони авторського права; лічильники; посилання на файл ліцензійної угоди; об'єкти, яким не знайшлося місця на основній сторінці (рис.2.4.)

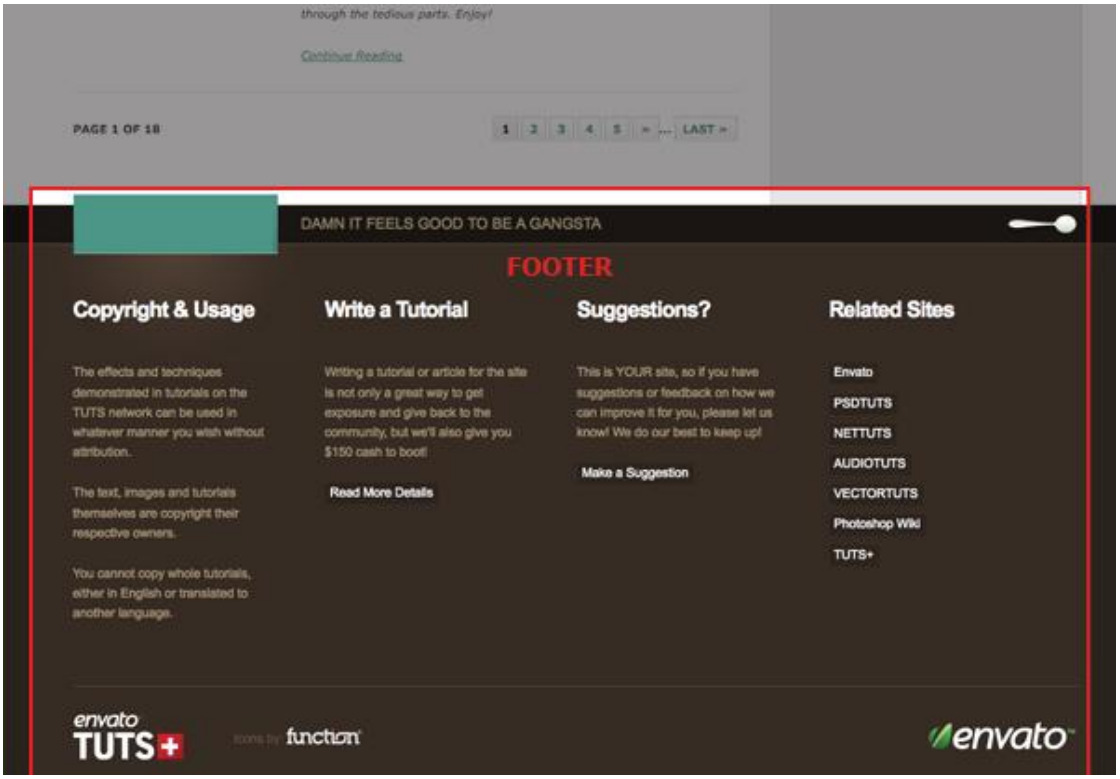


Рис. 2.4. Підвал сайту *(footer)*

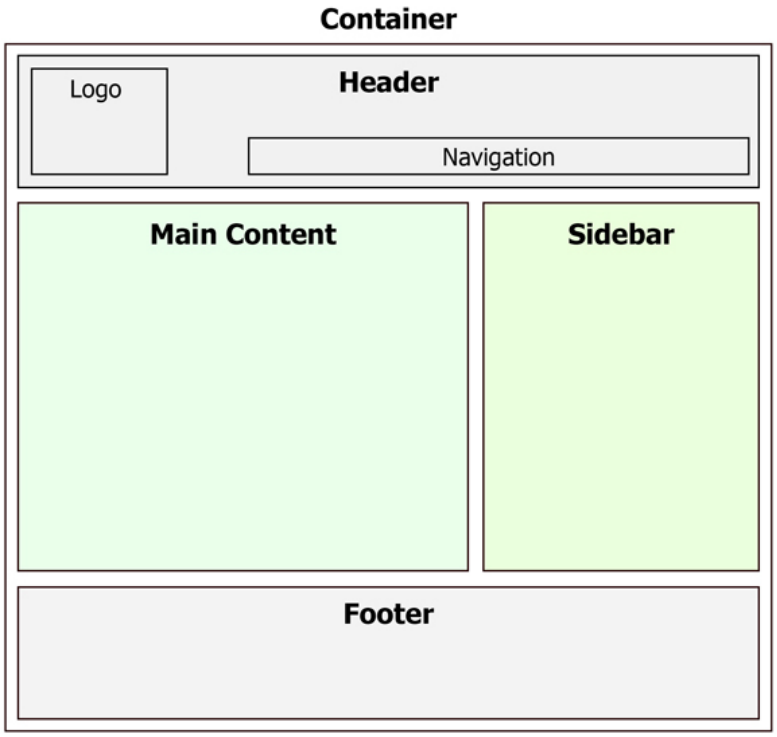


Рис.2.5. Анатомія простої веб-сторінки *(web-page)*

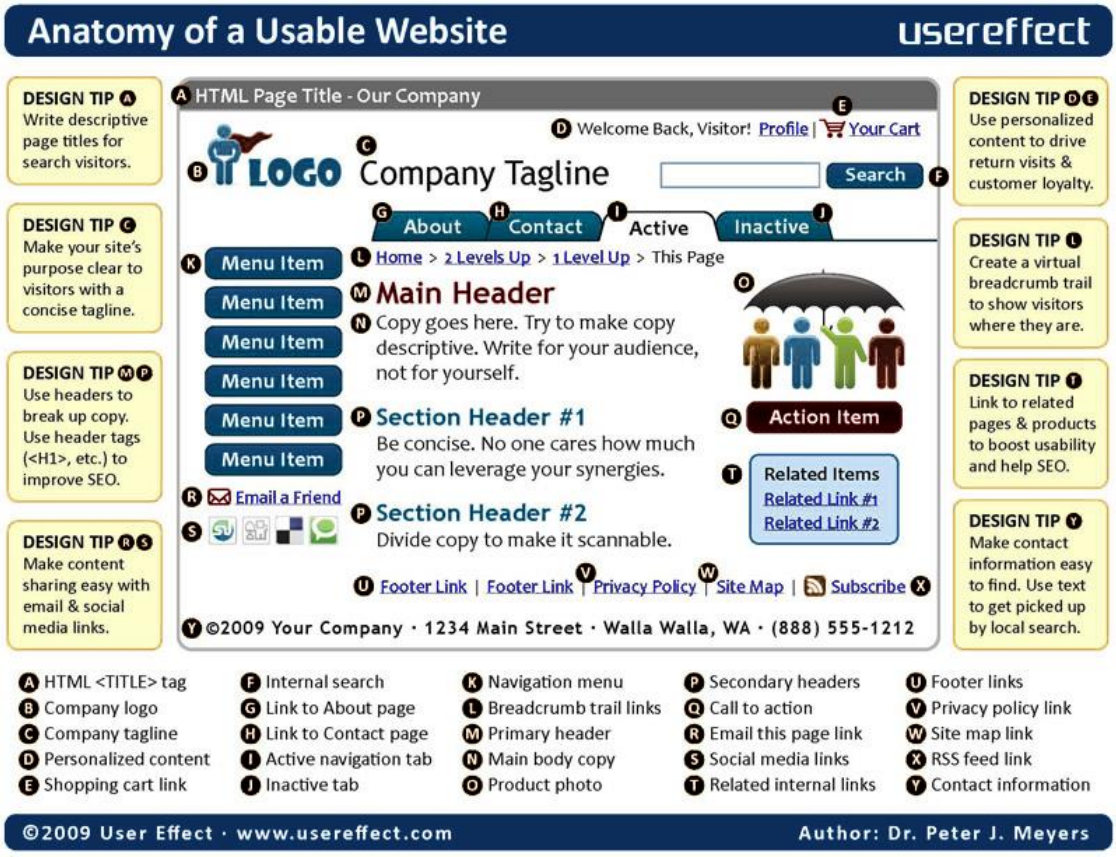


Рис.2.6. Анатомія складної веб-сторінки *(web-page)*

2.3 Тестування верстки

При тестуванні верстки *(layout)*, необхідно враховувати значну кількість факторів і виконувати у подальшому наступні **види перевірок:**

1. ***Візуальна частина:*** тестування проводиться в одному з найбільш часто використовуваному браузері і в наступній послідовності:
   - перевіряються помітні оку поламані блоки, невідповідність кольору, некоректне відображення тексту навколо зображень;
   - точність відповідності макета: використовується накладка в програмі "Photoshop";
   - перевіряється сітка (вертикальні / горизонтальні вирівнювання).

Часто перевірка сітки упускається в формах. Якщо сітка крива, але верстка відповідає дизайну, це питання можна обговорити з дизайнером / замовником;

- перевіряється в різних розширеннях:

Список класичних розширень		
1024x600	*1280x800*	*1680x1050*
1024x768	*1280x1024*	*1920x1080*
1152x864	*1440x900*	

- не повинно нічого “ламатися”;
- не повинна з'являтись горизонтальна прокрутка для обумовлених в ТЗ розширеннях;
- не повинні різко обриватися фони при великих розширеннях (що розміру картинки вистачило на велике розширення екрану).

Можна переводити екран в різні розширення або можна використовувати інструменти тестування (наприклад, в Firefox Меню -> Інструменти -> Веб-розробка -> Адаптивний дизайн або Resolution Test plugin в Chrome);

- зменшуємо розмір вікна менше мінімального по технічному завданню - не повинно нічого “ламатися”, фони не повинні "розпливатись";
- перевіряємо масштабування сторінки. У розумних масштабах (ми обмежуємося діапазоном 75-150%) сторінка повинна виглядати без візуальних неточностей;

- чи нормально підсвічуються поля у фокусі;
- чи нормально підсвічуються поля з помилками.

2. ***Доступність*** включає наступні **етапи:**
   - чи виділяється текст в текстових блоках;
   - чи нажимаються “клікабельні” елементи (посилання / кнопки);
   - чи встановлюється фокус в поля форм;
   - клікабельні елементи повинні мати покажчик "курсор", такі, що перетягуються - "лапка" або "ресайз", активні / недоступні - курсор default;
   - в ідеальному випадку всі активні елементи повинні реагувати на наведення, не доступні / неактивні - не повинні;
   - клікабельні елементи, призначення яких не очевидно, повинні бути забезпечені підказками (title);
   - лого на головній сторінці посиланням бути не повинно, на всіх внутрішніх сторінках - повинно;
   - перевірка друку сторінки (якщо було в технічному завданні);
   - написи (особливо логотип і головне меню сайту) повинні залишатися читабельними, для всіх інформаційних картинок повинні бути присутні підписи акуратним невеликим сірим шрифтом (так, для img можна задавати font - це зовнішній вигляд alt-тексту, що виводиться замість картинки). Картинки часто відключають при використанні дорогого Інтернету, що тарифікуються по трафіку (GPRS, роумінг). Перевіряється в FF через плагін Web Developer → Images → Replace Images With Alt Attributes.
   - Працездатність при вимкненому JavaScript. JS може бути вимкнений згідно корпоративних вимог безпеки. А в Opera Mini він працює тільки методом перезавантаження сторінки. Весь критично важливий функціонал сайту повинен бути доступний без JS. Перевіряється в FF через плагін Web Developer → Disable → Disable JavaScript → All JavaScript;
   - Повинна бути присутня іконка favicon.ico

3. ***Коректна робота при введенні реального тексту, надійність верстки.***

Верстка повинна “тягнутися”, чи не розвалюватися і не втрачати дизайнерський вигляд при зміні контенту на сторінці. Контенту може бути більше або менше, ніж на макеті.

Обов'язково потрібно перевіряти видалення заголовків (буває, що відступи між блоками після цього пропадають). Це розповсюджена

помилка і причина в тому, що відступи були задані не для блоків, а для внутрішніх елементів - заголовків.

- ***Перевірка введення і видалення даних.***
  - Перевіряється на сторінці з контентом: пробуємо додавати і видаляти вміст - «що буде, коли тексту багато?», «а коли мало?».
  - Обов'язково спробувати змінити розташування елементів: щоб після того, як змінюємо блоки місцями, не розвалилося оформлення (із-за каскаду).
- ***Перевірка коректності роботи стилів.***
  - На сторінках з контентом додаємо текст з абзацами і без абзаців (важливо! буває горе-верстальники прописують стилі тільки для абзаців), зі списками та картинками, таблицями і заголовками різних рівнів.

Це потрібно перевіряти, щоб на сайті потім не з”явились проблеми при заповненні контенту реальними даними.

Правки вмісту сторінки робляться в Mozilla FireFox через плагін Firebug: HTML → Edit - змінюємо / додаємо / видаляємо текст. Хороший приклад тексту для перевірки знаходиться в normalize.css в test.html між <body> і </ body>.

4. ***404-і запити,*** чи немає у верстці 404-х запитів (firebug).

ТЕМА 3

ЮЗАБІЛІТІ та ЧЕК-ЛИСТИ

3 ЮЗАБІЛІТІ ТА ЧЕК-ЛИСТИ

У даній лекції розглянуто основні поняття: тестування юзабіліті, методи проведення юзабіліті тестування та поняття чек-листа.

3.1 Визначення тестування зручності використання (usability testing)

Іноді виникають незрозумілі, нелогічні додатки, багато функцій та способів використання яких часто не очевидні. Після такої роботи рідко виникає бажання використовувати додаток знову, та користувач шукає більш зручні аналоги. Для того, щоб додаток був популярним, йому мало бути функціональним – повинен бути ще й зручним. Якщо задуматися, інтуїтивно зрозумілі додатки економлять нерви користувачам та витрати роботодавця на навчання. А значить такі додатки більш конкурентоспроможні. Тому, тестування зручності використання, про яке піде мова далі, є невід'ємною частиною тестування будь-яких продуктів масового використання.

Тестування зручності використання – це метод тестування, спрямований на встановлення ступеня зручності використання та навчання, зрозумілості та привабливості для користувачів розроблювального продукту в контексті заданих умов.

Тестування зручності користування дає оцінку рівня зручності використання програми ***за наступними пунктами:***

- ***продуктивність, ефективність (efficiency)*** - скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупка і т.д.? (*менше - краще*);

- ***правильність (accuracy)*** - скільки помилок зробив користувач під час роботи з додатком? (*менше - краще*);

- ***активізація в пам'яті (recall)*** – як багато користувач пам'ятає про роботу програми після припинення роботи з нею на тривалий період часу? (*повторне виконання операцій після перерви має проходити швидше ніж у нового користувача*);

- ***емоційна реакція (emotional response)*** – як користувач себе почуває після завершення завдання - розгублений, в стані стресу? Чи порекомендує користувач систему своїм друзям? (*позитивна реакція - краще*).

Питання які виникають при розгляді тестування зручності використання (usability testing):

1. ***Чи важливе юзабіліті?*** – Дуже важливе для всіх видів сайтів. Неважливо скільки трафіку приведете на сайт, якщо трафік не буде зручний та зрозумілий, користувачев піде та не захоче більше повертатися. Тому, над сайтом потрібно постійно працювати, тестувати зручність, тестувати конверсії та виявляти кращий варіант дизайну, структури та логіки сайту.
2. ***Що можна тестувати на юзабіліті?*** – Будь-який інтерфейс користувача, за допомогою якого користувач виконує задачі. ***Це можуть бути:***
   - веб-сайти;
   - інтранет- і настільні додатки;
   - «коробочні» програмні продукти;
   - побутова електроніка та мобільні телефони. Будь-який пристрій з інтерфейсом треба тестувати. Особливо важливі етапи початку експлуатації та інсталяції;
   - прототипи всього вищезазначеного.

Виявлення юзабіліті-проблем на ранніх етапах дозволяє не витрачати ресурси на розробку невдалих рішень.

Важливо звертати увагу на:

- ✓ простоту використання сайту або інтерфейсу;
- ✓ ефективність використання;
- ✓ запам'ятовуваність;
- ✓ помилки, їх кількість та серйозність;
- ✓ задоволення користувача *(суб'єктивне)*.

При створенні, наприклад, сайту завжди важливо спочатку розуміти аудиторію: «що робить?», «як шукає?», «до чого звикла?», «що само собою зрозуміле?» та проектувати дизайн або інтерфейс під аудиторію, тому для визначення основних вимог ***використовують наступні підходи:***

- спліт тести *(різні дизайни с тим же трафіком)*;
- фокус групи.

3.2 Коротка історія юзабіліті

Сфера юзабіліті в тому вигляді, в якому існує на теперішній час, бере початок у 1980-х. Більшість методик, що використовуються в юзабіліті, своїм корінням сягають до ергономіки, що зародилася на початку 20 століття та значною мірою проявила свій вплив в Другу Світову війну. Розглянемо коротко ключові події історії, найбільш вплинули на дану сферу.


SEE THE PROOF!

PROOF 1 LOWER OPERATING COST

An accurate electric meter measures the amount of current used by Frigidaire and proves how little current the Meter-Miser uses. Compare it with others.

PROOF 2 SAFER FOOD PROTECTION

Frigidaire dares to place a Food-Safety Indicator in the cabinet to prove that Safety-Zone Temperature is maintained — at lowest cost. Compare it with others.

PROOF 3 FASTER FREEZING—MORE ICE

The Thermo-Gauge proves Frigidaire's ability to freeze larger quantities of ice—faster. Compare it with others.

PROOF 4 MORE USABILITY

Wider, roomier, handier to use. Automatic Reset Defroster. Full-Width Sliding Shelves. Portable Utility Shelf. Double-Range Cold Control. Saves steps, Saves work. Compare it with others.

1911 - Фредерік Тейлор публікує «Принципи наукової організації праці», що описують дослідження часу та руху об'єктів в процесі виробництва й методики, що відносяться до ефективності цих процесів.

1916 - Френк і Ліліан Гілбрет розбивають робочі процеси на більш дрібні кроки і пропонують нові способи оптимізації робіт різного роду, починаючи з цегляної кладки і закінчуючи діловодством. Вони використовують свою методику під час Першої Світової війни, демонструючи солдатам алгоритм збирання-розбирання зброї в темряві.

Palm Beach Post публікує рекламу нового Frigidaire, підкреслюючи поліпшені якості юзабіліті *(рис)*.

1943 - Альфонс Чапаніс, лейтенант армії США, показує можливість зниження ймовірності «помилки пілота» за рахунок більш інтуїтивного розташування елементів панелі приладів в кабіні пілота.

1947 - Джона Карліна призначають директором нового відділу по «перевагам користувачів» в Bell Labs. Пізніше його перейменовують в відділ, що займається «людським фактором». Джон допомагає в розробці сучасної системи чисельного набору номера, яка використовується і до цього дня. Карлін проводить знамените дослідження, що визначило граничну довжину телефонного шнура для моделей телефонів AT & T. Ночами він вкорочує телефонні шнури в офісі корпоративної лабораторії до тих пір, поки співробітники не починають скаржитися в технічну підтримку.

1954 - Пол Фіттс публікує статтю в Журналі експериментальної психології *(Journal of Experimental Psychology)*, в якій описує математичну

модель, зв'язуючи час руху з точністю руху і з відстанню переміщення, що стала відомою в якості Закону Фіттса *(чим далі або точніше виконується рух, тим більше корекції необхідно для його виконання, і відповідно, більше часу потрібно для внесення цієї корекції).*

1956 - Психолог Джордж Міллер вперше використовує фразу «чарівне число сім плюс-мінус два», отриману ним у результаті ряду експериментів, які говорять про те, що людина не може думати про більш ніж 5-9 речей одночасно.

1957 - Сформовано співтовариство по ергономіці Human Factors and Ergonomics Society.

1967 - Майкл Скрівен пише про Формативну і сумативну оцінки якості навчання студентів. Дані терміни надалі застосовувалися для різного роду оцінок, які стосуються юзабіліті.

1979 - У лабораторіях таких компаній як IBM проводиться те, що ми сьогодні називаємо сумативним юзабіліті тестуванням.

Виходить перша наукова публікація зі словом «юзабіліті» у назві: «Економічний ефект юзабіліті в інтерактивних системах» Джона Беннета *(«The Commercial Impact of Usability in Interactive Systems» by John Bennett»).*

1980 - Еріксон і Симон публікують «Verbal Reports as Data», де використовується метод «думок вголос», який надалі буде домінувати в юзабіліті-тестуванні.

Джефф Келлі, студент Альфонса Чапаніса, вперше використовує термін «Oz Paradigm», який ми сьогодні знаємо як метод «Wizard of Oz».

1981 - Альфонс Чапаніс разом з колегами публікує «Керівництво для початківця користувача комп'ютера», в якому юзабіліті описана в більшій мірі з формативної, а не сумативної точки зору, і передбачається, що для виявлення більшості проблем за допомогою юзабіліті-тестування достатньо попрацювати з 5-6 користувачами.

1982 - Домагаючись більш точних результатів ніж при вибірці в 5-6 чоловік, Джим Льюїс публікує першу роботу, що описує використання біноміального розподілу для визначення розміру вибірки, необхідного для пошуку проблемних місць в юзабіліті. Він ґрунтується на ймовірності відшукання проблеми з імовірністю «p» для певного ряду завдань і користувацької вибірки розміру «n».

Клейтон Льюїс публікує технічний звіт IBM *(«Cognitive Interface Design»),* в якому він використовує метод «думок вголос» Герба Саймона.

Професіонали, які цікавляться взаємодією людини і комп'ютера вперше зустрічаються в Гейтерсберзі *(Меріленд).* Конференція спонсорується Human Factors Society і Association for Computing Machinery *(ACM).* Пізніше в цьому році ACM формує відповідний підрозділ Special Interest Group on Computer-Human Interaction *(SIGCHI).*

Дослідники Carnegie Mellon і Xerox Park - Стюард Кард, Томас Моран і Аллен Ньюелл публікують «Психологію взаємодії людини і комп'ютера».

У Бостоні проводиться перша конференція з людино-комп'ютерного взаємодії під егідою підрозділу SIGCHI асоціації ACM.

1984 - Під час матчу Super Bowl в 1984 році Apple представляє Macintosh, виконаний в єдиному корпусі *(рис).*

Сміт і Мозер публікують рекомендації по «розробці ПЗ з інтерфейсом».

«Людський фактор» *(The Human Factor*) Гаррі Херша і Діка Рубінштейна публікується в рамках Digital Equipment Corporation і є першим книжковим варіантом опису людино-машинного взаємодії.

1985 - Дж. Гулд і Клейтон Льюїс публікують знамениту статтю «Проектування юзабіліті: ключові принципи і підходи проектувальників». Вони обговорюють акцент на користува wmre аудиторію на ранніх етапах разом з емпіричними вимірами і ітеративним проектуванням.

Річард Спенсер публікує «Комп`ютерне юзабіліті-тестування і оцінка».

1986 - Джон Брук, що працює в Digital Equipment Corporation, створює «короткий і чорновий» опитувальник з юзабіліті ПО. System Usability Scale *(SUS)* є одним з найбільш широко використовуваних опитувальників для оцінки юзабіліті систем.

Через кілька місяців після проведення щорічної конференції SIGCHI в Бостоні відкривають відповідний локальний філія.

1987 - Бен Шнейдерман публікує перше видання книги «Проектуючи користувацький інтерфейс». На сьогоднішній день вийшло вже п'яте видання цієї книги.

На основі робіт Бена Шнейдермана в HCI lab університету Меріленда публікується опитувальник за ступенем задоволеності користувачів - Questionnaire For User Interaction Satisfaction *(QUIS).*

Поява сучасної професії юзабіліті.

1988 - Джон Уайтсайд з Digital Equipment Corporation і Джон Беннет з IBM публікують ряд статей на тему «юзабіліті-інжинірингу», які акцентують увагу на визначення цілей на ранньому етапі, прототипуванні і ітеративній еволюції проекту. Джо Дюмас, один з основоположників юзабіліті, говорить про ці статті і цей період як про період зародження юзабіліті як професії.

Дон Норман публікує «Психологію повсякденних речей» яку пізніше перевидають під назвою "Дизайн повсякденних речей".

1989 - Фред Девіс публікує «модель прийняття технологій» (Technology Acceptance Model - TAM), що включає опитувальник по оцінці корисності і юзабіліті.

1990 - Шекел публікує "Людські фактори і юзабіліті", визначаючи юзабіліті як функцію ефективності та задоволення користувача *(стандарт ISO 9241 pt 11).* Незважаючи на запропоновані доповнення, ми все ще формуємо наше розуміння юзабіліті на цих аспектах.

Пітер Полсон і Клейтон Льюїс публікують ряд статей по когнітивному аналізу.

Джейкоб Нільсен і Ролф Молік публікують класичну статтю по «евристичної оцінкою користувацьких інтерфейсів», в якій описують знаменитий метод зменшення витрат в юзабіліті.

Роберт Вірзі описує три експерименти на конференції співтовариства людських факторів і ергономіки *(Human Factors and Ergonomics Society Conference)* і повторює ранню роботу Нілсена, використовуючи біноміальний розподіл для визначення розміру вибірки для юзабіліті-досліджень.

1991 - Група з CHI, що включає і Дженіс Джеймс, створює асоціацію юзабіліті професіоналів. Пізніше Дженіс і Джіні Редіш створять групу, яка цікавиться юзабіліті в співтоваристві по технічній взаємодії.

1992 - Роберт Вірзі публікує статтю про достатнє для проведення юзабіліті-тестування число суб'єктів «Refining the test phase of usability evaluation: How Many Subjects is Enough?».

У цій та попередній роботі з'ясував наступні факти:

1. Додаткові суб'єкти дають нову інформацію в ході юзабіліті-тестування з набагато меншою ймовірністю.
2. Перші 4-5 користувачів вказують приблизно на 80% проблем юзабіліті.
3. Серйозні проблеми з набагато більшою ймовірністю відшукуються першими користувачами.

Джим Льюїс публікує опитувальник по оцінці юзабіліті після дослідження системи - Post Study System Usability Questionnaire *(PSSUQ).*

1993 - Джейкоб Нільсен публікує книгу «Проектуючи юзабіліті» *(Usability Engineering).*

Джо Дюмас і Джіні Редіш публікують «Практичні поради з юзабіліті-тестування».

Юрек Кіраковскі публікує опитувальник по оцінці юзабіліті ПО - Software Usability Measurement Inventory (SUMI) в University of Cork.

Тарон Ховард запускає майданчик обговорення проблем юзабіліті, яка є популярною і сьогодні.

1994 - Рендолф Біас і Дебора Мейх'ю публікують книгу з економічного обґрунтування юзабіліті.

Джим Льюїс проводить аналіз тез Вірзі у статті про вибірках для юзабіліті-досліджень «Sample sizes for usability studies: Additional considerations» і приходить до висновку про те, що додаткові користувачі з набагато меншою ймовірністю повідомляють нову інформацію, і висновку про залежність розміру вибірки від частоти виникнення проблем. Він виявив і незалежність частоти виникнення від складності проблем.

Джеф Рубін публікує книгу по юзабіліті-тестування "Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests"

1995 - Джейкоб Нільсен публікує першу колонку з юзабіліті на useit.com, яка виходить і сьогодні.

Асоціація юзабіліті професіоналів проводить першу щорічну зустріч в Портленді.

1996 - Джон Брук публікує опитувальник для оцінки юзабіліті систем *(System Usability Scale - SUS)* після 10 років практичного використання в індустрії.

Підстава WebEx в Каліфорнії. Ця компанія розробить ПО для перегляду віддаленого робочого столу і конференц-зв'язку, яке буде використовуватися для віддаленого юзабіліті-тестування.

1998 - Юзабіліті стає стандартом *(ISO 9241 pt 11).*

Опублікована книга «Web Navigation: Designing the User Experience». Одна з книг, де використовувався термін "користувацький досвід".

Ролф Моліх проводить першу оцінку юзабіліті, залучаючи незалежні команди професіоналів, що працюють одночасно над оцінкою одного проекту *(Comparative Usability Evaluation - CUE).*

Джейкобс, Герцум і Джон, в свою чергу публікують «Ефект оцінки в юзабіліті-тестуванні: виявлення проблем і оцінка їх складності», що вперше документує вплив «оцінки».

Грей і Салзман публікують порівняльний аналіз методів оцінки юзабіліті, в якому наводиться детальна критика методологій порівняння методів зменшення витрат в юзабіліті: евристичної оцінки *(Heuristic Evaluation)* і юзабіліті-тестування *(Usability Testing).*

Агресті і Коулл публікують роботу «Adjusted-Wald binomial confidence interval» в журналі American Statistician. Запропонований інтервал добре працює в дослідженнях з невеликою вибіркою суб'єктів, які головним чином і проводяться в сфері юзабіліті.

2000 - Стів Круг публікує книгу «Do not Make me Think», яка сприяє популяризації юзабіліті-тестування, використовуючи старий добрий метод «думок вголос» 20-річної давності.

2001 - Американський національний інститут стандартів *(ANSI)* розробляє загальний формат для звітів по юзабіліті-тестів (*Common Industry Format for Usability Test Reports - CIF).*

Юзабіліті працює на відстані.

2002 - З'являються перші публікації по віддаленому юзабіліті-тестуванню, що включають і публікацію Тома Тулліса по емпіричному порівнянні лабораторного та віддаленого юзабіліті-тестування Веб-сайтів «An Empirical Comparison of Lab and Remote Usability Testing of Web Sites».

2003 - Опублікована книга «Спостерігаючи користувацький досвід».

2006 - Опубліковані методи зниження витрат на проведення юзабіліті-досліджень за рахунок автоматизації та краудсорсингу.

2008 - Том Тулліс і Білл Альберт публікують книгу по «виміру» користувацького досвіду «Measuring the User Experience» - першій книзі, присвяченій вимірюванню юзабіліті, яке за останнє десятиліття принесло свої плоди.

2010 - Опублікована книга «Beyond the Usability Lab: Conducting Large-Scale User Experience Studies», присвячена масштабним юзабіліті-дослідженням.

2012 - UPA перейменовують в UxPA *(User Experience Professionals Association).*

Саур і Льюїс публікують першу книгу по статистичному аналізу юзабіліті-даних «Quantifying the User Experience: Practical Statistics for User Research».

3.3 Мета та види Usability Testing

Для того щоб організувати результативне юзабіліті-тестування необхідно визначитися з його метою. Мета тестування являє собою очікуваний результат дослідження.

Після того, як була сформульована мета тестування, необхідно вибрати відповідний вид тестування.

Основні види юзабіліті-тестування:

- метод карткового сортування;
- метод зворотного карткового сортування;
- метод тестування сподівань;
- метод оцінки сприйняття дизайну;
- метод eye tracking;
- метод “думки вголос” ;
- конструктивна взаємодія;
- метод фокус-груп;
- експертна оцінка;
- евристична оцінка;
- паперове прототипування;

Метод карткового сортування – це класифікаційний метод, при якому користувачі сортують різні елементи веб-сайту, по декількох категоріях. Щоб провести ефективну карткову сортування необхідно грамотно вибрати список параметрів для класифікації. На кожній картці записується один параметр. Користувачів просять згрупувати найбільш

логічно дані набору карток, а так само дати назву отриманим групам. На основі результатів тестування будується навігація.

Метод зворотного карткового сортування – це верифікація прямого карткового сортування, тобто перевірка того, що спроектована структура навігації зрозуміла користувачеві. Зворотне карткове сортування можна проводити декількома способами, найпоширеніші - текстовий та графічний спосіб. У випадку з текстовим способом користувачеві дається верхній список пунктів меню та дається завдання: наприклад, знайти контактну інформацію на сайті. Респондент повинен сказати де, на його думку, розташована дана інформація, після чого дається список меню другого рівня. Графічний спосіб передбачає, що користувач розкладе меню по пунктам таким чином, який на його думку вірний.

Метод тестування сподіванньь – тест на розуміння користувацьких процесів, що відбуваються в системі. Процедура тестування: користувачеві видається роздрукований варіант інтерфейсу, задаються питання про очікувану поведінку того або іншого елемента.

В юзабіліті-тестуванні є поняття «парадокс активного користувача». Це означає, що користувач робить дію раніше закінчення питання, тому при тестуванні очікувань користувачеві необхідно надавати роздрукований варіант інтерфейсу. Метод тестування очікувань дозволяє обчислити дефекти.

Дефекти –це розбіжність очікувань користувача і запланованого поведінки системи.

Вищеперелічені методи дозволяють провести тестування на ранньому етапі, ще не маючи графічного дизайну. В результаті таких тестувань виявляють критичні помилки.

Критична помилка - це помилка респондента через нерозуміння структури інтерфейсу, яка призвела до інших помилок.

Метод оцінки сприйняття дизайну – дозволяє зрозуміти чи викликає дизайн цільові емоції. Разом з макетом інтерфейсу дизайнерові видається список прикметників (емоцій), які повинні характеризувати дизайн сайту. Як правило, цей список емоцій розробляють маркетологи. Готовий інтерфейс в декількох варіантах колірного оформлення надають респондентам і пропонують вибрати допоміжні прикметники, що характеризують загальне враження. В результаті тестування вибирають той дизайн, який викликає у користувачів необхідні емоції.

Метод Eye tracking – це тестування за допомогою системи Eye tracking, використовуваної юзабіліті-фахівцями для реєстрації руху очей. Цей метод здійснюється з використанням веб-камери і спеціального програмного забезпечення. Таким шляхом складається теплова карта інтерфейсу. Теплова карта - карта сайту, на якій відзначені найбільш гарячі ділянки - це місця, де найдовше фіксується погляд користувачів. Цей метод хоч і є ефективним, в плані розуміння які зони потрапляють до уваги користувача, а які навпаки - проблемні, однак, він не дає пояснення, чому так відбувається.

Метод "думки вголос". В тесті "думки вголос" ви попросите учасників тестування використовувати систему, безперервно думаючи вголос, - тобто, просто весь час висловлювати вголос свої думки, в той час як вони рухаються через інтерфейс користувача.

Для запуску тестування юзабіліті за методом думки вголос необхідно:

1. Набрати відповідних користувачів.
2. Дати їм відповідні завдання для виконання.
3. Замовкнути і дати можливість користувачам говорити.

Метод має безліч переваг. Найголовніше він дозволяє виявити те, що користувачі дійсно думають про ваш проект. Зокрема, ви чуєте їх помилки, які, як правило, перетворюються в практичні рекомендації по перебудові та переоформленню: наприклад, коли користувачі неправильно розуміють елементи дизайну, потрібно їх змінити. Іноді можна навіть дізнатися причини чому користувачі думають неправильно про деяких частинах інтерфейсу і чому вони вважають інші простими у використанні.

Переваги методу "думки вголос":

- ***Дешевий.*** Не потрібно ніякого спеціального устаткування, необхідно знаходиться поруч з користувачем або чути і бачити його роботу будь-яким іншим способом і приймати до відома. Збір найбільш важливих думок від декількох користувачів і виявлення проблемних місць займе близько доби.

- ***Стійкий.*** Більшість людей не можуть провести складні дослідження зважаючи на необхідність виконання належної методології, іноді помилки проведення зводять нанівець всі зусилля витрачені на саме дослідження. У разі використання методу "думки вголос", процедура є досить простою для ведучого і не повинна привнести помилок в продукт *(тільки якщо ви не намагаєтеся переконати користувача або підказати йому)*.

- ***Гнучкий.*** Ви можете використовувати метод на будь-якому етапі життєвого циклу розробки, від ранніх прототипів до в повному обсязі працюючих систем. Думки вголос особливо підходить для Agile проектів. Ви можете використовувати цей метод для оцінки будь типу користувацького інтерфейсу з будь-якою формою технології *(хоча трохи складніше використовувати метод "думок вголос" з мовними інтерфейсів)*. Сайти, програми, інтранет, споживчі товари, корпоративне програмне забезпечення або мобільний дизайн не має значення - "думки вголос" підходить для всього, так як розрахований на мислячих користувачів.

- ***Переконливий.*** Найбільш вперті і незворушні розробники, дизайнери та керівники, як правило, пом'якшуються, коли отримують дані про те, як користувачі оцінюють їхню роботу. Отримання рештою командою записів (або хоча б результатів) проведення такого дослідження це найкращий спосіб, щоб мотивувати їх звернути увагу на зручність використання.

- ***Легко навчитися.*** Навчиться проводити юзабіліті тестування методом "думки вголос" можна не більше ніж за один день, досить провести хоча б кілька подібних тестів.

Дешевизна та надійність виділяють даний метод тестування серед інших. Але з іншого боку даний метод не дасть докладних статистичних даних, якщо ви не запустите величезне, дороге дослідження.

Існуючи недоліки методу:

- ***Неприродність ситуації***. Більшість людей не сидять і не говорять самі з собою. Трудністю стає необхідність скласти монолог. На щастя, більшість користувачів швидко звикають до нової ситуації. Для додаткової мотивації та звикання користувача до ситуації ви можете показати їм коротке демонстраційне відео аналізу вголос і пояснити, що від них очікується.

- ***Фільтрація звітності.*** Люди повинні говорити речі, як тільки вони приходять на думку, а не розмірковувати про свій досвід, що забезпечує "відредагований" коментар після якогось дії або реакції. Тим не менше, більшість людей хочуть говорити "розумними словами", від чого з'являється ризик, що вони не будуть говорити, поки вони не продумали ситуацію в деталях.

- ***Зсув поведінки користувача.*** Підказки та уточнюючі питання, як правило, необхідні, але отримуючи їх від недосвідченого посередника дуже легко змінити поведінку користувача. У таких випадках, результати дослідження не представляють реальної користі. Необхідно постаратися виділити частини, які піддаються результатами впливу ведучого, і виключити їх з результатів.

- ***Не панацея.*** Метод "думки вголос" служить багатьом цілям, але не всім. Необхідно проводити дослідження і іншими методами для отримання повної картини.

Конструктивна взаємодія – різновид методу «думки вголос», який включає спільну роботу відразу двох учасників в експерименті користувачів з системою. Основна перевага в тому, що спілкуючись між собою користувачі виявляються в більш природній ситуації, ніж коли їх просто просять промовляти свої дії поодинці. Потенційною проблемою для забезпечення конструктивної взаємодії може виявитися несумісність двох респондентів та відмінність в їхніх підходах до роботи з інтерфейсом. Методика найкраще підходить для ситуацій, коли є можливість залучити до тестування велику кількість учасників, що застосовується при створенні масових продуктів.

Метод фокус-груп – має давню історію й застосовується в різних маркетингових дослідженнях ще з 50-х років XX століття. Фокус-група - це метод групового інтерв'ю. Мета фокус-групи полягає в зборі якісних даних на основі докладного обговорення учасниками конкретної проблеми *(питання)*. Метод фокус-груп дозволяє зрозуміти, що насправді люди думають і відчувають при роботі з продуктом, що проектується. У роботі кожної фокус-групи беруть участь від 4 до 12 осіб. Це - люди, попередньо відібрані відповідно до визначених критеріїв. Обговорення проблеми *(питання)* у фокус-групі проходить в умовах, близьких до природних, що виникають при роботі з продуктом. Групове обговорення фокусується на певній тематиці. Питання до учасників та план проведення фокус-групи ретельно готують заздалегідь. Групову дискусію організовує професійний інтерв'юер. Він виконує функції модератора. Його завдання - створити в групі безпечну і комфортну атмосферу, спонукати учасників вільно висловлюватися по заданій темі і вислуховувати думку всіх. В середньому участь у фокус-групі триває від 1 до 2 годин. Після завершення дискусії результати обговорення піддають ретельному аналізу, проводять узагальнення та систематизацію зібраної інформації.

У дослідженнях, пов'язаних з юзабіліті-характеристиками програмних продуктів, метод фокус-групи має сенс застосовувати лише на початкових етапах проектування, щоб виявити основні потреби та

очікування потенційних користувачів. Застосування їх для оцінки існуючих або зпроектованих інтерфейсів не виправдано і не дає позитивних результатів.

Експертна оцінка. На відміну від методик, що були описані раніше, учасниками експертної оцінки не є звичайні користувачі, а експерти в галузі юзабіліті і професійні дизайнери інтерфейсів. Вони досліджують продукт і намагаються виявити наявні, на їх погляд, проблеми. Експертна оцінка часто не може замінити собою повноцінне юзабіліті-тестування, але у неї є свої переваги. Існують різні формальні способи проведення такого роду оцінок (наприклад, уявне переміщення по інтерфейсу, коли експерт продумує всі можливі шляхи руху користувача і шукає потенційно небезпечні місця).

Евристична оцінка – один з підвидів експертної оцінки. Основною особливістю даної методики (Heuristic Evaluation) є наявність списку певних юзабіліті-принципів (або евристик). Беруть участь фахівці по черзі досліджують продукт, виділяють існуючі, на їх погляд, недоліки й класифікують ці недоліки як порушення одного або декількох з цих принципів. Список евристик заздалегідь визначається організаторами дослідження. Вони можуть варіюватися, але найбільш відомий набір з 10 правил, запропонований Я. Нільсеном.

Евристики сформульовані як універсальні для всіх програмних продуктів:

1. Видимість стану системи. Користувач завжди повинен бути проінформований про те, що відбувається в системі з допомогою наочних засобів і протягом розумного часу.

2. Відповідність між системою і аудиторією. Система повинна спілкуватися з користувачем звичною для нього мовою.

3. Свобода дій користувача. Користувачі часто вибирають ту чи іншу дію помилково, і у них повинна бути можливість без зайвих зусиль вивести систему з небажаного стану.

4. Позначення і стандарти. Користувачі не повинні сумніватися, що означають ті чи інші позначення або назви. Потрібно слідувати загальноприйнятим в даній області стандартам і домовленостям.

5. Запобігання помилок. «Краще займатися не дизайном повідомлень про помилки, а дизайном, який би запобіг виникненню помилок».

6. Розуміння краще за запам'ятовування. Всі об'єкти, функції і дії в інтерфейсі повинні знаходитися перед очима користувача. Не потрібно змушувати його тримати в пам'яті ту інформацію, що може бути в пам’яті системи. Інструкції по роботі з конкретним елементом продукту завжди повинні бути доступні за першої ж необхідності.

7. Гнучкість і ефективність використання. У продукті повинні бути присутні функції, які не очевидні для новачків і не заважають їм, але дозволяють підвищити швидкість і ефективність роботи досвідчених користувачів. Потреби обох груп повинні бути задоволені.

8. Естетичний і мінімалістичний дизайн. Будь-яка зайва інформація знижує рівень наочності інтерфейсу і ступінь ефективності роботи з ним.

9. Визначення, діагностування та виправлення помилок. Повідомлення про помилки повинні бути зрозумілі користувачеві, чітко визначати причину виниклих проблем і пропонувати конкретний спосіб їх вирішення.

10. Довідка та документація. Вся довідкова інформація повинна бути доступна, орієнтована на потреби користувача, містити конкретні кроки і мати розумний обсяг.

Евристична оцінка, яку дає кожен конкретний учасник, багато в чому залежить від його особистих характеристик та особливостей сприйняття, тому, хоча дану методику й можна використовувати навіть при наявності єдиного експерта, ефективність в такому випадку буде значно нижчою, ніж при груповій оцінці. Одна людина ніколи не зможе виявити всі наявні в інтерфейсі проблеми. Три людини – це мінімальне число учасників, при якому має сенс проводити оцінку.

Паперове прототипування – це один з різновидів юзабіліті-тестування, під час якої користувачі виконують реалістичні завдання, взаємодіючи з паперовою версією інтерфейсу, якою управляє людина, що «грає роль комп'ютера, але не пояснює, яким чином працює інтерфейс ». Паперові прототипи відносяться до групи прототипів низької точності. Виділяють чотири ролі, в яких виступають учасники тестування із застосуванням паперових прототипів. Це – користувач, консультант *(facilator),* «комп'ютер» і спостерігач.

Тестування проходить наступним чином – спочатку організатори визначають декілька типових задач, які ймовірно будуть вирішувати користувачі розроблювального ресурсу. Потім вони роблять скріншоти

(*якщо вже є які-небудь напрацювання і макети)* або просто намальовані від руки начерки різних елементів інтерфейсу *(різних сторінок у випадку з веб-сайтом)*. З ними й буде взаємодіяти користувач. Його просять виконати кілька завдань, взаємодіючи безпосередньо з паперовим прототипом. Один або декілька осіб з числа організаторів грають роль «комп'ютера», викладаючи перед користувачем аркуші паперу таким чином, щоб симулювати взаємодію з справжнім інтерфейсом, але при цьому не дають жодних пояснень про те, як він працює. Єдиний, хто спілкується з користувачем, - це консультант, який пояснює йому суть експерименту, дає завдання і відповідає на виникаючі питання. Решта учасників з числа організаторів спостерігають за подіями зі сторони. Спостерігачі можуть перебувати в приміщенні, де проводиться тестування, або стежити за ним за допомогою будь-яких технічних засобів.

Незважаючи на те, що далеко не всі типи проблем виявляються охопленими методом прототипування, у цього методу є значні переваги. Основними перевагами паперових прототипів є вкрай низькі витрати ресурсів на їх створення при досить високому рівні ефективності. Вони не вимагають особливих технічних навичок, з їх допомогою можна отримати важливу інформацію від користувачів на самій ранній стадії розробки до того, як будуть витрачені зусилля на реалізацію того чи іншого підходу. Крім того, їх простота і дешевизна дають можливість випробувати багато різних варіантів замість одного. Цікаво, що, незважаючи на описані переваги методу прототипування, багато розробників ігнорують його. Вони мислять таким чином: «Це занадто просто, щоб могло працювати». Однак для подібних ідей немає ніяких підстав. Навпаки, різні дослідження незалежно один від одного показують, що тестування з використанням прототипів високої точності не дає скільки-небудь серйозного виграшу в кількості знайдених проблем у порівнянні із застосуванням аналогів низької точності *(паперових)*. При цьому витрати на їх створення помітно вищі.

3.4 Що таке чек-лист?

Чек-лист – це документ, який описує що має бути протестовано. При цьому чек-лист може бути абсолютно різного рівня деталізації. На скільки детальним буде чек-лист залежить від вимог до звітності, рівня знання продукту співробітниками й складності продукту.

Чек-листи – один з фундаментальних інструментів тестування, які дозволяють не забувати про важливі тести, фіксувати результати роботи та відстежувати статистику про статус програмного продукту.

Іноді чек-листами називають докладні інструкції про продукт, який тестують, що містять послідовність дій, безліч деталей та ін. Це не так!

Головний принцип чек-листів полягає в тому, що кожен тестувальник по-своєму проходить їх, розширюючи тестовий набір своєю експертизою.

Навіщо потрібен чек-лист?

- ✓ не забути необхідні тести;
- ✓ для поділу завдань за рівнем кваліфікації;
- ✓ для збереження звітності та результатів тестування.

Що має бути в чек-листі?

- ✓ список перевірок *(з необхідної ступенем деталізації);*
- ✓ статус перевірок:
  - збірка, на якій проводилося тестування;
  - тестове оточення *(якщо є);*
  - тестувальник;
- ✓ результат перевірки.

Таблиця 3.1 Приклад чек-листа (частина)

№	Перевірка	Збірка	Тестове оточення	Тестувальник	Результат
1	Наявність сторінки логіна	1.0	OS X, Safari	Ольга Д.	failed - відсутня сторінка логіна тікет 89
2	Логін с правильними даними	1.1	Win7-64, Chrome	Анна П.	passed
3	Логін с неправильними даними	1.1	Win7-32, FF	Ольга Д.	failed - не виводяться помилки - виконується логін з використанням чужого паролю тікети 209, 210
4	Логін с порожніми полями	1.0	OS X, Safari	Анна П.	blocked - відсутня сторінка логіна
5	Доступ до сторінки профілю без логіна				
6	Реєстрація з правильними даними				
7	Лист підтвердження реєстрації				
...	...	...	...	...	...

3.5 Переваги чек-листів у порівнянні з тест-кейсами:

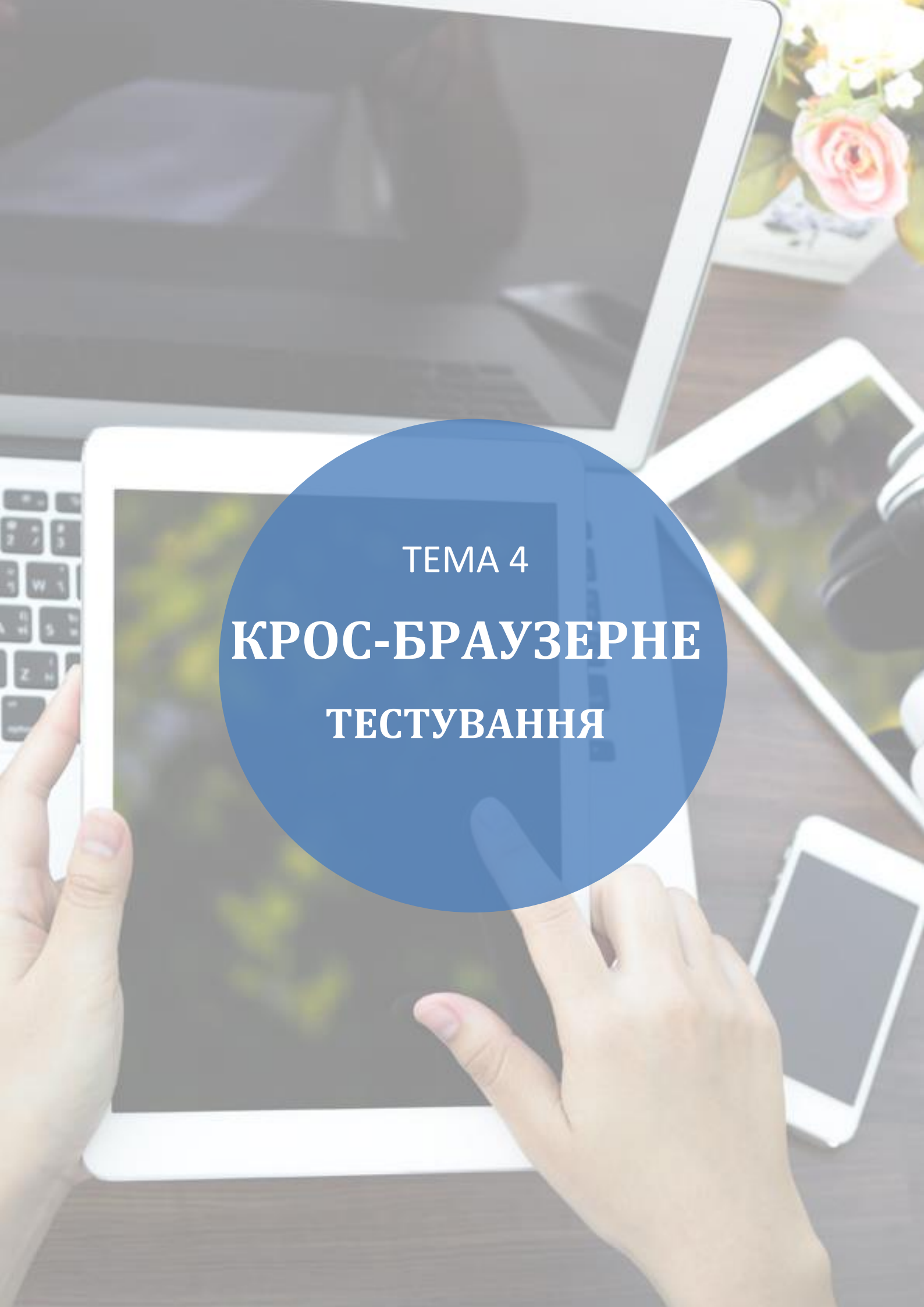
- ✓ нівелювання ефекту пестициду в регресійному тестуванні *(коли в результаті прогонки тестових сценаріїв помилки перестають виявлятися)*;
- ✓ розширення тестового покриття за рахунок відмінностей при проходженні (кожен тестувальник по своєму розуміє один й той же пункт перевірки та може доповнити щось додаткове, але не менш важливе);
- ✓ скорочення витрат на утримання та підтримку тестів: не треба писати багато букв;
- ✓ відсутність рутини, яку так не люблять кваліфіковані тестувальники;
- ✓ можливість проходити та комбінувати тести по-різному, залежно від уподобань співробітників;

При цьому, чек-листи зберігають **безліч плюсів,** за які так популярні детальні тест-кейси:

- \+ статистика: хто, коли, що проходив *(з деталізацією по збірці продукту та оточенню, на якому проводилося тестування);*
- \+ пам'ятка, яка допомагає не забути важливі тести;
- \+ можливість оцінити стан продукту, готовність до випуску;

Недоліки чек-листів:

- \- початківці тестувальники не завжди ефективно проводять тести без достатньо докладної документації;
- \- чек-листи неможливо використовувати для навчання початківців співробітників, так як в них недостатньо докладної інформації;
- \- замовнику або керівництву може бути недостатньо того рівня деталізації, який пропонують чек-листи.



ТЕМА 4

КРОС-БРАУЗЕРНЕ ТЕСТУВАННЯ

4 КРОС-БРАУЗЕРНЕ ТЕСТУВАННЯ

У даній лекції розглянуто кросбраузерне тестування, також основні браузери та відмінності між ними.

4.1 Визначення кросбраузерності

Під кросбраузерністю мають на увазі властивість сайту відображатися та працювати у всіх браузерах ідентично. Під ідентичністю розуміється відсутність розвалів верстки та здатність відображати матеріал з однаковим ступенем читабельності. Поняття «кросбраузерність» дуже часто плутають з попіксельною відповідністю, що насправді є різними поняттями. Так як веб-технології весь час розвиваються, прийнятну кросбраузерність можна забезпечити тільки для останніх версій різних браузерів. На практиці зазвичай обмежуються тільки найпопулярнішими браузерами, що суттєво може скоротити час на розробку сайту.

4.2 Історія виникнення терміну

Термін «кросбраузерність» виник наприкінці 90-х років під час так званих «браузерних війн», які проходили між двома основними браузерами того часу: Microsoft Internet Explorer та Netscape Navigator. Основною причиною цієї боротьби була жорстка конкуренція між браузерами, яка вилилася в дуже швидкі темпи розробки нових версій. У цей час браузери придбали такі компоненти, як JavaScript та безліч html-тегів. Internet Explorer почав наздоганяти браузер Netscape до 1996 року, в третій версії, що додавалася до ОС Windows 95 OSR2, придбавши підтримку скриптів та першу на ринку комерційну підтримку Cascading Style Sheets (CSS). Цей момент можна вважати початком війни браузерів, що закінчилася повним падінням Netscape та тріумфом Internet Explorer, який посів понад 95% ринку. Основним прийомом у боротьбі стало додавання специфічних, нестандартних можливостей до браузера. Найбільші відмінності виникали у підтримці JavaScript – мови сценаріїв, що додає інтерактивність документам. В результаті багато документів було «оптимізовано» для конкретного браузера та абсолютно не читалися в іншому. World Wide Web Consortium (організація, яка розробляє та впроваджує технологічні стандарти для Всесвітньої павутини) приймає безліч ретельно обговорюваних стандартів (різних версій HTML, JavaScript, CSS та ін.), але дотримання цих стандартів повністю лягає на розробників браузерів. В той час звичайним для веб-дизайнерів було вирішення питання про розміщення на своїх сайтів знаків «найкраще

переглядати цей сайт в Netscape» або «найкраще переглядати цей сайт в Internet Explorer». Ці картинки часто відносилися до певної версії браузера й часто при натисканні відправляли користувача до місця, де він міг завантажити цей браузер. Логотипи відображали різницю між стандартами, підтримуваними різними браузерами та позначали браузер, використаний при тестуванні сторінок. У відповідь на такий крок прихильники того, що всі браузери повинні підтримувати стандарти World Wide Web Consortium створили спеціальний значок «Коректно відображається в будь-якому браузері» й розміщували на перших сайтах, зверстаних з урахуванням кросбраузерності.

4.3 Основні браузери

В даний момент існує безліч браузерів, але увага буде приділено найпопулярніші з них:

- Google Chrome (48,71% ринку);
- Internet Explorer (18,91% ринку);
- Mozilla Firefox (16,53% ринку);
- Safari (10,21% ринку);
- Opera (1,63% ринку);
- Інші (4,01% ринку).

Google Chrome – браузер, що розробляється компанією Google на основі вільного браузера Chromium та движка Blink (до квітня 2013 використовувався WebKit). Google Chrome спрямований на підвищення безпеки, швидкості та стабільності. Chrome, як правило, створює для кожної вкладки окремий процес, щоб не допустити ситуації, при якій зміст однієї вкладки має можливість впливати на зміст іншої (також у випадку, якщо процес обробки змісту вкладки зависне, можна буде завершити без ризику втрати даних інших вкладок). Браузер підтримує безліч розширень, в тому числі розширення, які допомагають у тестуванні (такі як EditThisCookie для роботи з куками сайтів, PerfectPixel для перевірки відповідності сторінок макетів, Page Ruler для піскельного вимірювання елементів на сторінках та багато інших).
Google Chrome включає в себе такі інструменти для веб-майстрів, які можуть стати в нагоді й при тестуванні:

- ***веб-інспектор:*** для запуску необхідно натиснути правою кнопкою миші на будь-якому компоненті веб-сторінки й вибрати «Перегляд коду елемента», з'явиться діалогове вікно, в якому зазначаються елементи та ресурси, пов'язані з цим компонентом, а також є можливість переглянути ієрархічне представлення моделі DOM та консоль JavaScript;

- ***диспетчер завдань:*** для запуску необхідно відкрити меню «Сторінка» та вибрати «Розробникам» (в ОС Windows). Вибрати «Диспетчер завдань». Також можна скористатися швидкими клавішами ⇧ Shift + Esc. У диспетчері завдань показуються всі процеси, запущені в Google Chrome, й використовувані цими процесами ресурси (пам'ять, процесор, мережа);
- ***відладчик JavaScript:*** для запуску необхідно: відкрити меню «Сторінка» і вибрати «Розробникам» (в ОС Windows). Вибрати «Налагодження JavaScript». Відкриється командний рядок відладчика JavaScript, який можна використовувати для налагодження запущених процесів.

В табл.4.1. представлено ряд службових сторінок з різною інформацією про браузер та сторінки.

Таблиця 4.1. Службові сторінки з інформацією про браузер та сторінки

Адреси	***Значення***
chrome://about	Список службових сторінок
chrome://accessibility	Доступність
chrome://extensions	Список встановлених розширень
chrome://cache	Показати вміст кеша
chrome://crash (застаріла, не використовується) chrome://kill	Відображає сторінку з повідомленням про неполадку. У російській версії Chrome повідомлення починається з тексту «Він мертвий, Джим!», «Він полетів! Але він обіцяв повернутися ... »(в старих версіях« О ні, ми її втратили! »,« От чорт ... »)
chrome://credits	Розробки, застосовувані в Chrome
chrome://dns	Показує записи DNS
chrome://flags	Розблокувати приховані (експериментальні) можливості
chrome://flash	Перегляд інформації про Flash-плагін
chrome://histograms	Графіки різних статистичних параметрів
chrome://inducebrowsercrashforrealz	Викликати збій Google Chrome. Для перегляду розробниками процесів, що відбуваються в системі при збої
chrome://memory	Показує інформацію про використовувану додатком і вкладками пам'ять
chrome://shorthang	Викликати «зависання» вкладки. Для тестування розробниками захисту від зависання
chrome://net-internals	Інформація про з'єднання
chrome://terms	Умови надання послуг Google Chrome

Продовження таблиці 4.1.

Адреси	*Значення*
chrome://version	Номер версії і збірки. Аналог - about:
view-source:[URL]	Перегляд вихідного коду сторінки

Internet Explorer – браузер, що розроблявся корпорацією Microsoft з 1995 по 2015 рік. Входе до комплекту операційних систем сімейства Windows аж до Windows 10. Є сімейством різних версій браузера, які мають між собою суттєві відмінності. Кожна версія IE містить свої унікальні помилки, особливості відображення веб-сторінок, а також не підтримує деякі властивості CSS і Java-script. У наш час найбільш часто сайти позиціонуються як підтримуючі версії IE9 +, але для деяких окремих може бути необхідне тестування на IE8 і навіть IE7. Більш ранні версії браузера в наш час майже не використовуються.

При тестуванні в різних версіях IE слід пам'ятати про те, що в оголошених "не підтримуваними" розробником (замовником, клієнтом) версіях повинно бути повідомлення-"заглушка" при переході на відповідні сторінки або сайти. В іншому випадку відсутність такої заглушки визнається проблемою зручності використання (юзабіліті).

- ***IE7*** – сьома версія оглядача від Microsoft, випущена в жовтні 2006 року. У даній версії було вперше додано розбиття на вкладки, підтримку alpha-каналу PNG-зображень, поліпшено підтримку стандартів W3C, вбудовано механізм роботи з RSS, покращено захист від шахраїв, додано підтримку інтернаціональних доменних імен та ін. IE7 була версією за замовчуванням при поставках Windows Vista і Windows Server 2003;
- ***IE8*** – восьма версія оглядача від Microsoft, випущена в 2009 році. Дана версія вперше володіла підтримкою як 32-бітових, так і 64-бітових операційних систем і була версією за замовчуванням для Windows 7 і Windows Server 2008. Також Internet Explorer 8 є останнім браузером з гілки IE для Windows XP і Windows Server 2003. У восьмій версії IE були додані: автоматичне відновлення вкладок після аварійних ситуацій; «Прискорювачі» - швидкі команди, доступні з контекстного меню: пошук в Live Search, пошук на карті, відправка поштою, переклад іншою мовою, додавання в онлайн-закладки і ряд інших; WebSlices (веб-фрагменти) - підписка користувачів на окремі ділянки сторінок; «Розумний адресний рядок» - при введенні адреси браузер повертає результат, заснований не тільки на URL раніше відвіданого вами сайту, але і на заголовку сторінки та інших її властивостях; підсвічування доменного імені в адресному рядку; приватний режим роботи InPrivate, що дозволяє заходити на сайти, не залишаючи слідів в історії браузера; швидке

повносторінкове масштабування (управляється натисканням клавіші Ctrl і обертанням колеса миші); ***IE8*** – перша версія браузера, в яку включений режим емуляції декількох попередніх версій Internet Explorer (подання сторінок у вигляді в якому б вони відображалися на більш старих версіях браузера). Однак, емуляція в IE8 (і наступних версіях) не є повною заміною для перегляду і тестування, так як є коректною не у всіх випадках (тому слід тестувати сайти в "живих" версіях браузерів, встановлюючи їх, наприклад, на віртуальні машини);

- **IE9** – дев'ята версія оглядача від Microsoft, випущена в 2011 році. IE9 підтримує деякі компоненти ще незакінченої специфікації HTML5 (а саме аудіо, відео та canvas елемент), багато компонентів CSS 3, формат шрифтів Web Open Font Format, векторну графіку у форматі SVG, колірні профілі ICC версій 2 і 4, забезпечує більш швидку обробку JavaScript . Також, в IE9 представлено апаратне прискорення відтворення графіки за допомогою Direct2D. IE9 призначений тільки для Windows 7, Windows Server 2008 R2, Windows Vista (тільки SP2) і Windows Server 2008 (з встановленим оновленням Platform Update). З іншого боку, IE9 неповністю підтримує HTML5 (наприклад, HTML5-атрибути maxlength, placeholder, required у елементів форм, в елементі audio підтримується тільки формат mp3 та інше). Також увагу слід звертати на відображення 3D графіки (особливо з використанням програмного интерфеса WebGL).
- ***IE10*** – десята версія оглядача від Microsoft, фінальна версія випущена в 2012 році. В операційній системі Windows 8 встановлюється за замовчуванням, також доступна версія для Windows 7. Не сумісний з Windows Vista і більш ранніми версіями. У даній версії додано для підтримки більшу кількість елементів HTML5 та CSS3, однак не всі. Так само з'явилися функції перетягування (Drag and Drop) і роботи з файлами (File API). Недоліками версії все так само залишаються неповна підтримка елементів HTML5 і відсутність підтримки WebGL;
- ***IE11*** – одинадцята і, швидше за все, остання версія оглядача Internet Explorer від Microsoft (далі буде замінений на розроблювальний зараз Microsoft Edge, деталі не відомі). Поставляється з Windows 8.1 і Windows Server 2012 R2, так само доступна для Windows 7, Windows Server 2008 R2, Windows Phone 8.1. У даній версії була додана підтримка WebGL, збільшено кількість підтримуваних елементів HTML5 (проте не всі), додана можливість для додатків працювати в повноекранному режимі (завдяки реалізації FullScreen API). На відміну від інших браузерів, в IE11 URL-кодовані рядки в адресному рядку раніше відображаються в нечитабельному закодованому вигляді, ускладнюючи роботу з сайтами на кшталт Вікіпедії, де

переважна більшість URL-адрес сторінок містять нелатинські символи (що позначається при тестуванні зручності користування - юзабіліті).

Mozilla Firefox – вільний браузер на движку Gecko, розробкою та поширенням якого займається Mozilla Corporation. Третій за популярністю браузер у світі й перший серед вільного ПЗ. Коефіцієнт помилок в Safari - 0,15%, Opera - близько 0,125%, Google Chrome - 0,12% і Firefox - 0,11%. Таким чином, Firefox показав кращий результат - відсоток збоїв виявився найменшим серед найпопулярніших веб-браузерів. Майже з початку свого існування Firefox є достатньо гнучким браузером з широкими можливостями налаштування: користувач може встановлювати додаткові теми, які змінюють зовнішній вигляд програми, плагіни і розширення, що додають нову функціональність. При розробці Firefox особлива увага приділялася підтримці стандартів W3C. Інструментів для веб-розробника в базовій конфігурації Firefox немає, але їх цілком замінюють плагіни, наприклад:

- ***HTMLValidator*** – розширення для перевірки коду на відповідність стандартам W3C;
- ***View Source Chart*** – розширення для зручної візуалізації розмітки сторінки;
- ***Firebug*** – багатофункціональне розширення, що включає в себе відладчик, DOM-навігатор та JavaScript-консоль;
- ***Web Developer*** – розширення, яке пропонує безліч функцій для налагодження сторінки.

Mozilla Firefox так само надає набір вбудованих вкладок з різною інформацією, яка може бути корисною при розробці та тестуванні (табл.4.2).

Таблиця 4.2.Набір вбудованих вкладок в Mozilla Firefox

Адреси	***Значення***
about:addons	Показує сторінку з встановленими доповненнями, розширеннями і темами.
about:blocked	Показує сторінку з попередженням про те, що веб-сторінка є шахрайською і атакує комп'ютери.
about:buildconfig	Показує аргументи і параметри, використані при компіляції даної збірки.
about:cache?device=disk	Індивідуально показує вміст дискового кеша.
about:cache?device=memory	Індивідуально показує вміст кешу пам'яті.
about:cache-entry	При правильних параметрах виводить інформацію про елементи кеша.

about:certerror	Показує сторінку про те, що це з'єднання є недовірене.
about:config	Показує інтерфейс для перегляду і налаштування безлічі змінних конфігурації, багато з яких не доступні через GUI (панель налаштувань).
about:crashes	Показує список звітів про нестабільну роботу, створених утилітою Breakpad, які були відправлені на сервер Mozilla (з'явився в нічних збірках Firefox 3, починаючи з 24 січня 2008).
about:credits	Показує список тих, хто допомагав Mozilla.
about:feeds	Показує сторінку перегляду і підписки стрічки новин.
about:home	Показує домашню сторінку (FireFox 4.0 і вище).
about:jetpack	Перегляд і установка опцій розширення JetPack
about:license	Показує Mozilla Public License (і Netscape Public License) для частин програми.
about:logo	Показує логотип використовуваний в about:
about:memory	Показує інформацію про використовувану додатком пам'ять (для версій 3.6 і вище).
about:mozilla	Показує версію The Book of Mozilla
about:neterror	Показує сторінку «Помилка» Firefox не може завантажити цю сторінку з невизначеної причини.
about:neterror?e=error&u=url &d=desc	Показує сторінку помилки використовувану, коли браузер не може отримати доступ до запитуваної сторінці. (Причому після & d = може бути будь-яке Ваше слово або словосполучення)
about:newtab	Показує сторінку швидкого запуску з мініатюрами часто використовуваних сторінок сайтів (для версій 13.0 і вище).
about:permissions	Показує сторінку управління політикою зберігання даних окремих сайтів (для версій 6.0 і вище).
about:plugins	Показує встановлені доповнення.
about:privatebrowsing	Дозволяє включити режим приватного перегляду, при якому браузер не веде журнал відвідуваних адрес і не зберігає жодної інформації про сесію.
about:rights	Показує короткий опис прав користувача з посиланнями на більш докладні описи (приблизно з версії 3.0.5).
about:robots	Показує сторінку «Welcome Humans» з жартівливими цитатами. Актуально для Firefox 2 і вище.

about:sync-tabs	Показує сторінку зі списком вкладок, відкритих на інших комп'ютерах і синхронізованих за допомогою Mozilla Sync.
about:sessionrestore	Показує сторінку, яка дозволяє відновити вікна і вкладки, загублені в результаті падіння браузера (для версій 3.5 і вище).
about:support	Показує інформацію, корисну для усунення технічних проблем (для версій 3.6 і вище).

Safari – браузер, розроблений корпорацією Apple та входить до складу OS X та iOS. Safari заснований на коді движка WebKit, який вільно розповсюджується. Даний браузер створювався, коли добігав кінця термін договору Apple з Microsoft про розробку Internet Explorer для OS X. Незабаром після появи Safari робота над Internet Explorer for Mac була припинена. Деякий час випускалася версія для ОС Windows, але після версії 5.1.7, випущеної в 2012 році, випуск та підтримка браузера на Windows припинилися. На даний момент Safari для Windows не є підтримуваним сайтами браузером і визнаний застарілим. Safari для Windows була єдиною версією Safari, випущеної для будь-якої ОС, відмінної від OS X і iOS. Версій Safari для Linux і Android (так само як і для інших мобільних платформ, крім iOS) ніколи не існувало.

Safari включає в себе за замовчуванням засіб для розробників Web Inspector, консоль налагодження JavaScript, можливість вмикати / вимикати різні елементи сторінки та багато іншого входить в меню "Розробка", яке не є видимим за замовчуванням.

Для включення меню "Розробка" необхідно зробити ***наступні кроки:***

1. у браузері вибрати меню Правка → Налаштування;
2. у вікні клікнути «Додатки»;
3. поставити галочку біля «Показати меню« Розробка »у рядку меню» (рис. 4.1).

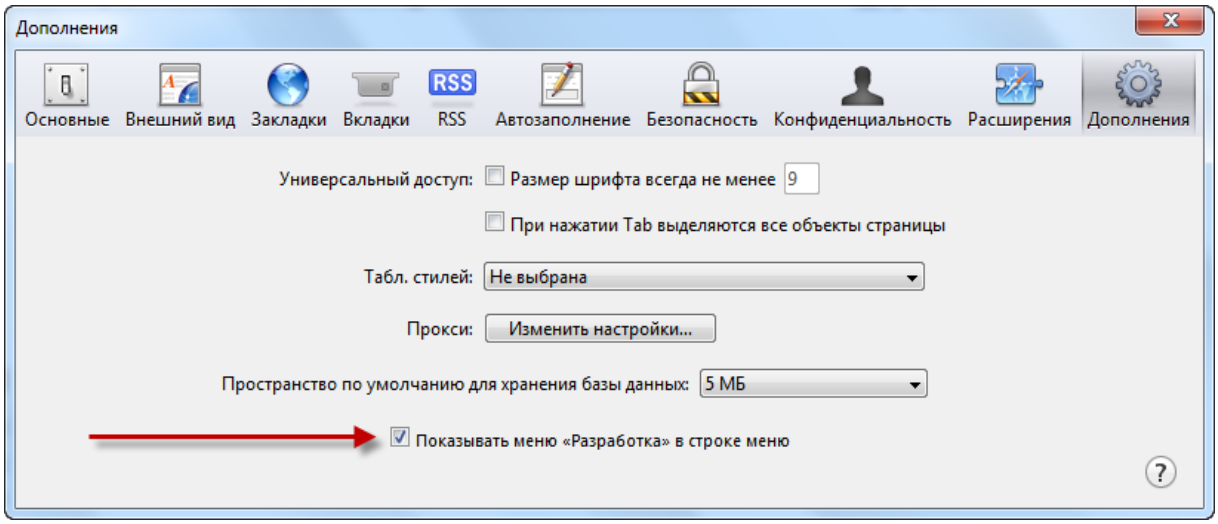


Рис. 4.1. Включення меню "Розробка" в налаштуваннях браузера

Після проходження вищевказаних кроків у рядку меню Safari з'явиться меню "Розробка" (рис. 4.2).

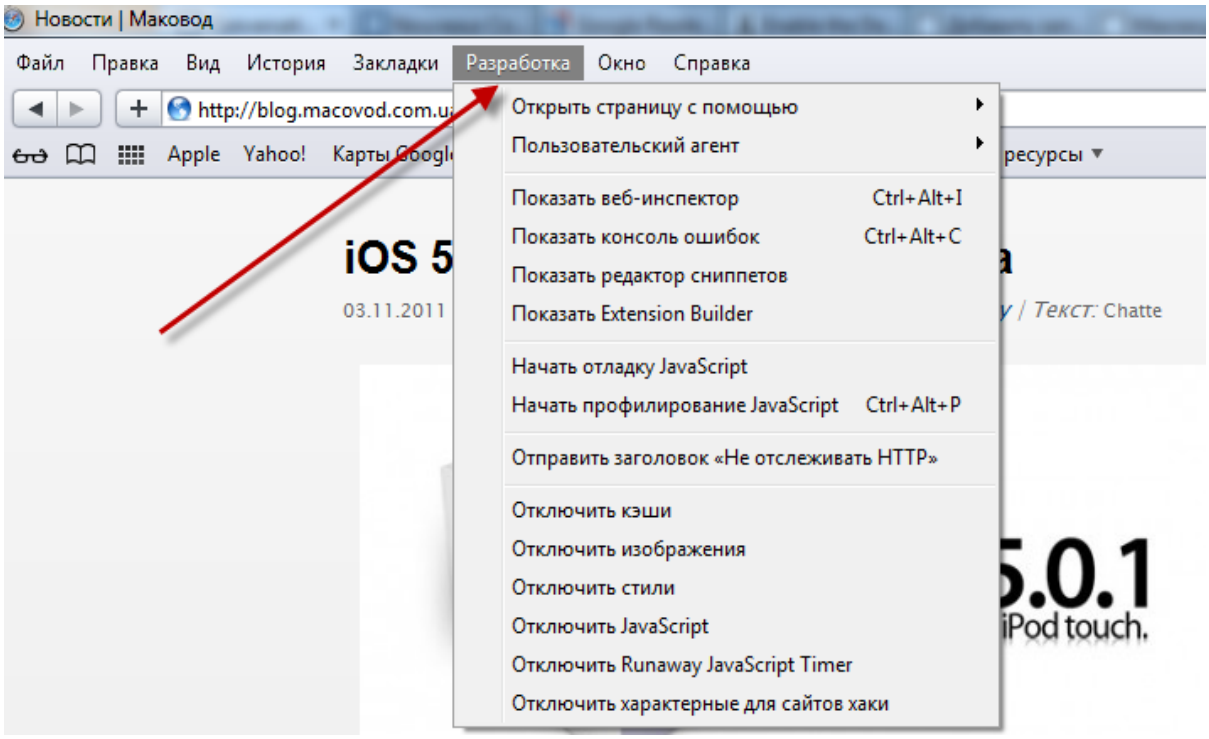


Рис. 4.2. Меню "Розробка" в рядку меню Safari

Дане меню так само дозволяє проводити емуляцію перегляду сторінок в Safari на різних пристроїв Apple (iPad, iPhone) за допомогою Safari на ПК.

Opera **–** веб-браузер та пакет прикладних програм для роботи в Інтернеті, що випускається компанією Opera Software з 1995 року. Вихідний код браузера закритий, що істотно впливає на розвиток браузера силами ентузіастів. Примітно, що в Східній Європі відсоток користувачів набагато вище середньосвітового. На початку 2013 року проект перейшов на движки WebKit та V8 в оболонці Chromium, а ще через кілька місяців – движок Blink, що є відгалуженням від WebKit.

Браузер Opera портовано під безліч операційних систем (включаючи Windows, OS X, Linux; раніше були версії для FreeBSD, Solaris). Також існують версії браузера для мобільних платформ на основі Symbian OS, MeeGo, Java, Android, Windows Mobile, bada, iOS. Крім видання Opera для персональних комп'ютерів, існують версії браузера для інших пристроїв - Opera Mobile (для КПК і смартфонів), Opera Mini (для мобільних телефонів, що підтримують Java ME), Nintendo DS Browser (для портативної ігрової консолі Nintendo DS), Internet Channel (для портативної ігрової консолі Nintendo Wii). Принцип роботи зберігається, але є деякі відмінності у функціях та інтерфейсі.

Особливу увагу при тестуванні в браузері Opera слід приділити:

- ✓ роботі редактора повідомлень в популярній CMS WordPress;
- ✓ скачуванню torrent-файлом через вбудований torrent-клієнт;
- ✓ скачуванню odt-файлів.

4.4 Тестування Debug Flash Player

Для тестування веб-додатків, реалізованих за допомогою Flash Player, необхідно встановити особливу відладочну версію (debug version), яка дозволяє при помилках в додатку отримувати повідомлення, що містять необхідні відомості для їх усунення. Існують окремі версії debug Flash Player для Internet Explorer та для інших браузерів.

Debug-версії не оновлюються автоматично, тому завжди необхідно перевіряти актуальність встановленої версії flash.

Для перевірки поточної версії кожного конкретного браузера необхідно:

1) перейти в за адресою https://helpx.adobe.com/flash-player.html (рис. 4.3);

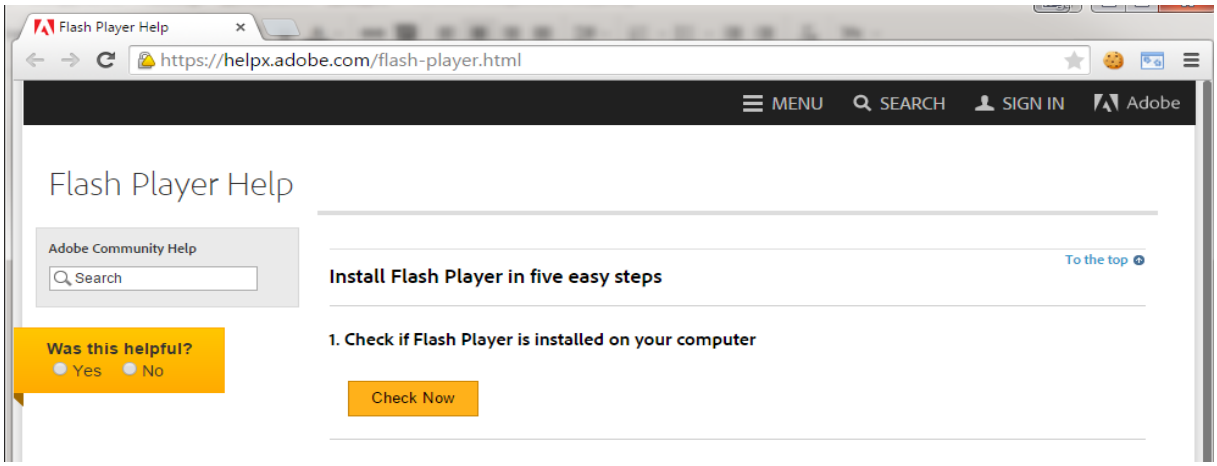


Рис. 4.3. Сторінка перевірки версії flash (виконується перевірка для браузера Google Chrome)

2) натиснути на кнопку “Check Now”;
3) переглянути результати перевірки версії flash в браузері (рис. 4.4).

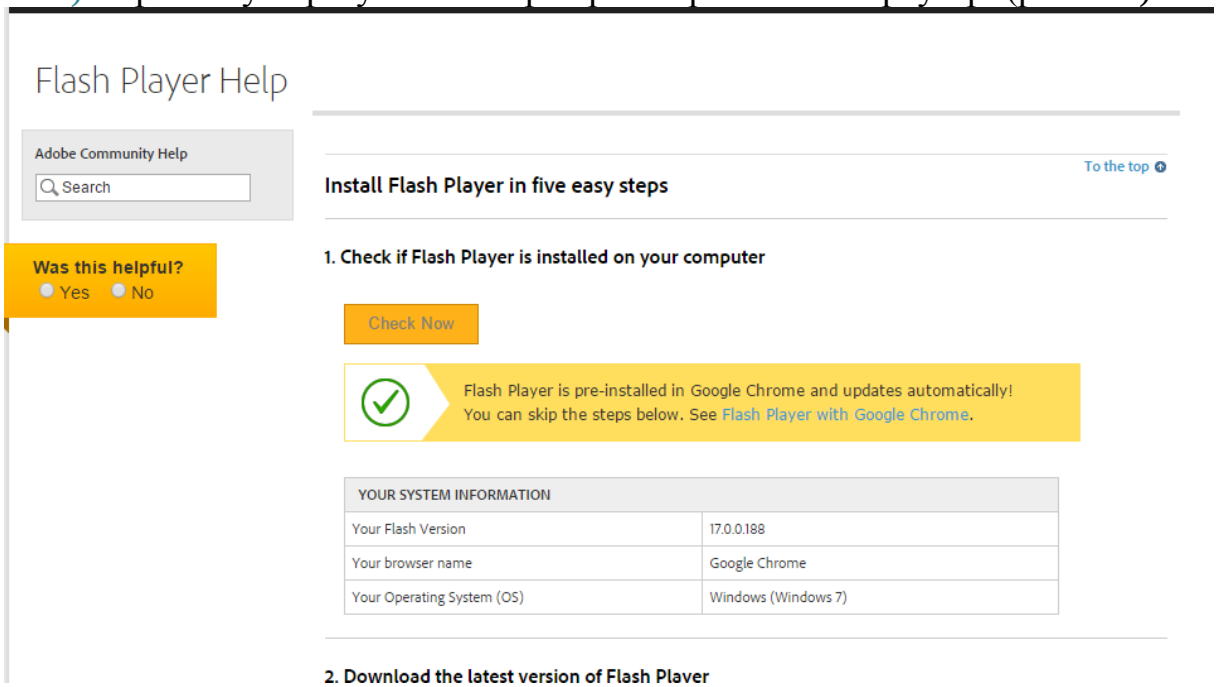


Рис. 4.4. Приклад результатів перевірки версії flash для браузера Google Chrome - встановлена остання версія

Для установки debug Flash Player необхідно:

1) завантажити інсталяційний файл з сайту adobe (зазвичай 1 для IE та ще 1 для інших браузерів);
2) встановити скачаний файл (буде потрібно закрити всі відповідні браузери на час установки);
3) відредагувати файл конфігурації flash:

- для Win7: перейти в директорію C: \ Windows \ SysWOW64 \ Macromed \ Flash (для інших ОС розташування необхідного файлу може відрізнятися). Для відображення деяких з цих папок необхідно включити відображення прихованих файлів і папок;
- відкрити файл конфігурації mm.cfg в блокноті або подібному редакторі;
- відредагувати відкритий файл (додати або змінити)
  ErrorReportingEnable = 1
  TraceOutputFileEnable = 1
  TraceOutputFileName = вказати шлях до лог-файлу (наприклад C: \ Users \ USER \ AppData \ Roaming \ Macromedia \ Flash Player \ Logs, де USER - ім'я вашого користувача - Windows 7)
  Для OS X TraceOutputFileName = / Users / your_username / Library / Preferences / Macromedia / Flash \ Player / Logs /;

4) перевірити установку на http://helpx.adobe.com/flash-player.html - при успішній установці debug-версії це буде вказуватися в результатах перевірки червоним (рис. 4.5)

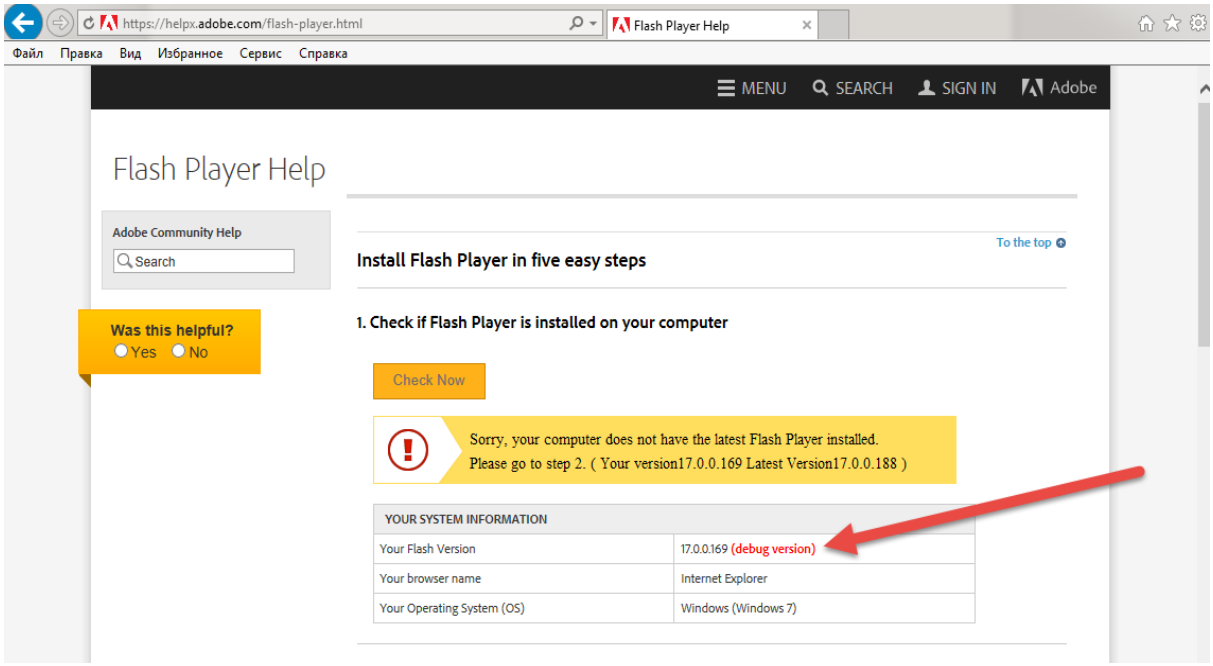


Рис. 4.5. Приклад результату перевірки flash для IE - встановлена debug-версія, але не найновіша, запропоновано оновлення

Існують деякі особливості установки debug-версії для браузера Google Chrome: для коректної роботи тестового flash необхідно пройти ще кілька кроків для ручної активації встановленого плагіна перед перевіркою поточної версії.

Таким чином, при установці debug-версії для браузера Google Chrome необхідно:

1) пройти кроки 1-3 з вищеописаної загальної інструкції;
2) у браузері Chrome перейти за адресою chrome: // plugins - відкриється список всіх встановлених плагінів (рис. 4.6);
3) знайти в списку Adobe Flash player;
4) відключити стандартну версію і включити debug-версію (рис. 4.7);
5) перевірити установку на http://helpx.adobe.com/flash-player.html;

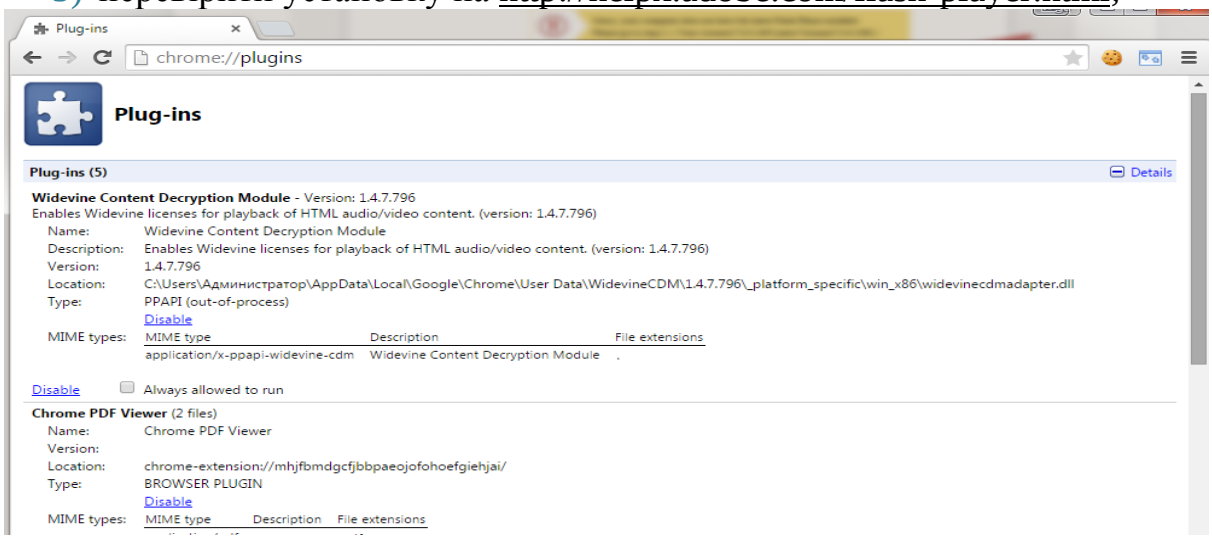


Рис. 4.6. Служебная страница chrome://plugins

Слід пам'ятати, що при оновленні debug-версії зі сторінки перевірки версії flash буде встановлена НЕ debug-версія flash. Для поновлення debug-версії на більш нову debug-версію необхідно заново завантажувати файл установки з офіційного сайту розробника (наприклад, https://www.adobe.com/support/flashplayer/downloads.html).

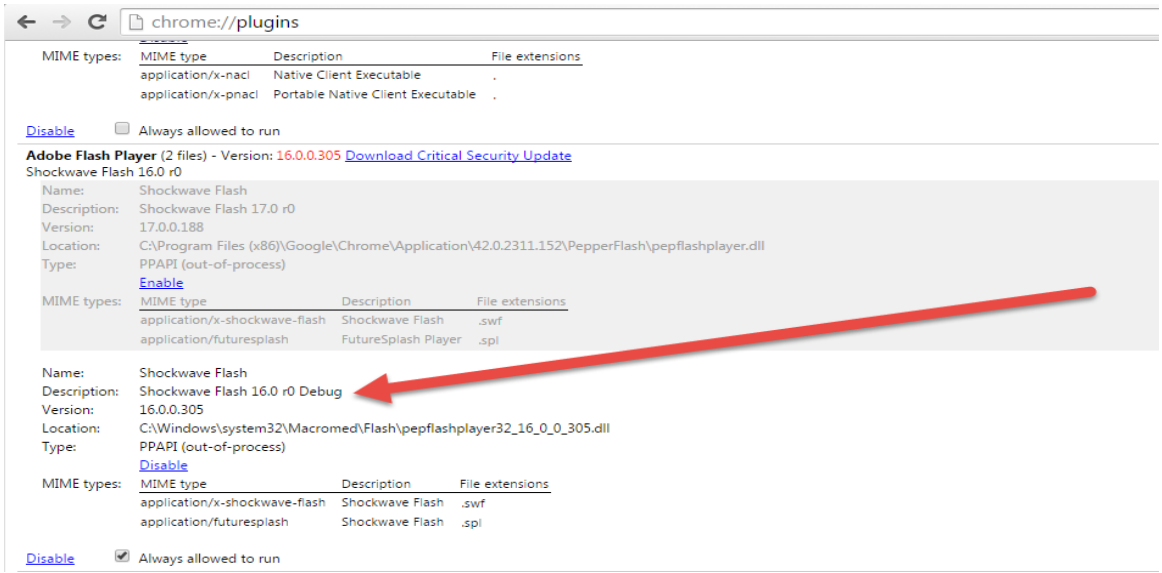


Рис. 4.7. Включена тільки debug-версія Falsh Player у вкладці chrome: // plugins

Ще одним показником правильної установки debug-версії є поява пункту Debugger в контекстному меню при натисканні правою клавішею мишки на flash контенті (рис. 4.8).

Рис. 4.8. Пункт Debugger в контекстному меню

Правильно налаштовані debug-версії при помилках в flash будуть відображати спливаючі (блокуючі екран) вікна з інформацією про причини помилок (рис. 4.9).

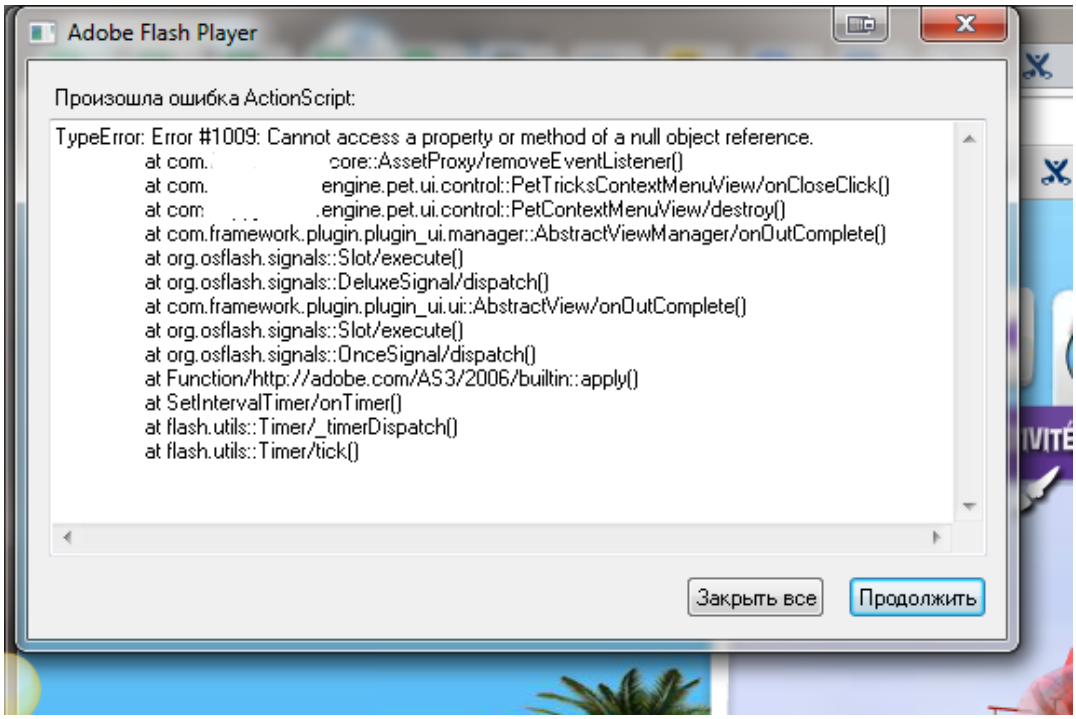
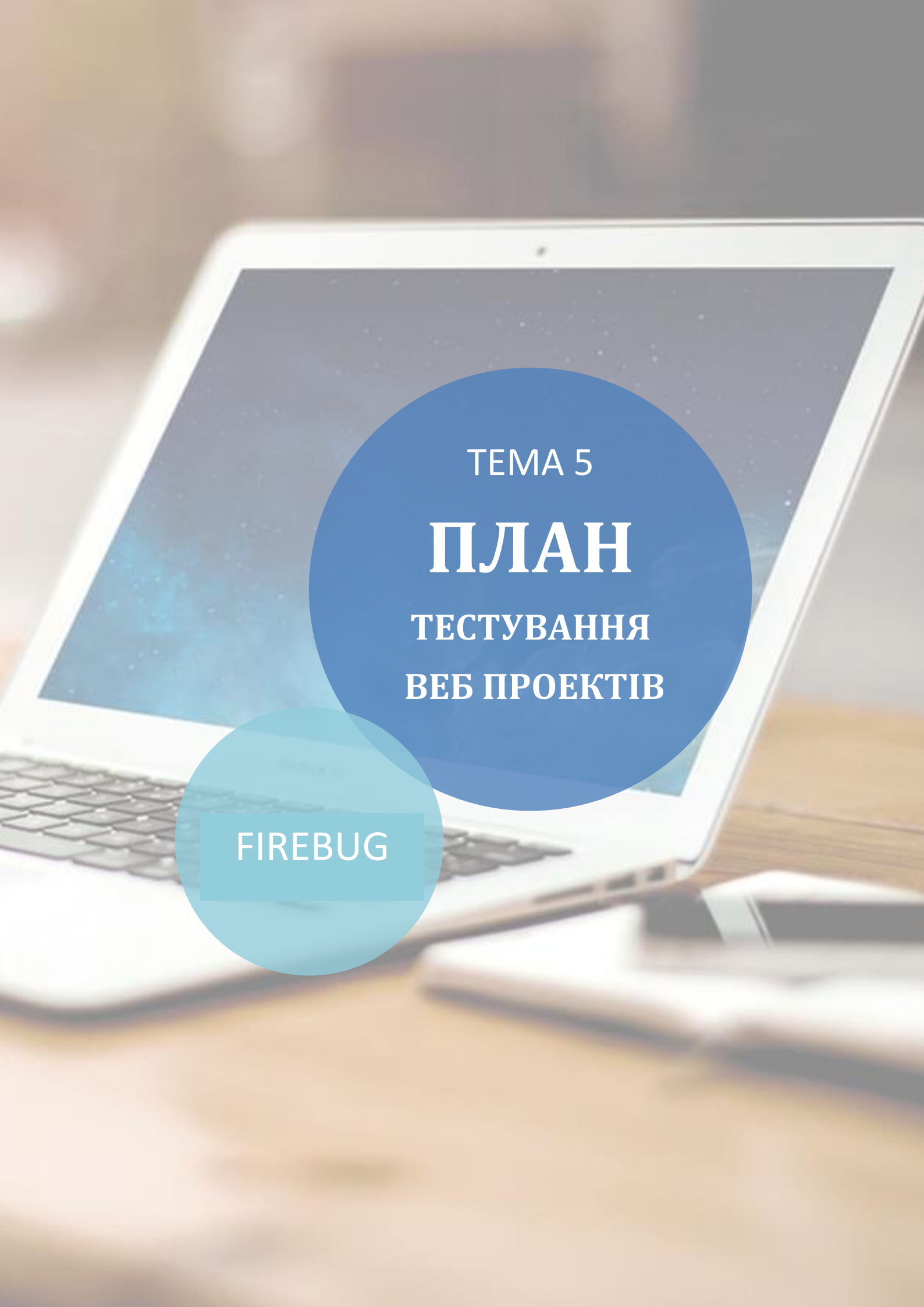


Рис.4.9. Приклад помилки у спливаючому вікні браузера з debug-версією flash

Такі ж помилки (і попередні отриманої помилку процеси) записуються в лог-файл, розташований за адресою, яка була вказана в mm.cfg файлі. Саме цей файл необхідно прикріплювати в баг-репорти про помилки flash. Однак слід пам'ятати, що лог-файл є "самоперезаписуючийся", або іншими словами він очищається відразу після кліка по будь-якій з кнопок спливаючого flash-вікна. Тому рекомендується копіювати лог-файл в будь-яку іншу робочу директорію перед продовженням роботи з браузером. Так само для коректного тестування веб-додатків, в яких використовується flash, необхідно завжди очищати кеш браузера і перезапускати його після отримання подібних помилок для того, щоб вони не вплинули на подальшу роботу тестованого програми.



ТЕМА 5

ПЛАН

ТЕСТУВАННЯ ВЕБ ПРОЕКТІВ

FIREBUG

5 ПЛАН ТЕСТУВАННЯ ВЕБ-ПРОЕКТІВ. FIREBUG

5.1 Загальне уявлення про тестування веб проектів

Тестування, як завершальний етап розробки веб-сайту, відіграє життєво важливу роль в процесі створення якісного програмного забезпечення. Чим складніший сайт, тим більше часу потрібно на його перевірку і налагодження. На жаль, існує безліч прикладів, коли розробники і замовники упускають етап тестування сайту, що практично завжди призводить до великих фінансових і тимчасових витрат в майбутньому, невдоволенню користувачів ресурсу, і, в результаті, необхідності доопрацювання (або навіть повторної розробки) ресурсу. Залежно від специфіки проекту, на тестування може виділятися до 50% загального бюджету і часових ресурсів.

Будь-який веб-додаток складається з клієнтської і серверної частини, де клієнтом виступає браузер, для відображення даних при взаємодії з сервером. Перевага використання веб-додатків в тому, що всі стандартні функції в браузері виконуються незалежно від операційної системи. Проблеми виникають від різної реалізації HTML, CSS, DOM та інших специфікацій в продукті, що призводить до проблем при разрабці та підтримці веб-додатків.

Тестування сайту вирішує кілька основних завдань:

- дає впевненість у якості кінцевого продукту, підтверджує, що всі заявлені функціональні вимоги реалізовані, веб сайт їм відповідає і не має помилок у програмному коді;
- підтверджує, що сайт здатний виконуватися у всіх заявлених режимах і на всіх підтримуваних ОС або Інтернет-браузерах коректно;
- гарантує, що дані, які зберігаються і обробляються ,надійно захищені від стороннього доступу і злому;
- дозволяє переконатися в тому, що користувач може "інтуїтивно" використовувати Ваш продукт, не плутаючись в складних переплетеннях інтерфейсів.

Як же відбувається тестування сайту?

Після завершення основних робіт зі створення програмної частини веб-ресурсу, фахівець контролю якості розробки *(іншими словами - Тестувальник)* отримує всю необхідну документацію та матеріали й приступає до тестування сайту. Для організації тестування веб-сайту

передбачена спеціально розроблена методика, відповідно до якої й здійснюється перевірка.

Отже, ***розглянемо докладніше тестування сайту по пунктах:***

1) Починається все з підготовчих робіт – тестувальник вивчає отриману документацію (аналізує функціонал по тех. завданням, вивчає кінцеві макети сайту та складає план тесту для подальшого тестування);
2) Функціональне тестування – найбільш тривалий етап перевірки ресурсу, суть процесу полягає у перевірці всього описаного функціоналу:
3) Перевірки роботи всіх обов'язкових функцій сайту;

При тестуванні обов'язкової функціональності веб-додатків основну увагу необхідно приділяти:

- інтеграції даних *(Data Integration);*
- перевірці полів даних *(Data field checks);*
- перевірці числових полів *(Numeric field checks);*
- перевірці букво-цифрових полів *(Alphanumeric field checks).*

5.1.1 Тестування інтеграції даних для веб-додатків

Інтеграція даних (Data Integration) в контексті перевірки веб-додатків відноситься до того, як введені на сайті дані сприймаються, перетворюються, відслідковуються *(моніторяться),* передаються по каналах передачі даних та зберігаються в базі даних в реальному часі для бізнес-додатків.

При введенні даних у форми, перевіряйте, яка максимальна величина текстових символів може помістится в поле для тексту, а відповідно, й передатися на сервер або в базу даних. Наприклад, у текстове поле можна ввести тільки 100 символів й при введенні тексту більшого розміру, частина даних, починаючи з 101 символу, буде загублена.

Якщо ввести в числове поле веб-форми від'ємне значення *(-100),* то воно може бути успішно прийнято, оброблено веб-додатком та передано через сервер в базу даних. У базі даних це числове значення може бути збережено як позитивне *(100),* а не як негативне число, так як в базі даних може бути задекларовано зберігання тільки позитивних значень для цього типу змінної.

При роботі з цілими числами, введення в поле даних дійсного числа (наприклад, 2,93), може призвести до помилки, яка пов'язана з округленням числових значень, і як результат, в базу даних може потрапити число 2 або 3.

5.1.2 Тестування полів даних для веб-додатків

При перевірці полів даних *(Data field checks)* в веб додатках необхідно переконатися, що дані які вводяться, відповідають необхідному для цього поля типу даних, і дані в поле можна вводити тільки певної довжини.

Назва полів для веб-форми повинні мати правильний синтаксис тобто – не мати граматичних помилок. Це означатиме, що при перевірці веб сторінки або веб-форми треба перевірити назви полів або текст,що відображається , на відсутність помилок. Також, на веб сторінці може бути присутня інформація про авторське право *(Copyright)* автора або компанії в такому форматі: © 2014 Azarskyy. Дата повинна відповідати поточному року розробки програми. Може бути і такий формат: © 1999-2014 Company Name.

При введенні даних в поля повинні правильно перевірятися високосні роки та числа 28-29 лютого, а також 30-31.

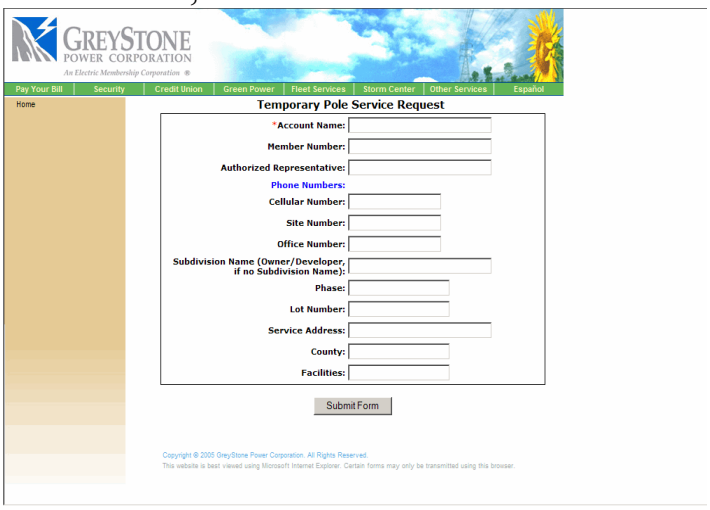


Рис. 5.1. Приклад полів даних

5.1.3 Тестування числових полів для веб-додатків

Перевірка числових полів *(Numeric fields checks)* виконується для перевірки того, що:

- найбільше таі найменше значення оброблені коректно;
- порожні числові поля в додатку повинні оброблятися у вигляді появи підказки або повідомлення про помилку;
- позитивні (+) та негативні (-) числові значення обробляються коректно;
- верхні та нижні значення обробляються коректно;
- забезпечується перевірка при діленні на нуль (0).

5.1.4 Тестування букво-цифрових полів для веб-додатків

a) ***Перевірка букво-цифрових полів (Alphanumeric field checks) виконується для перевірки того, як обробляються:***

- дані з пропуском та без пропуску*(пробіл);*
- найбільше та найменше значення;
- правильні букви для поля веб-форми;
- неправильні букви для поля веб-форми;
- неправильні символи для поля веб-форми;
- дані з пропуском в першій позиції;
- дані з пропуском в останній позиції;
- мінімальна довжина допустимих символів для введення;
- максимальна довжина допустимих символів для введення;
- мінімальна кількість слів допустимих для введення;
- максимальна кількість слів допустимих для введення;
- слова з "чорного списку";
- звичайний текст;
- регулярні вирази;

b) **Тестування працездатності користувальницьких форм на сайті *(наприклад, зворотній зв'язок, додавання коментаря в блог);***

Форми *(Forms)* призначаються для обміну даними між клієнтською частиною *(браузером)* і веб-сервером *(Web server*) або поштовим сервером (*Mail server).* Форма може містити інформацію у вигляді тексту, розмітку, а також спеціальні елементи, які називають елементи управління *(Control elements*). При роботі з формою, користувач заповнює обов'язкові *(Mandatory)* і необов'язкові *(Optional)* елементи, наприклад, вводить текст, вибирає елементи меню, вводить пароль й так далі.

Прикладами форм можуть бути, наприклад, форми при реєстрації користувача, які можуть включати такі ***елементи управління (рис.5.2):***

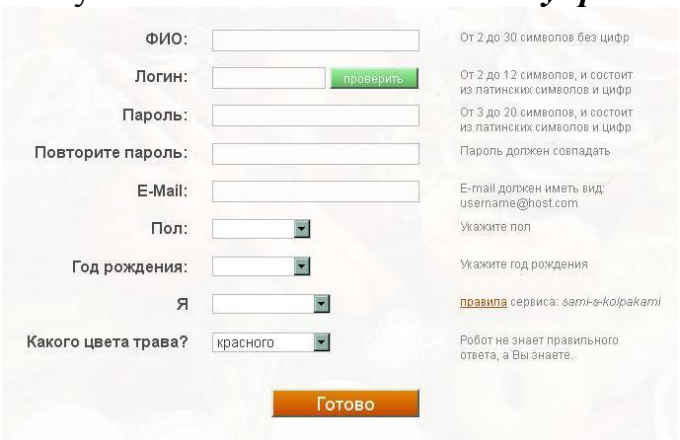


Рис.5.2. Приклад форми реєстрації

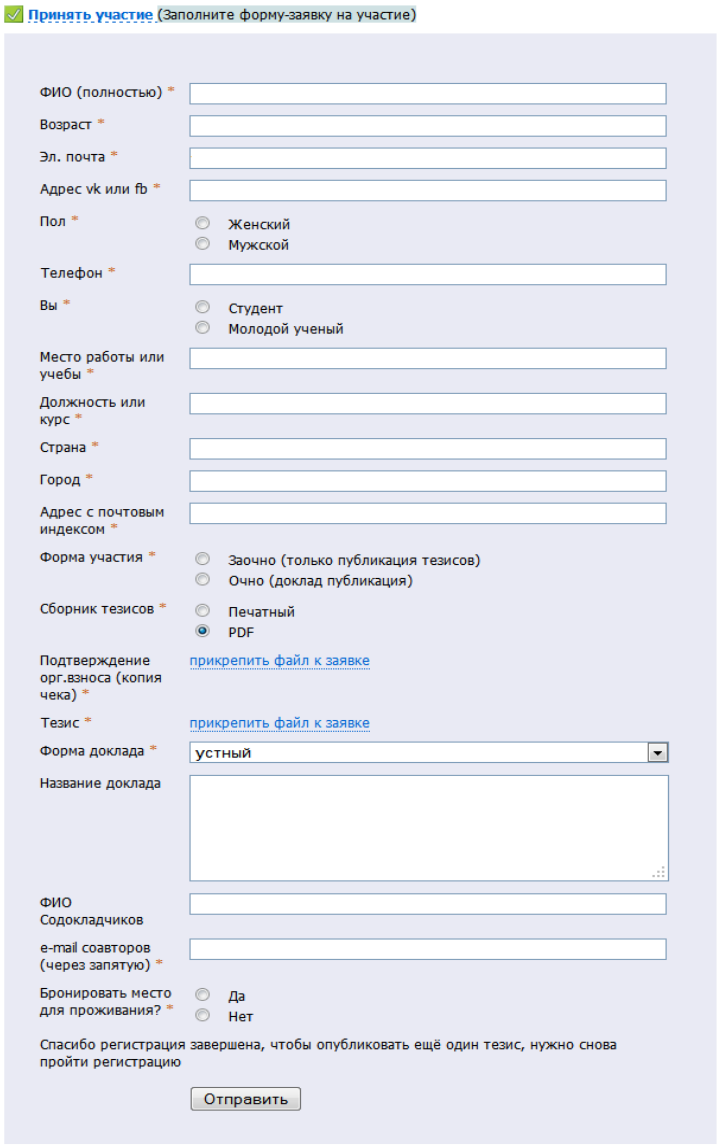


Рис. 5.3. Приклад форми реєстрації

1) ***Кнопки (Buttons).***

Кнопки бувають різних типів:

- тип кнопки відправити *(Submit)* відповідає за відправку форми;
- активізація кнопки з типом скидання (Reset) призводить до скидання на формі всіх елементів управління в їх початкові значення;
- кнопки з типом натиснути *(Push)* не мають поведінки за замовчуванням, і кожна така кнопка може мати свій клієнтський сценарій.

2) ***Прапорці*** – чекбокси *(Checkboxes)*, можуть бути переключені користувачем у стан включити (*On*) / виключити *(Off*), і тільки включені прапорці беруться до уваги при відправці форми.

3) ***Перемикачі*** *(Radio buttons)* працюють, як і прапорці, тільки при виборі одного із значень в стан включити, інші елементи перемикаються в стан вимкнути.

4) ***Введення тексту*** *(Text input)* призначений для введення тексту в однорядковому *(Singleline)* і багаторядковому *(Multi-line)* режимах.

5) ***Вибір файлу*** *(File select)* дає користувачеві можливість вибрати файл, який буде відправлений з формою, наприклад Ваше прикріплене резюме на сайті роботодавця.

6) ***Пароль*** (*Password*) – в цьому елементі вводиться текст таким чином, щоб символи приховувалися від "випадкових" спостерігачів, наприклад, за допомогою ряду зірочок *(Series of asterisks)* - **********.

При заповненні і відправленні форми необхідно, щоб на формі була обробка таких подій:

- прийняття допустимих вхідних значень;
- блокування неприпустимих вхідних значень;
- обов'язкові поля для введення повинні бути заповнені;
- необов'язкові поля у формі можуть залишатися незаповненими;
- перемикачі дозволяють вибрати тільки одне значення;
- прапорці / чекбокси можуть бути в станах включити / виключити;
- чи з'являється смуга прокрутки (*Scrollbar)* при необхідності?
- послідовне перемикання між елементами форми можливо за допомогою клавіші Tab на клавіатурі;
- чи всі дані в списках розташовані в хронологічному порядку?
- чи розташовані всі елементи форми відповідно до затвердженого дизайну?

c) ***Перевірка роботи пошуку (включаючи релевантність результатів);***
Шукане значення:
- Абревіатури.
- Одне неповне слово.
- Словосполучення.
- Кілька неповних слів.
- Спецсимволи.
- Цифри (позитивні, негативні, нуль, цілі, дробові, спец. Значення (типу "Infinity")).

- Регістр введення.
- Різні мови / розкладки.
- Пусте значення.
- Морфологія.

Додатково:

- Аналіз граничних значень і класи еквівалентності для довжини.
- Пошук по всіх полях.
- Пошук по деяких полях.
- Пошук без заповнення обов'язкових полів.
- Пошук по обов'язковим полях.
- Врахування різноманітних параметрів.
- Варіанти введення:
- З клавіатури.
- З буфера обміну (через контекстне меню або поєднання гарячих клавіш).
- З екранної клавіатури.

Очищення поля:

- Через клавішу <Esc>.
- Через спец. клавішу додатки.
- Через клавіші <Delete> і <Backspace>

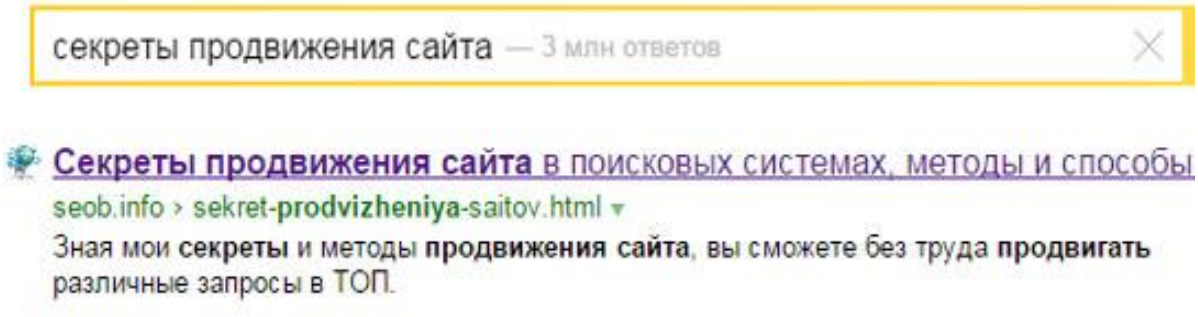


Рис. 5.4 Приклад пошуку і результат

d) Перевірки гіперпосилань, пошук неробочих посилань;

Посилання *(Links)* або гіперпосилання *(Hyperlinks)* дозволяють в електронному документі активувати або відображати інший документ або програму. Кожне посилання на сайті повинна бути в певному місці, її назва повинна відповідати вимогам і бути без граматичних помилок. Також, вона правильно повинна виконувати покладені на неї навігаційні властивості - вести на певну іншу сторінку або файл. На веб-сторінці сайту також необхідно перевірити, чи працює навігація не тільки на внутрішні ресурси сайту, але і на інші сайти. Бувають такі помилки, коли існує веб-сторінка з певною адресою і на неї можна зайти за прямим посиланням *(Direct Link)*, але в теж час, на неї не існує посилань з головної сторінки або з інших сторінок сайту - таку сторінку називають сторінка-сирота *(Orphan page)*.

Перевіряються на сторінці також всі поштові посилання *(Mail to links) і* чи досягає відправлений з сайту електронний лист адресата.

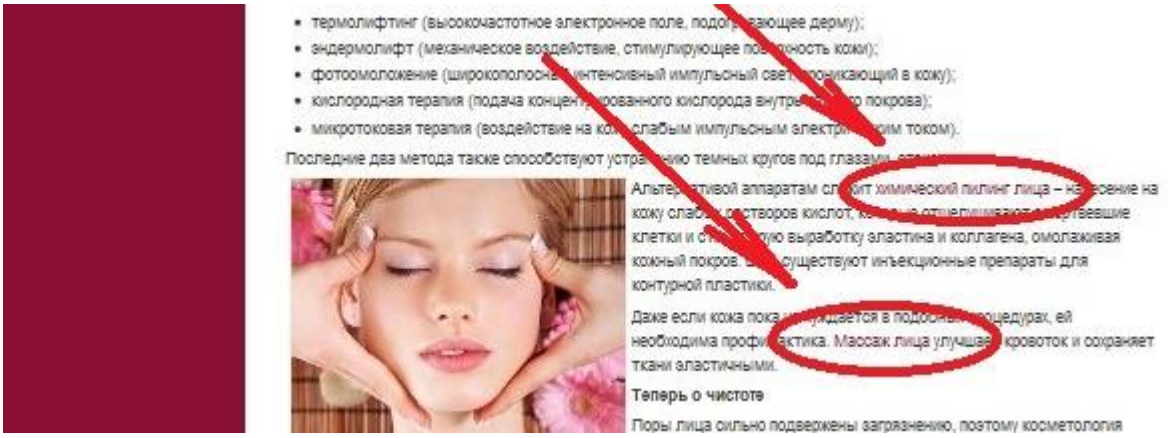


Рис. 5.5 Приклад посилань на сайті

Коли існує проблема при роботі з посиланнями, то програма для навігації та перегляду веб-сторінок – браузер *(Browser)* – обробляє цю подію та показує повідомлення про помилку у вигляді коду стану HTTP *(HTTP status code).*

Коди помилок бувають (рис.5.6):

- 1xx Інформаційні (*1xx Informational);*
- 2xx Успішно *(2xx Success);*
- 3xx Перенаправлення *(3xx Redirection);*
- 4xx Помилки клієнта *(4xx Client Error);*
- 5xx Помилки сервера *(5xx Server Error);*

Рис. 5.6 Приклад помилки 404

е) Перевірки загрузки файлів на сервер

- Відкриття вікна провідника для вибору файлу для завантаження;
- Можливість завантаження файлу з локальних дисків комп'ютера;
- Можливість завантаження файлу із знімних дисків;

- Можливість завантаження декількох файлів;
- Завантаження файлів різних форматів і розмірів;
- Завантаження файлів, що містять у назві спеціальні символи *(! * <>? / \ Т т.п.)*

f) Перегляд на відповідність вмісту сторінок сайту вихідного контенту, наданого замовником.

Контент –це все, що присутнє на сайті: текстовий зміст, зображення, аудіо, відео та інші файли будь-яких розширень. Для користувача контент може бути представлений для перегляду, скачки або інших дій. Все це перевіряється порівнянням з матеріалами наданими замовником.

g) Тестування взаємодії веб-додатків з сервером

Для коректної роботи веб-додатки на стороні сервера перевіряють конфігураційні налаштування, які повинні відповідати вимогам до програмного продукту, що розробляється.

При перевірці веб-додатків необхідно враховувати, як:

- поведе додаток, якщо сервер "впаде";
- чи відреагує веб-додаток після спрацьовування механізму перезапуску сервера;
- чи відреагує веб-додаток після неспрацьовування механізму перезапуску сервера;
- стабільно відновлює свою роботу веб-додаток при періодичному перезапуску сервера;
- повинна бути оброблена транзакція при передачі даних, якщо був збій при з'єднанні з сервером;
- обробити відключення від інтернету;
- реалізована інтелектуальна обробка помилок та наскільки інформативні для користувача повідомлення про помилки.

5.1.5 Тестування верстки веб-сайтів

Тестування верстки – при перевірці верстки першим ділом тестувальник перевіряє розташування елементів, відповідність їх позицій наданим макетам, а так само перевіряє оптимізацію зображень і графіки. Далі здійснюється перевірка валідності коду. У процесі верстки важливо дотримуватися коректної ієрархії об'єктів, і важливо упевнитися в її валідності за фактом завершення робіт. Браузери, незважаючи на явно невірний код, в будь-якому випадку постараються відобразити веб-сторінку. Але оскільки не існує єдиного регламенту про те, як же має бути показаний «кривий» документ, кожен браузер намагається зробити це по-

своєму. А це в свою чергу призводить до того, що один і той же документ може виглядати по-різному в різних браузерах. Виправлення явних промахів і систематизація коду призводить, як правило, до стабільного результату. Завершивши перевірку на валідність, фахівець приступає до перевірки на кроссбраузерність, тобто перевіряє працездатність сайту в різних браузерах, а так само при різних параметрах настройки екрану.

Навіщо перевіряти сайт на кроссбраузерність? На сьогоднішній день існує ряд найбільш популярних веб-браузерів, таких як Google Chrome, Safari, Mozilla Firefox, Internet Explorer і Opera. Кожен з них дотримується загальних рекомендацій візуалізації розмітки сторінки, проте в той же час кожен обробляє код відповідно до особливості власного движка. Ускладнюється все тим, що досить часто з'являються нові версії браузерів, і ресурс, який відмінно виглядає, наприклад, в IE9, не обов'язково буде виглядати коректно в IE7 або IE8. Тому в процесі тестування враховується перелік браузерів, підтримка яких обумовлювалася із замовником на ранніх етапах обговорення проекту. Етап перевірки сайту на кроссбраузерність при різних розширеннях досить довгий за часом, але результат того вартий - з вашим сайтом зможе ознайомитися кожен представник цільової аудиторії.

5.1.6 Usability тестування

Usability тестування – проводиться для оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників і аналіз отриманих результатів.

Незважаючи на той факт, що опрацювання зручності використання ресурсу здійснюється в процесі складання технічного завдання, розробки макетів, бувають ситуації, коли отриманий результат не є оптирисьним. Хоча таке і відбувається досить рідко, оптирисьне рішення в даному випадку - внести зміни до реалізованого товару.

Тестування проводиться за участю кількох людей з цільової аудиторії, так званих респондентів. Для проведення тестування досить 4-6 чоловік. Існує правило 80/20, яке свідчить, що 20% користувачів дають 80% результату. Тому така кількість респондентів максирисьно ефективна з точки зору економії часу і витрат.

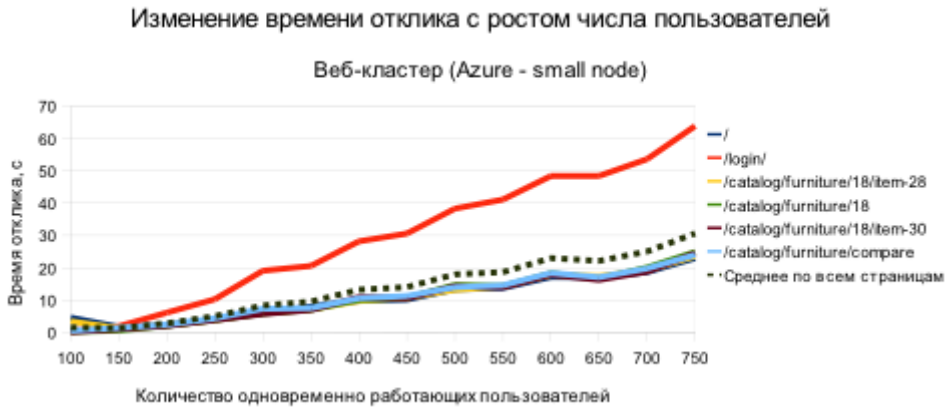
5.1.7 Тестування безпеки

Тестування безпеки. На даній стадії тестування фахівець перевіряє – чи немає у користувачів доступу до службових / закритих сторінок, а так само проводить перевірку захисту всіх критично важливих сторінок *(наприклад, розділу адміністрування сайту)* від зовнішнього впливу.

5.1.8 Тестування продуктивності сайту

Тестування продуктивності сайту – проводиться з метою визначення швидкодії сайту або частини під певним навантаженням. Тестування продуктивності включає в наступні ***види тестування:***

1. ***Навантажувальне тестування*** – найпростіша форма тестування продуктивності. Тестування навантаження зазвичай проводиться для того, щоб оцінити поведінку сайту (або програми) під заданим очікуваним навантаженням. Цім навантаженням може бути, наприклад, очікувана кількість одночасно працюючих користувачів на сайті, що вчиняють задане число транзакцій за інтервал часу. Такий тип тестування зазвичай дозволяє отримати час відгуку всіх найважливіших бізнес-функцій.



2. ***Тестування швидкодії*** – перевірка швидкості завантаження сайту для визначення швидкості відпрацювання скриптів, завантаження зображень і контенту. Цей тест проводиться з метою оптимізації процесу завантаження сайту, а так само визначення оптимальності налаштувань сервера.

Залежно від спрямованості тестування, перевіряється та чи інша особливість веб-сайту: корпоративного сайту, інтернет-магазину, сайту-візитки. Як правило, процес тестування документується у вигляді тестового плану та тест-кейсів. Тестовий план описує стратегію тестування, методи та засоби тестування, порядок та інші особливості. Тест-кейси описують послідовні покрокові операції перевірки функціоналу програми або веб-сайту.

Обробка помилок

Протягом усього етапу тестування, фахівець створює та доповнює звіт про виявлені помилки. Даний звіт передається учасникам проекту, після чого керівник проекту визначає відповідального за виправлення кожної з помилок *(частина обов'язків з часом розподіляється самим тестувальником*). Далі визначається графік виправлення помилок, після

чого проводиться повторне тестування з метою контролю якості виправлення помилок, а так само відсутності нових. Дана процедура повторюється поки сайт не буде відповідати специфікаціям тех. завдання. Саме тому тестування – настільки довгий процес.

По завершенню тестування проект готовий до розміщення на сервері і повноцінної роботи, ефективно і стабільно виконуючи покладені на нього бізнес-функції. Саме тестування є гарантом спокійного сну як для замовника, так і для команди розробників веб-сайту.

5.2 Firebug

Firebug – розширення для браузера Firefox, що є консоллю, і DOM-інспектором JavaScript, DHTML, CSS, XMLHttpRequest.

Основні функції Firebug:

- перегляд HTML, зміна стилів і верстки в режимі реального часу;
- перегляд CSS-метрик;
- аналіз використання та продуктивності мережі з великою точністю;
- налагодження та профілювання JavaScript;
- зручна консоль, яка дає Вам знати і дає детальну і корисну інформацію про помилки в JavaScript, CSS і XML;
- набір функцій логування сценаріїв JavaScript;
- інструменти для перегляду DOM сайту і багато-багато іншого.

Розширення Firebug безкоштовне, тому без зусиль можно знайти файл для установкуи в інтернеті. Після установки необхідно перезавантажити браузер. Консоль Firebug можна викликати як за допомогою клавіші F12, так й натиснувши по іконці програми у вікні браузера (рис.5.7).

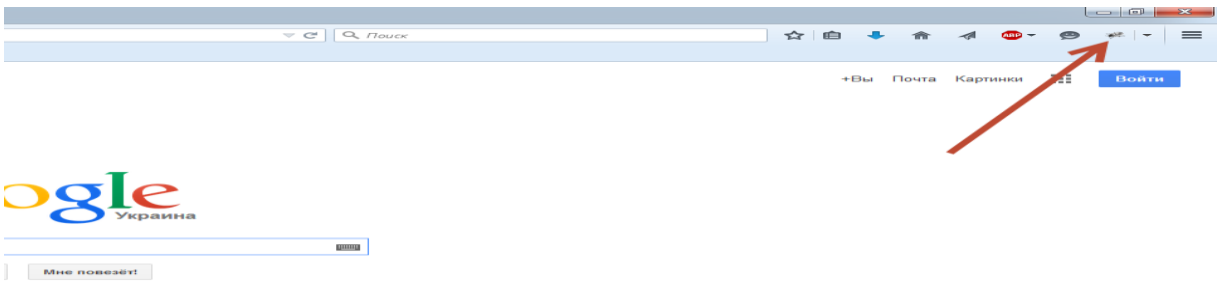


Рис. 5.7. Виклик консолі Firebug

Нижче наведено 10 причин, за якими варто використовувати Firebug.

1. *Console (Консоль)*

Першим, що можна помітити при відкритті Firebug *(з панелі статусу або використовуючи комбінацію клавіш ctrl + F12)* буде консольна панель – Console. Після швидкого перегляду, можна припустити, що це альтернативна версія консолі помилок (*Ctrl + Shift + J*). Основні можливості зводяться до двох наступних:

- реєстрація *(логування)* помилок, попереджень і повідомлень;
- можливість працювати з Javascript кодом;

Але Firebug розширює функціональність браузера Firefox, тому може виконувати набагато більше, а саме:

- реєстрація помилок Javascript, CSS, XML, XMLHttpRequest *(AJAX)* і Chrome *(ядро Firefox);*
- виконання Javascript коду на поточній веб-сторінці;
- передає додатковий Javascript об'єкт для управління *(в консоль)*.

Приклади роботи об'єкта console. Припустимо, дано наступний HTML файл:

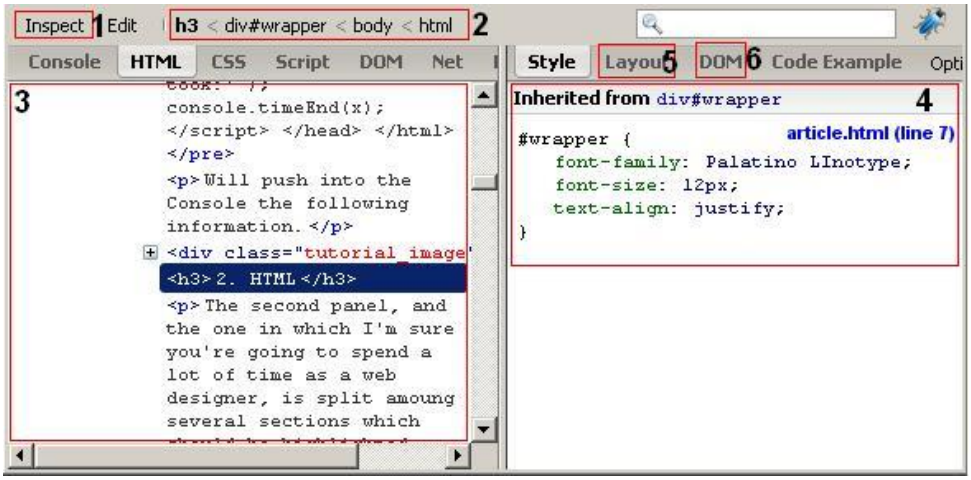
```
<Html>
  <Head>
      <Script type = "text / javascript">
          console.time (1);
          console.log ('the script section has started executing');
          console.warn ('warning message');
          console.error ('error message');
          console.info ('info message');
          console.log (
              'Finishing script execution \ n',
              'Execution took:'
          );
          console.timeEnd (1);
      </ Script>
  </ Head>
</ Html>
```

На рис.5.8. представлений результат вище зазначеного коду:

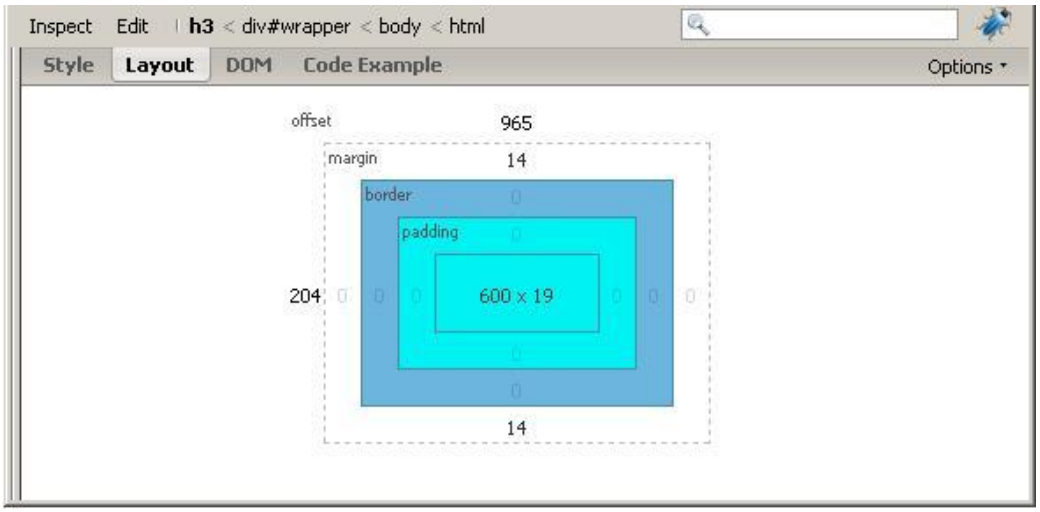
Рис.5.8. Результат при використанні HTML файл

2. HTML

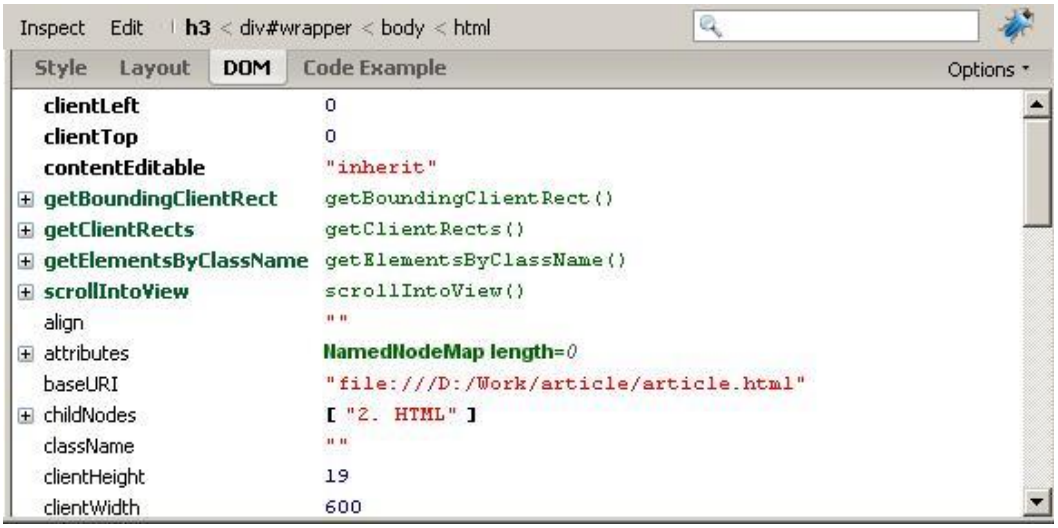
Наступна панель розбита на кілька секцій, які розглянемо нижче.



1. кнопка є еквівалентом "Inspect Element" в контекстному меню на сторінці. Крім того, що корисна для швидкого пошуку елемента на сторінці, а також виділяє вибраний елемент.
2. показана ієрархічна структура обраного елемента та можливість виконання серії дій (для кожного окремого компонента цієї структури), наприклад:
   - копіювання вмісту HTML коду;
   - створення XPath виразів;
   - приєднання спостереження за подіями (і логування в консольній панелі Console);
   - видалення елемента;
   - редагування елемента і його дочірніх вузлів;
   - переміщення елемента у вкладку DOM для відстеження;
3. головне вікно панелі: корисно *(використовується useful)* для переміщення по HTML документу, швидкої зміни коду та виявлення помилок в коді *(наприклад, раннє закриття div елементу)*. Контекстне меню пропонує той же функціональний набір як й в другій вкладці.
4. обчислений стиль поточної сторінки або відображуваного елемента. Можливість на ходу міняти стилі і перевіряти спадкування стилів CSS є її найбільш значущою рисою.
5. можна досліджувати box-модель елемента: розмір вмісту *(content size)*, заповнення *(padding)*, зміщення *(offsets)*, відстань від кромки *(margins)* і межі *(borders)*.

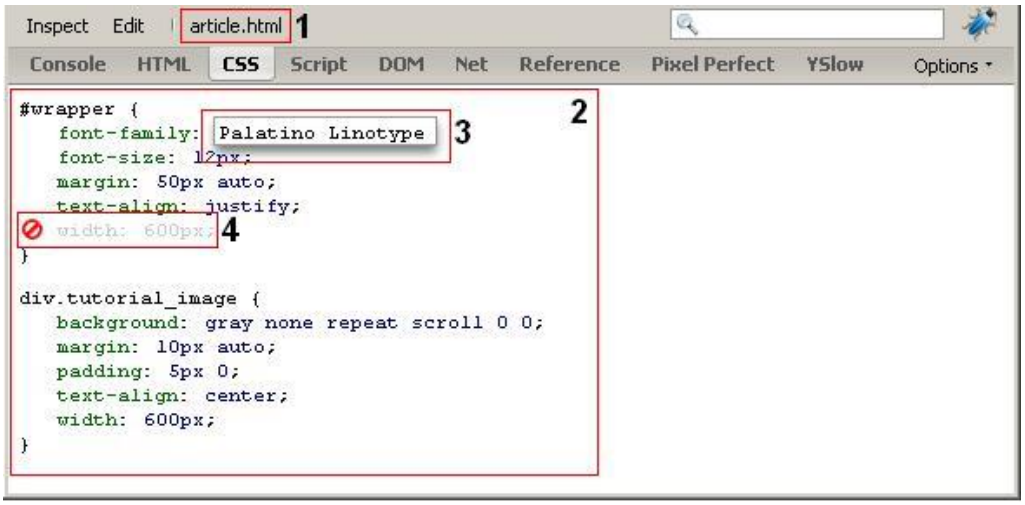


6. вкладка DOM показує список всіх методів та властивостей обраного на даний момент елемента.



3. CSS

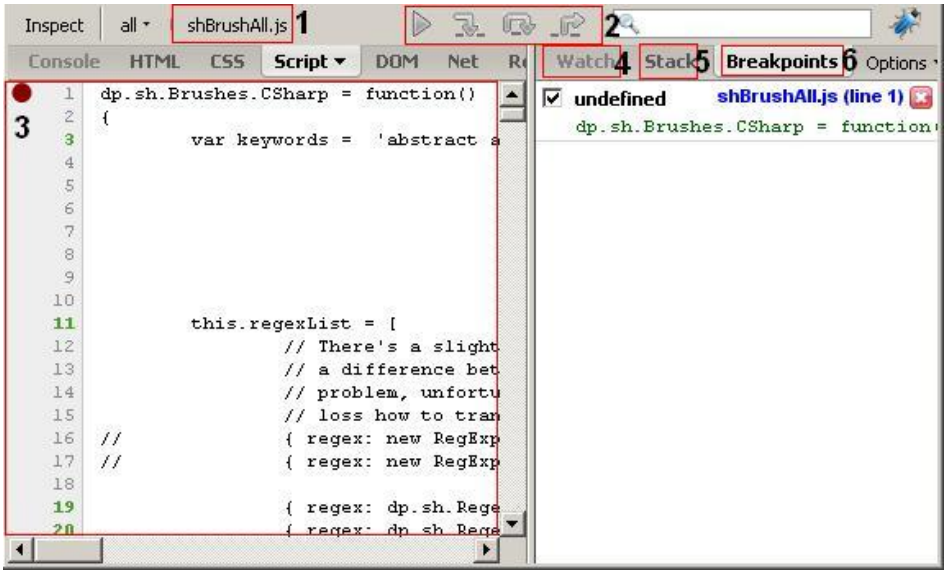
Основна різниця між цією панеллю та вкладкою Style панелі HTML є те, що можете працювати з не вичислювальними стилями.



1. Якщо сторінка, на якій працюєте, містить кілька таблиць стилів, тоді можете вибрати потрібну таблицю стилів.
2. Основна область, де показаний CSS код.
3. Легкість зміни CSS властивості.
4. Легко заборонити CSS правило.

4. Script

Іноді вам доводиться стикатися з труднощами, коли ви пишете Javascript код. Основну частину часу ви працюєте з панеллю Console; тільки в екстрених випадках це змусить вас перейти на панель Script. Передбачаючи ці екстрені випадки *(які напевно відбудуться)*, давайте розглянемо цю панель, і познайомимося з нею ближче

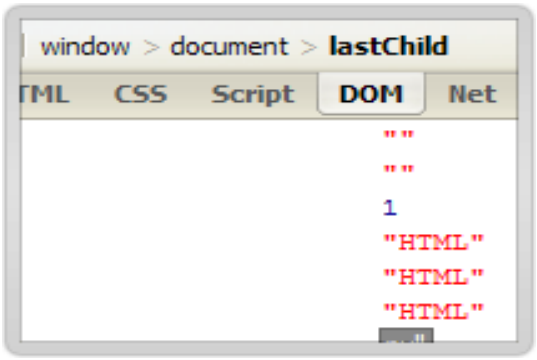


1. Випадаюча вниз кнопка, за допомогою якої ми можемо вибрати бажаний файл скрипта.

2. Функції відладки: continue, step in, step over і step out. Вони діють лише тоді, коли виконання коду досягає точки зупинки.
3. Основне вікно. Тут ми встановлюємо (і видаляємо) точки зупинки, а також перевіряємо код Javascript.
4. Схожа на панель DOM, вкладка Watch показує методи і параметри об'єкта для відладжуваного коду.
5. Показує вміст стека функцій в реальному часі.
6. Список активних на даний момент точок зупинки. Звідси можна тільки видалити точку зупинки.

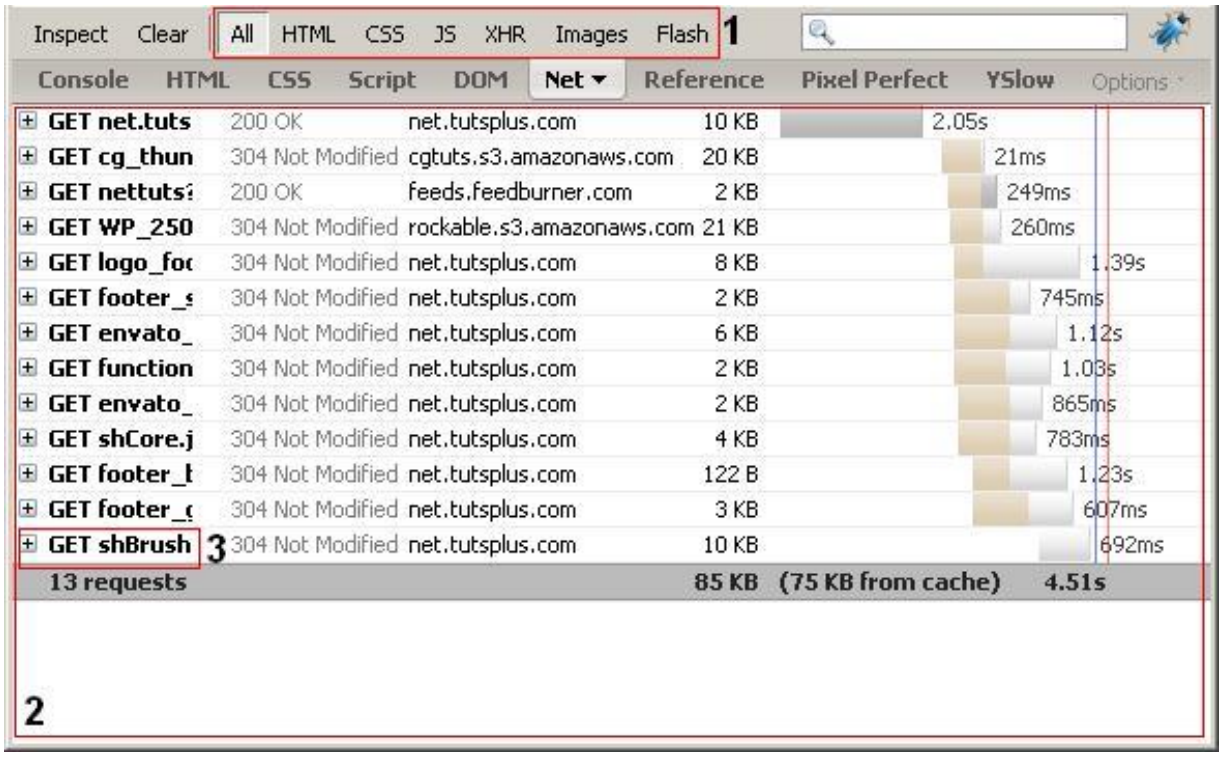
5. DOM

Такий же, як і HTML→ DOM. Об'єктна модель документа *(Document Object Model, DOM)* – величезна ієрархія об'єктів та функцій, доступних з javascript. Firebug дозволяє швидко знаходити DOM-об'єкти, й швидко редагувати.



6. Net

Цікавить скільки часу зайняло завантаження вашої сторінки? Або ви хочете знати, відповідь на який запит зайняв найбільше часу? На щастя, це теж може бути здійснено через панель Net.



1. Запити можуть бути відфільтровані відповідно до їх типів.
2. Кожен запит показаний в цій вкладці. В кінці списку запитів ми бачимо сумарний підрахунок: кількості запитів, обсягу, скільки було закешовано і загальний час виконання всіх запитів.
3. Може бути показано ще більш детально, наприклад: HTTP заголовки, відповідь і кеш *(такий же як і відповідь)*

Тестування виконання

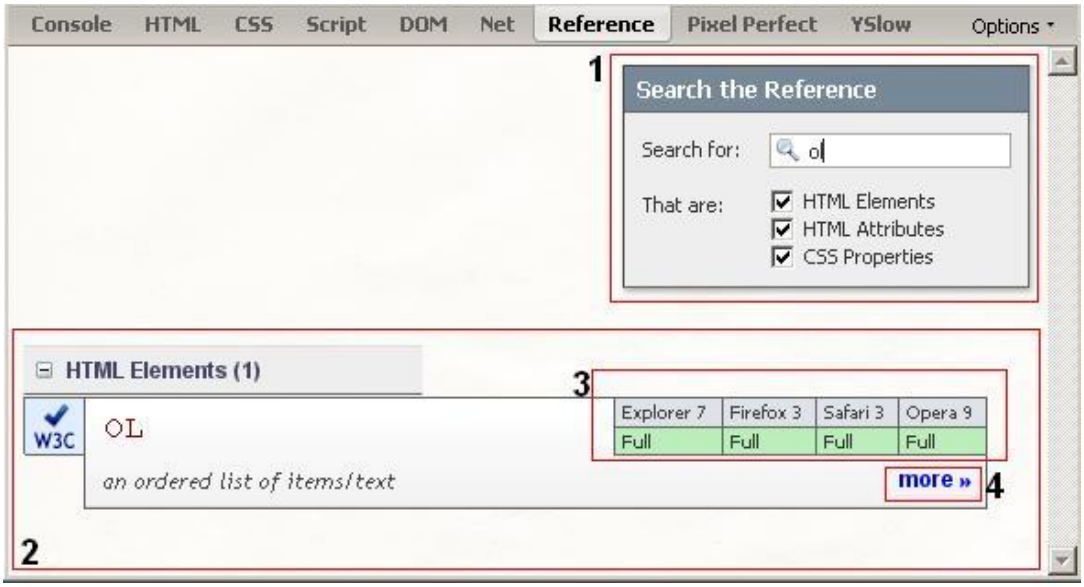
Необхідно протестувати виконання особливої функції або циклу? Використовуйте характеристику "timer" в Firebug.

```
function TimeTracker () {
   console.time ("MyTimer");
   for (x = 5000; x> 0; x -) {}
   console.timeEnd ("MyTimer");
}
```

Три кроки. Починаємо з виклику "console.time" в якій вказуємо унікальний ключ. Далі, вставляємо необхідний код. Потім, викликаємо console.timeEnd, й знову вводимо унікальний ключ.

7. *Reference*

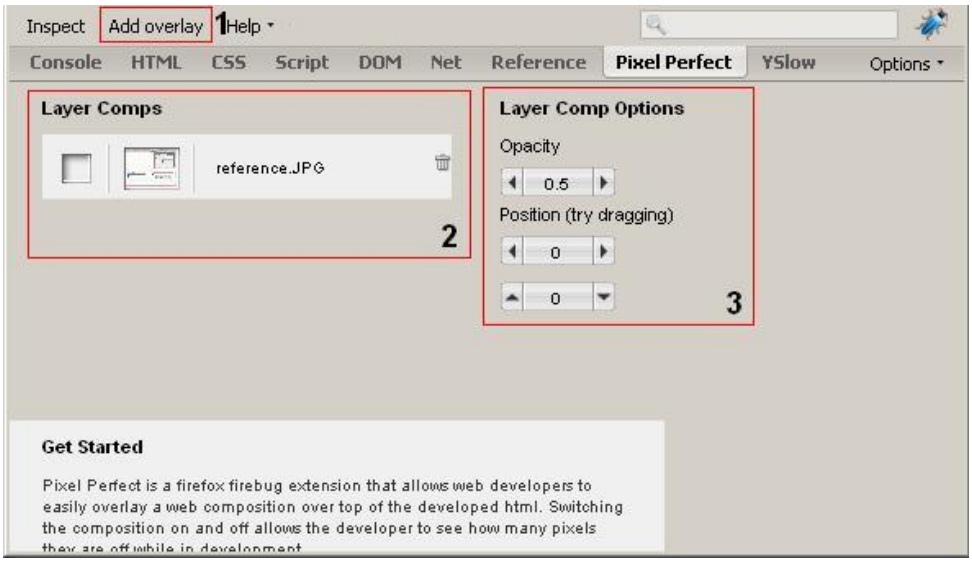
Додаткова панель, яка надається CodeBurner, розширенням Firebug'a, за допомогою цієї панелі маємо швидкий доступ до HTML та CSS коду.



1. Секція пошуку і фільтрації.
2. Тут один над іншим розташовані результати, в нашому випадку тільки один результат.
3. Область сумісності з основними версіями браузерів. Так, Chrome не присутній в цьому списку, але Chrome використовує той же движок що і Safari, з назвою Webkit, так що якщо є сумісність з Safari, то і сумісність з Chrome буде така ж хороша.
4. І якщо інформації що відображається в цій панелі буде недостатньо, ви можете перейти за цим посиланням для отримання більшої кількості інформації, такої як: приклади, сумісність з великою кількістю версій браузерів, опис та ін.

8. *PixelPerfect*

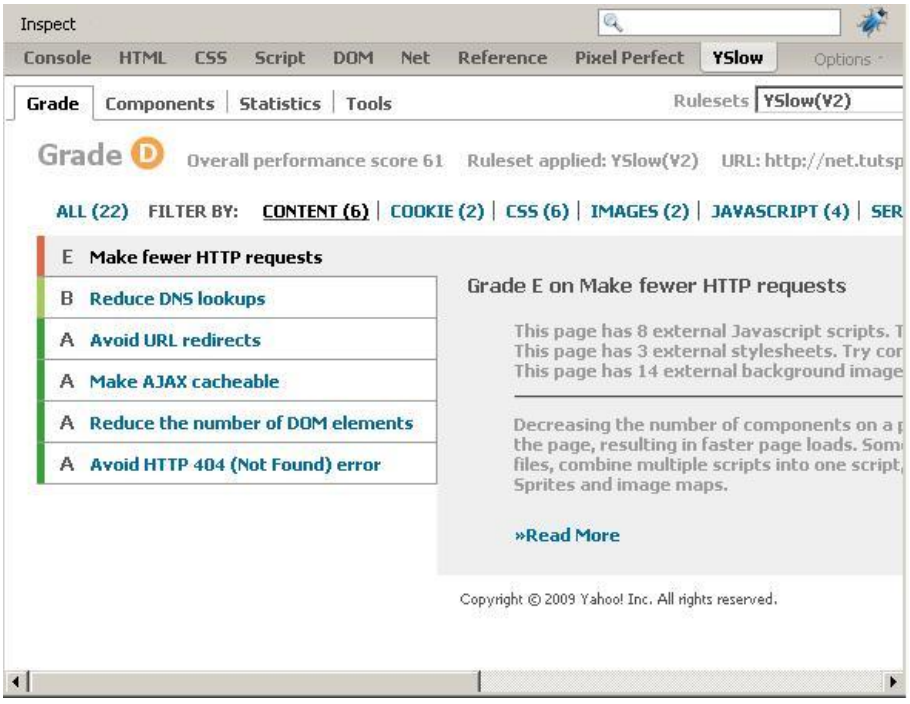
Якщо часто виконуєте нарізку з PSD документа, тоді знаєте скільки часу займає замір відстаней між можливими структурними елементами. Саме тут PixelPerfect показує свою силу. Наявність даного доповнення у панелі інструментів, допоможе робити бездоганну нарізку.



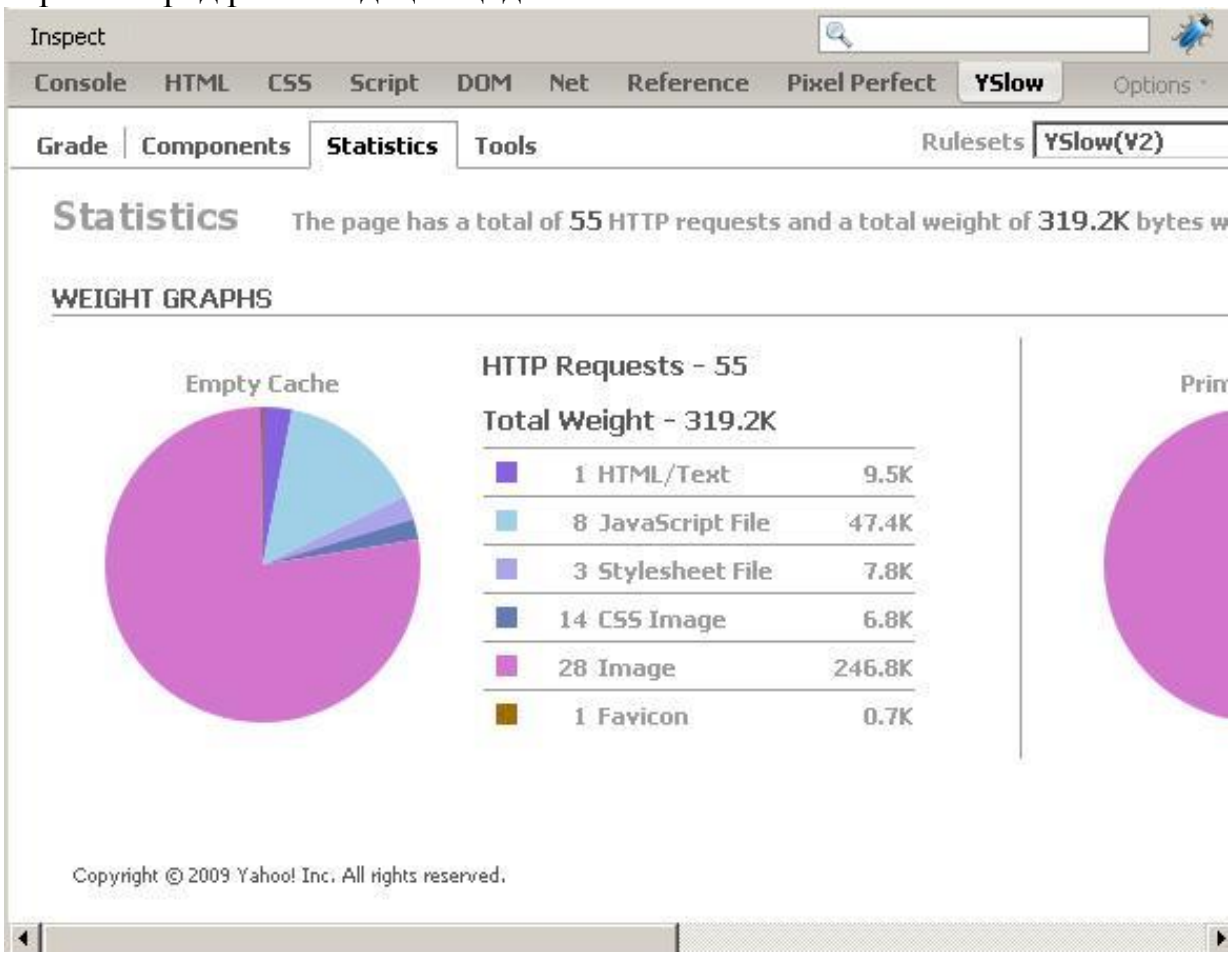
1. За допомогою цієї кнопки можемо додавати безліч шарів зображень для поточної сторінки.
2. Список шарів, де можемо застосувати або видаляти шар.
3. Область параметрів шару.

9. *YSlow*

Інше доповнення Firebug'a було розроблено Yahoo !, яке може запропонувати поліпшення швидкісних характеристик, грунтуючись на серії виконаних тестів.



За допомогою YSlow, можемо оцінити загальну продуктивність сайту та легко виявити місця, які можуть бути поліпшені, а також отримати ряд рекомендацій щодо поліпшення.

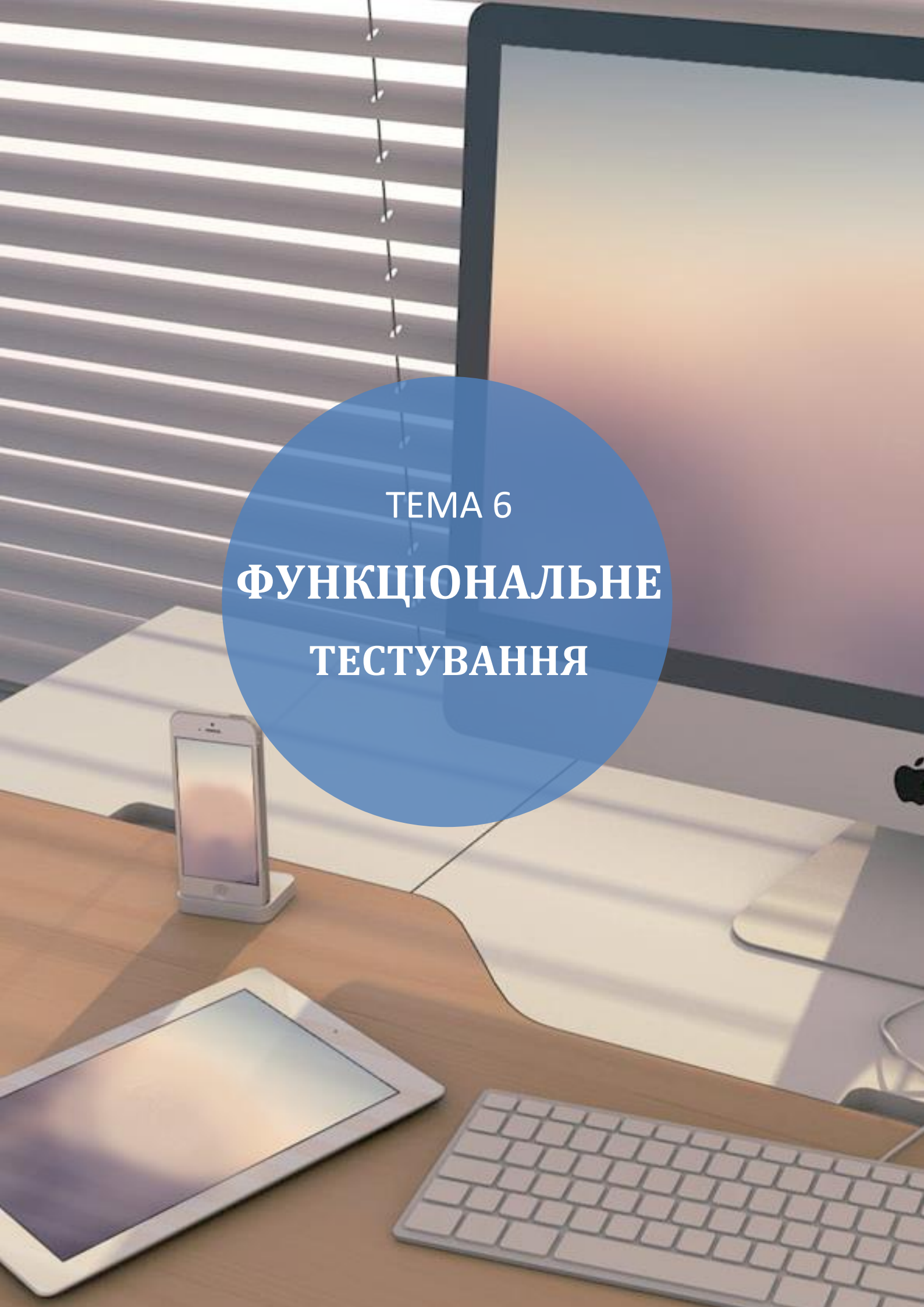


Крім кругової діаграми, також маємо JSLint та Smush.it у розпорядженні.

10. FirePHP

Останній, але не єдиний, важливий компонент Firebug'a це FirePHP. За допомогою цього доповнення, можете прозоро відправляти інформацію *(попередження, помилки, логи, дані)* в панель Console з вашого PHP коду. Приклад використання FirePHP на стороні сайту:

```
<? Php
    FB :: log ('Log message');
    FB :: info ('Info message');
    FB :: warn ('Warn message');
    FB :: error ('Error message');
?>
```



ТЕМА 6

ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ

6 ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ (FUNCTIONAL TESTING)

Сучасні IT-системи являють собою складно організовані програмні комплекси, об'єднані множинними інтеграційними зв'язками. При такій організації, вплив змін у функціоналі будь-якої з систем, складно піддається прогнозуванню та потенційно може призвести до збоїв й виникнення критичних помилок, які перешкоджають подальшому веденню бізнесу.

6.1 Основні поняття функціонального тестування (functional testing)

Функціональне тестування (functional testing) – вид тестування, при якому виявляється некоректна / неправильна робота функціоналу програми.

Функціональне тестування (functional testing) – процес перевірки відповідності поведінки системи спочатку заявленим функціональним вимогам.

Функціональне тестування (functional testing) – це тестування програмного забезпечення з метою перевірки реалізованості функціональних вимог, тобто здатності програмного забезпечення в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить програмне забезпечення, які завдання вирішує.

При функціональному тестуванні перевіряється коректність роботи системи, яка включає перевірку кожної з функцій програми, адекватність збережених і вихідних даних, методи їх обробки, обробка даних, що вводяться, методи зберігання даних, методи імпорту та експорту даних і т.д. залежно від специфіки додатку. При перевірці проектів проводиться документація функціональних вимог, що спрощує перевірку.

Цілями процесу функціонального тестування (functional testing) в області якості є:

- підтвердження якості програмного продукту, що відповідає показникам, встановленим замовником;
- виявлення та документування дефектів програмного продукту;

- прийняття об'єктивного рішення, зафіксованого в звіті про результати тестування, про можливість поставки програмного продукту замовнику.

Зазвичай, функціональне тестування проводиться на двох рівнях:

- ***компонентне (модульне) тестування*** – вид тестування окремих компонентів програмного продукту, сфокусоване на їх специфіці, призначенні та функціональних особливостях;
- ***інтеграційне тестування*** – вид тестування, що проводиться після компонентного тестування і спрямований на виявлення дефектів взаємодії різних підсистем на рівні потоків управління та обміну даними [19].

Підходи функціонального тестування (functional testing):

- тестування веб-форм;
- практика: пошук функціональних багів;
- техніки тестування і еквівалентне розбиття;
- граничні сценарії;
- чек-лист для перевірки функціоналу сайту;
- практика: доповнення чек-листа для перевірки функціоналу сайту;
- тестування без вимог;
- неформальні техніки тестування.

Розрізняють тестування (testing) за ознакою позитивності сценаріїв [13,20,21]:

- ***позитивний тестовий сценарій (positive testing scenarios)*** використовує тільки коректні дані і перевіряє, що додаток правильно виконав функцію, тобто покликаний показати, що програма працює так, як і годиться, за умови, що користувач вносить коректні дані і не виходить за рамки передбаченого сценарію поведінки.

Основною метою "позитивного" тестування (*positive testing*) є перевірка здатності системи працювати з виконанням тих функцій, для яких розроблялася. Як правило, перед початком тестування створюються тестові сценарії, які передбачають коректне функціонування системи.

- ***негативний тестовий сценарій*** (negative testing scenarios) призначений для перевірки сценаріїв, відмінних від коректного, вхідних даних з одним або більше неправильних параметрів і т.д.

Основною метою "негативного" тестування (*negative testing*) є перевірка стійкості системи до впливів різного роду, валідації невірного

набору даних, перевірка обробки виняткових ситуацій (як в реалізації самих програмних алгоритмів, так і логікою бізнес-правил).

Негативне тестування веб додатків [50] передбачає обробку тих сценаріїв, які можуть статися з системою, наприклад: користувач допустить помилку при введенні даних. В даному випадку система повинна бути готова "відповісти" на запит користувача повідомленням про помилку.

Негативні тести створюються з урахуванням людського фактора - тестувальник передбачає найбільш часто повторювані помилкові запити, які можуть вводити користувачі при роботі з системою.

Найбільш популярним негативним тестом в практиці тестувальників є так звана "техніка двох вкладок", коли система може давати збій, якщо відкрити програму у двох різних вкладках браузера. Дана техніка може бути застосована не тільки для онлайн-магазинів, але практично в будь-якому програмному продукті.

В рамках негативного і позитивного тестування виконують димове тестування, яке передбачає проведення коротких стадій тестів, які підтверджують, що система в цілому працездатна і виконує ключові функції. У випадку, коли в ході димового тестування ніяких явних дефектів не виявляють, тест оголошується успішно пройденим.

У багатьох ресурсах з тестування, при описі послідовності виконання видів тестування, негативне тестування відокремлюють в окрему процедуру і ставлять на 4-е місце.

При цьому наводиться наступний план тестування:

1. Вивчення документації.
2. Димове тестування (*smoke testing*) - перший прогін програми, щоб зрозуміти чи працює вона взагалі.
3. Позитивне тестування - перевірка роботи програми при отриманні "правильних" вхідних даних.
4. Негативне тестування включає наступні найбільш поширені негативні тест кейси (*negative test case*):
    - обов'язкове введення (*Required Data Entry*);
    - типи даних полів (*Field Type Test*);
    - розмір поля (*Field Size Test*);
    - числові граничні значення (*Numeric Bounds Test*);
    - граничні значення дати (*Date Bounds Test*);
    - валідність дати (*Date Validity*);
    - правила заповнення полів (Rules for filling fields);

- внутрішні одинарні лапки (Embedded Single Quote);

Позитивне і негативне тестування відіграє ключову роль нарівні з іншими категоріями тестування, включаючи функціональні види тестування, тестування безпеки, а також тестування взаємодії, і вимагають глибокої експертизи та спеціалізації.

При функціональному тестуванні (functional testing) проводиться перевірка наступних модулів сайту [17,18]:

1. Реєстрація (registration, logging) – модуль дозволяє користувачам реєструватися на сайті для реалізації деяких подальших дій (наприклад, для доставки товару, відправки інформаційної розсилки, надання доступу до закритої інформації).

2. Авторизація (authorization) – підтвердження особи користувача в мережі. Для авторизації необхідно ввести логін та пароль (рис.6.1).

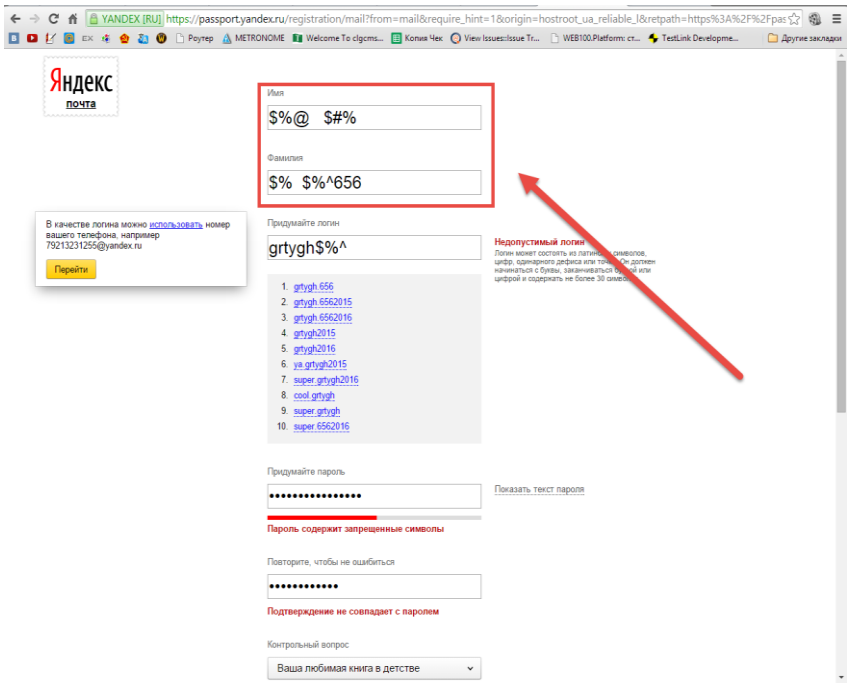


Рис. 6.1. Авторизація

3. Новинний модуль (news) – призначений для публікації новин на сайті, забезпечує спрощення роботи як адміністраторів сайту по додаванню новин на сайт, так й засвоюваність інформації відвідувачами (рис.6.2).

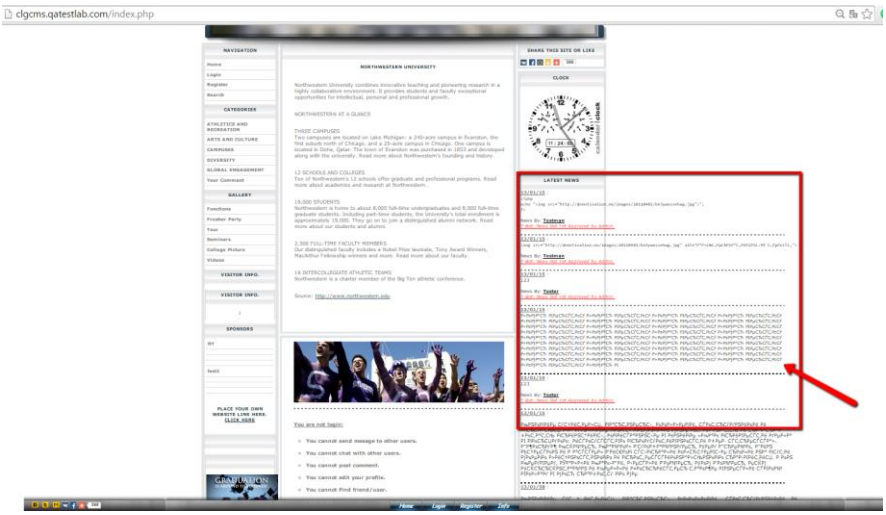


Рис. 6.2. Новинний модуль

4. Пошук по сайту (search) – модуль дозволяє здійснювати пошук по всіх сторінках сайту. Результатом роботи є видача списку знайдених сторінок з уривками текстів та посиланнями, що містять пошуковий запит (рис.6.3).

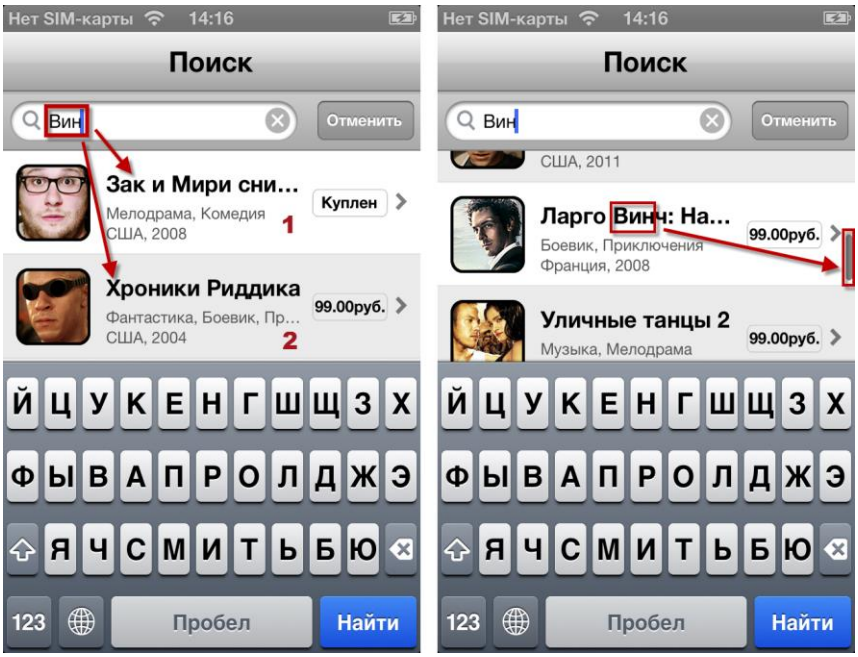


Рис. 6.3. Пошук по сайту

5. Зворотній зв'язок (feedback) –модуль дозволяє вивести текстову інформацію та форму з полями: ім'я, e-mail, повідомлення. Після натискання на кнопку "Відправити" введена інформація відправляється на e-mail користувача (рис.6.4).

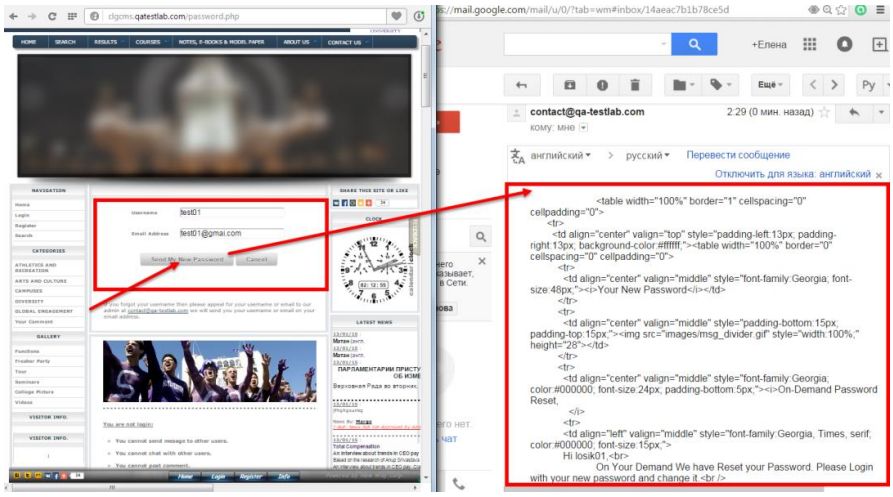


Рис. 6.4. Зворотній зв’язок

6. Банери (banner) – графічне зображення рекламного характеру, що дозволяє на сторінках сайту в певних місцях створити рекламні місця та змінювати інформацію, яка була виведена (рис.6.5).

Рис. 6.5. Банери

7. Фотоальбоми (photo album) – модуль, призначений для створення каталогу зображень, який дозволяє створювати зручні каталоги з категоріями зображень, фільтрами та іншими засобами, які спрощують навігацію і роботу з фотоальбомами (рис.6.6).

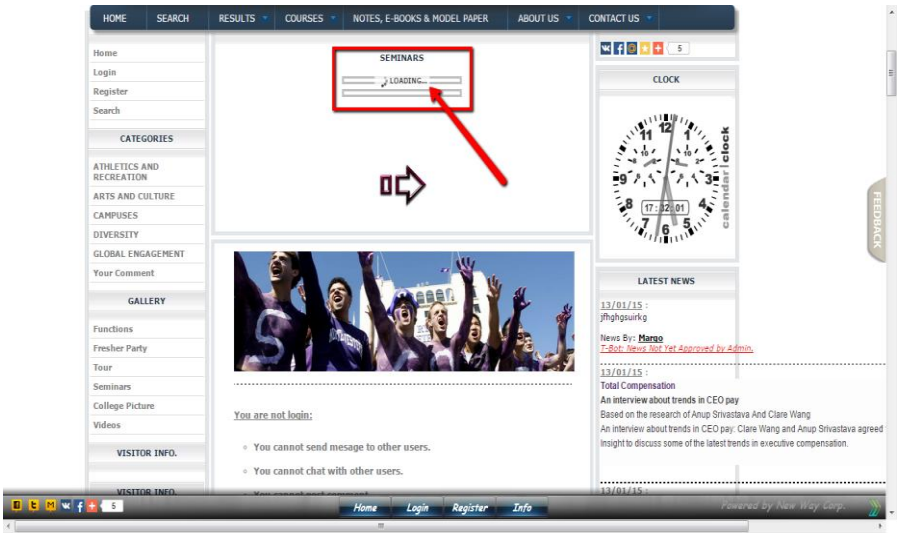


Рис. 6.6. Фотоальбоми

8. Форум (forum) – важливий модуль, що дозволяє створювати форуми, при цьому відрізняється широким функціоналом і можливістю тонкої настройки й може використовуватися як для створення повноцінного форуму на сайті, так і для створення окремого сайту-форуму (рис. 6.7).

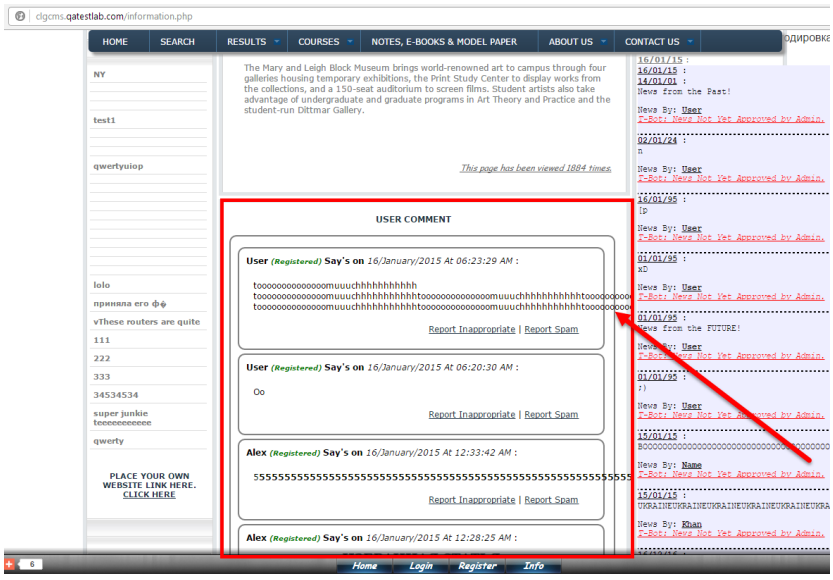


Рис. 6.7. Форум

9. Інтернет-магазин (online shop, e-shop) – один з найважливіших модулів системи, необхідний для створення повноцінного інтернет-магазину, володіє широкими настройками та відрізняється гнучким настроюванням, а також доступний в рамках платних пакетів послуг "Бізнес" та "Розширений" (рис. 6.8).

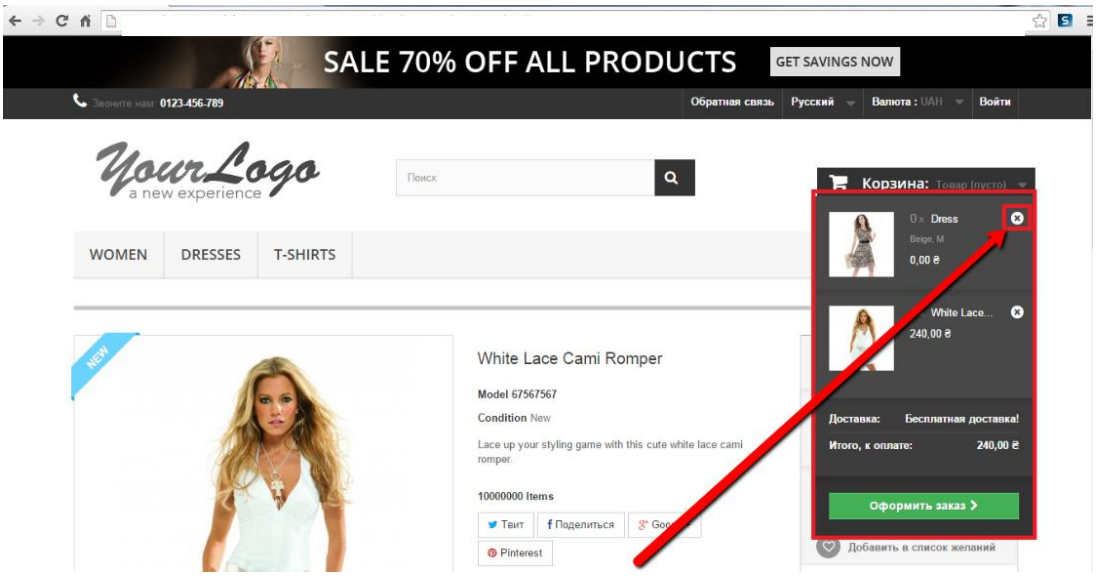


Рис. 6.8. Інтернет-магазин

10. Випадаючий список (drop-down list) –елемент списку одиночного вибору (рис. 6.9).

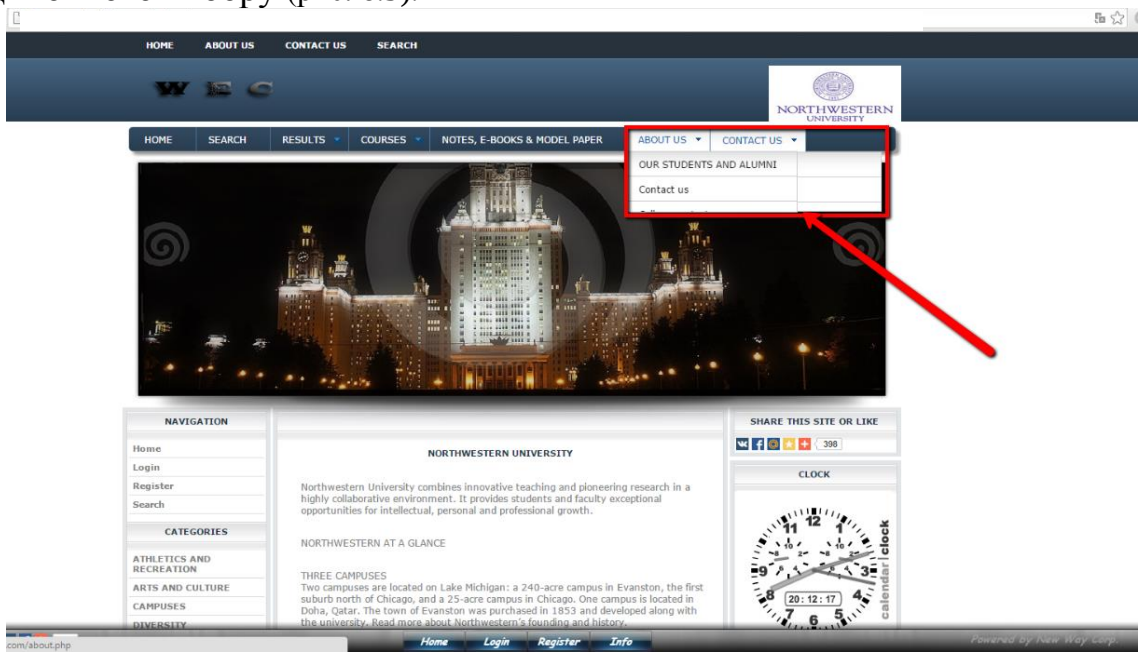


Рис. 6.9. Випадаючий список

11. Коментарі в соціальних мережах – новий модуль для автоматичної публікації контенту, опублікованого на сайті, в соціальних мережах.

12. Відео – модуль, що дозволяє розміщувати на сайті відеофайли, за допомогою, якого можна додавати на сайт власні відео або розміщувати переналаштовані ліцензійні відео і канали.

13. Модуль розсилки (mailout module) **–**модуль розсилок дозволяє організувати розсилання повідомлень зареєстрованим відвідувачам сайту, існує можливість автоматичної відправки тематичних повідомлень на електронну пошту.

14. Форма (form) **–** набір полів для введення даних на веб-сторінці. Дані відправляються на сервер, коли користувач завершує заповнення всіх полів і відправляє форму. Форум дозволяє створювати теми та вести їх обговорення.

Функціональне тестування (functional testing) складається з таких компонентів:

- 90% часу тестування;
- перевірка функціональних вимог: логіки і бізнес-правил додатків або системи;
- повноцінне системне / функціональне тестування є найбільш трудомістким процесом і може займати *(Ф.Брукс)* до 80% всього бюджету проекту з тестування.

Функціональні вимоги включають:

- функціональну придатність (*suitability*);
- точність (*accuracy*);
- здатність до взаємодії (*interoperability*);
- відповідність стандартам і правилам (*compliance*);
- захищеність (*security*);

6.1.1 Валідні та невалідні дані

При роботі з даними важливу роль відіграє валідація даних (*data validation*). Перш ніж використовувати отримані від користувача дані, необхідно переконатися, що дані введені правильно і являють коректні значення.

Валідація (validation) – це процес перевірки даних на відповідність певним, заздалегідь відомим правилам *(форматам, вимогам)*.

При тестуванні необхідно перевіряти узгодженість валідаторів вхідних даних з логікою обробки цих даних додатком, для розуміння того, як треба тестувати затвердження даних, треба розуміти, як вони повинні працювати, що можна вважати правильним, а що ні.

«Невалідні» дані, що не задовольняють певним обмеженням, можуть викликати збій у роботі програми.

Припустимо, в якомусь місці програми виникає виняток при спробі перетворити рядок в число, якщо рядок має некоректний формат. Зрозуміло, якщо винятку не буде ніде перехоплено, це може призвести до аварійного завершення програми. Але це малоймовірний сценарій розвитку подій. Швидше за все в якомусь місці спрацює перехоплювач, який або видасть користувачеві якесь повідомлення про помилку в програмі, або зробить запис у журнал помилок, після чого програма постарається відновитися від збою і продовжити роботу. Тобто навіть якщо валідацію не виконувати, цілком імовірно, що нічого страшного не трапиться.

Проблеми, що виникають при відсутності валідації (validation):

1. ***Неможливість відновитися після збою.*** Не завжди програма здатна «повернути все назад». Можливо, в процесі роботи програма виконала якісь незворотні дії - видалила файл, відправила дані по мережі, надрукувала щось на принтер, запустила різець верстата і він частково справив обробку заготовки деталі. Але навіть якщо відновлення в принципі можливо, алгоритм відновлення може теж містити помилки, і це іноді призводить до зовсім сумних наслідків.

2. ***Додаткове навантаження на систему.*** Відновлення після збою - це зайва робота. Вся робота, яка була виконана до моменту збою - теж зайва. А це означає додаткове навантаження на систему, якою можна уникнути, якщо заздалегідь перевірити дані. З іншого боку, валідація - це теж додаткове навантаження, причому відновлення доводиться робити лише зрідка, а перевірку треба виконувати кожен раз, так що ще невідомо, що вигідніше.

3. ***Ін'єкції не викликають збоїв.*** Один з основних способів експлуатації вразливостей в програмах полягає в тому, щоб «обдурити» валідатори, тобто передати дані, які валідатор визнає коректними, але при цьому вони інтерпретуються належним чином, так, що зловмисник може отримати несанкціонований доступ до даних або деяким можливостям програми, або здатний зруйнувати дані або програму. Якщо валідації немає взагалі, завдання зловмисника максимально спрощується.

4. ***Складність ідентифікації, причини, проблеми.*** Якщо виключення вилетіло звідкись із глибини програми, визначити причини його виникнення не так-то просто. І навіть якщо це можливо, може

виявитися нелегко пояснити користувачеві, що збій викликаний даними, які він ввів деякий час тому в якомусь зовсім іншому місці програми. А якщо перевірка виконана негайно після введення даних, ніяких складнощів з ідентифікацією джерела проблеми не виникає.

Наявність валідації дозволяє запобігти серйозним збоям, спрощує ідентифікацію проблем, але за це доводиться розплачуватися продуктивністю, оскільки додаткові перевірки збільшують навантаження на систему.

Валідація даних здійснюється наступним чином:

1. ***Посимвольна перевірка.*** Як правило такі перевірки виконуються в інтерфейсі, у міру введення даних. Але не тільки. Наприклад, лексичний аналізатор компілятора теж виявляє неприпустимі символи безпосередньо в процесі читання компілюючого файлу. Тому такі перевірки можна умовно назвати «лексичними».

2. ***Перевірка окремих значень.*** Для користувача інтерфейсу це перевірка значення в окремому полі, причому виконуватися вона може як у міру введення (перевіряється неповне значення, яке введено до справжнього моменту), так і після завершення введення, коли поле втрачає фокус. Для програмного інтерфейсу (API) це перевірка одного з параметрів, переданих в процедуру. Для даних, одержуваних з файлу, це перевірка якогось прочитаного фрагмента файлу. Такі перевірки, знову-таки за аналогією з компіляторною термінологією, можна назвати «синтаксичними».

3. ***Сукупність вхідних значень.*** Можна припустити, що в програму спочатку передаються якісь дані, після чого подається деякий сигнал, який ініціює їх обробку. Наприклад, користувач ввів дані у форму або в кілька форм (у так званому «Візард») і нарешті натиснув кнопку «OK». У цей момент можна виконати так звані «семантичні» перевірки, націлені на валідацію не тільки окремих значень, але і взаємозв'язків між ними, взаємних обмежень.
4. Цілком можлива ситуація, коли кожне окреме значення «синтаксично» коректно, але разом вони утворюють неузгоджений набір. Для програмного інтерфейсу цей різновид валідації передбачає перевірку набору вхідних параметрів процедури, що викликається, для випадку отримання даних з файлу це перевірка всіх прочитаних даних.

5. ***Перевірка стану системи після обробки даних.*** Нарешті, є останній спосіб, до якого можна вдатися, якщо валідацію безпосередньо вхідних даних виконати не вдається - можна спробувати їх обробити, але залишити можливість повернути все до вихідного стану. Такий механізм часто називається транзакційним.

Транзакція – це послідовність дій, які або всі завершуються успішно, або відбувається якийсь збій при виконанні окремої дії, і тоді скасовуються результати всіх попередніх дій цього ланцюжка. Так от, валідацію можна виконувати в процесі виконання транзакції, а остання перевірка може бути виконана в самому кінці транзакції по обробці даних. При цьому ми валідуємо вже не самі дані, а той стан, який вийшло після їх повної обробки, і якщо цей стан не задовольняє якимось обмеженням, тоді ми визнаємо вхідні дані невалідний і повертаємо все до вихідного стану [51].

Щоб знайти дефект валідації даних, треба розуміти, де шукати й на що звертати увагу.

Розглянемо декілька прикладів проведення тестування поля логін/пароль [52]:

1. Реєстрація нового користувача:

- зареєструвати нового користувача з логіном new_user: ***очікуваний результат:*** можна;
- зареєструвати нового користувача з логіном new_user_test: ***очікуваний результат:*** можна;
- зареєструвати нового користувача з логіном new-user: ***очікуваний результат:*** можна;
- зареєструвати нового користувача з логіном new1234user: ***очікуваний результат:*** можна;
- зареєструвати нового користувача з логіном new @ user: ***очікуваний результат:*** негативна відповідь;
- зареєструвати нового користувача з логіном newuser і паролем newuser *(повний збіг)*: ***очікуваний результат:*** негативна відповідь;
- використання тільки ASCII символів у логіні: ***очікуваний результат:*** негативна відповідь;
- реєстрація користувача з логіном, що містить пробіли або складається з одних пробілів: ***очікуваний результат:*** негативна відповідь;
- реєстрація користувача з паролем, що містить пробіли або складається з одних пробілів: ***очікуваний результат:*** негативна відповідь;
- реєстрація користувача з логіном містить XSS або SQL injections: ***очікуваний результат:*** негативна відповідь;

- а чи можна зареєструвати користувача "admin", і користувача "admin" (*де а - з російської розкладки)?*
- в деяких випадках розробники перевіряють користувача в базі за допомогою LIKE, і не обробляють user input. Тому потрібно перевірити комбінацію %%% / %%% (знак% повторюється 3 рази, щоб обійти валідацію на мінімальну довжину).
- логін під існуючим користувачем – зміна пароля:
  - створити акаунт з максимально можливим числом символів у логіні:
    - спробувати зайти в систему;
    - спробувати змінити пароль;

Причина: можлива розбіжність максимумів між рядками введення нового пароля, введення пароля, зміни пароля, і в БД.

Додатково: виконати ті ж кроки, але з максимальною кількістю символів + 1.

Додатково: виконати ті ж кроки, але з максимальною кількістю дозволених символів + пробіл (та інші нешкідливі);

З максисмальною кількістю дозволених символів + 1 заборонений.

  - створити акаунт з максимально можливим числом символом в паролі:
    - спробувати зайти в систему;
    - спробувати змінити пароль (і логін);

Причина: можлива розбіжність максимумів між рядками введення нового пароля, введення пароля, зміни пароля, і в БД.

2. Введення некоректних даних

- ввести коректний логін і коректний пароль.
  очікуваний результат: успішний вхід в систему.
  Вийти з системи, очистити кеш і куки (відкрити/закрити браузер?);
- залишити обидва поля порожніми і натиснути на кнопку "*Login*": ***очікуваний результат:*** негативна відповідь;
- залишити порожнє поле "Login" і натиснути на кнопку "*Login*": ***очікуваний результат:*** негативна відповідь;
- залишити порожнє поле "Password" і натиснути на кнопку "*Login*": ***очікуваний результат:*** негативна відповідь;
- ввести коректний логін і некоректний пароль:
  очікуваний результат: негативна відповідь;
- ввести некоректний логін, але коректний пароль:
  очікуваний результат: негативна відповідь;
- ввести некоректний логін і некоректний пароль:
  очікуваний результат: негативна відповідь.
- в поле логіна ввести коректний пароль, а в поле пароля ввести коректний логін:

очікуваний результат: негативна відповідь;

- ввести логін <script> alert (123) </ script> і коректний пароль:
  очікуваний результат: негативна відповідь;

3. Зміна / видалення логінів

- в базі або настройках сайту вказати, що термін придатності певного логіна минув, увійти в систему під цим логіном:
  очікуваний результат: негативна відповідь;

- увійти в систему під коректним логіном / паролем, потім змінити пароль та увійти в систему під новим паролем:
  очікуваний результат: пароль змінений, можна зайти.
  - зміна пароля і вхід під старим:
    - запам'ятати пароль;
    - увійти в систему;
    - поміняти пароль;
    - вийти з системи;
    - увійти в систему назад зі старим паролем: очікуваний результат: негативну відповідь;
- увійти в систему під коректним логіном/паролем, потім перейменувати аккаунт і перезавантажити браузер:
  - увійти в систему під старим логіном/паролем:
    очікуваний результат: негативна відповідь;
  - увійти в систему під новим логіном/паролем:
    очікуваний результат: можна;
- увійти в систему під коректним логіном / паролем, потім видалити акаунт і перезавантажити браузер:
  - увійти в систему під старими логіном / паролем:
    очікуваний результат: негативну відповідь;

6.2 Формування простого чек-листа (checklist) для функціонального тестування (functional testing)

Чек-лист (checklist) – контрольний список, що містить ряд необхідних перевірок для тестування.

Чек-лист (checklist) створений, щоб не пропустити жодного етапу. Оформляється у вигляді списку, де в міру руху проставляються галочки у перевірених або виконаних, залежно від функцій чек листа, пунктів.

Мета чек-листа (checklist) – контроль виконання всіх вимог до нового сайту.

Чек-лист *(checklist)* створюється на основі специфікації вимог програмного забезпечення. Визначаючи набір необхідних тестів, слід керуватися трьома основними правилами:

- чек-лист *(checklist)* повинен охоплювати весь функціонал продукту, що розробляється і жодна заявлена в специфікації вимога не повинна залишитися без уваги;
- число тестів потрібно мінімізувати. Чим більше вимог перевіряється одним тестом - тим краще;
- набір тестів повинен не повторювати вимоги, а перевіряти їх.

Приклад функціонального чек-листа (checklist):

Шапка

- *Іконка і назва на вкладці*
- *Логотип*
- *Live Chat*
- *Twitter*
- *Facebook*
- *Like*

Фотогалереї

- *Перелистування фото*
- *Коректний зум (якщо є)*
- *Коректний поворот фотографій (якщо є)*
- *Коректне відображення списку фотографій і самих фото*

Особистий кабінет

- *Редагування анкети*
- *Можливість видалення анкети*
- *Відображення статуса користувача*
- *Відображення статуса підписки*
- *Можливість відмінити/підтвердити підписку*
- *Вихід користувача із особистого кабінету*

Реєстрація/Авторизація

- *Реєстрація користувача*
- *Авторизація користувача*
- *Анонімний користувач*
- *Відновлення пароля*
- *Редагування облікового запису*

Статті

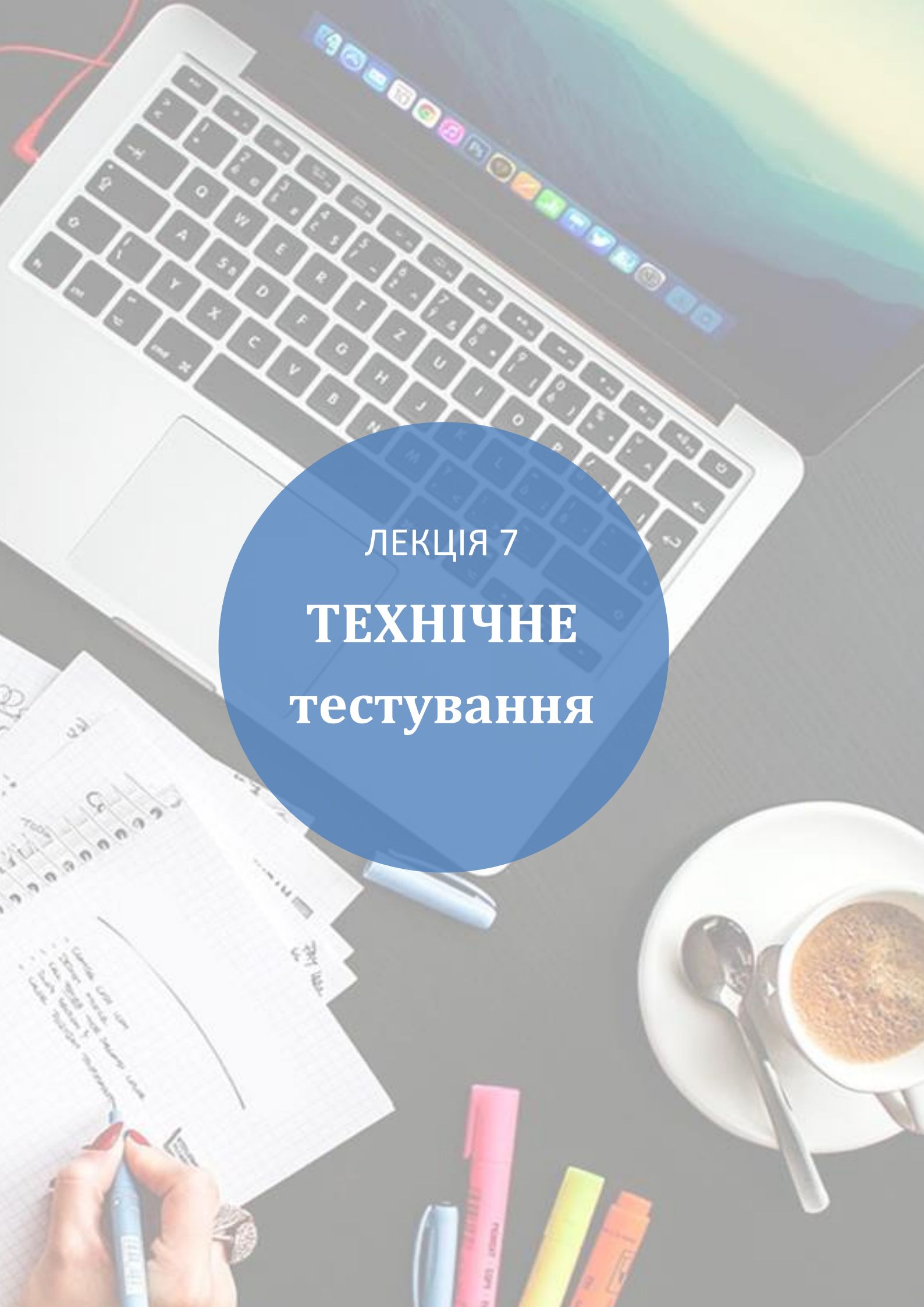
- *Шаринг статтей у всіх запропонованих соц. сітях*
- *Відображення фото*
- *Коректність переходу по посиланням*
- *Коректність повернення на головну сторінку*

Пошук

- *Пошук за новинами, розділами*
- *Перехід по посиланням*
- *Відображення дат новин*

Зворотній зв'язок

- *Робота при правильному заповненні полів*
- *Робота при неправильному заповненні полів*



ЛЕКЦІЯ 7

ТЕХНІЧНЕ тестування

В даній лекції розглянуто поняття технічне тестування та деякі програми, які дозволяють здійснювати тестування, а також розглянуто поняття «Системи управління змістом» (CMS) та представлені системами Joomla та Drupal.

7.1 Технічне тестування (performance testing)

Технічне тестування – група видів тестування, що проводиться на завершальному етапі готовності програмного продукту (сайту) й спрямована на перевірку технічних характеристик, таких як:

- нагрузоздатність сервера
- швидкості завантаження сторінок
- трастовість (такі показники як вік, тИЦ, PR)
- цілісність посилань
- індексація сторінок, наявність в каталогах пошукових машин
- мета-теги, HTML-теги
- відповідність стандартам W3C і багато іншого.

7.2 Основні програми для технічного тестування

Розглянуто такі програми як Firebug (а саме його консольну частину) та Xenu.

7.2.1 Firebug (console)

Firebug – розширення для браузера Firefox, що є консоллю, відладчиком та DOM-інспектором JavaScript, DHTML, CSS, XMLHttpRequest. Firebug показує в консолі функцію, що викликала помилку, стек викликів функцій, що викликали цю помилку, а також попереджає, що правило CSS або метод / властивість JavaScript, яке намагаєтесь використовувати, не існує.

До основними можливостями Firebug відносять:

- ***зручний перегляд HTML-коду сторінки:*** функція Inspect дозволяє точно визначити місцезнаходження тега того чи іншого елемента, переглянути всі «прив'язані» до нього властивості та стилі;
- ***редагування HTML та CSS прямо в браузері:*** можна змінювати атрибути тегів та значення властивостей для того, щоб спостерігати зміни. Це зручно для тих випадків, коли потрібно шляхом

експериментів знайти найбільш прийнятний варіант оформлення створеної сторінки;
- ***налагодження JavaScript***;
- ***відстеження процесу завантаження сторінки***;
- ***перегляд HTTP-заголовків звичайних та AJAX-запитів***.

Особливості використання Firebug:

1. ***Вихідний код:*** *н*а відміну від звичайного вихідного коду HTML, Firebug враховує зміну коду після перетворення через JavaScript. Таким чином, надається можливість відстежити елементи, що додаються через скрипти, й подивитися стилі, які неможливо виявити іншим способом.

2. ***Швидке редагування HTML:*** за допомогою Firebug можна вносити зміни безпосередньо в код документа та бачити результат впливу на сторінці, а також можете видаляти, додавати, редагувати атрибути тегів, просто натискаючи по ним.

3. ***Інспектування коду CSS:*** для будь-якого обраного елемента показується стиль, окремі властивості при цьому можна відключати, додавати нові властивості, змінювати значення існуючих. Результати змін відображаються в браузері негайно.

4. ***Пошук будь-якого елементу мишкою:*** за допомогою інструменту «Аналізувати елемент» можете вибрати будь-який текст, зображення або об'єкт на сторінці, просто натиснувши по ньому. При цьому відкриється HTML-код обраного елемента та його стиль.

5. ***робота плагіна:*** перш за все, що помітите при відкритті Firebug (з панелі статусу або використовуючи комбінацію клавіш ctrl + F12), буде консольна панель – Console (рис.7.1).

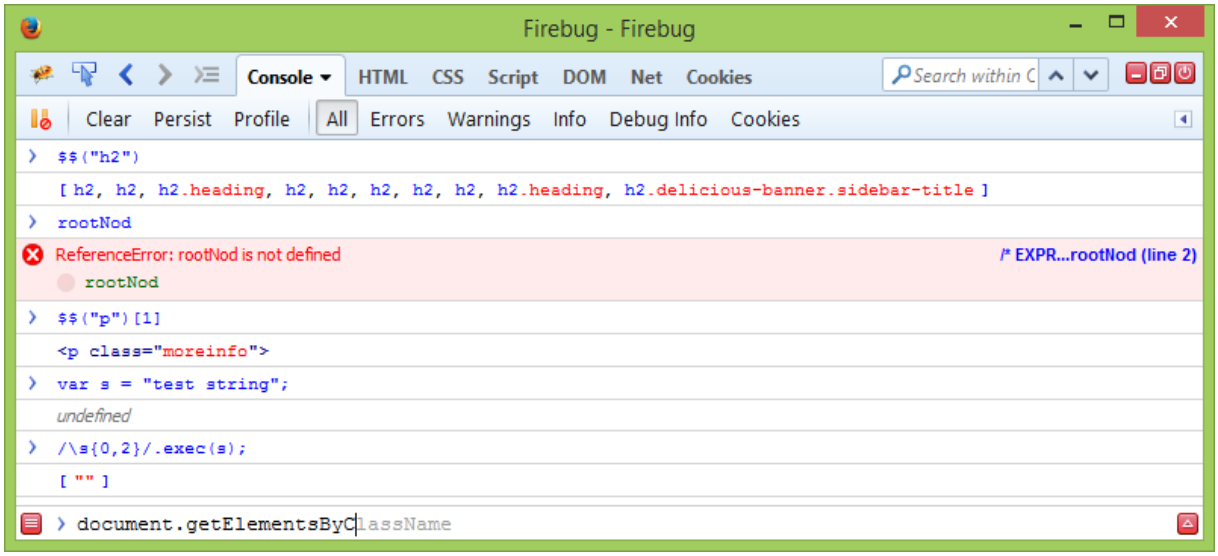


Рис 7.1. Консольна панель Firebug

Після швидкого перегляду можна припустити, що це альтернативна версія звичайної консолі помилок, яку можна викликати комбінацією клавіш Ctrl + Shift + J. Основні можливості зводяться до двох наступних:

- реєстрація (логування) помилок, попереджень та повідомлень;
- можливість працювати з Javascript кодом.

Але Firebug розширює функціональність браузера Firefox, тому він може виконувати набагато більше, а саме:

- реєструвати помилки Javascript, CSS, XML, XMLHttpRequest (AJAX) і Chrome (ядро Firefox)
- виконувати Javascript код на поточній веб-сторінці
- передавати додатковий Javascript об'єкт для управління (в консоль) – ***console***.

Об'єкт ***console*** є глобальною змінною та включає в себе безліч методів для отримання інформації, що проходить через скрипти, безпосередньо у вікні консолі. У контексті даної теми буде розглянуто основні методи.

`console.log (object [, object, ...])` – вписує рядок в консоль та можна вибирати скільки завгодно аргументів та всі будуть виведені в одному рядку. Перший аргумент для `log` може бути рядком, що містить патерни підстановки типу `printf`.

Наприклад:

`console.log ("The% s jumped over% d tall buildings", animal, count);` – можна переписати без підстановки рядків з тим же результатом:

```
console.log ("The", animal, "jumped over", count, "tall buildings");
```

Ці дві техніки можна комбінувати. Якщо використовуєте підстановку рядків, але даєте більше аргументів, ніж місць для підстановки, то аргументи, що залишилися, будуть додані через пробіл, як наприклад:

```
console.log ("I am% s and I have:", myName, thing1, thing2, thing3) ;
```

Якщо об'єкти логуються, то пишуться не як статичний текст, а у вигляді інтерактивних гіперпосилань, на які можна натиснути, щоб проінспектувати об'єкт у вкладках HTML, CSS, Script або DOM, а також можете використовувати патерн `%o`, щоб підставити гіперпосилання в рядок.

`console.debug (object [, object, ...])` – пише в консоль повідомлення, включаючи гіперпосилання на рядок, з якого стався виклик.

`console.info (object [, object, ...])` – друкує повідомлення на консоль з іконкою "info" та відповідним кольором коду, додаючи гіперпосилання на рядок, з якого стався виклик.

`console.warn (object [, object, ...])` – друкує повідомлення на консоль з іконкою "warning" та відповідним кольором коду, додаючи гіперпосилання на рядок, з якого стався виклик.

`console.error (object [, object, ...])` – друкує повідомлення на консоль з іконкою "error" та відповідним кольором коду, додаючи гіперпосилання на рядок, з якого стався виклик.

`console.assert (expression [, object, ...])` – перевіряє на вірність вираз expression, якщо помилковий, то буде видане повідомлення на консоль та exception.

`console.dir (object)` – друкує інтерактивний список всіх властивостей об'єкта. Виглядає аналогічно тому, що можно бачити у вкладці DOM.

`console.dirxml (node)` – друкує вихідне дерево XML для елемента HTML або XML. Виглядає аналогічно тому, що можно бачити у вкладці HTML, а також можете натиснути будь-який вузол, щоб інспектувати у вкладці HTML.

`console.trace ()` – друкує інтерактивний стек виконання javascript на момент виклику.

Стек в подробицях показує функції, разом з параметрами виклику кожної з них, можете натиснути на кожну функцію, щоб перейти на код у вкладці Script, та натиснути будь-який аргумент, щоб інспектувати у вкладці DOM або HTML.

`console.group (object [, object, ...])` – виводить повідомлення на консолі та відкриває вкладений блок відступів для всіх майбутніх повідомлень консолі. Викликайте `console.groupEnd ()`, щоб закрити блок.

`console.groupEnd ()-` закриває останній відкритий блок відступів, створений викликом console.group.

`console.time (name) -` створює новий таймер з даним ім'ям. Викликайте console.timeEnd (name) з тим же ім'ям, щоб зупинити таймер і видати значення, що замірялось до цієї дії.

`console.timeEnd (name) -` зупиняє таймер, створений викликом console.time (name) та пише значення, що замірялось до цієї дії.

`console.profile ([title]) -` включає javascript профілювальник. Необов'язковий аргумент title може містити заголовок, який буде надруковано в звіті профілювання.

`console.profileEnd ()-` вимикае javascript профілювальник і друкує його звіт.

`console.count ([title]) -` друкує число виконань. Необов'язковий аргумент title може містити повідомлення, що доповнює це число.

Представлений повний набір патернів, які можна використовувати для підстановки:

Патерни підстановки в рядок

%s	Рядок
%d, %i	Ціле (форматування чисел поки-що не підтримується)
%f	Число с плаваючою крапкою (форматування чисел поки-що не підтримується)
%o	Гіперпосилання на об'єкт

Використання Firebug для дослідження HTML-коду сторінки

Firebug дозволяє реалізувати дві дуже цікаві можливості, що допомагають розібратися з пристроєм коду відкритої в браузері сторінки. Можна водити мишею по самому HTML тегу у вікні цього плагіна, а на веб-сторінці будуть виділятися ті елементи, за формування яких цей тег відповідає (натискаєте, наприклад, по div контейнерам і тут же побачите вгорі підсвічування цього контейнера в дизайні веб-сторінки) .

А можна, навпаки, активувати режим, що мишею ви будете водити по самій веб-сторінці, а HTML код, відповідний тегам (що знаходиться під курсором миші), буде в реальному часі показуватися в вікні Firebug. Просто незамінна річ при вивченні пристрою того чи іншого веб-ресурсу, особливо для початківців.

Після того, як потрібний фрагмент знайдений, ви можете натиснути по ньому лівою кнопкою миші і перейти в ліве поле вікна Firebug для подальшої роботи з тегами цього елемента дизайну. З тегами у вікні плагіна можна виконувати будь-які маніпуляції і результат ви побачите відразу ж на відкритій в браузері сторінці.

При цьому, звісно, ніяких змін в реальних файлах сайту проводитися не буде. Але зате ви зможете в реальному часі промоделювати різні варіанти побудови дизайну веб-сторінки.

Використання Firebug для дослідження CSS-стилів сторінки

У правому вікні цього розширення відображаються всі властивості CSS, які відповідають за оформлення виділеного в лівому вікні фрагмента коду. Ці CSS властивості будуть визначати зовнішній вигляд даного html-елементу. Це може бути його розміщення щодо інших блоків даної сторінки, розмір, тип і колір шрифту, використовуваного в даному блоці для відображення кольору або посилань і багато іншого.

Розташовані в правій частині властивості можна редагувати ак само, як ми редагували теги в лівій частині вікна цього розширення. Всі внесені зміни у властивості з правої області будуть тут же відображатися на відкритій в Mozilla FireFox веб-сторінці, але ніяк не вплинуть на реальні властивості CSS, прописані у файлі таблиць каскадних стилів.

Використання Firebug для вимірювання швидкості завантаження веб-сторінки

Швидкість, з якою завантажується ресурс, є дуже важливим показником. Якщо у тестованого проекту низька швидкість завантаження, то це може спричинити за собою досить неприємні наслідки як з боку пошукових систем, так й з боку лояльності відвідувачів.

Найважливіше те, що відвідувачі можуть відмовитися від роботи з представленим ресурсом, тому що дуже довго завантажуються веб-сторінки. Крім цього пошукові системи, особливо Google, враховують швидкість завантаження при загальній оцінці корисності того чи іншого ресурсу, а також може впливати на стан веб проекту в пошуковій видачі.

У плагіна Firebug є можливість виміряти швидкість завантаження веб-ресурсів. Досить просто натиснути кнопку «Мережа» та оновити відкриту веб-сторінку.

В результаті, буде надана повна інформація щодо завантаження всіх елементів веб-сторінки.

7.2.2 Xenu Link Sleuth

Xenu Link Sleuth – це один з найбільш корисних інструментів в пошуковій оптимізації.

До основни завдання, які виконує програма відносять:

- ***шукати биті (не працюючі) посилання на заданому ресурсі:*** своєчасне виявлення несправних посилань дозволяє не тільки поліпшити показники сайту в пошукових системах, але й явно підвищити інтерес та лояльність користувачів сайту;
- ***складати карту сайту:*** для динамічних сайтів скласти карту не складає проблеми, однак, для статичних HTML ресурсів створювати карту сайту вручну вельми довго й трудомістко. Xenu вирішує цю задачу за кілька хвилин в залежності від розміру сайту та швидкості інтернет-з'єднання;
- ***шукати сторінки з великим часом віддачі:*** вимірювання швидкості завантаження веб-сторінок – дуже важливий процес тестування. Але заміряти вручну кожну сторінку багатосторінкового сайту занадто затратно по ресурсам. За допомогою Xenu можна побачити картину відгуку всіх сторінок в цілому та визначити проблемні місця;
- ***знайти неунікальні заголовки:*** кожен заголовок на сторінці повинен бути унікальним, тоді жоден з них не буде знаходитися в додаткових результатах пошуку та фільтруватися, як дубльований контент.
- ***знайти сторінки з великим рівнем вкладеності:*** всі сторінки на сайті по можливості повинні знаходитися не далі, ніж у двох-трьох рівнях від головної. Чим далі знаходиться сторінка, тим складніше до неї добратися як користувачам, так й пошуковим системам. Якщо знайшлися подібні сторінки, які являються достатньо важливими, але знаходяться далі, ніж в 3-х рівнях від головної, варто прийняти які-небудь заходи для поліпшення навігації;
- ***подивитися, які зі сторінок мають найбільшу та найменшу кількість внутрішніх посилань:*** перевірте внутрішню перелінковку в чисельному вигляді. Які з сторінок заслужили більше уваги, а які менше (виходячи з внутрішніх посилань);
- ***знайти картинки з відсутнім атрибутом "alt":*** атрибут "alt" є важливим при оптимізації сайту або окремих сторінок під певні запити. Перевірте, можливо відсутня важлива інформація чи опис зображень на вашому сайті.

Робота з програмою Xenu Link Sleuth

Щоб почати аудит якогось сайту, виберіть пункт меню "File" Check URL". У вікні необхідно ввести адресу сайту та виділити чекбокс "Check external links" (перевіряти зовнішні посилання) (рис. 7.2).

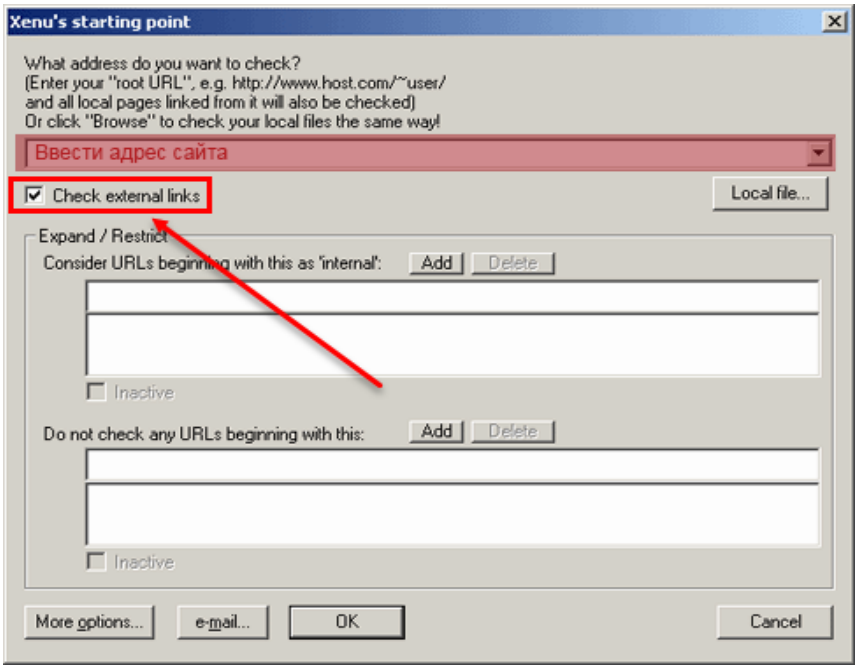


Рис. 7.2. Меню опцій програми Xenu

У програмі передбачено безліч налаштувань, доступних в пункті меню "Options"→"Preferences" (Рис. 7.3):

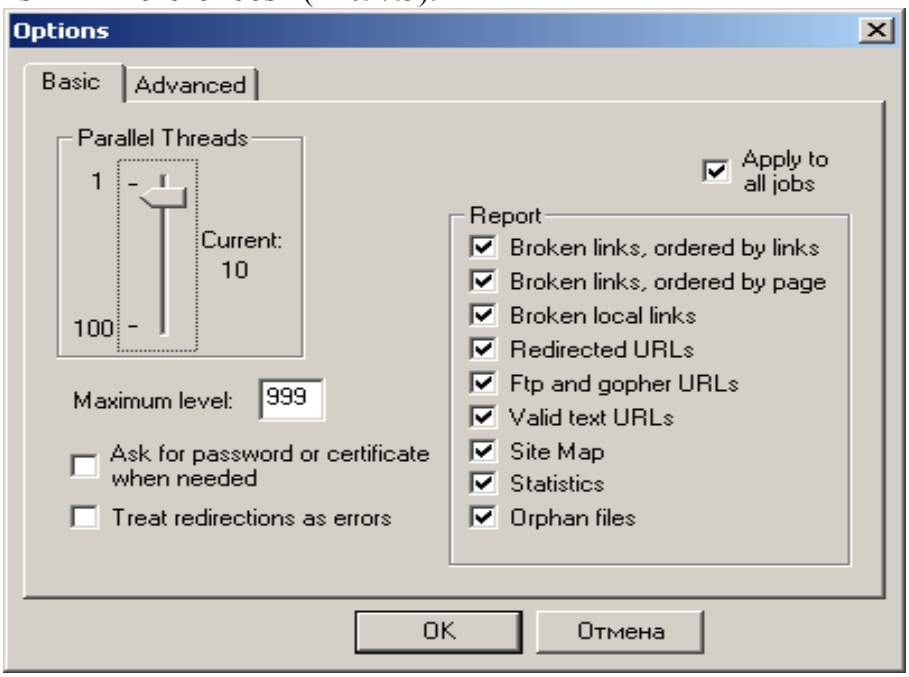


Рис. 7.3. Вікно налаштувань програми Xenu

Функції, які виконують відповідні елементи пункту меню "Options":

- ***"Parallel threads"*** встановлює кількість синхронних (паралельних) запитів;
- ***"Apply to all jobs"*** дозволяє зберігати налаштування для всіх проектів та звіт дозволяє включати наступні пункти:
  - ***"Broken links, ordered by link"*** – биті посилання, відсортовані за адресою;
  - ***"Broken links, ordered by page*** – список битих посилань, відсортованих по сторінках;
  - ***"Broken local links"*** – биті внутрішні посилання;
  - ***"Redirected URLs"*** – адреси з 301/302 перенаправленням;
  - ***"Ftp and gopher URLs*** - ФТП й інші не HTTP адреси;
  - ***"Valid text URLs"*** – список робочих текстових посилань;
  - ***"Site Map"*** – карта сайту;
  - ***"Statistics"*** – статистика;

Відразу ж після закінчення перевірки посилань сайту, в головному вікні з'явиться детальна таблиця, де дані можна відсортувати по кожному із стовпців: адреса посилання, статус, mime-тип, розмір, заголовок, рівень вкладеності, кількість зовнішніх та внутрішніх посилань, час віддачі сторінки. Файл звіту роботи Xenu дуже великий й містить в собі всю необхідну інформацію (рис. 7.4).

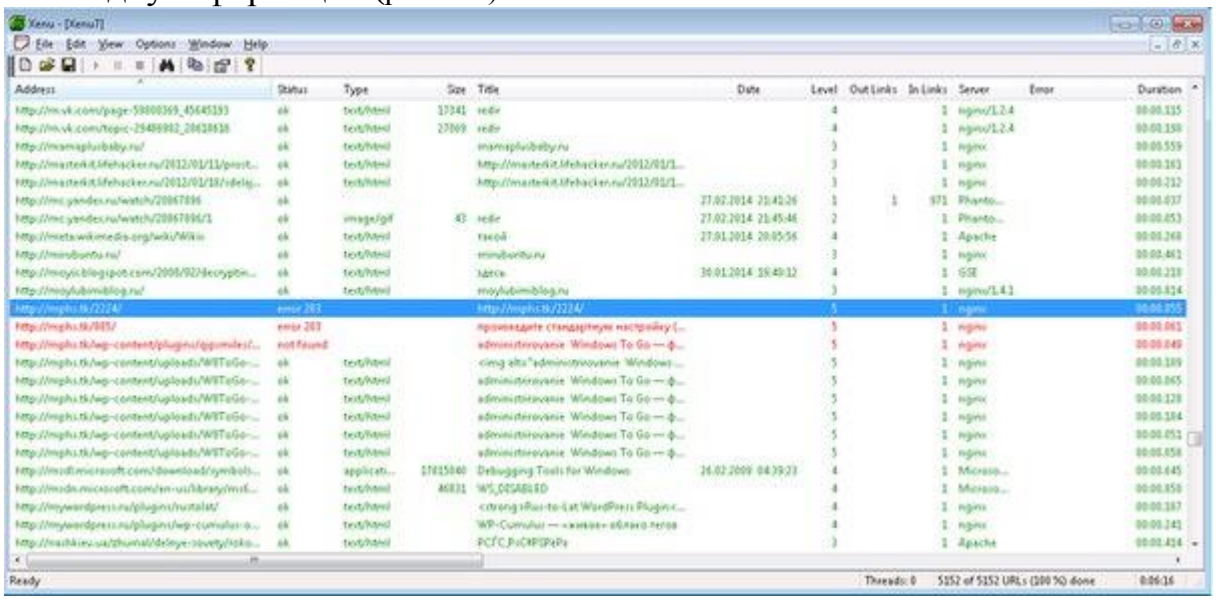


Рис. 7.4. Вікно звітної інформації програми Xenu

В головному вікні програми виводиться список всіх знайдених посилань та інформація у вигляді таблиці, яка складається з наступних стовпців (перерахую найбільш цікаві):

- ***Address*** – власне знайдені URL, якщо відсортувати за алфавітом, то можна побачити які є вихідні посилання (не на ваш домен), оцінити їх кількість, оцінити внутрішню структуру вашого сайту, скільки сторінок, файлів в тій чи іншій категорії. Подвійний клік відкриває сторінку в браузері.

- ***Status*** – в цьому стовпці можна дізнатися статус завантаження сторінки: ok - сторінка завантажилася (зелений колір), червоний колір - будь-яка помилка (timeout, 404 та інше). сортуючи з цього стовпця можна знайти мертві посилання. Виконайте натискання на праву клавішу мишки і виберіть "url properties", щоб дізнатися, на якій сторінці перебуває це мертве посилання. Якщо багато сторінок на вашому сайті показують помилку timeout, то можливо ваш хостинг або сайт занадто повільні і не можуть віддавати сторінки в стільки потоків. Зменшуйте кількість потоків в опціях, але якщо це спостерігається при менш ніж 5-ть потоків - терміново міняйте хостинг або сервер на хостингу.

- ***Type*** – тип сторінки \ контенту: це може бути просто html сторінка, може бути зображення (jpeg, png), js скрипти, css-файли, архіви та інше.

- ***Size*** – розмір сторінки. Сортуючи з цього стовпця ви можете виявити аномально великі сторінки або файли. Можна, наприклад, знайти великі зображення і стиснути їх більш оптимальним чином - це призведе до економії простору на хостингу і збільшення швидкості завантаження.

- ***Title*** – title посилання (якщо виходить на зовнішній сайт), сторінки (якщо посилання внутрішнє), зображення. Пам'ятайте, що Google не любить, коли на одному сайті або сторінці кілька однакових title? Якщо вказане значення "redir" - значить це посилання з перенаправленням і таким чином можна виявити всі посилання з перенаправленням.

- ***Level*** – рівень вкладеності сторінки. Пам'ятайте про те, що чим більша вкладеність - тим гірше (для індексації, для доступності користувачам, вартості посилання), тому рекомендують тримати сторінки сайту не далі 3-го рівня вкладеності? Можна проаналізувати, які сторінки мають занадто великий рівень вкладеності і подумати над структурою сайту. Якщо раптом ви виявите, що деякі посилання мають явно завищений рівень

вкладеності, то це значить, що на неї потрапили через редиректи: 1 редирект - +1 вкладеність.

- ***Out Links*** – кількість посилань на сторінці (як на внутрішні сторінки, так і на зовнішні). Виконайте натискання на праву клавішу мишки і виберіть "url properties" і ви побачите, які саме посилання виявлені на сторінці. Сортуючи з цього стовпця можна виявити сторінки з явно завищеною кількістю посилань, та й в цілому можна оцінити якість перелинковки.

- ***In Links*** – кількість внутрішніх вхідних посилань на цю сторінку. Завдяки цьому стовпцю можна побачити, на які сторінки найбільше звертається увага. Виконайте натискання на праву клавішу мишки і виберіть "url properties" і ві побачите, які саме сторінки посилаються.

- ***Duration*** – час завантаження сторінки. Тут можна виявити повільні сторінки. Повільними вони можуть бути через обсяг, завантаження файлів з повільних зовнішніх сайтів або із-за великого навантаження на базу даних.

7.3 Системи управління контентом (CMS - Content Management System)

✓ ***CMS*** – інформаційна система або комп'ютерна програма, яка використовується для забезпечення та організації спільного процесу створення, редагування та управління контентом.

Основні функції CMS:

- надання інструментів для створення наповнення, організації спільної роботи над наповненням;
- управління наповненням: зберігання, контроль версій, дотримання режиму доступу, управління потоком документів та ін;
- публікація наповнення;
- представлення інформації у вигляді, зручному для навігації, пошуку.

✓ ***Joomla!*** – система управління наповненням (CMS), написана на мовах PHP і JavaScript, що використовується в якості сховища бази даних СУБД MySQL або іншої індустріально-стандартної реляційної СУБД.

Являється вільним програмним забезпеченням, що поширюється під ліцензією GNU GPL.

CMS Joomla! включає в себе мінімальний набір інструментів при початковій установці, який доповнюється в міру необхідності. Це знижує заповнення адміністративної панелі непотрібними елементами, а також знижує навантаження на сервер і економить місце на хостингу.

Основні можливості:

- функціональність можна збільшувати за допомогою додаткових розширень (компонентів, модулів та плагінів);
- присутній модуль безпеки для багаторівневої аутентифікації користувачів та адміністраторів (використовується власний алгоритм аутентифікації і «ведення» сесій);
- система шаблонів дозволяє легко змінювати зовнішній вигляд сайту: розташування модулів, шрифти та інше. Присутня можливість надання користувачам вибирати одне з декількох відображень. У мережі існує величезний вибір готових шаблонів, як платних, так і безкоштовних. Також існує програмне забезпечення для самостійного створення оригінальних шаблонів;
- передбачені схеми розташування модулів, ща легко налаштовуються, включаючи ліве, праве, центральне і будь-яке інше довільне положення блоку. При бажанні вміст модуля можна включити у вміст матеріалу. Наприклад, вираз {loadposition mod_fpslideshow}, введене (разом з фігурними дужками) в довільне місце статті, виведе вміст модуля, якому задана позиція виводу як «mod_fpslideshow»;
- до переваг системи можна віднести те, що всі компоненти, модулі, плагіни і шаблони можна написати самому, розмістити їх в структурованому каталозі розширень або відредагувати існуюче розширення на свій розсуд;
- відбувається регулярний вихід оновлень. Існує публічний «баг-трекер» (система відслідковування помилок). (Див. список офіційних трекерів). Існують також трекери міграції зі старих версій Joomla, трекер побажань розширення функціоналу і так далі, де користувачі Joomla можуть залишати зауваження з приводу роботи CMS, які згодом вивчаються її розробниками і при необхідності включаються в чергове оновлення Joomla, вирішуючи ті чи інші проблеми;
- починаючи з версії 1.6 вбудована багатомовність;
- починаючи з версії 2.5 розширена підтримка баз даних. Реалізована підтримка Microsoft SQL Server, а з версії 3.0 - PostgreSQL[53].Надалі планується додати підтримку Oracle, SQLite. Оскільки оновлення з'являються постійно, бажано спостерігати за ними на офіційному сайті компанії.

Можливості адміністрування:

- для кожної динамічної сторінки можна створити свій опис і ключові слова з метою підвищення рейтингу в пошукових системах;
- початок та закінчення публікації будь-яких матеріалів можливо запрограмувати за календарем;
- можливість обмежити доступ до певних розділів сайту тільки для зареєстрованих користувачів, а з виходом Joomla 1.6 доступ як до розділу, так й до певного матеріалу з точністю до конкретного зв'язку матеріал↔користувач;
- схеми розташування елементів, що налаштовуються по областях шаблону;
- різні модулі (останні новини, лічильник відвідувань, докладна статистика відвідувань, гостьова книга, форум та інші);
- у версії 1.6 була сильно покращена система установки і управління розширеннями. Тепер можливо одночасно встановлювати кілька розширень, об'єднаних в один інсталяційний пакет. Більше того, реалізована можливість автоматичного оновлення встановлених розширень (за умови, що розробник задіює цей механізм);
- у версії 1.6 з'явилася можливість публікації вмісту на декількох мовах;
- у версії 1.6 з'явилася можливість визначення часу початку і завершення публікації модулів. Так само в новій версії Joomla поліпшені можливості по управлінню відображенням вмісту;
- можливість створення не однієї, а декількох форм зворотного зв'язку для кожного контакту;
- модуль прийому новин, статей і посилань від віддалених авторів;
- ієрархія об'єктів;
- менеджер розсилки новин. Підтримка більш ніж 360 служб розсилки новин по всьому світу;
- вбудований візуальний редактор TinyMCE;
- ЧПУ - «Загальнозрозумілі URL»;
- більше 15000 готових розширень.

✓ ***Drupal*** – система управління вмістом, що також використовується як каркас для веб-додатків (CMF). Система написана на мові PHP та використовує як сховище даних реляційну базу даних (підтримуються MySQL, PostgreSQL та інші). Drupal є вільним програмним забезпеченням, захищеним ліцензією GPL, й розвивається зусиллями ентузіастів зі всього світу.

Технічні можливості:

- архітектура Drupal застосовуватися при побудови різних типів сайтів – від блогів й сайтів з новинами до інформаційних архівів або соціальних мереж. Наявну за замовчуванням функціональність можна збільшувати підключенням додаткових розширень – «модулів» в термінології Drupal.

Найбільш важливі функції, надані Drupal «з коробки»:

- єдина категоризація всіх видів наповнення (таксономія) – від форумних повідомлень до блогів та статей;
- широкий набір властивостей при побудові рубрикаторів: списки, ієрархії, синоніми, споріднені категорії та інше;
- вкладеність категорій будь-якої глибини;
- пошук по наповненню сайту, в тому числі пошук по таксономії та користувачам;
- розмежування доступу користувачів до матеріалів (рольова модель);
- динамічну побудову меню;
- підтримка XML-форматів:
  - виведення документів в RDF / RSS;
  - агрегація матеріалів з інших сайтів;
  - BlogAPI для публікації матеріалів за допомогою зовнішніх програм;
- авторизація через OpenID;
- символьні осмислені URL (інакше «людино-зрозумілі» або “загально-зрозумілі” - ЧПУ);
- переклади інтерфейсу сайту на різні мови, а також підтримка ведення різномовного контенту;
- можливість створення сайтів з пересічним вмістом (наприклад загальною базою користувачів або загальними налаштуваннями);
- роздільні конфігурації сайту для різних віртуальних хостів (мультісайтинг), у тому числі власні набори модулів і тем оформлення для кожного під сайту;
- повідомлення про випуск оновленнях модулів.

В Drupal пропонується гнучка схема організації структури сайту на основі таксономії.

Таксономія – механізм, що дозволяє створювати довільну кількість тематичних категорій для наповнення сайту та асоціювати з модулями, що забезпечують введення та виведення інформації. Категорії можуть представляти плоскі або ієрархічні списки, або складні структури, де елемент може мати декілька «батьківських» та кілька дочірніх елементів.

За допомогою подібної схеми одними й тими ж модулями можлива організація різних варіантів структуризації вмісту. Наприклад, легко створюється список «ключових слів» для всіх документів сайту та ін.

Інша парадигма з'явилася зі створенням в Drupal розширення Content Construction Kit (CCK). CCK дозволяє доповнювати документи новими полями різних типів – від полів введення URL і email, до полів зберігання та відображення мультимедійних файлів. Також за допомогою додаткових модулів до CCK (наприклад Node reference) можна організувати зв'язки між документами, не використовуючи механізм таксономії.

ТЕМА 8

РЕГРЕСІЙНЕ ТЕСТУВАННЯ

8 РЕГРЕСІЙНЕ ТЕСТУВАННЯ (REGRESSION TESTING)

8.1 Основні визначення

Ручне тестування – це кропіткий й часом рутинний процес. Однією з проблем є те що при внесенні змін в код складно передбачити які тести слід виконати заново, щоб переконатися що все працює так як слід. Для цього вдаються до регресійного тестування та повторного прогону всіх тестів.

Регресійне тестування – це вид тестування спрямований на перевірку змін, зроблених в додатку або навколишньому середовищі (усунення дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатки), для підтвердження того факту, що існуюча раніше функціональність працює як й раніше. Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки та тестування. Це дає гарантію того, що зміни в новій версії програми не пошкодили вже існуючу функціональність.

З досвіду розробки ПЗ відомо, що повторна поява одних й тих же помилок – випадок досить частий. Іноді це відбувається через слабку техніки управління версіями або з причини людської помилки при роботі з системою управління версіями. Але настільки ж часто вирішення проблеми буває «недовго живуть»: після наступного зміни в програмі рішення перестає працювати. І нарешті, при переписуванні якої-небудь частини коду часто спливають ті ж помилки, що були в попередній реалізації.

Тому вважається хорошою практикою при виправленні помилки створити тест на неї і регулярно проганяти його при подальших змінах програми. Хоча регресійне тестування може бути виконано і вручну, але найчастіше це робиться за допомогою спеціалізованих програм, що дозволяють виконувати всі регресійні тести автоматично. У деяких проектах навіть використовуються інструменти для автоматичного прогону регресійних тестів через заданий інтервал часу. Зазвичай це виконується після кожної вдалої компіляції (у невеликих проектах) або щоночі або щотижня.

Регресійне тестування є невід'ємною частиною екстремального програмування. У цій методології проектна документація замінюється на розширюване, повторюване і автоматизоване тестування всього програмного пакету на кожній стадії процесу розробки програмного забезпечення.

Регресійне тестування може бути використано не тільки для перевірки коректності програми, а також використовується для оцінки якості отриманого результату. Так, при розробці компілятора при прогоні

регресійних тестів розглядається розмір одержуваного коду, швидкість виконання й час компіляції кожного з тестових прикладів

У Microsoft Test Manager визначено основний тестовий план, й для кожного PBI відповідні тести функцій (рис.8.1) та збирається білд й починаємо тестувати додаток згідно з планом (рис.8.2).

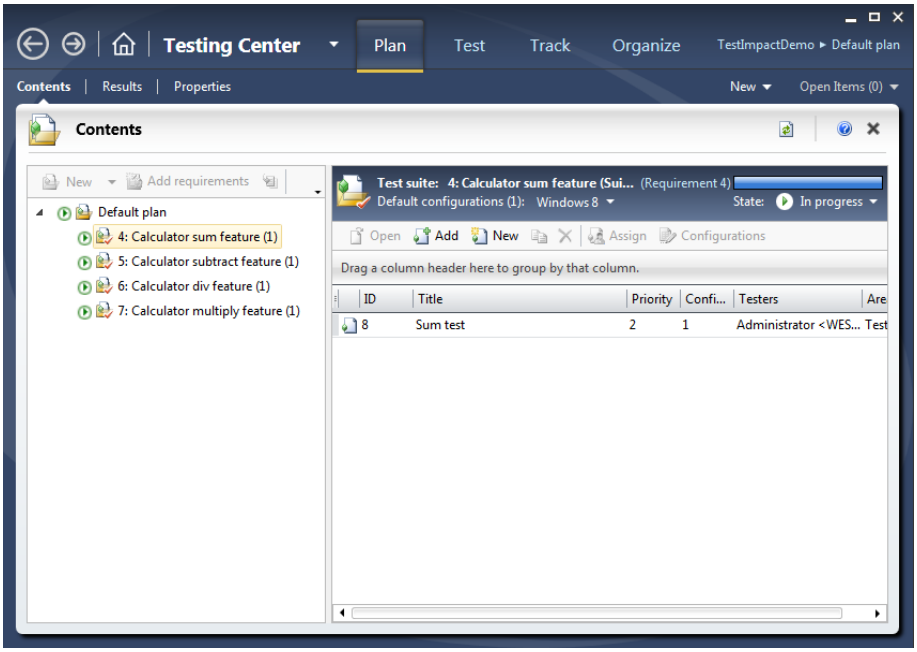


Рис. 8.1. Визначення основного тестового плану та відповідні тести функцій

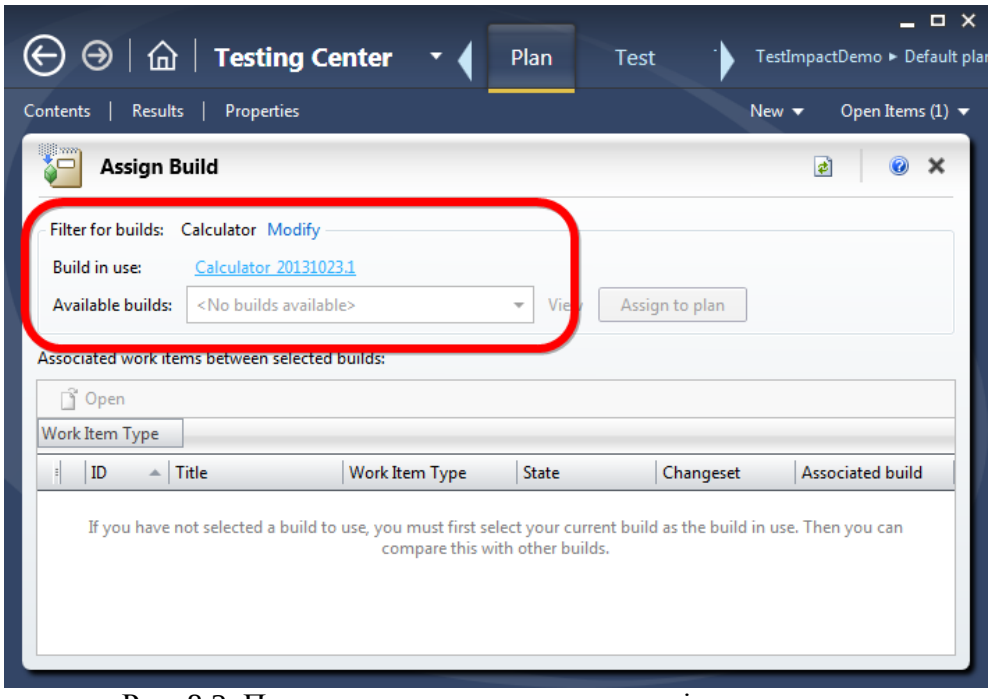


Рис. 8.2. Початок тестування додатоку згідно з планом

Проходження всіх особливостей проекту представлене на рис.8.3.

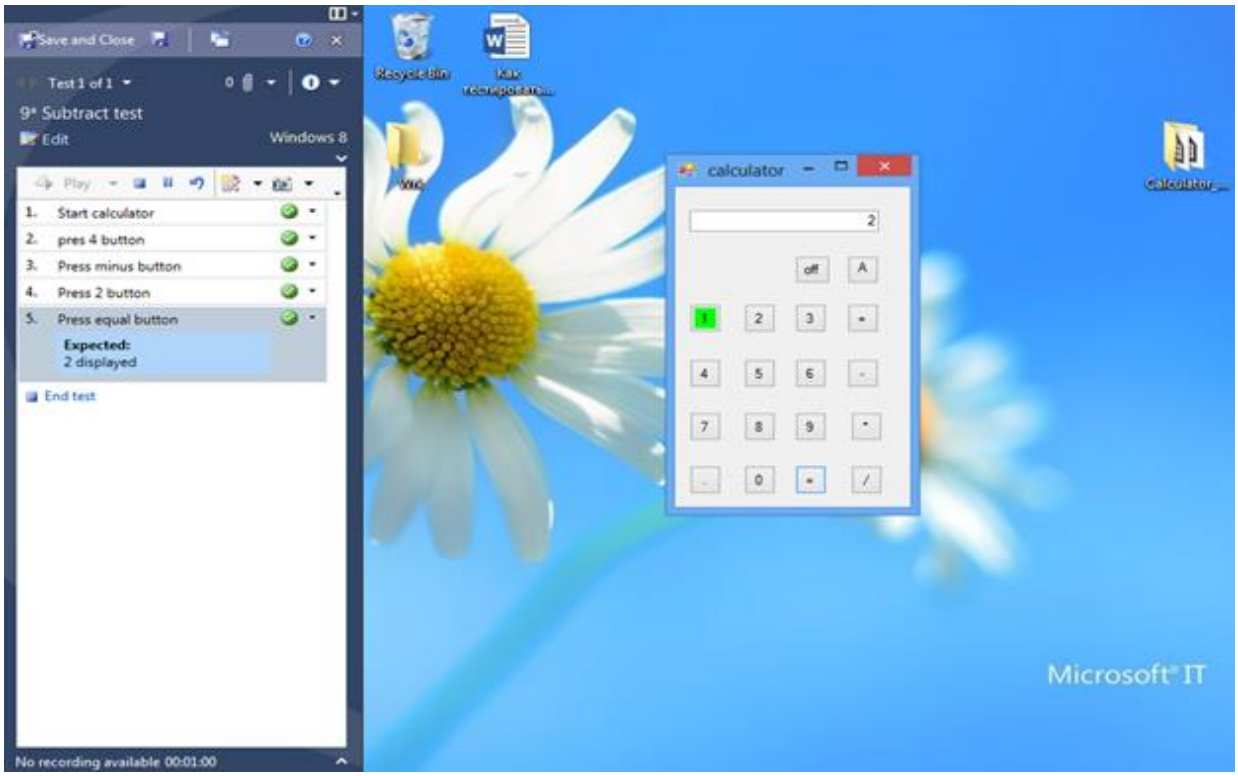


Рис. 8.3. Проход всіх тестів для продукту

На рис. 8.4. представлені результати проходження всіх особливостей проекту.

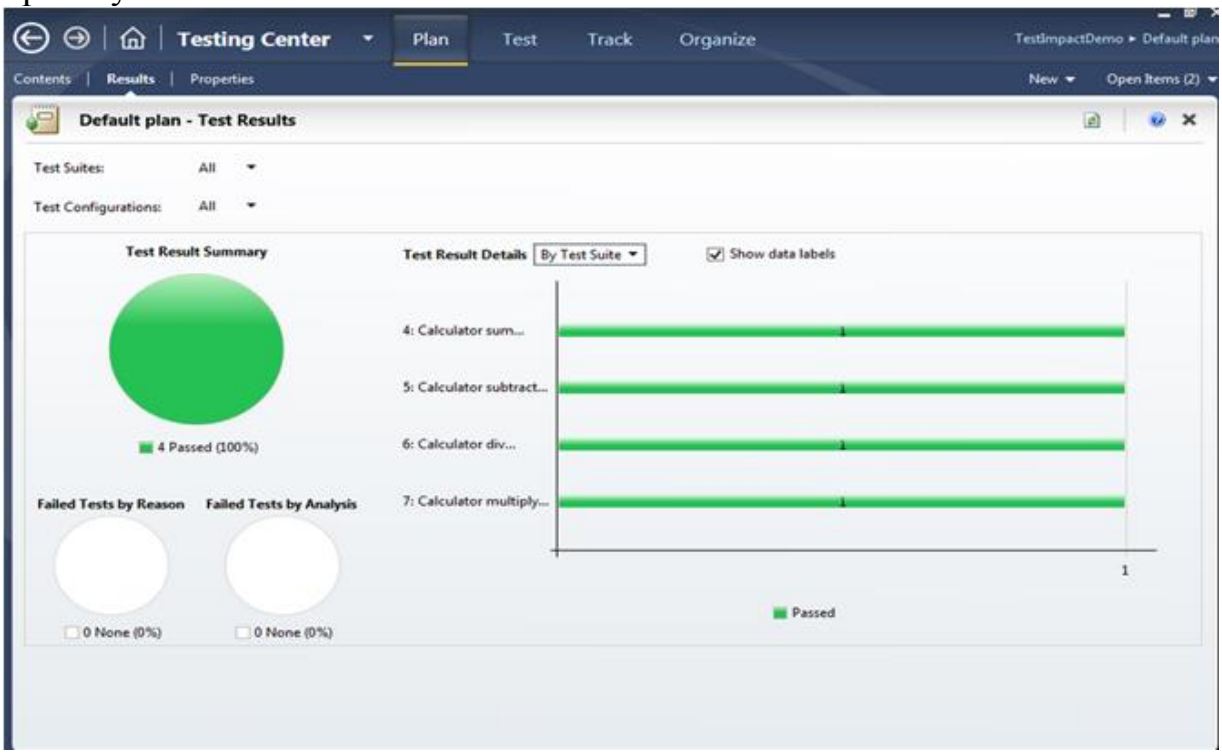


Рис. 8.4. Результати проходженння

Вносимо необхідні зміни в код продукту (рис.8.5).

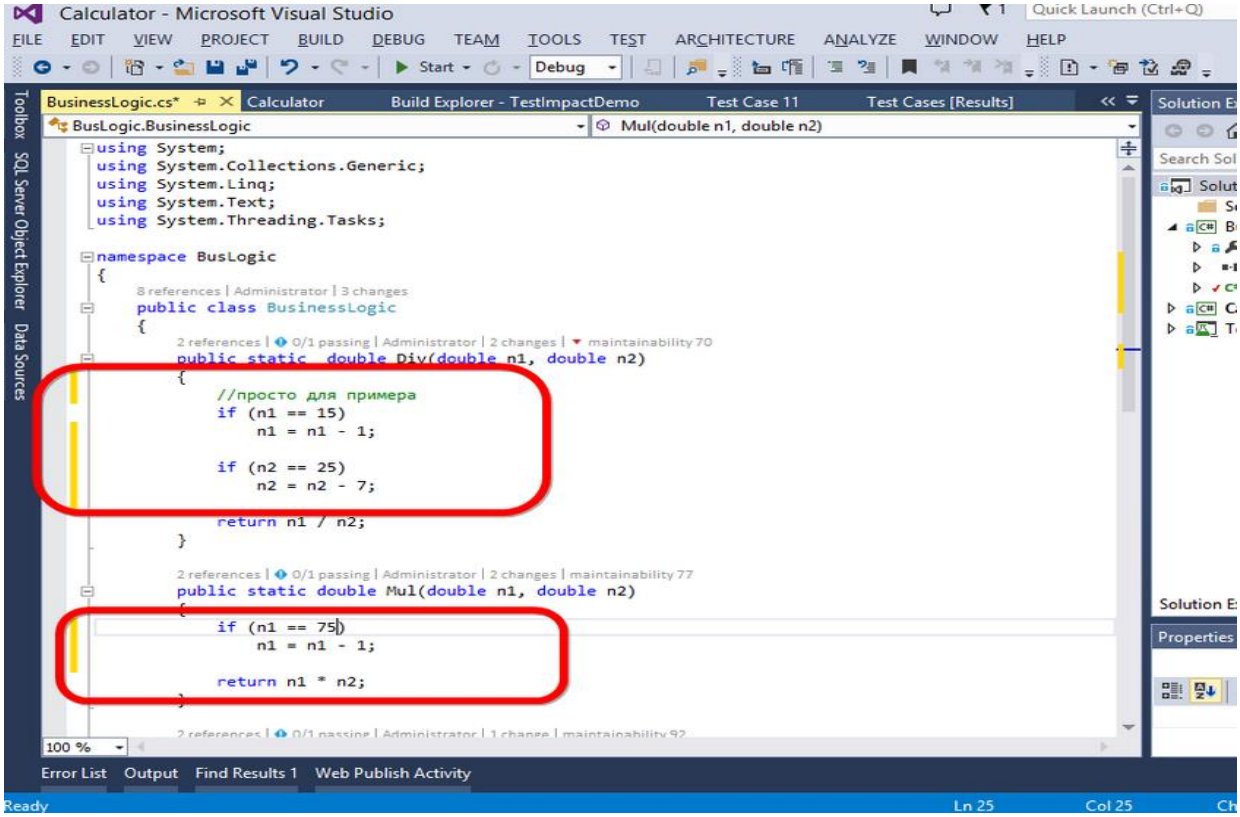


Рис 8.5. Зміни в коді програми

Після внесення змін необхідно провести повторне тестування (рис.8.6).

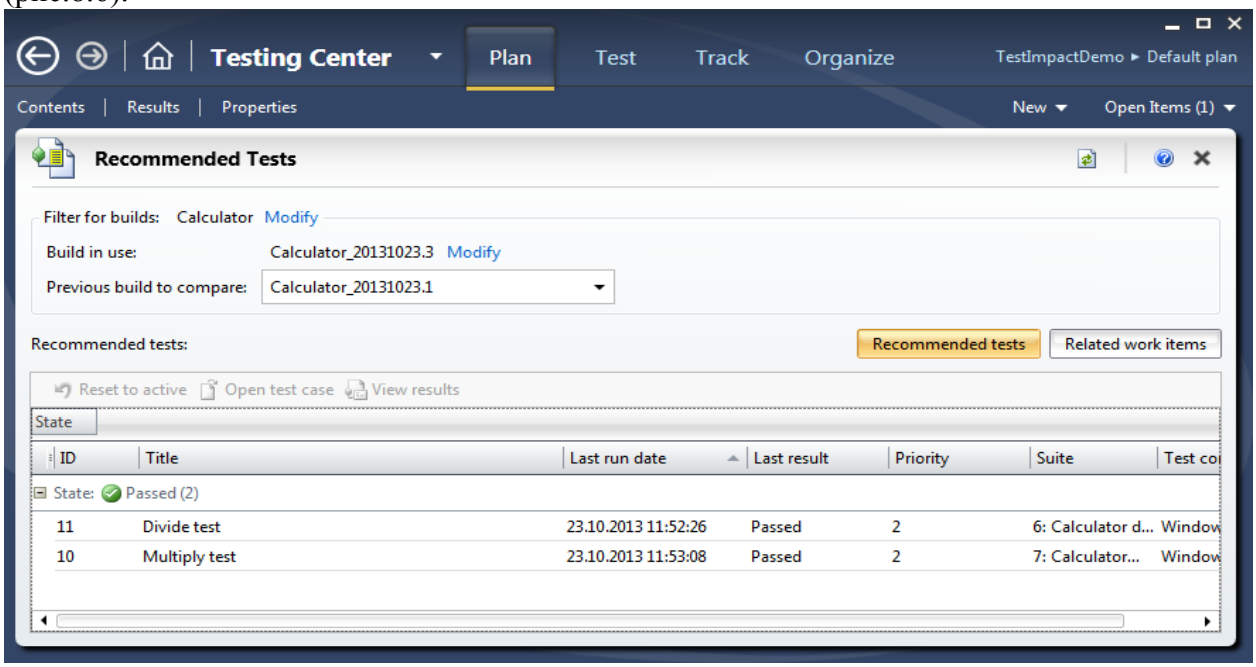


Рис. 8.6. Тестування нових функцій

Тим самим тестувальник може значно заощадити час на перевірку білду, й провести тільки ті тести, в код програми яки було внесено зміни програмістами.

Залежно від контексту терміну "Регресійне тестування", може мати різний зміст. Сем Канер, наприклад, описав три основних визначення регресійного тестування:

- ***Регресія багів (Bug regression)*** – спроба довести, що виправлена помилка насправді не виправлена.
- ***Регресія старих багів (Old bugs regression)*** – спроба довести, що нещодавня зміна коду або даних зламало виправлення старих помилок, тобто старі баги стали знову відтворюватися.
- ***Регресія побічного ефекту (Side effect regression)*** – спроба довести, що нещодавня зміна коду або даних зламало інші частини додатку що розробляється.

На запитання, коли і як проводити регресійне тестування, та які тести ставити в першу чергу відповісти не просто. Усе визначається видом розроблювального ПЗ, тривалістю життєвого циклу, термінами тестування, кількістю членів команди.

Загальні положення проведення регресійного тестування:

1. Регресійне тестування проводиться в кожній новій версії.
2. Починають регресійне тестування з тестів верифікації версії. Якщо програма приходить від розробника у вигляді повноцінної інсталяції, то тести верифікації починаються з перевірки інсталяції, після чого проводиться короткий набір тестів функціональності. Якщо хоча б один з тестів failed, версія передається на доопрацювання, регресійне тестування припиняється, а тестер повертається до тестування останньої "робочої" версії.
3. Після успішного проходження тестів верифікації версії, проводять серію тестів верифікації багів.
4. З тестів регресії проводять лише ті, які пов'язані зі зміненою в новій версії ділянкою коду.
5. Аналогічним чином (п.4) відбираються тести в групу регресії на "закритих" багах.
6. Тести регресії, виконані успішно (pass) двічі вважаються "закритими". Подальше їх використання проводиться так, як описано в пункті 4.
7. Для тестів регресії, які передбачається проводити більше 3-5 разів рекомендується писати скрипти для автоматизації процесу. Це відноситься до всіх груп тестів регресії.
8. Відбір тестів для фінального регресійного тестування здійснюється за такими принципами:

- у першу чергу відбирають тести забраковані (failed) два та більше разів, а також й ті, які виявляли баги, вимагають доопрацювання (re-do);
  - у другу чергу відбираються тести забраковані один раз, й успішно пройдені повторно;
  - далі відбираються всі тести, які були пройдені успішно (pass), але проводилися тільки один раз.
  - потім проводяться всі інші тести, залежно від поставленого завдання.

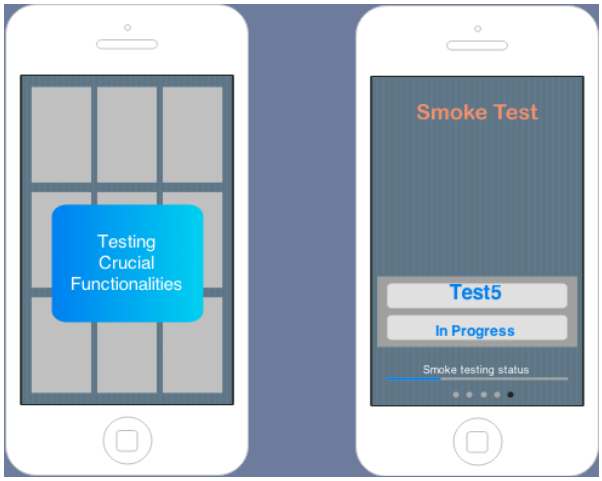
Для наочності при проведенні регресійного тестування можна використовувати наступну таблицю:

№ теста	№ версии	№ бага	№ версии	№ бага

Ключові переваги регресійного тестування:

- при регулярному проведенні регресійного тестування – значне скорочення кількості дефектів в системі до моменту релізу;
- виключення деградації якості системи при зростанні функціональності;
- зменшення ймовірності критичних помилок в дослідно-промисловій експлуатації.

8.2 Smoke testing



Іронічна назва цього виду тестування походить від швидкого способу перевірки інженерами електроприладів: якщо при включенні в розетку пішов дим - прилад потребує доопрацювання. Мета такого тестування перевірити, що після чергової збірки програмного продукту немає явних грубих помилок.

Smoke тести повинні бути швидкими та легкими для того, щоб

їх можна було запускати часто.

Installation / Build Verification Scenario –викачується білд, перевіряється інсталяція.

UI Verification – відкривається кожне вікно програми та перевіряється, що всі кнопки існують й немає змін. Якщо тести валяться на цьому етапі, то продовжувати тестування немає сенсу. Потрібно спочатку виправити зміни.

Functional – кілька тестів з кожної функціональності.

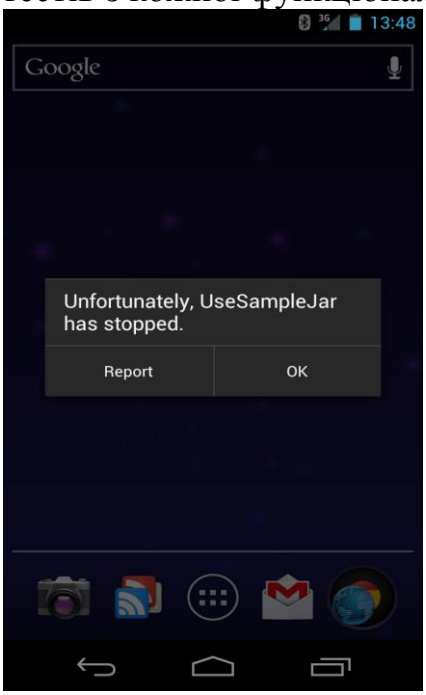


Рис. 8.7. Креш додатки при запуску

Сучасні методології розробки, практикують підхід безперервної інтеграції (Continuous Integration), який передбачає щоденну збірку програмного продукту. Збірки не завжди бувають належної якості, й можуть містити помилки в роботі критичного для бізнесу функціоналу, тому перевірка ключового функціоналу повинна здійснюватися безпосередньо після збирання та перед передачею на тестування. Це дозволяє скоротити втрату часу на тестування збірки, що містить блокуючі помилки.

Висновок про працездатність основних функцій робиться на підставі результатів поверхневого тестування найбільш важливих модулів програми на предмет можливості виконання необхідних завдань та наявності критичних і блокуючих дефектів, які швидко знаходяться. У разі відсутності таких дефектів димове тестування оголошується пройденим, і додаток передається для проведення повного циклу тестування, в іншому випадку, димове тестування оголошується проваленим, і додаток йде на доопрацювання.

Аналогами димового тестування є Build Verification Testing і Acceptance Testing, що виконуються на функціональному рівні командою тестування, за результатами яких робиться висновок про те, приймається чи ні встановлена версія програмного забезпечення в тестування, експлуатацію або на поставку замовнику.

Ще одне використання **Smoke -тест -** це перевірка інтеграції між системами і коректності їх перенесення на нові стенди. А так само з метою визначення коректності настройки взаємодії між системами.

Приклади:

а) Помилки інсталяції: якщо програмний продукт не встановлюється, його тестування, швидше за все, виявиться неможливим.

б) Помилки при з'єднанні з базою даних, актуально для архітектури клієнт-сервер.

8.3 Sanity testing

Sanity testing або санітарне тестування – це вузьконаправлене тестування, що є достатнім для доказу того, що конкретна функція працює згідно заявленим в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або навколишньому середовищу. Зазвичай виконується вручну.

Відмінність санітарного тестування від димового (Sanity vs Smoke testing). У деяких джерелах помилково вважають, що санітарне та димове тестування – це одне й теж. Насправді, ці два види тестування мають два різних "вектори руху", напрямки в різні боки. На відміну від димового (Smoke testing), санітарне тестування (Sanity testing) направлено в глиб функції, що перевіряється, в той час як димове направлено в ширину, для покриття тестами якомога більшої кількості функціоналу в найкоротші терміни.

Наприклад, Gmail додав нову функцію «Авто підтвердження». Ця функція працює як повідомлення про те що email успішно доставлений адресату. Оскільки це новий функціонал, необхідно протестувати, та є

стабільний білд з новим функціоналом який тестуємо. Це й називається санітарне тестування

8.4 Build Verification Test

Тестування спрямоване на визначення відповідності, випущеної версії, критеріям якості для початку тестування. Є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Може проникати далі, залежно від вимог до якості випущеної версії.

Відразу після отримання нотифікації про встановлення нової версії, команда тестування повинна негайно приступити до приймального тестування. Це можуть бути як автоматизовані, так й ручні тести, суть яких полягає в тому, щоб у стислі терміни охопити найбільший функціонал та прийняти рішення, що дана версія задовольняє критеріям якості та може бути прийнята для проведення повного або запланованого тестування.

У випадку, якщо встановлена версія не відповідає критеріям якості, команда тестування має право забракувати (reject), приклавши список помилок, що є причиною відмови від подальшого тестування.

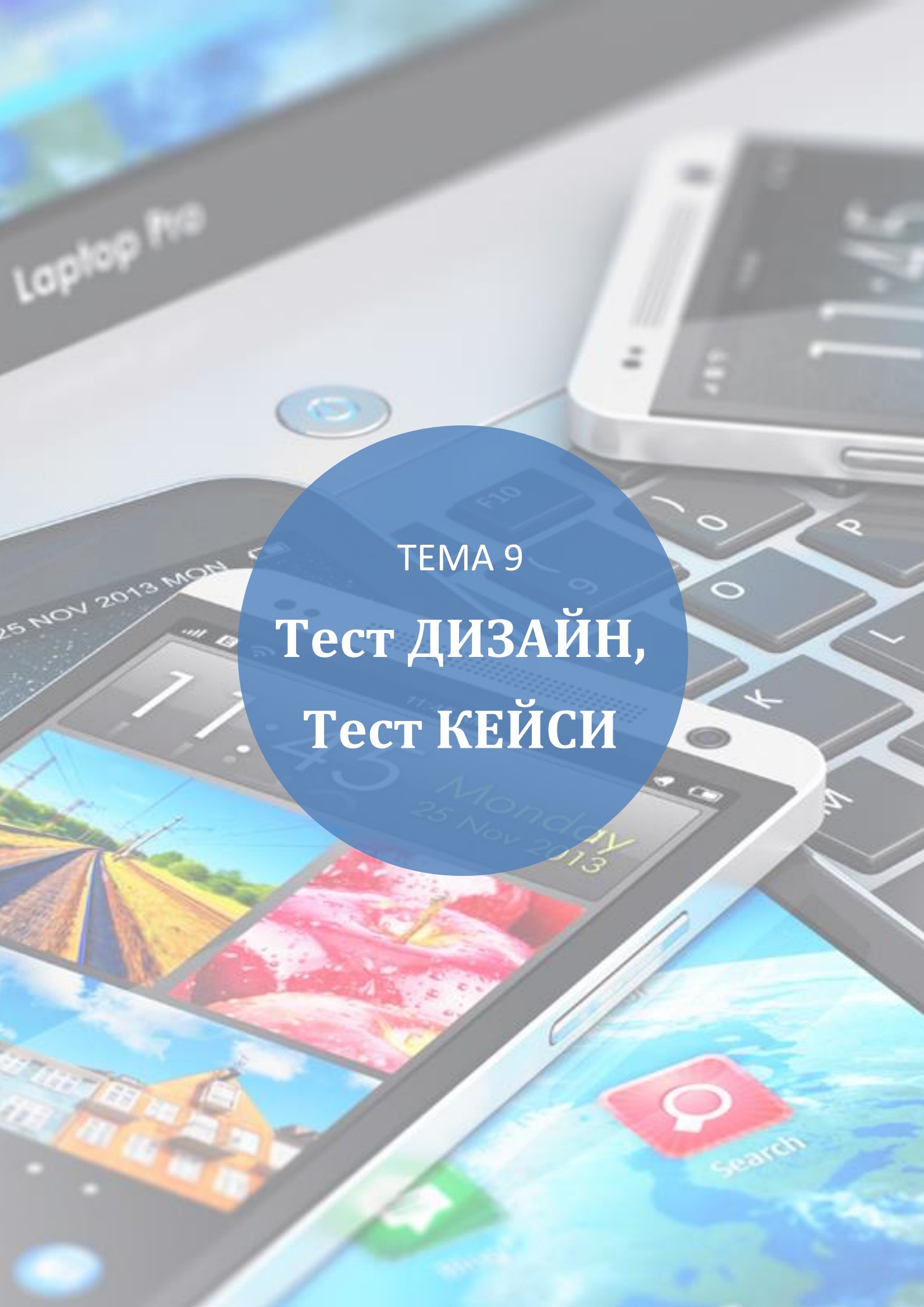
Отримавши повідомлення, що встановлена версія була забракована (rejected) командою тестування, керівник розробки повинен провести аналіз причин.

В цьому випадку існують наступні варіанти подальших дій:

- якщо з'ясується, що помилки, які послужили причиною відмови, з'явилися внаслідок неправильної установки, тобто з вини білд майстра, то рекомендується провести переустановлення версії (Redeploy);
- якщо ж установка була виконана коректно та встановлена версія дійсно не задовольняє критеріям якості, то проводиться відкат на попередню версію (Roll Back);
- у випадку, якщо команда тестування приймає встановлену версію, то на всю проектну групу розсилається повідомлення, що версія прийнята в тестування.

Добре підготовлена збірка повинна відповідати наступним умовам:

- успішно компілюються всі файли, бібліотеки та інші компоненти;
- посилання на всі файли, бібліотеки та інші компоненти дійсні;
- не містяться ніяких стійких системних, що виключають можливість правильної роботи прикладної програми, що блокують роботу помилок;
- усі димові тести проходяться.



TEMA 9

Тест ДИЗАЙН, Тест КЕЙСИ

9 ТЕСТ ДИЗАЙН (TESTING DSING) ТА ТЕСТ КЕЙСИ (TEST CASE)

Тестувальники повинні тестувати додатки за допомогою всіх можливих і мислимих засобів. Серед цих пристосувань числяться і тест кейси *(test cases)*, які допомагають провести перевірку продукту без ознайомлення з усією документацією. Написаний один раз, зручний у підтримці тест-кейс *(test case)* заощадить багато часу і сил тестувальникам.

9.1 Основні поняття та визначення

9.1.1 Тест дизайн (test design)

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

Головні цілі тест дизайну (test design) [68]:

- придумати тести, які виявлять найбільш серйозні помилки продукту;
- мінімізувати кількість тестів, необхідних для знаходження більшості серйозних помилок.

До завдань тест дизайну відносять:

- написання тест - кейсів;
- аналіз ризиків;
- створення приймальних перевірок;
- опис процесу тестування;
- складання списку функцій продукту;
- аналіз вимог;
- розстановка пріоритетів тестування;
- аналіз скарг користувачів;
- побудова таблиць прийняття рішень.

План роботи над тест - дизайном (test design) [68]:

- аналіз наявних проектних артефактів: документація *(специфікації, вимоги, плани),* моделі, виконуваний код і т.д.;
- написання специфікації по тест дизайну *(test design specification);*

- проектування і створення **тестових випадків** *(test cases)*

Ролі, відповідальні за тест дизайн:

- тест аналітик – визначає ***"ЩО тестувати?"***;
- тест дизайнер – визначає ***"ЯК тестувати?"***.

Навички тест дизайнера

Розробка тестів **–**це непросте та тривале заняття, яке вимагає серйозного поглиблення, тому виділяють наступні **навички, необхідні тест дизайнерові*:***

- ***вміння поділяти систему на складові*** (робити декомпозицію), тобто, потрібно вміти бачити систему як ціле, але й уміти розкладати її на складові частини.

Це дуже корисна навичка для проведення функціонального тестування, де перевіряється кожна особливість продукту.

- ***вміння збирати та аналізувати вимоги до продукту***, якщо немає формально описаних вимог - потрібно вміти їх збирати у розробників, у аналітиків, у користувачів;

- ***вміння розставляти пріоритети*** **–** вміти відрізняти більш важливе від менш важливого, і розставляти пріоритети тестування;

- ***вміння формулювати свої думки*** *(письмово й усно)* – вміння важливе для тестувальника в принципі, яке значно допоможе при створенні тест кейсів;

- ***знання і вміння застосовувати техніку тест дизайну на практиці.***

Техніки тест дизайну *(test design technics)* містять теоретичну частину *(деякі рекомендації з використання),* але головна їх частина – практична. Тобто, кожна техніка дає поради щодо застосування. Тому особливо важливо не тільки вивчити техніку, а й спробувати на практиці. Техніки можуть містити рекомендації не тільки по тест дизайну (розробці тестів), але й з виконання тестів.

Найпоширеніша техніка тест - дизайну (test design) [69]:

- ***еквівалентний поділ (equivalence partitioning - EP),*** як приклад, у вас є діапазон допустимих значень від 1 до 10, ви повинні вибрати одне вірне значення усередині інтервалу, скажімо, 5, і одне невірне значення поза інтервалом - 0.

- ***аналіз граничних значень (boundary value analysis - BVA),*** якщо взяти приклад вище, в якості значень для позитивного тестування виберемо мінімальну і максимальну межі (1 і 10), і значення більше і менше кордонів (0 і 11). Аналіз граничних значень може бути застосований до полів, записів, файлів, або до будь-якого роду сутностей, які мають обмеження.

- ***причина / наслідок (cause/effect - CE)*** – введення комбінацій умов (причин), для отримання відповіді від системи (Наслідок). Наприклад, ви перевіряєте можливість додавати клієнта, використовуючи певну 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)*** – коли тест аналітик використовує свої знання системи і здатність до інтерпретації специфікації на предмет того, щоб "передбачити" за яких вхідних умов система може видати помилку. Наприклад, специфікація каже: "користувач повинен ввести код". Тест аналітик думатиме: "Що, якщо я не введу код?", "Що, якщо я введу неправильний код?", І так далі. Це і є передбачення помилки;

- ***вичерпне тестування (exhaustive testing - ET)*** – це крайній випадок. В межах цієї техніки ви повинні перевірити всі можливі комбінації вхідних значень, і в принципі, це має ідентифікувати всі проблеми. На практиці застосування цього методу не представляється можливим через величезну кількість вхідних значень.

Вважається, що кожна техніка може укладати рекомендації з семи питань:

1. Що тестується?
2. Наскільки ретельно?
3. Ким тестується?

4. Які проблеми шукаються?
5. Як тестується?
6. Як оцінюється результат тестів?
7. Чи існує певна мета тестування?

Але кожна техніка зазвичай дає відповіді не на всі питання, а на 2-3. Решта нюансів лишається на розсуд тих, хто буде використовувати техніку.

Завдання тест аналітиків і дизайнерів зводиться до того, щоб, використовуючи різні стратегії і техніки тест-дизайну, створити набір тестових випадків, що забезпечує оптимальне ***тестове покриття (test coverage)*** - одна з метрик оцінки якості тестування, що представляє собою щільність покриття тестами вимог або виконуваного коду тестованого додатку. Однак, на більшості проектів ці ролі не виділяються, а довіряють звичайним тестувальникам, що не завжди позитивно позначається на якості тестів, тестуванні і, як з цього випливає, на якості програмного забезпечення (кінцевого продукту).

9.1.2 Тест кейси (test cases)

Тестовий випадок (test case) - це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується, або її частини.

Тестовий набір (test suite):

- набір тестів, що реалізують бізнес-завдання, виконуване тестованою системою;
- тестовий набір включає, окрім тестових сценаріїв, ще й тестові дані або правила їх генерації.

До стандартних атрибутів тест-кейса (test case) відносять ***(рис. 9.1):***

- ***ідентифікатор (identifier)*** – включає в себе як ідентифікатор ТС - WPAD-15, який присвоюється автоматично, так і ім'я ТЗ: Вхід в систему, який вказується тестувальником;

- ***ім'я*** тест-кейсу повинно бути осмисленим та повинно дозволяти зрозуміти призначення тест-кейсу;

- ***версія (version) 3*** – це версія, присвоюється автоматично;

- ***дата*** створення ... ім'я автора та дата останньої зміни ... ім'я користувача, що вніс останні зміни *(created on ... by, last modified ... by)* - визначається автоматично;

- ***коротка інформація*** *(summary)* – описується тестувальником те, що буде перевірятися;

- ***попередні умови*** *(preconditions)* – список всіх необхідних підготовчих дій *(налаштування програми, середовища тестування)* для виконання даного ТЗ;

- ***кроки для відтворення*** *(step actions)* – потребує активних дій: «зробіть те, зробіть це». Повинен бути коротким і зрозумілим;

- ***очікувані результати*** *(expected results)* - визначає правильну реакцію програми на виконання даного кроку. Повинен бути зрозумілим, однозначним, простим;

- ***спосіб виконання*** *(execution type)* – можливі два варіанти виконання: ручне і автоматизоване;

- ***важливість тесту*** *(test importance)* – відображає те, наскільки даний тест і вимога, яка перевіряється, критичні для перевірки. Можливі три варіанти: high, medium, low;

- ***ключові слова*** *(keywords)* – використовуються для спрощення пошуку тестів за ключовими словами;

- ***використовуваність в тест планах*** *(test plan usage)* – список тест планів, до яких включено даний тест кейс;

- ***додані файли*** *(attached files)* – може бути прикладений відповідний файл для тестування або файл з тестовими акаунтами.

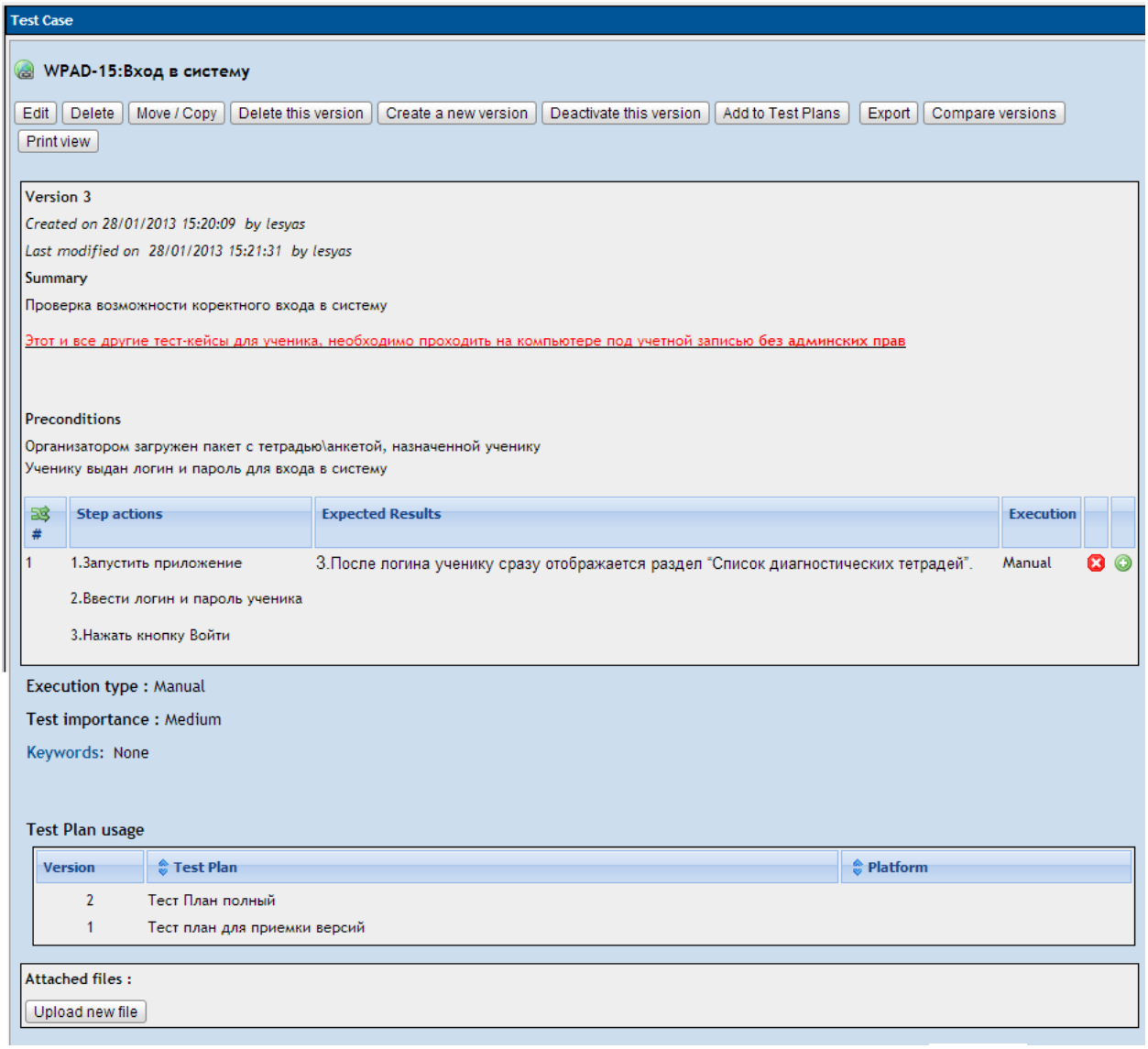


Рис.9.1. Опис тест-кейсу в системі управління тестуванням ***Testlink***

Якісний тест-кейс (test case) повинен відповідати таким критеріям [13]:

- ***тест кейс повинен бути точно визначеним;***

- ***тест кейс повинен бути відтворюваним***, тобто можна відтворити нескінченне число разів, не руйнуючи середу тестування (тестові дані, програми, устаткування), й тест кейс має відтворювати знайдені помилки при наступному виконанні, а також відтворення в різних конфігураціях середовища тестування;

- ***набір тестів не повинен бути надмірним***, якщо два тести призначені для виявлення однієї і тієї ж помилки, немає сенсу проходити один сценарій двічі;

Така ситуація часто зустрічається, коли над створенням тесту працюють двоє або більше фахівців.

- ***тест повинен бути найкращим у своїй категорії:*** у групі схожих тестів одні можуть бути ефективніше інших, тому, вибираючи тест, потрібно взяти той, який з найбільшою ймовірністю виявить помилку;
- ***тест кейс повинен бути коротким і простим:*** в описі кроків і очікуваних результатів повинні бути тільки необхідні слова (подумайте про людину, якій це все доведеться читати і виконувати), а також слід уникати довгих, заплутаних складнопідрядних речень (краще розділити один крок на кілька), сленгу, метафор, алегорій;

- ***тест кейс повинен бути однозначним і зрозумілим:*** значення кожного слова повинно розумітися в єдиному можливому сенсі й слід також виключити слова, які не дають чіткого розуміння: «добре», «очевидно», «погано», «і так далі»;

- ***тест кейс не повинен бути занадто простим або занадто складним;***

- іноді можна об'єднати два тести в один, але при цьому важливо не перестаратися.

Величезний і складний тест важко зрозуміти, важко виконати і довго створювати, тому краще всього дотримуватися золотої середини, розробляючи прості, але все ж не зовсім елементарні тестові приклади.

Тестовий випадок проходить ряд змін протягом розробки продукту, при цьому може бути змінений, автоматизований, вилучений; наочно зобразити життєвий цикл тестового випадку можна блок-схемою (рис. 9.2).

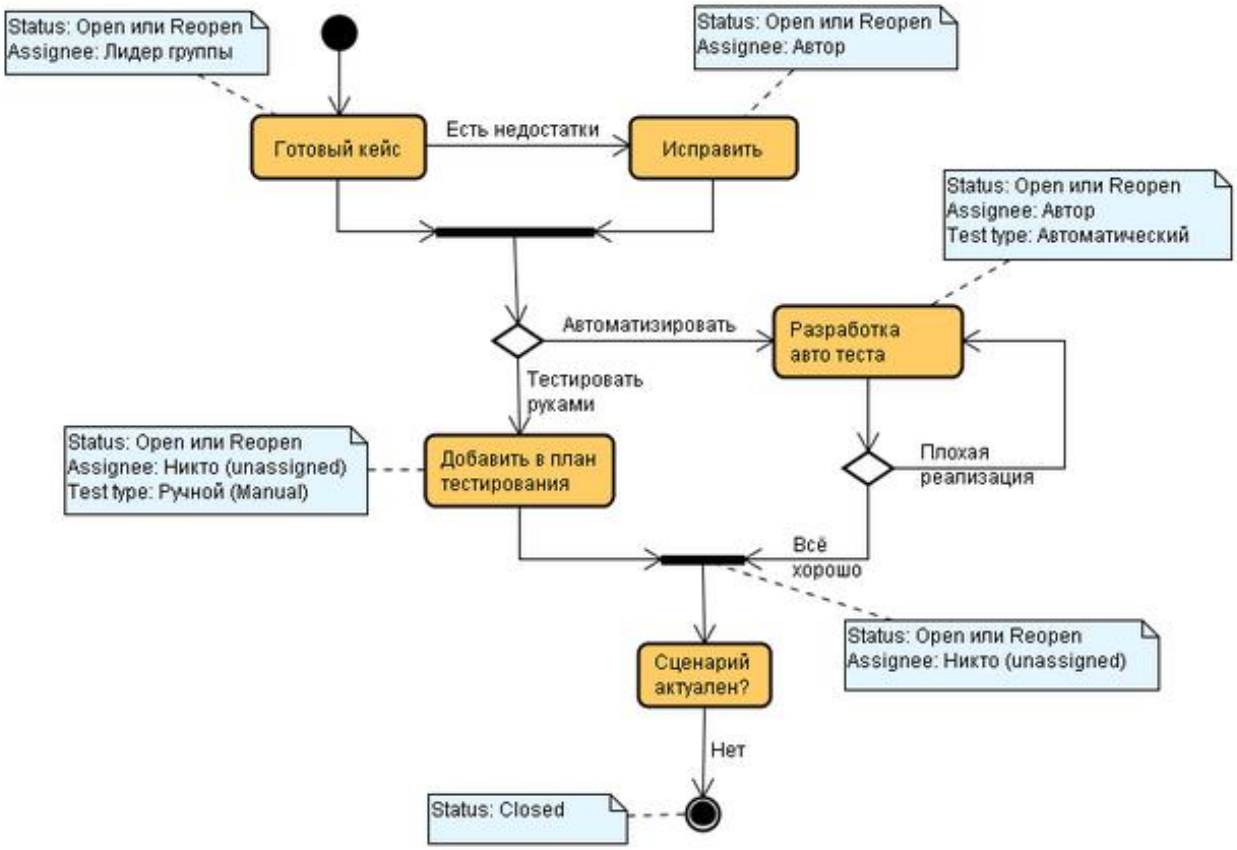


Рис. 9.2. Життєвий цикл тест-кейсу (test case)

Кожен вид тестування має як й переваги, так йнедоліки.

До переваг тестування за сценаріями відносять:

+ Вимагає менше навичок від тестера, який виконує тести;
+ Дає можливість легше оцінити тестове покриття;
+ Стабільність та незмінність тест-кейсів дає можливість простежити якість продукту з часом;
+ Забезпечує більш структурований підхід до тестування;
+ Як правило, виконання тестів не вимагає додаткових досліджень та спілкування з розробниками, аналітиками, роботи зі специфікаціями, що дає виграш за часом.

До недоліків тестування за сценаріями відносять:

– Тестування з тест-кейсами обмежує свободу тестера;
– Формує звички у тестуванні, тому що тестер звикає робити такі перевірки, як описані в тест-кейсах;

- Тест-кейси потихеньку втрачають свою ефективність і перестають знаходити баги;
- Постійне виконання одних і тих же тестів з часом стомлює;
- Вимагає багато часу на підтримку тест-кейсів: оновлення, видалення, виправлення;
- Тестер повільно розвиває своє знання продукту і контексту в порівнянні з дослідницьким тестуванням.

До переваг дослідницького тестування відносять:

+ Добре розвиває навички, дає тестеру більше свободи;
+ Не вимагає часу для написання тестових сценаріїв, весь час приділяється тестуванню;
+ Заохочує вивчення тестером продукту.

До недоліків дослідного тестування відносять:

- Недосвідченим тестерам в перший час буває важко через те, що вони надані самі собі, у них немає хоча б прикладів тестів, які потрібно виконувати;
- Без тест-кейсів складніше оцінити тестове покриття;
- Вивчення продукту забирає час від безпосереднього тестування.
- Найкраще комбінувати ці два підходи разом: використовувати і дослідницьке тестування, і тестування за сценаріями.

Наприклад, можна використовувати тестування з тест кейсами для регресійного тестування, а новий функціонал тестувати дослідницьким методом.

При складанні тест-кейсів необхідно дотримуватися наступних принципів:

- теми тест кейса (за принципом що перевіряється? Де перевіряється? З яким типом даних (коректні, некоректні)?);
- наявності опису тест кейса;
- попередніх умов, де має бути вказано, яким шляхом відкрити ту форму, яка перевіряється в кроках;
- на кожен крок повинен бути очікуваний результат;
- в кроках зазначений тип даних, що вводяться (коректні, некоректні), у разі некоректних повинні бути наведені приклади таких даних (наприклад, для поля введення ел. пошти некоректними будуть не латинські символи, неприпустимі спец. Символи, без @, без домена, та ін.);

- тест кейс повинен бути завершеним і цілісним (наприклад, функція відновлення пароля повинна закінчуватися на успішній зміні пароля, а не на відправці листа з посиланням для зміни пароля).

9.2 Системи управління тестуванням (test management system) TestRail и Testlink

Системи управління тестуванням (test management system) використовуються для зберігання інформації про те, як належним чином проводити тестування, здійснення черговості проведення тестування відповідно до плану, а також для отримання інформації у вигляді звітів про стадії тестування і якості продукту, що тестується. Інструменти мають різні підходи до тестування і, таким чином, включають різні набори функцій. Зазвичай використовуються для планування ручного тестування, збору даних про результати проходження чек-листів та тест-кейсів, а також для отримання оперативної інформації у вигляді звітів.

Системи управління тестуванням допомагають оптимізувати процес тестування і забезпечують швидкий доступ до аналізу даних, засобам спільної роботи і більш якісну взаємодію між декількома проектними групами. Багато систем управління тестування включають можливість роботи з вимогами.

Після старту тестування проекту члени команди можуть взаємодіяти через одну із систем управління тестування шляхом створення тест-кейсів, чек-листів, призначаючи відповідальних за проходження їх осіб, що спрощує і покращує якість взаємодії осіб, що проводять тестування в рамках конкретного проекту. При створенні або проходженні тестів і чек-листів, користувачі можуть отримати доступ до різних функцій систем управління тестуванням, які автоматизують дану діяльність і благотворно впливають на швидкість і якість її виконання. Список популярних інструментів систем управління тестуванням наведено в таб. 9.1.

Таблица 9.1 Список популярних інструментів систем управління тестуванням

Продукт/ Характеристики	Sitechco	TestLink	TestRail
Автор	ООО Лаборатория качества	TeamTest	Gurock Software
Лицензия	Проприета́рное	GNU General Public License	Проприета́рное
Автоматизированные unit-тесты	Нет	Нет	Нет
Автоматизированные GUI-тесты	Нет	Нет	Нет
Ручное тестирование	Да	Да	Да
Менеджер дефектов	Да	Да	Да
Дата выпуска	2010	2004	Неизвестно
Комментарии	Интеграция с системами отслеживания ошибок, таких как Redmine и Jira	Интеграция с множеством систем отслеживания ошибок, в том числе с Bugzilla	Интеграция с системой отслеживания ошибок Jira
Ссылки	http://www.sitechco.ru	http://www.TestLink.org	http://www.gurock.com/testrail/

Інструменти управління тестуванням дають командам можливість консолідувати і структурувати процес тестування за допомогою однієї з систем управління тестуванням замість установки декількох додатків, які призначені для управління тільки одним процесом або його частиною. Деякі програми включають передові інструментальні панелі для ретельного відстеження ключових показників, що дозволяє легко отримувати необхідну інформацію про стадії процесу тестування і якість продукту, що тестується.

9.2.1 Системи управління тестуванням Testlink

TestLink – система управління тестами з веб-інтерфейсом, яка дозволяє імпортувати вимоги, генерувати на їх основі тестові сценарії або створювати тестові сценарії з нуля, генерувати тестові специфікації, звіти про тестування, призначати виконання тестів на фахівців з контролю якості, відстежувати їх активність в реальному часі, пов'язувати не пройдені тести з баг-трекінговою системою і збирати різного роду статистику.

До переваг системи управління тестами TestLink відносять:

+ доменна аутентифікація, для якої досить доставити модуль php5-ldap і трохи поправити конфігурацію;

+ оповіщення по e-mail про призначення завдань на тестування, коли призначаєте набори тестових випадків на спеціаліста з контролю якості, якому може автоматично приходити нотифікація на пошту;

+ генерація тестової документації по шаблону, наближеному до корпоративного;

+ інтеграція з Jira, що дає можливість пов'язувати внутрісистемні повідомлення (ticket) в Jira с заблокованими або не пройденими тестовими випадками.

При цьому інформація по стану внутрісистемного повідомлення в Jira постійно оновлюється в TestLink і відповідає актуальній. Плоди інтеграції також видно при генерації звітності, коли вибірка всіх не пройдених тестових випадків супроводжується інформацією про дефекти в баг-трекінговій системі.

Системи управління тестами TestLink має наступну структуру (рис.9.3).

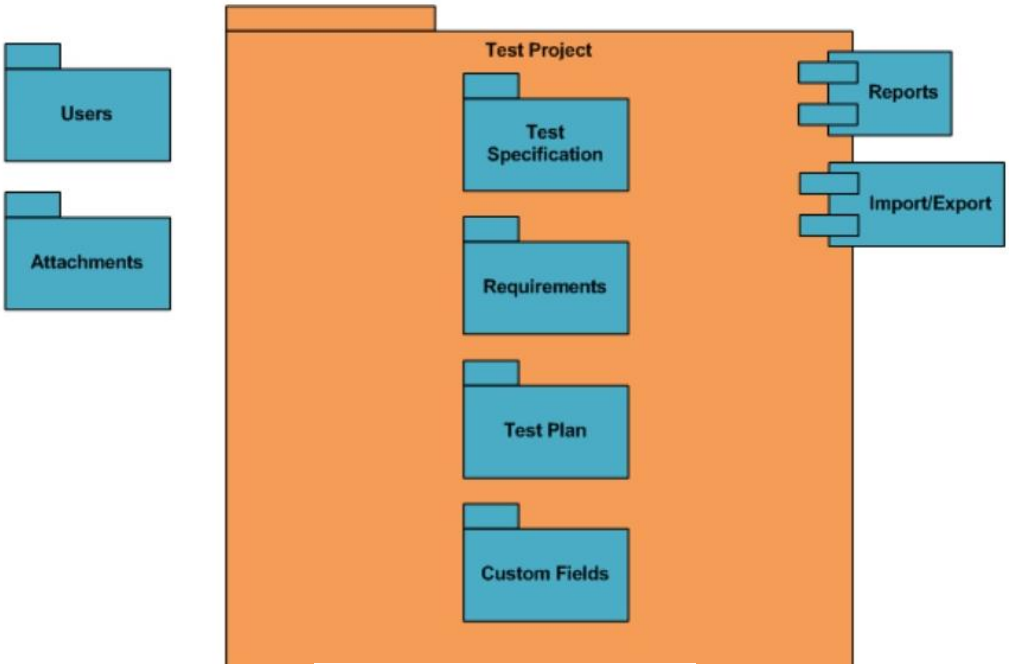


Рис. 9.3. Структура системи управління тестами TestLink

У системі управління тестами TestLink виділяють наступні користувальницькі ролі та сутності:

- ***користувальницькі ролі:***
  - ***гість (guest)*** – тільки перегляд тест-кейсів, звітів та параметрів і нічого не може редагувати;
  - ***test executor (tester)*** – має можливість переглядати і виконувати тести, призначені йому;
  - ***(test designer)*** – може переглядати і редагувати специфікації і вимоги;
  - ***test analyst (senior tester)*** – переглядає, створює, редагує і видаляє тест-кейси, виконує їх і не може керувати тест-планами і проектами або розпоряджатися правами;
  - ***test leader (leader)*** – ті ж права, що і в аналітика, крім того може керувати тест-планами і призначати права;
  - ***administrator (admin)*** – повний набір прав (як у лідера, плюс можливість управляти проектами і користувачами), при цьому права користувачів можна редагувати і створювати свої власні ролі, надаючи їм будь-який набір прав;

- ***сущності TestLink***
  - ***тестовий випадок (test case)*** – опис тест-кейса у вигляді кроків і очікуваних результатів;

- ***тестовий набір (test suite, test case suite)*** – набір тест-кейсів, що дозволяє структурувати всі тести в логічній формі. Наприклад: "LoginTests", "ValidationErrorTests", "MainMenuTests" і т.п.
- ***тест-план (test plan)*** – створюється при переході до виконання тестів. Тест-плани складаються з будь-якого набору тест-кейсів та / або TestSuite поточного проекту. Наприклад: "Regression", "Manual", "Automation", "Daily" тощо;
- ***test project*** – ключова одиниця в TestLink. Проект існує протягом усього циклу тестування і відповідає додатку, що тестується. Тестовий проект протягом життєвого циклу може змінити кілька версій і розвиватися разом з додатком. Наприклад: "OurWebPortal", "Calculator" і т.п. Як правило носить ім'я додатка, або включає його ім'я в назву.

- ***допоміжні особливості TestLink:***
  - ***build*** – відповідає білду, або серйозній модифікації тестованої програми;
  - ***platform*** – платформа, на якій проводиться тестування. В якості платформи може виступати операційна система (*Windows, Linux etc.),* браузер для веб-додатків (Chrome, Firefox etc.), різні варіанти серверів (*Apache, Tomcat etc*.) і баз даних (*MySql, MSSQL etc.);*
  - ***keyword*** – ключове слово, що служить для групування тест-кейсів з будь-якою ознакою. Наприклад "UI-Tests";
  - ***requirements*** – вимоги до додатка, які необхідно покрити тестами *(для requirement-based testing).* До них здійснюється прив'язка тест-кейсів, на підставі якої виробляється формування звіту про покриття вимог.

На рис. 9.3. представлений інтерфейс системи управління тестами TestLink.

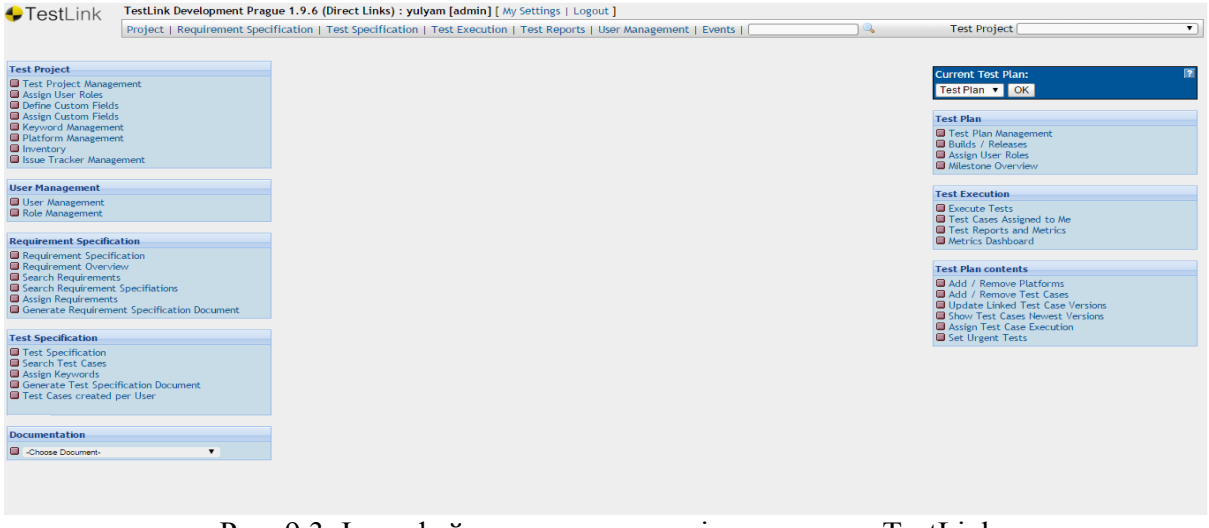


Рис. 9.3. Інтерфейс системи управління тестами TestLink

При створенні тестового випадку (test case) виконують наступні дії:

1. Спочатку необхідно створити тестовий проект (test project) (рис.9.4, 9.5):

   - натиснути на пункт *"Test Project Management"* в меню *"Test Project";*
   - натиснути на кнопку *"Create"*;
   - заповнити всі обов'язкові поля форми *"Test Project Management: Create a new project"*;
   - натиснути на кнопку *"Create"*;

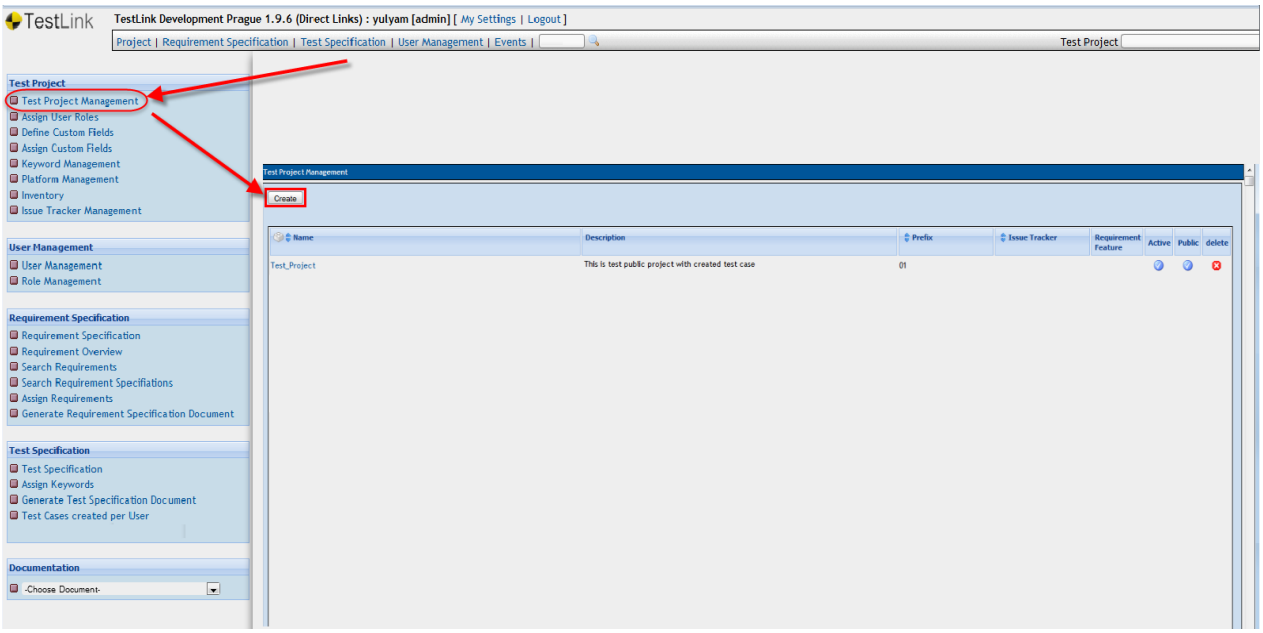


Рис.9.4. Створення тестового проекту (test project)

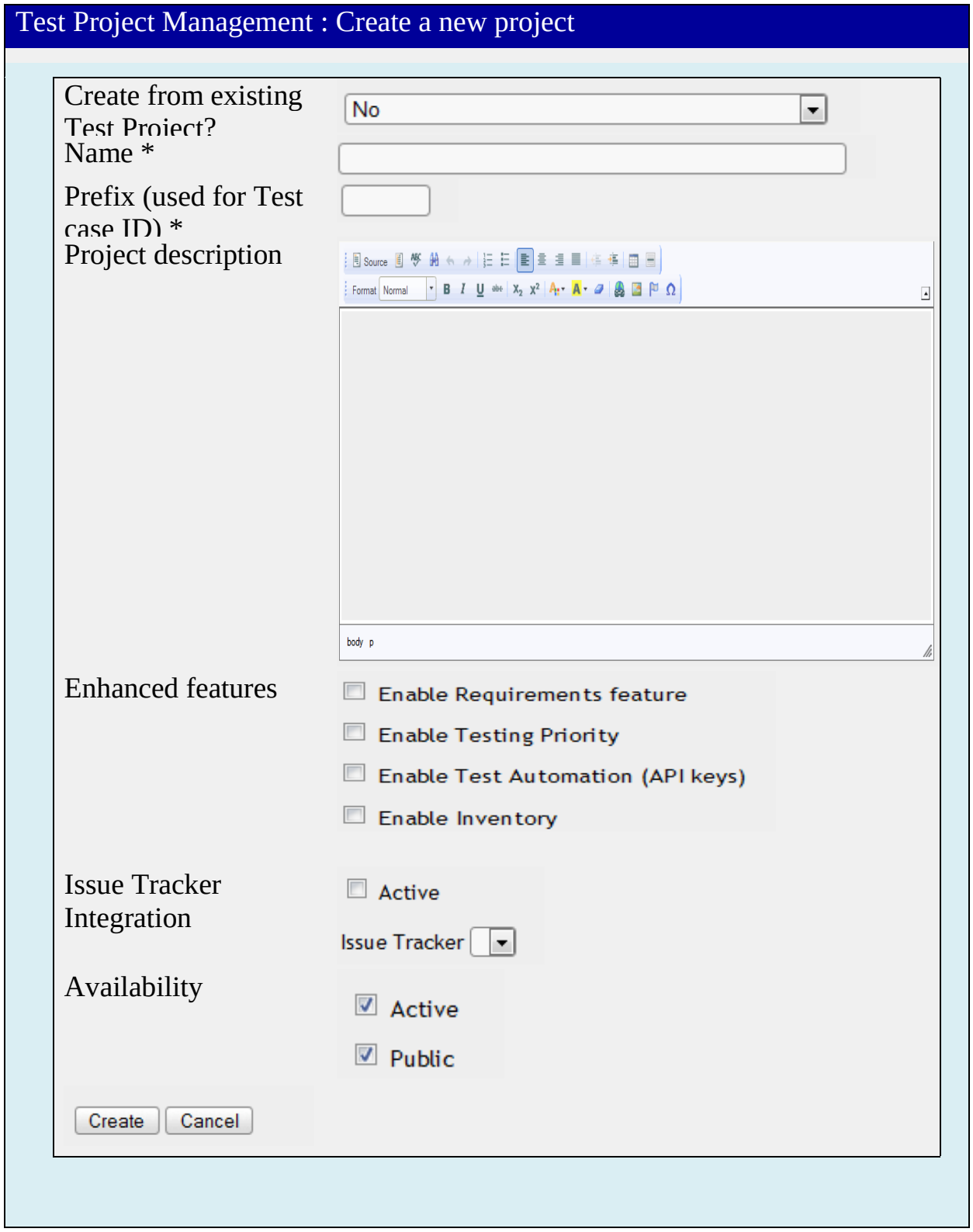


Рис.9.5. Форма *"Test Project Management: Create a new project"*

2. На другому етапі створюють тестовий набір *(test suite)* (рис.9.6):

- вибрати створений тестовий проект в випадаючому списку *"Test Project";*
- натиснути на пункт "Test Specification" в меню *"Test Specification";*
- натиснути на папку з ім'ям проекту в діалоговому вікні *"Navigator - Test Specification";*
- натиснути на кнопку "Create" в діалоговому вікні *"Test Project: <Name Project>";*

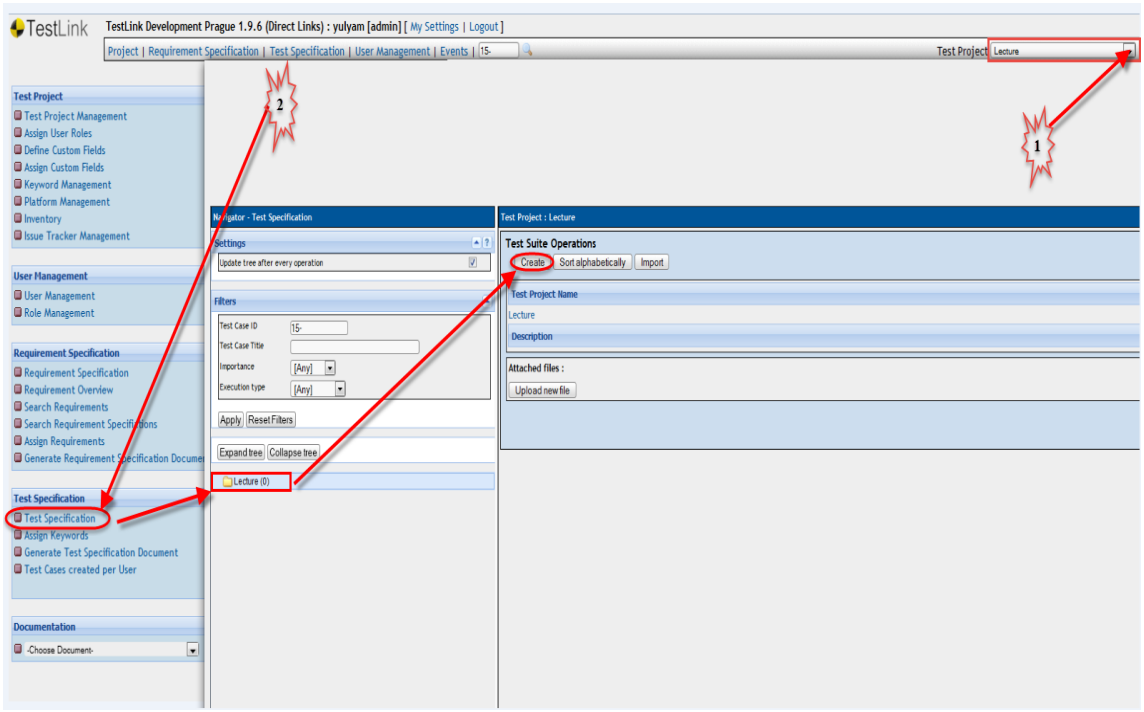


Рис.9.6. Створення тестового набору *(test suite)*

- присвоїти ім'я тестового набору в полі *"Test Suite Name"*;
- натиснути на кнопку *"Create Test Suite";*

3. Після створення тестового набору створюють тестовий випадок *(test case)* (рис.9.7):

- натиснути на папку з ім'ям тестового набору в діалоговому вікні *"Navigator - Test Specification";*
- натиснути на кнопку *"Create"* в пункті *"Test Case Operations";*

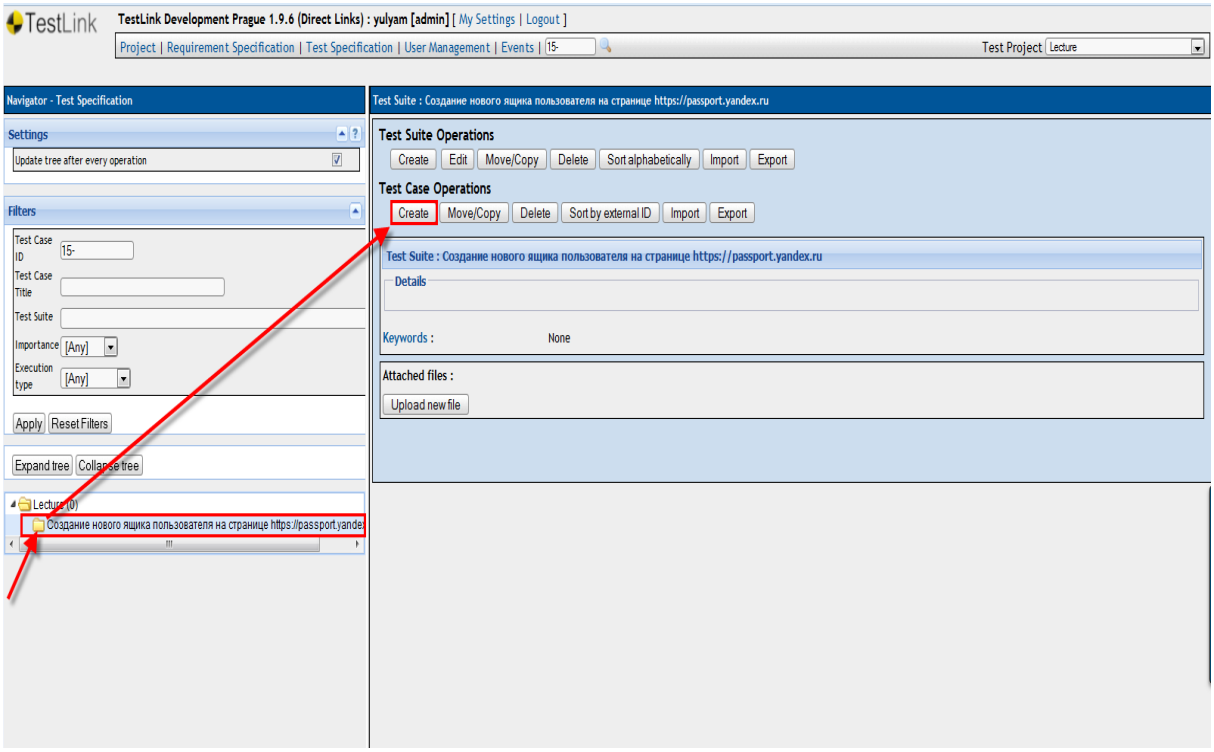


Рис.9.7. Створення тестового випадку *(test case)*

- в формі, що з'явилася, заповнити всі атрибути тестового випадку *(test case)* (рис.9.8);
- натиснути на кнопку *"Create"*;

4. Після створення тестового випадку *(test case)* створюють кроки відтворення (рис.9.9):

   - натиснути на тестовий випадок в діалоговому вікні *"Navigator - Test Specification"*;
   - натиснути на кнопку *"Create step"*;
   - заповнити поля *"Step actions" і "Expected Results"* (рис.9.10);
   - натиснути на кнопку *"Save"*;

5. Редагування тестового випадку *(test case):*

   - натиснути на тестовий випадок в діалоговому вікні *"Navigator - Test Specification"*;
   - натиснути на кнопку *"Edit"*;
   - внести необхідні зміни;
   - натиснути на кнопку *"Save"*;

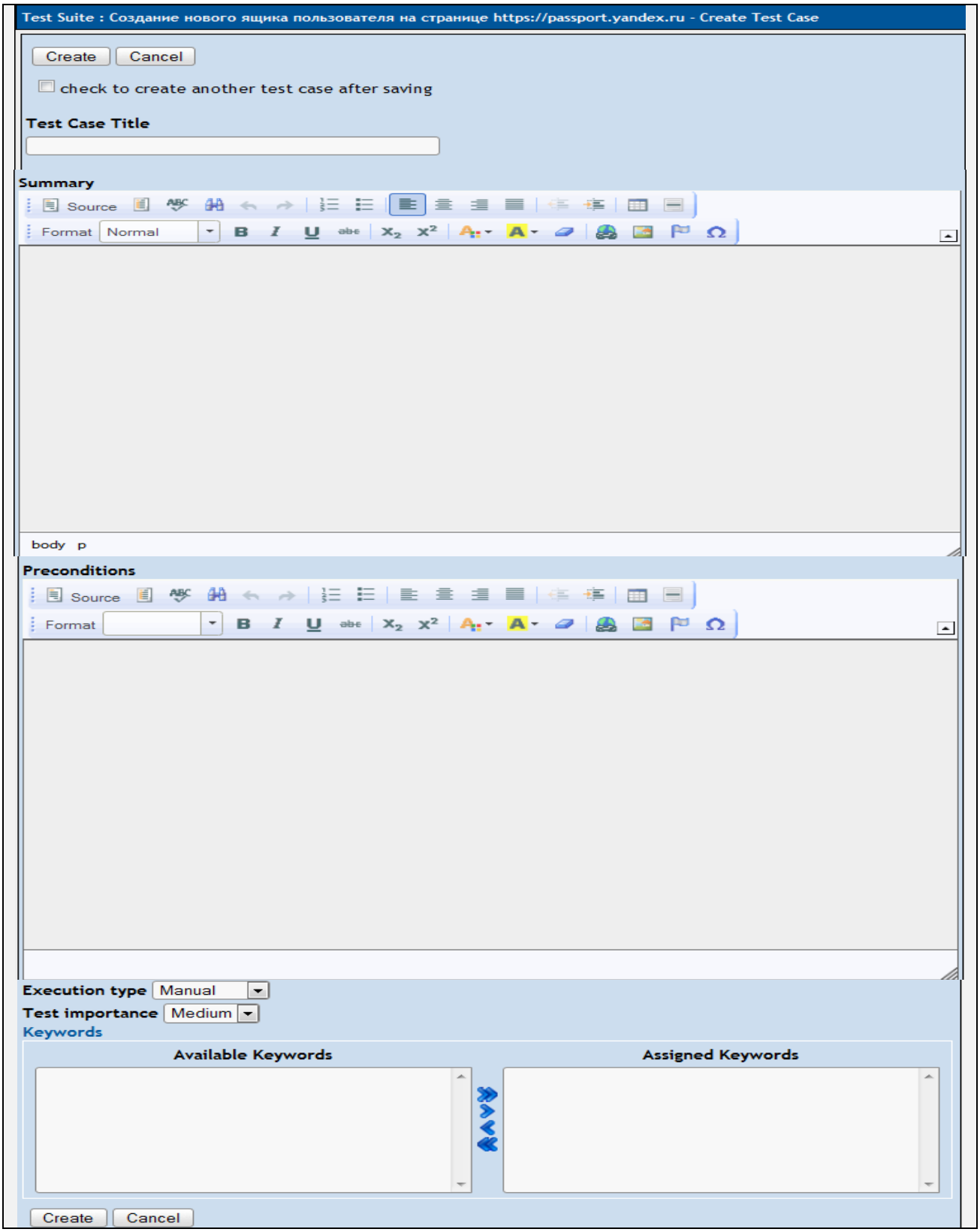


Рис.9.8. Форма тестового випадку *(test case)*

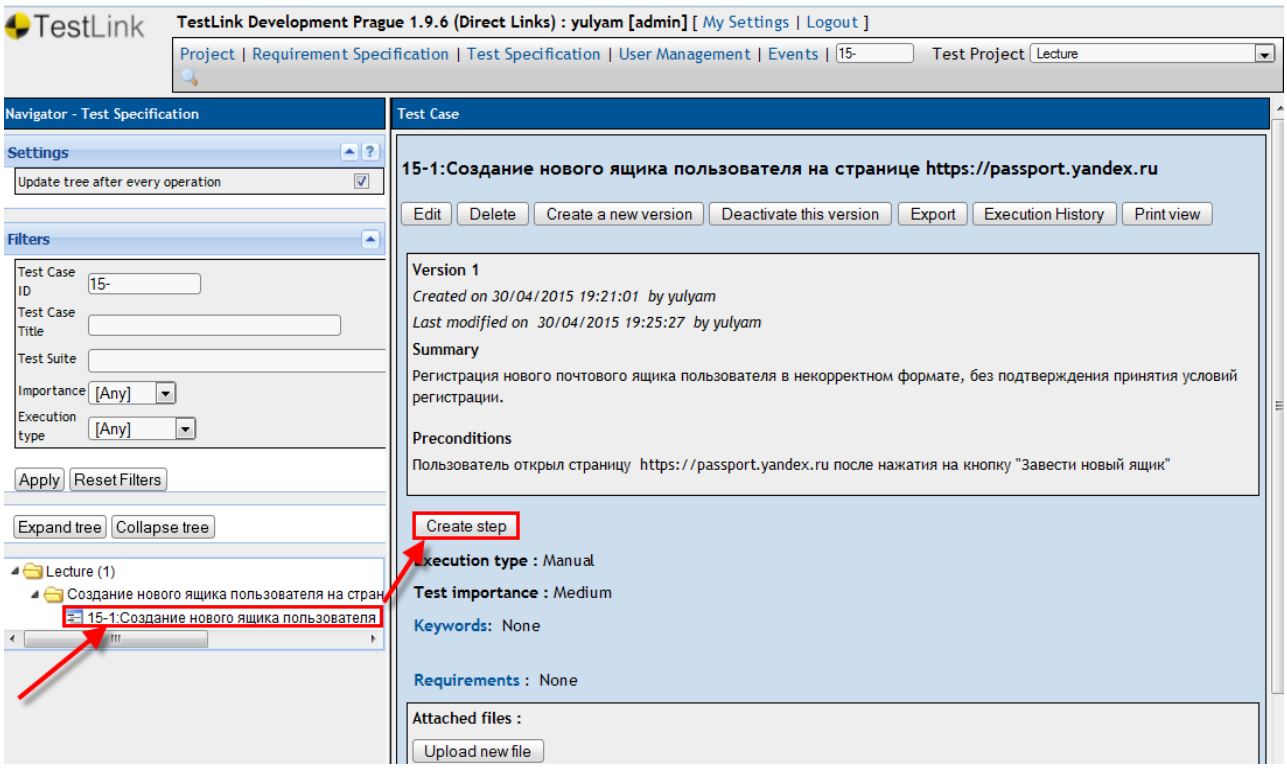


Рис.9.9. Створення кроків відтворення тестового випадку *(test case)*

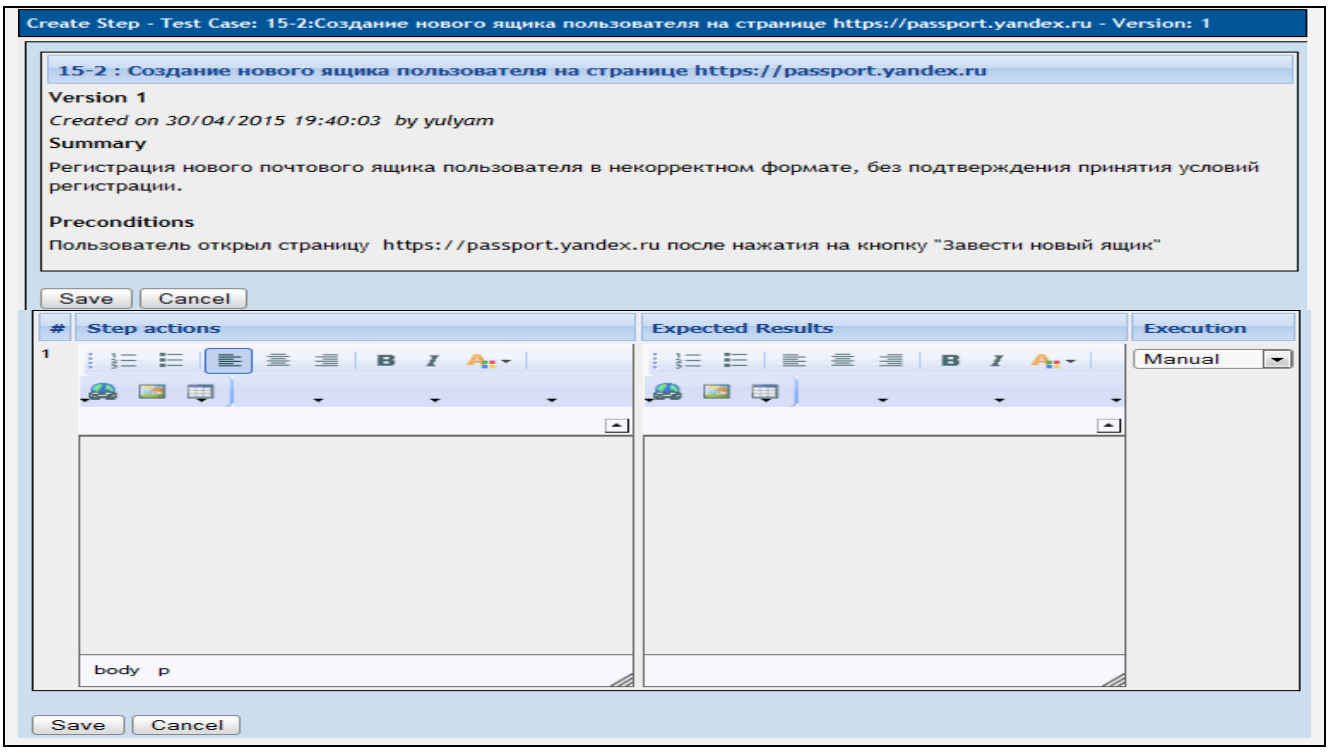


Рис. 9.10. Опис кроків відтворення *"Step actions" і "Expected Results"*

9.2.2 Системи управління тестуванням TestRail

TestRail – система для управління даними, що отримані в результаті тестування, яка дозволяє вести тестову документацію та облік результатів виконання тестів. Даний інструмент допомагає відслідковувати процеси, керувати програмним забезпеченням і організовувати команду.

За допомогою ***TestRail*** можна створювати тестові випадки *(test cases)*, управляти тестовими наборами *(test suite)* та координувати весь процес тестування програмного забезпечення. ***TestRail*** надає можливість підвищити продуктивність та отримати повний огляд ходу процесу тестування.

При відкритті ***TestRail*** спочатку виводиться сторінка із загальною підсумкової інформації по проектах *(global dashboard page)* (рис. 9.11), де можна отримати загальне уявлення про проект, доступності, нещодавнє оновлення та доступ до екрану користувача [70].

Так само можна переглянути більш детально статистичну інформацію про проект у вікні (рис.9.12).

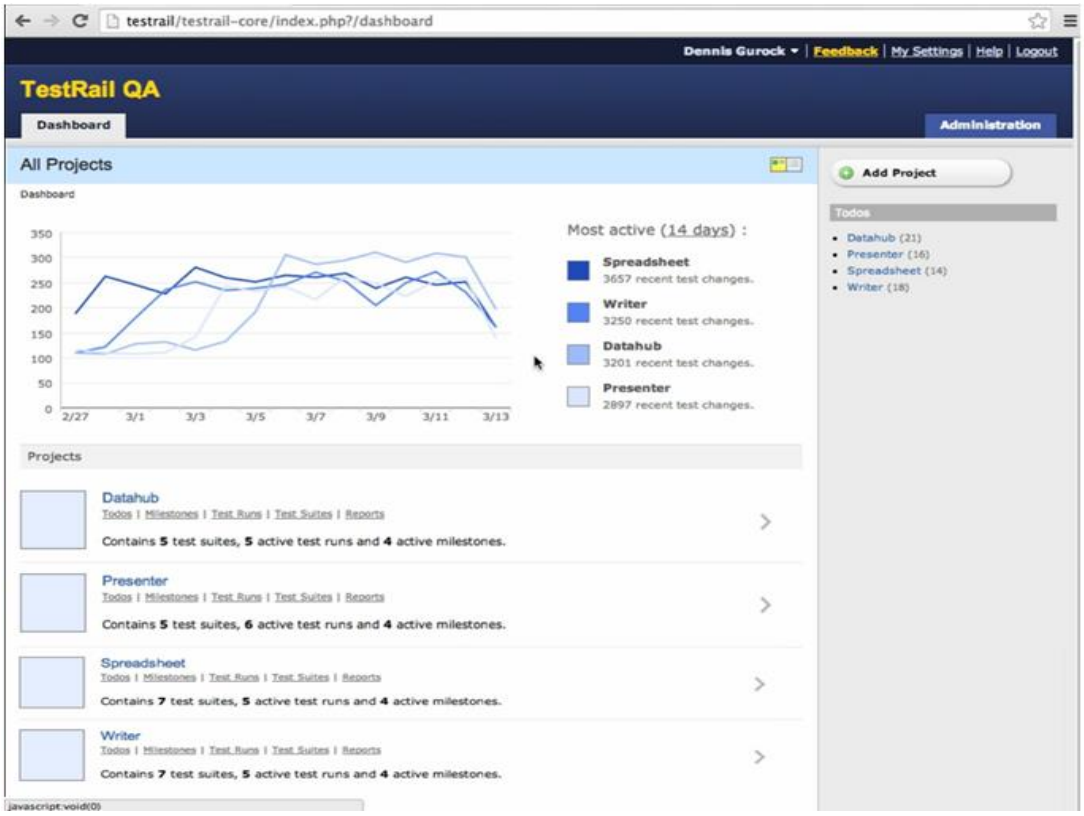


Рис. 9.11. Сторінка з загальною підсумковою інформацією з проектів *(global dashboard page)*

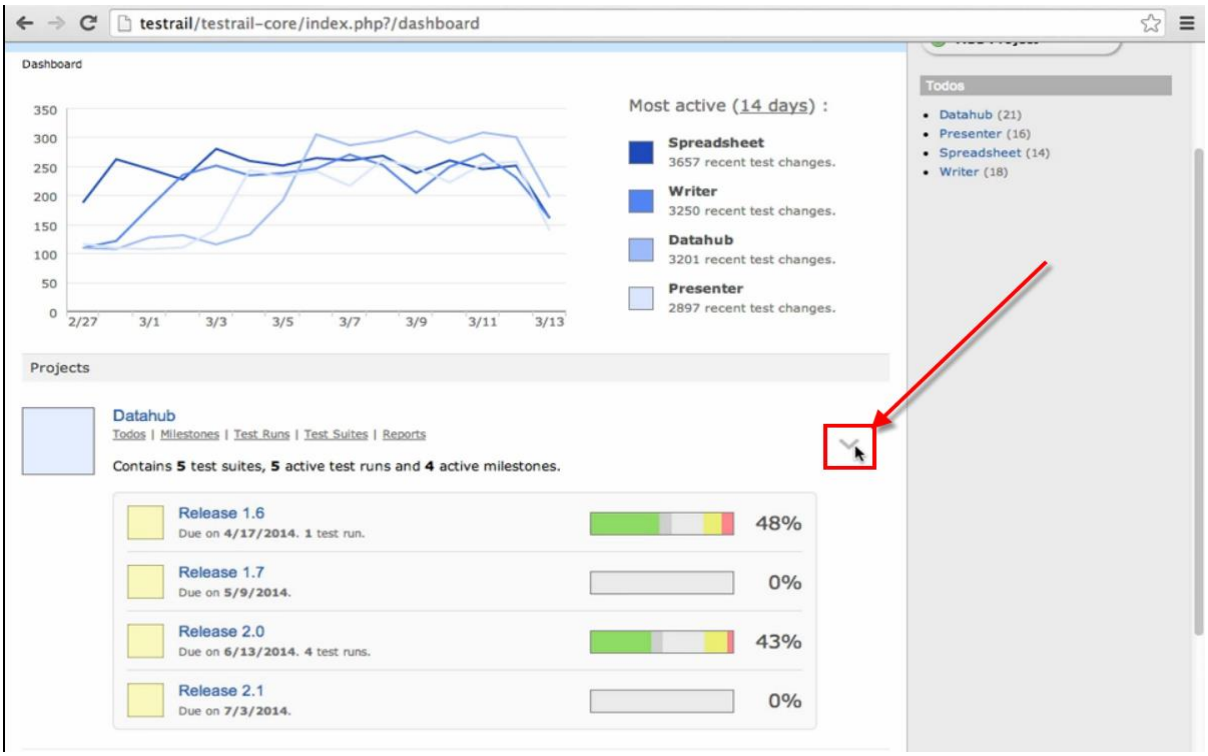


Рис. 9.12. Статистична інформація по проектах

Для того, щоб переглянути проект необхідно натиснути на ім'я проекту на сторінці із загальною підсумковою інформацією по проектах (рис. 9.13). Після переходу безпосередньо до проекту, у вікні проекту з'явиться додаткове навігаційне вікно, яке містить ***наступні вкладки:***

- загальна інформація *(overview);*
- контрольний термін *(milestones);*
- термін виконання *(todo);*
- тестовий прогін програми і результати *(test runs & results);*
- тестові набори й випадки *(test suites & cases);*
- звіти *(reports).*

На сторінці з загальною інформацією надається ***значна кількість корисної інформації про проект,*** а саме:

- графік виконання робіт *(project progress chart);*
- найближчий контрольний термін *(upcoming milestones);*
- останній тестовий прогін програми *(latest test runs);*
- звіт про виконану роботу *(activity report).*

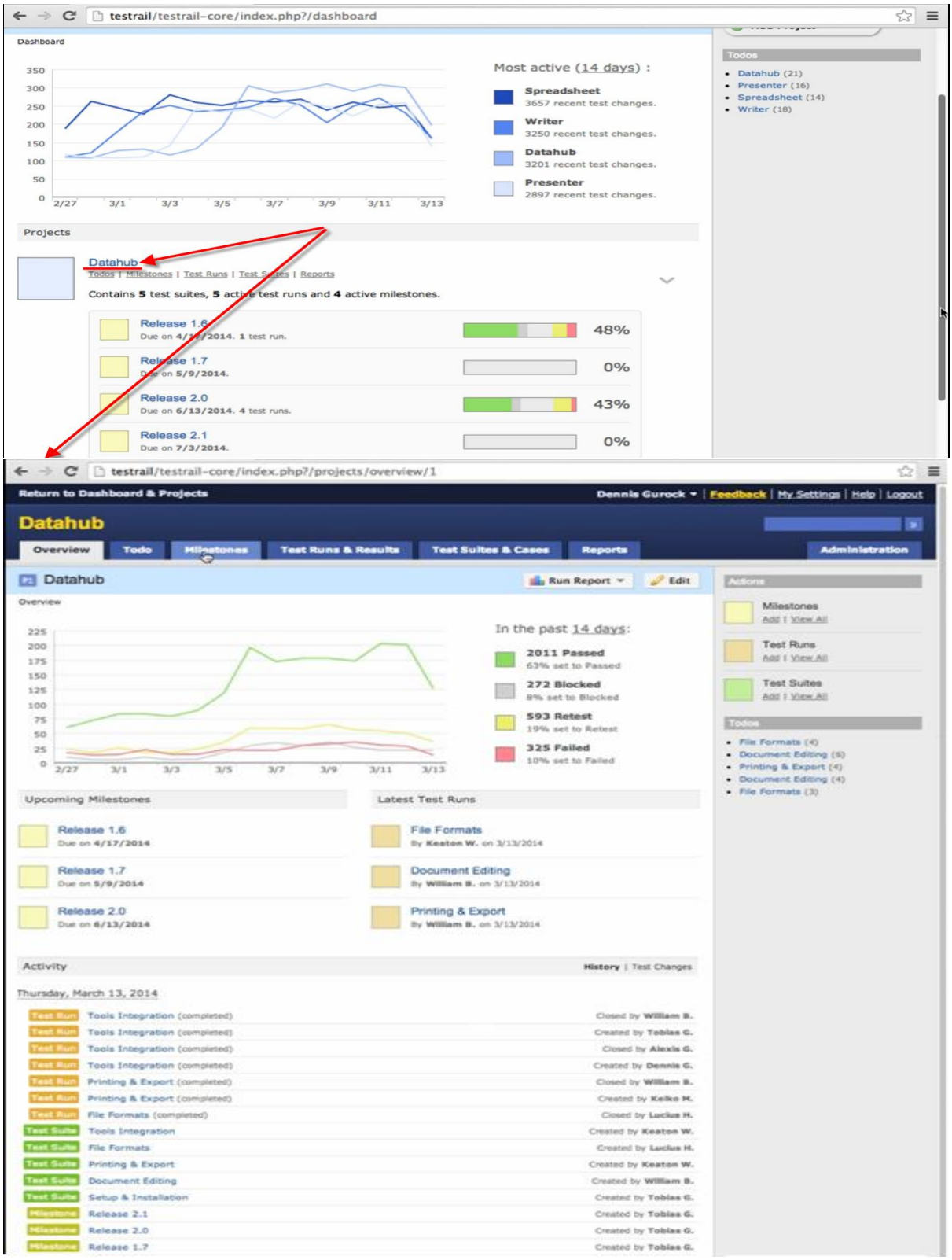


Рис. 9.13. Перегляд проекту в системі управління тестуванням ***TestRail***

У системі управління тестуванням TestRail надається можливість присвоєння статусу про завершення проекту. Ця функція дуже зручна для тестування тимчасових проектів (рис. 9.14). Після того як проекту було присвоєно статус про завершення на сторінці із загальною підсумковою

інформацією, проект переноситься в додатковий список із завершеними проектами.

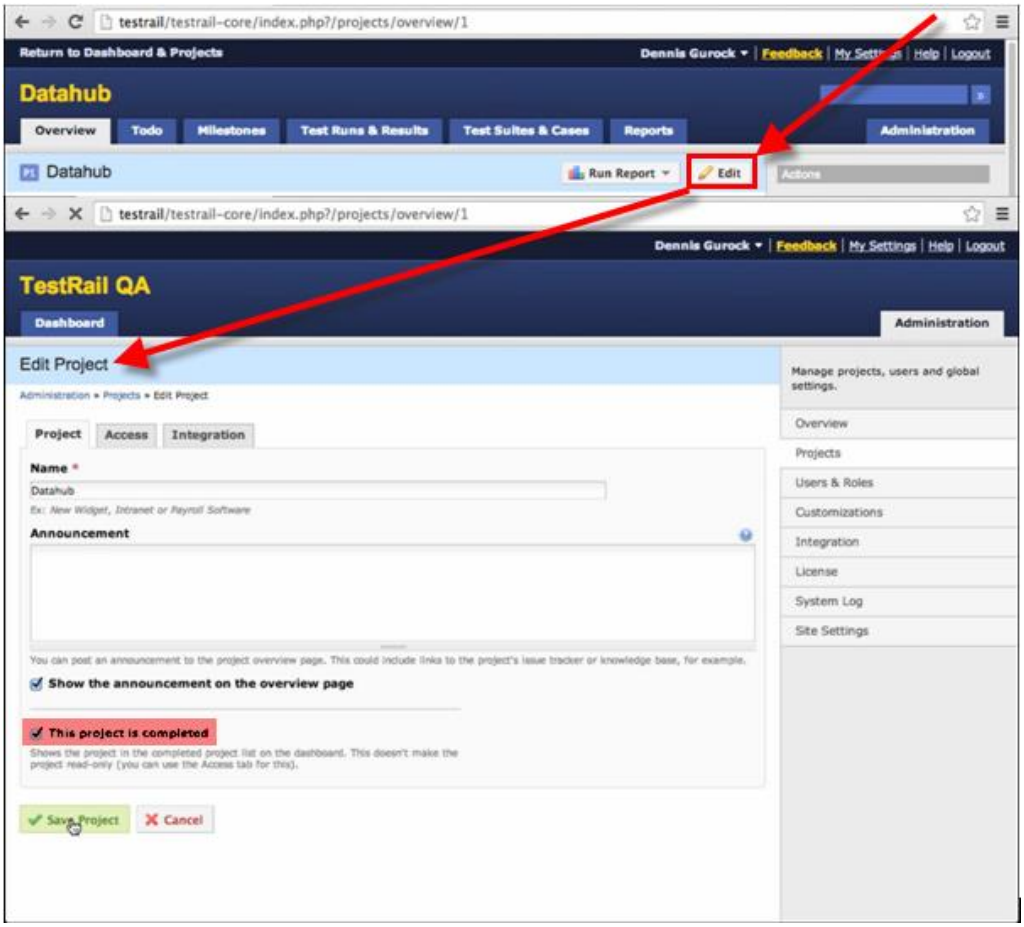


Рис. 9.14. Присвоєння статусу про завершення проекту

Робота з тестовими наборами та випадками (test suites & cases)

Для кращої якості роботи варто робити групування тестових випадків (test cases) за важливістю програми або по виконуваних функціях, при цьому необхідний спосіб організації тестових наборів (test suites) залежить від виконуваного проекту. Загалом, рекомендується створювати негроміздкі тестові набори для великих проектів, навіть якщо вони містять більше сотні тестових випадків. Це пов'язано із зручністю управління проектом та тестового прогону програми [29].

Для того, щоб переглянути тестовий набір, необхідно у вкладці "Test Suites & Cases" натиснути ім'я тестового набору. Після відкриття тестового набору надається список тестових випадків (test cases), які згруповані в різних розділах (рис. 9.15). Система управління тестуванням TestRail дозволяє додавати підрозділи (subsections) в основні розділи тестових випадків, що спрощує роботу з громіздкими тестовими наборами (test suites).

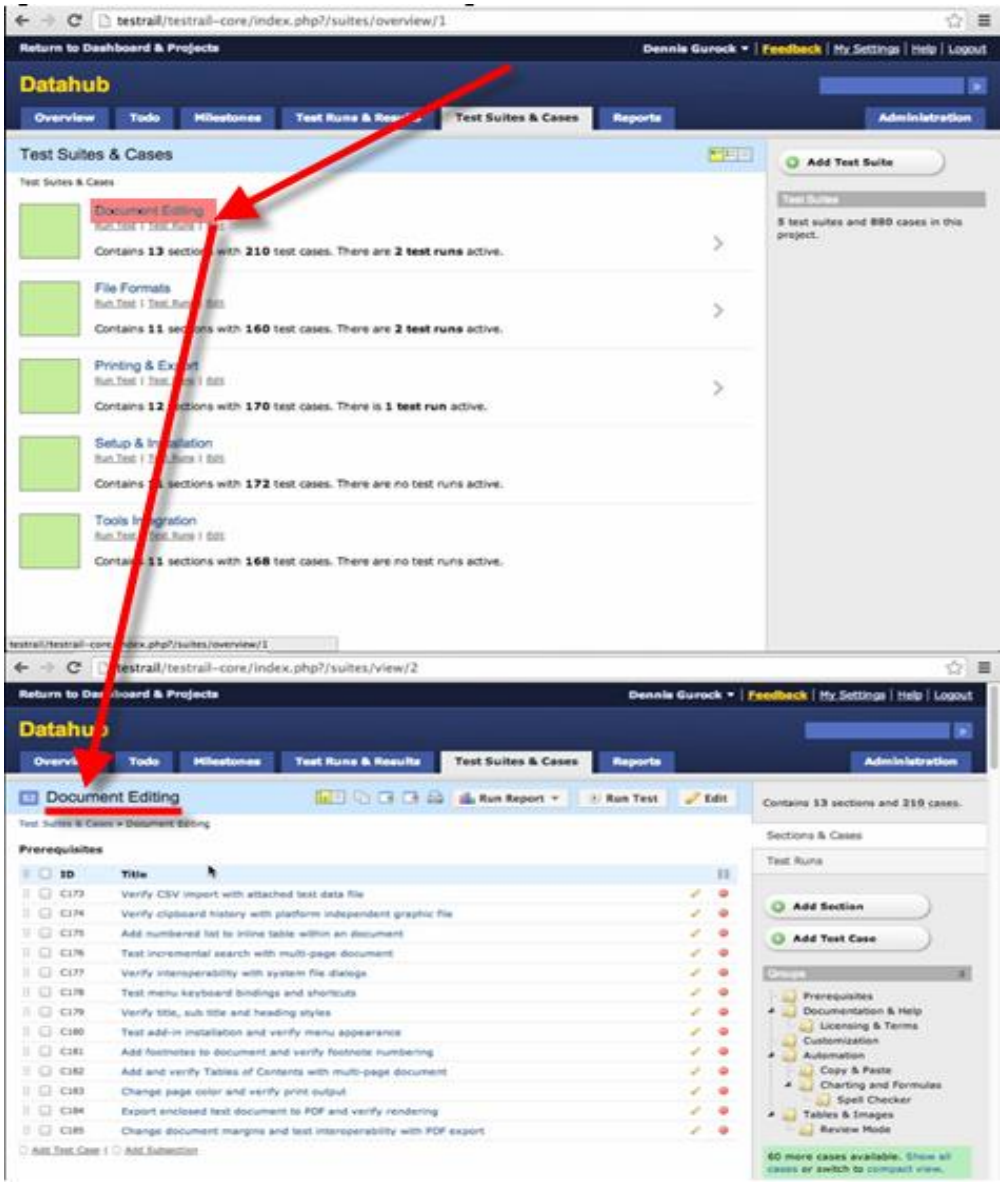


Рис. 9.15. Перехід до тестових *(test cases)*

Завдяки тому, що система управління тестуванням ***TestRail*** має значну кількість різних функцій, вона дозволяє спростити процес додавання тестових випадків в тестовий набір або додаткових розділів без переходу на іншу сторінку (рис. 9.16).

Для того, щоб додати тестовий випадок або підрозділ в тестовий набір необхідно натиснути посилання *"Add Test Case"* або посилання *"Add Subsection"* відповідно, які знаходяться внизу списку тестового набору, у вікні вказати ім'я тест-кейса або підрозділу (рис. 9.16 , 9.17).

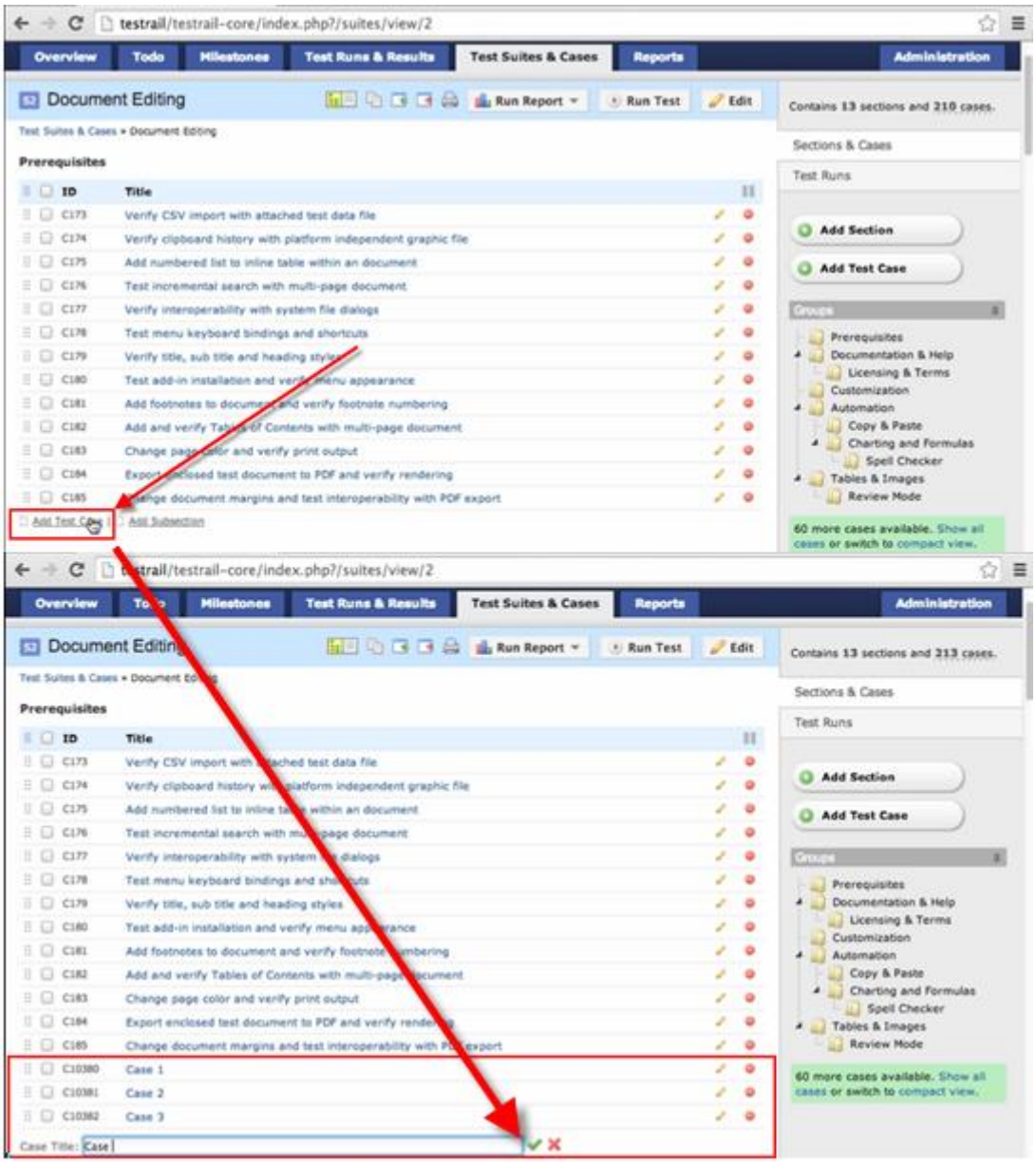


Рис. 9.16. Додання нових тестових випадків в тестовий набір

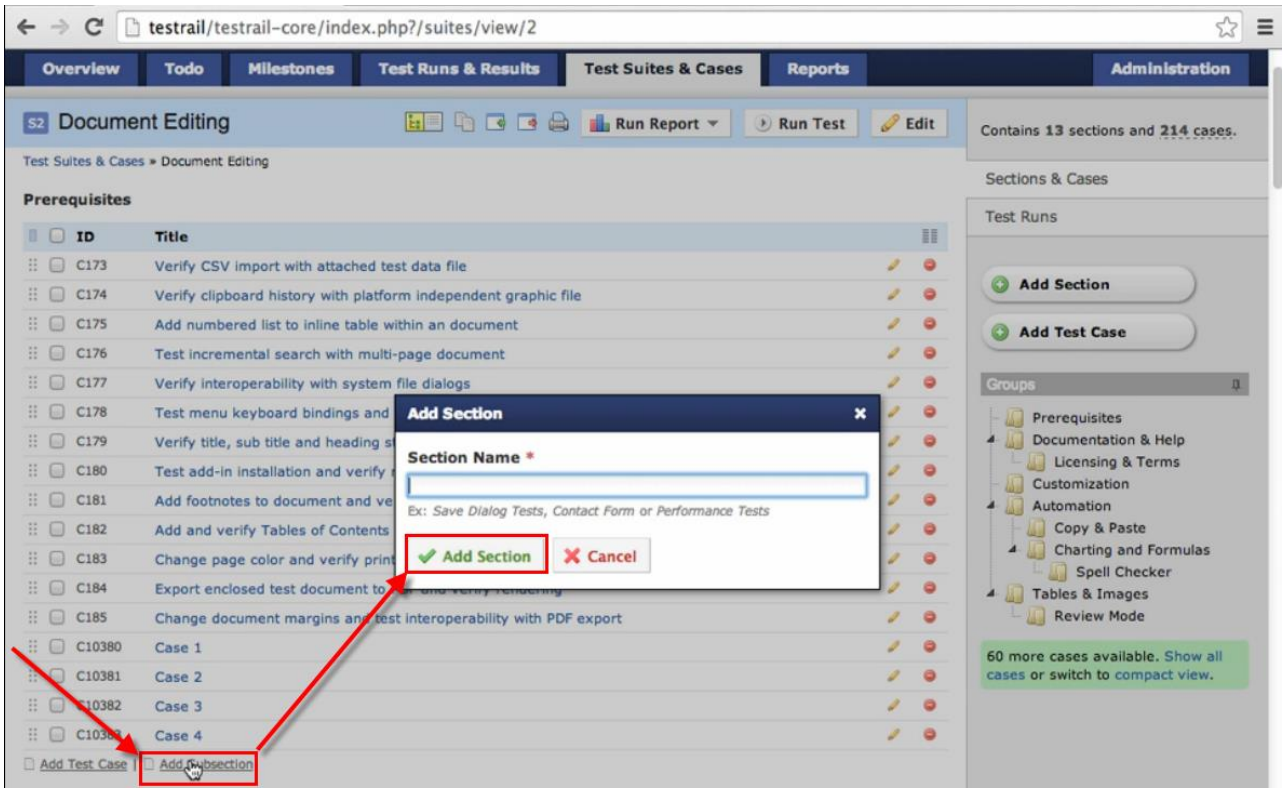


Рис. 9.17. Додання нового підрозділу в тестовий набір

Також система ***TestRail*** підтримує такі можливості як:

- ***функцію перетягування (drag and drop)*** тестових випадків з одного розділу в інший в тестовому наборі;
- ***функцію створення копій тестових випадків*** (*press the shift key and used the "drag and drop" function);*
- ***функцію копіювання і переміщення*** *(copy and move)* тестових випадків між проектами або між тестовими наборами;
- ***функцію редагування*** *(edit);*
- ***функцію експорту в xml, CSV і Excel формати*** (рис. 9.18).

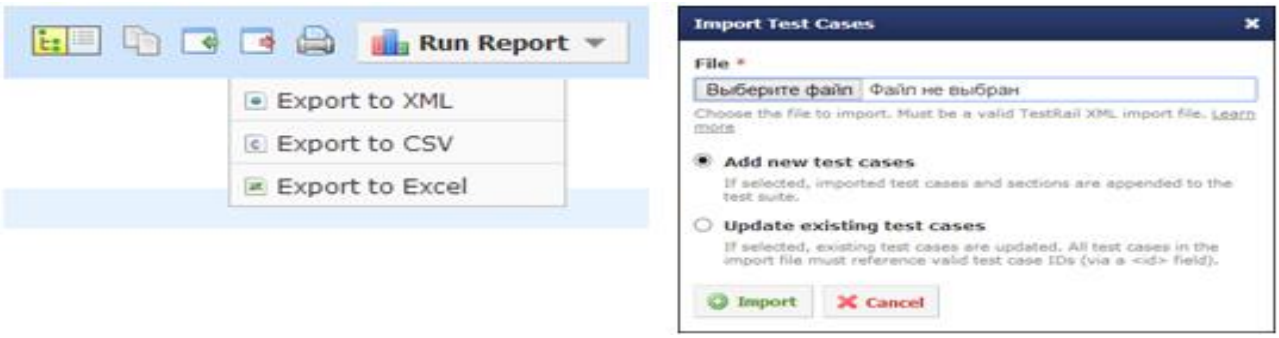


Рис. 9.19. Експорт даних із системи управління тестуванням ***TestRail***

До атрибутів тестового випадку *(test case)* (рис. 9.19) у системі управління тестуванням *TestRail* відносяться:

- назва *(title);*
- тип тестування *(test type);*
- пріоритет *(priority);*
- час, що виділяється на проведення тестування *(estimate);*
- контрольний термін *(milestones);*
- попередні умови *(preconditions);*
- кроки відтворення *(steps);*
- очікуваний результат *(expected result);*

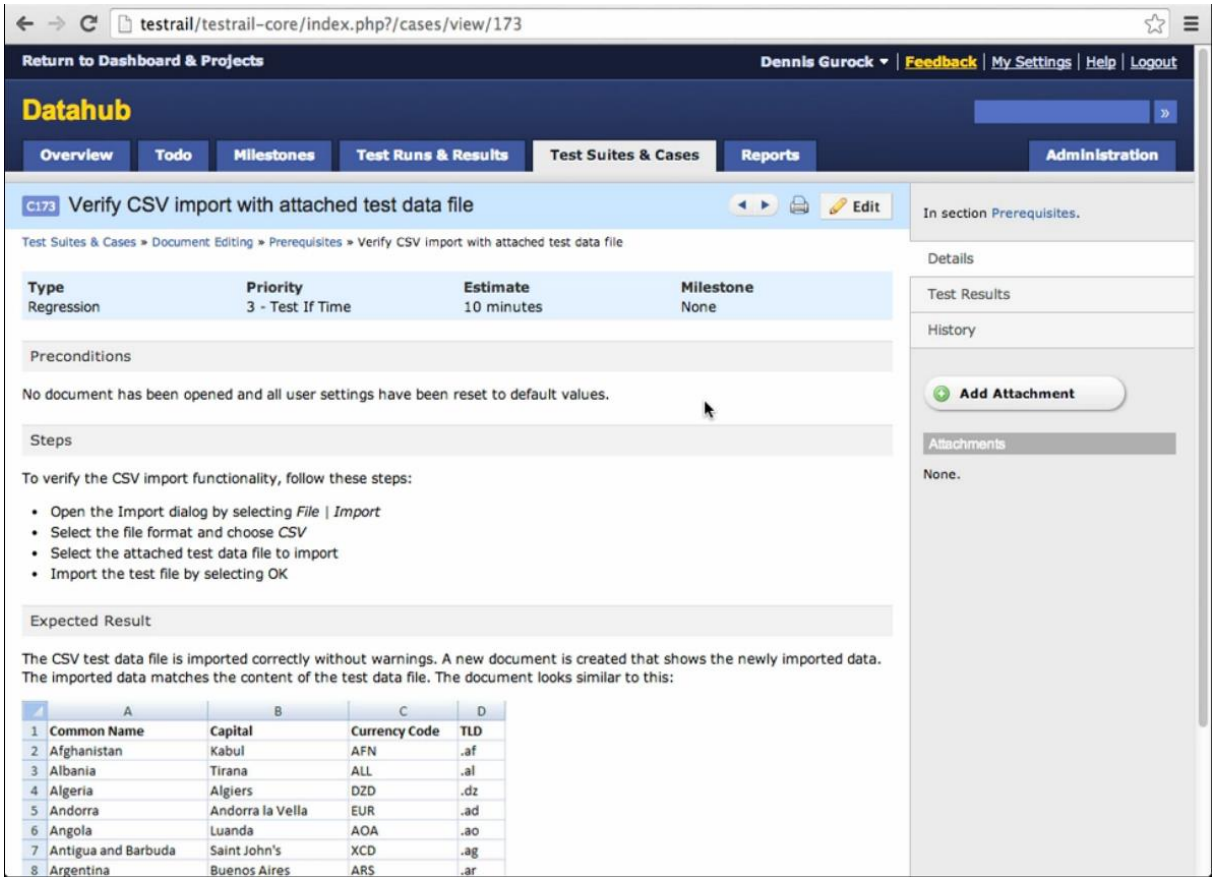
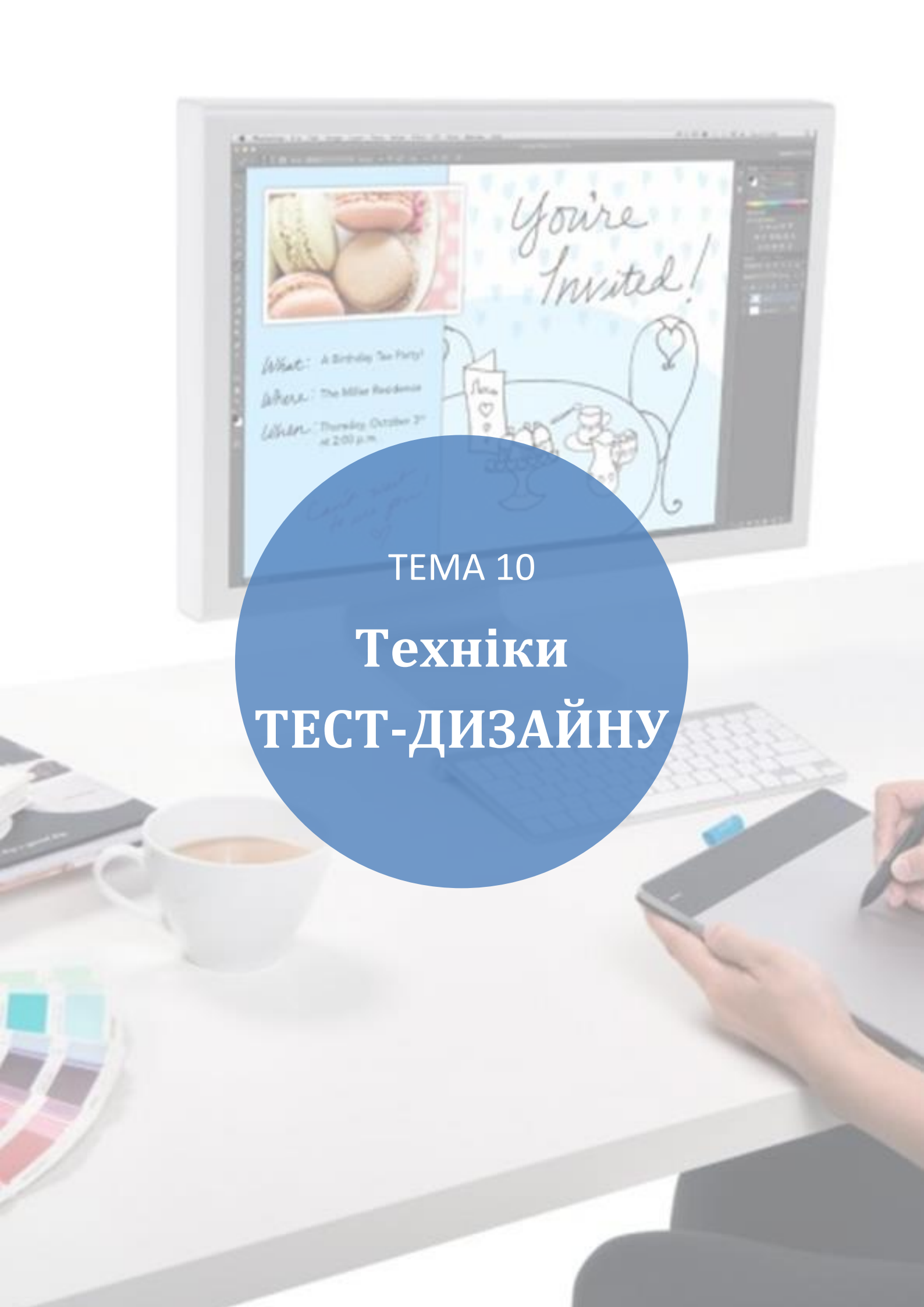


Рис. 9.19. Загальний вигляд тестового випадку в системі управління тестуванням ***TestRail***

Порівняльна характеристика систем управління тестами TestLink і TestRail представлена в таб. 9.2 [70].

Таблиця 9.2 Порівняльна характеристика систем управління тестами TestLink і TestRail

Критерії порівняння	TestLink	TestRails
Основні		
вартість ліцензії	–	$69 – $8,999
браузери	всі	всі
функціонал		
автозбереження / контроль версій	так	так
шаблони	немає	немає
друк / посилання на звіт	так	так
імпорт / експорт	так /немає	так / так
мобільна версія	немає	немає
оточення / зборки / вимоги	немає / так / так	немає / так / немає
підказки	немає	так
Елементи тест-менеджменту:		
тест-план	так	немає
тест-кейси	так	так
підтримка автотестів	немає	немає
чек-листи	так	немає
розмежування ролей користувачів	є	є
підтримка прикріплених файлів (файлове сховище)	немає	немає
e-mail notificare	немає	є
Робота з чек-листами		
налаштування полів в чек-аркуші (різні типи чек-листів, окремо налаштування тестового оточення, наприклад)	немає	немає
зручність редагування (суб'єктивна оцінка)	незручно	не дуже зручно
зв'язок з вимогами	так	немає
по зборках	так	немає
по оточенням	немає	немає
по тестерам	немає	немає
за датою виконання	немає	так
за результатом	так	так
відображення статусу тестованого	є	є
звітність за різними чек-листам відразу	так	так
звітність по покриттю вимог	так	немає
звітність за результатами тестування	так	немає
Простота використання		
простота	так	так
гарячі клавіші ***(hotkey)***	немає	немає
опис груп користувачів	так	так



TEMA 10

Техніки ТЕСТ-ДИЗАЙНУ

10 ТЕХНІКИ ТЕСТ-ДИЗАЙНУ

10.1 Тестове покриття (Test Coverage)

Тестове покриття – це одна з метрик оцінки якості тестування, що являє собою щільність покриття тестами вимог або виконуваного коду.

Якщо розглядати тестування як "перевірку відповідності між реальною та очікуваною поведінкою програми, здійснювану на кінцевому наборі тестів", то саме цей кінцевий набір тестів й визначатиме тестове покриття:

Чим вище необхідний рівень тестового покриття, тим більше тестів буде вибрано для перевірки тестованих вимог або виконуваного коду.

Складність сучасного програмного забезпечення та інфраструктури зробила неможливим проведення тестування з 100% тестовим покриттям. Тому для розробки набору тестів, що забезпечує більш менш високий рівень покриття, можна використовувати спеціальні інструменти або техніки тест-дизайну.

Існують наступні підходи до оцінки та виміру тестового покриття:

1. ***Покриття вимог (Requirements Coverage)*** – оцінка покриття тестами функціональних і не функціональних вимог до продукту шляхом побудови матриць трасування (traceability matrix).
2. ***Покриття коду (Code Coverage)*** – оцінка покриття виконуваного коду тестами, шляхом відстеження неперевірених в процесі тестування частин програмного забезпечення.
3. ***Тестове покриття на базі аналізу потоку управління*** – оцінка покриття, заснована на визначенні шляхів виконання коду програмного модуля та створення виконуваних тест-кейсів для покриття цих шляхів.

Відмінності:

Метод покриття вимог зосереджений на перевірці відповідності набору проведених тестів вимогам до продукту в той час, як ***аналіз покриття коду*** – на повноті перевірки тестами розробленої частини продукту (вихідного коду), а ***аналіз потоку управління*** – на проходженні шляхів у графі або моделі виконання тестованих функцій (*Control Flow Graph).*

Обмеження:

Метод оцінки покриття коду не виявить нереалізовані вимоги, оскільки він працює не з кінцевим продуктом, а з існуючим вихідним кодом.

Метод покриття вимог може залишити неперевіреними деякі ділянки коду, тому що не враховує кінцеву реалізацію.

Покриття вимог (Requirements Coverage)

Розрахунок тестового покриття щодо вимог проводиться за формулою:

$$T_{cov} = \left(\frac{L_{cov}}{L_{total}} \right) \cdot 100\% \qquad (10.1)$$

де T_{cov} – тестове покриття;

L_{cov} – кількість вимог, що перевіряються тест-кейсами;

L_{total} – загальна кількість вимог.

Для вимірювання покриття вимог необхідно проаналізувати вимоги до продукту і розбити їх на пункти. Опціонально кожен пункт зв'язується з тест-кейсами, які перевіряють його. Сукупність цих зв'язків і є матрицею трасування. Простеживши зв'язки, можна зрозуміти, які саме вимоги перевіряє тестовий випадок.

Тести, не пов'язані з вимогами, не мають сенсу. Вимоги, не пов'язані з тестами, – це "білі плями", тобто виконавши всі створені тест-кейси, не можна дати відповідь – реалізована дана вимога в продукті чи ні.

Для оптимізації тестового покриття при тестуванні на підставі вимог найкращим способом буде використання стандартних технік тест-дизайну. Приклад розробки тестових випадків за наявними вимогами розглянуто в розділі: " Практичне застосування технік тест-дизайну при розробці тест-кейсів "

Покриття коду (Code Coverage)

Розрахунок тестового покриття щодо виконуваного коду програмного забезпечення проводиться за формулою:

$$T_{cov} = \left(\frac{L_{tc}}{L_{code}} \right) \cdot 100\% \qquad (10.2)$$

де T_{cov} – тестове покриття;

L_{tc} – кількість рядків коду, покритих тестами;

L_{code} – загальна кількість рядків коду.

У даний час існує інструментарій (*наприклад*, *Clover*), що дозволяє проаналізувати, які рядки були перевірені під час проведення тестування, завдяки чому можна значно збільшити покриття, додавши нові тести для конкретних випадків, а також позбутися від дублюючих

тестів. Проведення такого аналізу коду та подальша оптимізація покриття досить легко реалізується в рамках тестування білого ящика (*white-box testing)* при модульному, інтеграційному та системному тестуванні; при тестуванні ж чорного ящика (*black-box testing)* завдання стає досить дорогим, оскільки вимагає багато часу й ресурсів для установки, конфігурації та аналізу результатів роботи як з боку тестувальників, так й розробників.

Тестове покриття на базі аналізу потоку управління

Тестування потоків управління *(Control Flow Testing)* – це одна з технік тестування білого ящика, заснована на визначенні шляхів виконання коду програмного модуля та створення виконуваних тест-кейсів для покриття цих шляхів [72].https://translate.google.com/translate?hl=ru&prev=_t&sl=ru&tl=uk&u=http://www.protesting.ru/testing/testcoverage.html%23%5B1%5D - [1]

Фундаментом для тестування потоків управління є побудова графів потоків управління *(Control Flow Graph*), основними блоками яких є:

- блок процесу – одна точка входу та одна точка виходу;
- точка альтернативи – одна точка входу, дві та більше точки виходу;
- точка з'єднання – дві та більше точок входу, одна точка виходу.

Для тестування потоків управління визначені різні рівні тестового покриття (табл. 10.1).

Таблиця 10.1. Рівні тестового покриття

Рівень	*Назва*	*Короткий опис*
Рівень 0	-	"Тестуй все що зможеш протестувати, користувачі протестують інше" На англійській мові це звучить набагато елегантніше: *"Test whatever you test, users will test the rest"*
Рівень 1	Покриття операторів	Кожен оператор повинен бути виконаний як мінімум один раз.
Рівень 2	Покриття альтернатив [73]/ Покриття гілок	Кожен вузол з розгалуженням (альтернатива) виконаний як мінімум один раз.
Рівень 3	Покриття умов	Кожну умову, що має TRUE і FALSE на виході, виконано як мінімум один раз.
Рівень 4	Покриття умов альтернатив	Тестові випадки створюються для кожної умови і альтернативи
Рівень 5	Покриття множинних умов	Досягається покриття альтернатив, умов та умов альтернатив (Рівні 2, 3 і 4)
Рівень 6	"Покриття нескінченного числа шляхів"	Якщо у разі зациклення, кількість шляхів стає нескінченним, допускається істотне їх скорочення, обмежуючи кількість циклів виконання для зменшення числа тестових випадків.
Рівень 7	Покриття шляхів	Всі шляхи повинні бути перевірені

Ґрунтуючись на даних цієї таблиці, можна спланувати необхідний рівень тестового покриття, а також оцінити вже наявний.

10.2 Техніки тест-дизайну

Тестуючи та пишучи тестові випадки (test cases), обмежена кількість тестувальників користується спеціальними ***техніками тест-дизайну***. Поступово набираючи досвід, вони усвідомлюють, що постійно роблять одну й ту ж роботу, що піддається конкретним правилам, й тоді знаходять всі ці правила вже описаними.

Опис найбільш поширених технік тест-дизайну:

Еквівалентний поділ *(Equivalence Partitioning – EP).* Як приклад, у вас є діапазон допустимих значень від 1 до 10, ви повинні вибрати одне вірне значення усередині інтервалу, скажімо, 5, і одне невірне значення поза інтервалом – 0.

Аналіз граничних значень *(Boundary Value Analysis – BVA).* Якщо взяти приклад вище, в якості значень для позитивного тестування виберемо мінімальну і максимальну межі (1 і 10) і значення більше і менше кордонів (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, введення комбінацій умов (причин) для отримання відповіді від системи (Наслідок). Наприклад, ви перевіряєте можливість додавати клієнта, використовуючи певну екранну форму. Для цього необхідно буде ввести кілька полів, таких як "Ім'я", "Адреса", "Номер телефону", а потім натиснути кнопку "Додати" – це "Причина". Після натискання кнопки "Додати" система додає клієнта в базу даних і показує його номер на екрані – це "Слідство".

Передбачення помилки *(Error Guessing – EG).* Це коли тест-аналітик використовує свої знання системи і здатність до інтерпретації специфікації на предмет того, щоб "передбачити", за яких вхідних умов система може видати помилку. Наприклад, специфікація каже: "користувач повинен ввести код". Тест-аналітик думатиме: "Що, якщо я не введу код?", "Що, якщо я введу неправильний код?" і так далі. Це і є передбачення помилки.

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

10.3 Практичне застосування технік тест-дизайну при розробці тест-кейсів

При тестуванні програм (сайтів) далеко не кожен вміє застосовувати тест-дизай. Щоб трохи прояснити ситуацію, вирішили запропонувати послідовний підхід до розробки тестових випадків (тест-кейсів), використовуючи найпростіші техніки тест-дизайну:

1. ***Еквівалентний поділ (Equivalence Partitioning) – EP;***
2. ***Аналіз граничних значень (Boundary Value Analysis) – BVA;***
3. ***Передбачення помилки (Error Guessing) – EG;***
4. ***Причина / Наслідок (Cause / Effect)– CE.***

Розробка тест-кейсів здійснюється за наступним планом:

1. Аналіз вимог.
2. Визначення набору тестових даних на основі EP, BVA, EG.
3. Розробка шаблону тесту на підставі CE.
4. Написання тест-кейсів на підставі первинних вимог, тестових даних та кроків тесту.

Протестувати функціональність форми прийому заявок, вимоги до якої представлені в таблиці 10.2.

Таблиця 10.2. Вимоги до функціональності форми прийому заявок

Елемент	***Тип елемента***	***Вимоги***
Тип звернення	combobox	Набір даних: 1 Консультація 2 Проведення тестування 3 Розміщення реклами 4 Помилка на сайті * На процес виконання операції прийому заявок не впливає.
Контактна особа	editbox	1. Обов'язкове для заповнення 2. Максимально 25 символів 3. Використання цифр та спец. символів не допускається
Контактний телефон	editbox	1. Обов'язкове для заповнення 2. Допустимі символи "+" та цифри

		3. "+" можна використовувати тільки спочатку номера 4. Допустимі формати: – починається з плюса – 11-15 цифр; – +31612361264; – +375291438884; – без плюса – 5-10 цифр, наприклад: 0613261264, 2925167
Повідомлення	text area	1. Обов'язкове для заповнення 2. Максимальна довжина 1024 символів
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 – відправка повідомлення 2. Якщо введені дані НЕ коректні – валідаційні повідомлення

Варіант використання *(іноді може і не бути):*

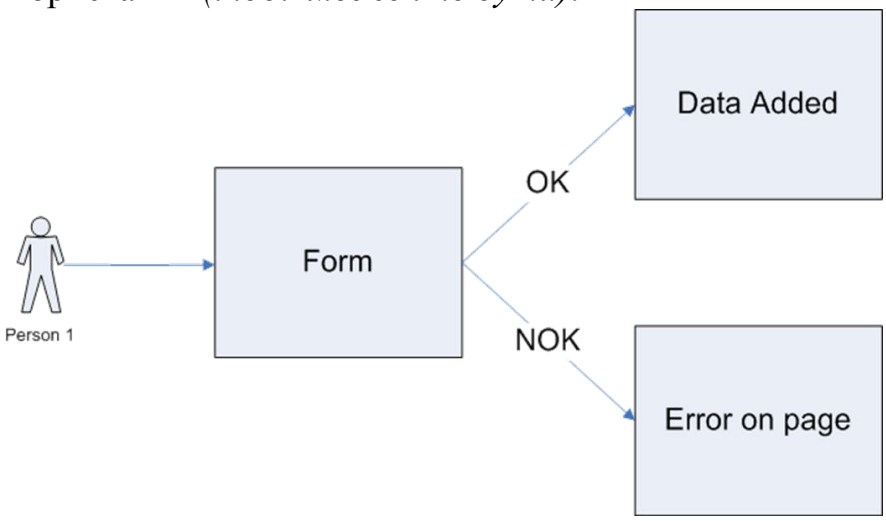


Рис.10.1. Функціональність форми прийому заявок

10.3.1 Аналіз вимог

Аналіз вимог – це процес збору вимог до програмного забезпечення (ПЗ), їх систематизації, документування, аналізу, виявлення суперечностей, неповноти, вирішення конфліктів у процесі розробки програмного забезпечення. В англомовному середовищі також говорять про дисципліну «інженерія вимог» *(requirements engineering)*. У процесі збору вимог важливо брати до уваги можливі протиріччя вимог різних зацікавлених осіб, таких як замовники, для розробників.

Повнота та якість аналізу вимог грають ключову роль в успіху всього проекту. Вимоги до ПЗ повинні бути документовані, здійсненні, тестовані, з рівнем деталізації, достатнім для проектування системи. Вимоги можуть бути функціональними та не функціональними.

Аналіз вимог включає три типи діяльності:

- ***збір вимог:*** спілкування з клієнтами та користувачами, щоб визначити, які їхні вимоги;
- ***аналіз вимог:*** визначення, чи є зібрані вимоги неясними, неповними, неоднозначними або суперечать одне одному; вирішення цих проблем;
- ***документування вимог:*** вимоги можуть бути задокументовані в різних формах, таких як простий опис, сценарії використання, призначені для користувача історії або специфікації процесів.

10.3.2 Визначення набору тестових даних

На підставі вимог до полів та використанні техніки тест-дизайну визначається набір тестових даних:

- в залежності від того, обов'язкове поле чи ні, визначають, які поля необхідно перевірити на порожнє значення, так як можна викликати помилку;

- так як вичерпне тестування не представляється можливим *через величезне число всіляких комбінацій значень,* в першу чергу, необхідно визначити мінімальний набір даних. Це можна зробити, використовуючи такі техніки, як EP та BVA;

- на формі присутнє поле, що має складовий тип (цифри використовуються спільно з символами), володіє спеціальним форматом даних, й тому виділення тестових даних для нього – досить трудомістке завдання;

- по завершенню генерації даних, використовуючи стандартні техніки, можна додати деяку кількість значень на підставі особистого досвіду (техніка EG) – це буде використання спец. символів, дуже довгих рядків, різних форматів даних, регістрів в рядках (Upper, Lower, Mixed cases), негативних та нульових значень, кейвордів Null – Na N – Infinity та ін., а також можна включити все, що вважаєте, може вивести додаток з ладу.

Примітка: Відзначимо, що кількість тестових даних після остаточної генерації буде досить велика, навіть при використанні спеціальних технік тест-дизайну. Тому обмежимося лише кількома значеннями для кожного поля, так як мета даної статті – показати саме процес створення тест кейсів, а не процес отримання конкретних тестових даних.

10.3.3 Вибір тестових даних для кожного окремо взятого поля

Поле Тип звернення.

Так як всі дані входять в 1 клас еквівалентності, тобто НЕ змінюють сам процес виконання прийому заявки, беремо будь-яку (1-шу) позицію в аркуші з очікуваним результатом ОК. Але так як поле реалізоване як список, є також сенс розглянути і граничні умови (техніка BVA), тобто беремо перший і останній елементи. Разом: 1-ша і остання позиції в листі. Очікуваний результат при використанні – ОК.

Поле Контактна особа.

Це обов'язкове поле розміром від 1 до 25 символів (включаючи граничні значення). Перевірка на обов'язковість додає до тестових даних порожнє значення. Проведемо аналіз граничних умов (BVA), отримаємо набір: 0, 1, 2, 24, 25 і 26 символів. Пусте 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, яке НЕ приведе до знаходження нових дефектів, а значить, повторне додавання в домен НЕ має сенсу). У зв'язку з тим, що значення 2 і 24 символу є, з нашої точки зору, некритичними, їх можна не додавати. У підсумку отримуємо, що мінімальний набір даних для тестування поля – це рядки 1 і 25 – ОК, і 0 (пусте значення), 26 символів – NOK.

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

- для номерів з плюсом

  За BVA отримаємо номера з 10, 11, 12 і 14, 15, 16 цифрами, де 10 і 16 – NOK, а 11, 12, 14, 15 – ОК

  Розглядаючи отримані дані з позиції EP виділимо, що 11, 12, 14, 15 входять в один клас еквівалентності. Тому при тестуванні ми можемо використовувати будь-яке з них, але так як 11 і 15 – це межі 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 – ОК, (+123456 78901, +123456789012345)
10 і 16 цифр – NOK; (+1234567890, +12345678901234567)

- для номерів без плюса:

За BVA отримаємо номери з 4, 5, 6 і 9, 10, 11 цифрами.

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

У підсумку маємо:

5 і 10 цифр – OK, (12345, 1234567890)

4 і 11 цифр – NOK; (1234, 12345678901)

Поле Повідомлення.

Підбір даних проводимо по аналогії з полем Контактна особа.

На виході отримуємо значення: рядки 1 і 1024 – OK, і 1025 символів – NOK.

Таблиця 10.3. Результуюча таблиця даних для використання при подальшому складанні тест-кейсів

Поле	OK/ NOK	Значення	Коментар
Тип звернення	OK	Консультація	перший у списку
		Помилка на сайті	останній у списку
	NOK		
Контактна особа	OK	йцукенгшщзйцукенгшщзйцуке	25 символів нижній регістр
		а	1 символ
		ЙЦУКЕНГШЩЗФИВАПРОЛДЖЯЧСМ І	25 символів верхнього регістру
		ЙЦУКЕНГШЩЗфивапролджЯЧСМІ	25 символів змішаний регістр
	NOK		порожнє значення
		йцукенгшщзйцукенгшщзйцукей	довжина більше максимальної (26 символів
		@ # $% ^ &;.?,> | \ / № "! () _ {} [<~	спец.символи (ASCII)
		+12345678901234567890123 45	тільки цифри
		adsadasdasdas dasdasd asasdsads (...) sas	дуже довгий рядок (~ 1Mb)
Контактний телефон	OK	+12345678901	з плюсом - мінімальна довжина
		+123456789012345	з плюсом – максимальна довжина
		1 2 3 4 5	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	з плюсом –букви замість цифр
		yyyxxxxzz	без плюса – букви замість цифр
		+ ### - $$$ -% ^ - & ^ - &!	спец.символи (ASCII)
		+1232132323123213231232 (...) 99	дуже довгий рядок (~ 1Mb)
Повідомлення	OK	йццуйцуйц (...) йцу	максимальна довжина (1024 символу)
	NOK		порожнє значення
		йццуйцуйц (...) йцуц	довжина більша максимальної (1025 символів)
		adsadasdasdas dasdasd asasdsads (...) sas	дуже довгий рядок (~ 1Mb)
		@ ## $$$% ^ & ^ &	тільки спец.символи (ASCII)

10.4 Розроблення шаблону тесту

На підставі техніки CE та наявних варіантів використання *(use case)* створюється шаблон планованого тесту, який описує кроки та очікувані результати в наслідок виконання даних дій (кроків) , але без конкретних даних, які підставляються на наступному етапі розробки тест-кейсів (табл.10.4).

Таблиця 10.4. Приклад шаблону тест-кейса

Дія	*Очікуваний результат*
1. Відкриваємо форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка "Відправити" не активна
2. Заповнюємо поля форми: Тип звернення Контактна особа Контактний телефон Повідомлення	Поля заповнені Кнопка "Відправити" – активна (Enabled)
3. Натискаємо кнопку "Відправити"	Якщо введені дані коректні: Повідомлення "Заявку відправлено" виведено на екран. Нова заявка з'явилася в списку на сторінці "Заявки". Якщо введені дані НЕ коректні: Валідаційні сполучення з усіма помилками виведено на екран. Заявка НЕ з'явилася в списку на сторінці "Заявки".

10.5 Написання тест- кейсів на підставі первинних вимог, тестових даних і шаблону тесту

Після того, як тестові дані та кроки тесту готові, розпочнемо безпосередньо розробку тест-кейсів. Тут нам допоможуть такі методи комбінування, як:

Послідовний перебір. Являє собою перебір всіх можливих комбінацій наявних значень. Таким чином, виходить, що кількість тест- кейсів буде дорівнювати добутку кількості варіантів тестових даних для кожного поля. Для нашого конкретного прикладу отримаємо 1170 тест- кейсів.

Попарний перебір (Pairwise Testing). Найчастіше збої викликає НЕ складне поєднання всіх параметрів,а поєднання лише пар и параметрів. Техніка попарного перебору , дозволяє створити тестові набори, які комбінують дані з двох полів. Завдяки цьому, кількість отриманих на виході тесткейсів у рази менша, ніж при комбінуванні того ж 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 отримаємо всього 118 тест-кейсів.
По завершенню підготовки комбінацій даних підставляємо їх в шаблон тест-кейса і в результаті маємо набір тестових випадків, що покриває тестовані нами вимоги до форми прийому заявок.

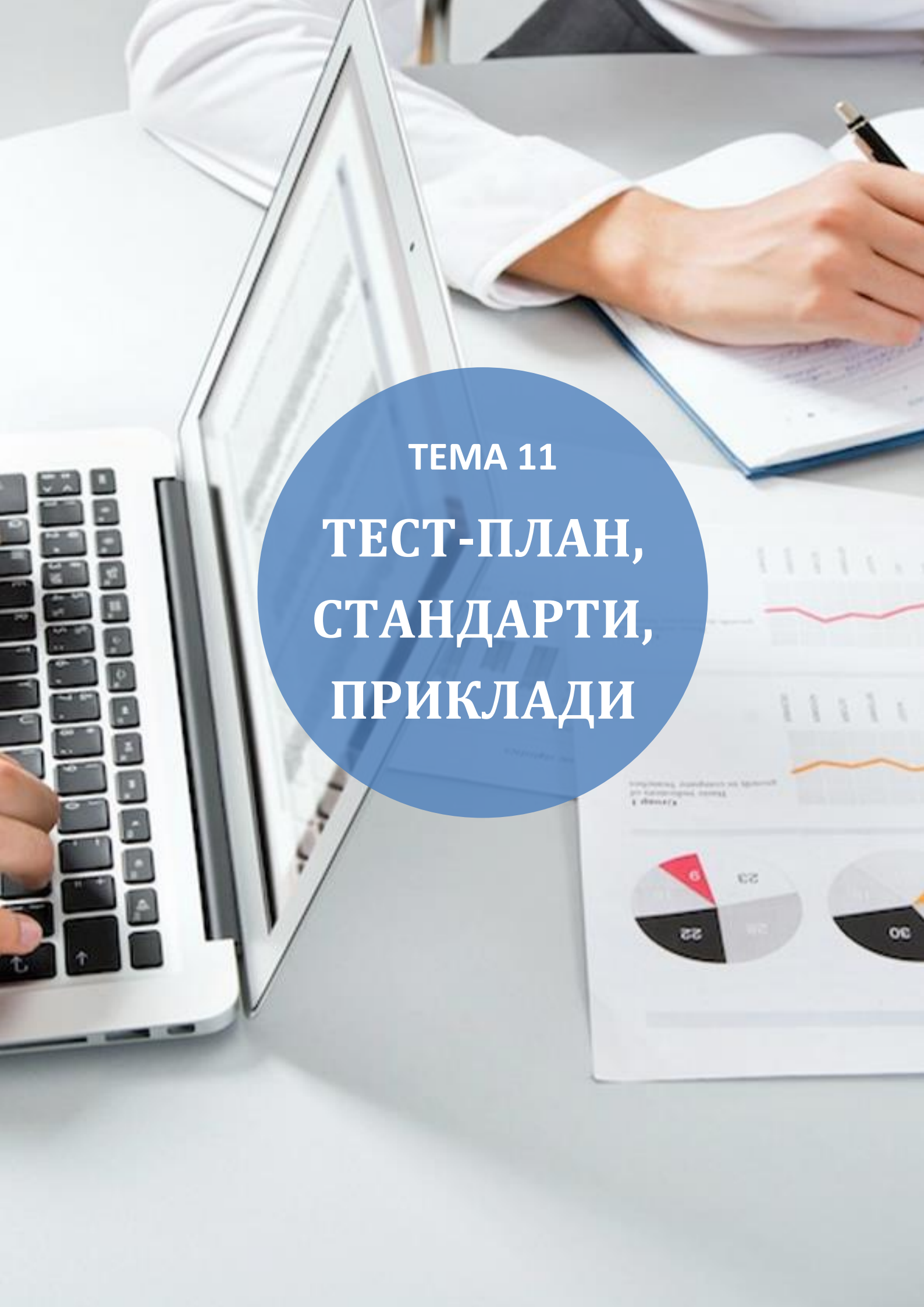
Примітка: *тест-кейси поділяються за очікуваним результатом на позитивні (табл.10.5) та негативні тест-кейси (табл.10.6).*

Таблиця 10.5. Позитивний тест-кейс (всі поля ОК)

Дія	***Очікуваний результат***
1. Відкриваємо форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка "Відправити" не активна
2. Заповнюємо поля форми: Тип звернення= Консультація Контактна особа = йцукенгшщзйцукенгшщзйцуке Контактний телефон = + 7-916-111-11-11 Повідомлення	Поля заповнені Кнопка "Відправити" – активна (Enabled)
3. Натискаємо кнопку "Відправити"	Повідомлення "Заявку відправлено" виведено на екран. Нова заявка з'явилася в списку на сторінці "Заявки".

Таблиця 10.6. Негативний тест-кейс (поле Контактна особа –NOK)

Дія	***Очікуваний результат***
1. Відкриваємо форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка "Відправити" не активна
2. Заповнюємо поля форми: Тип звернення = Консультація Контактна особа = @ # $% ^ &;.?,> | \ / № "! () _ {} [<~ Контактний телефон = (916) 333-33-33 Повідомлення = йццуйцуйц (...) йцу - 1024 символу	Поля заповнені Кнопка "Відправити" – активна (Enabled)
3. Натискаємо кнопку "Відправити"	Валідаційні сполучення з усіма помилками виведено на екран: У полі "Контактна особа " заборонено використання цифр і спец. символів. Заявка НЕ з'явилася в списку на сторінці "Заявки".



ТЕМА 11

ТЕСТ-ПЛАН, СТАНДАРТИ, ПРИКЛАДИ

11 ТЕСТ-ПЛАН, СТАНДАРТИ, ПРИКЛАДИ

Тест-план (Test Plan) – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладів, критеріїв початку та закінчення тестування, вимог до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення [76].

11.1 Цілі створення тест-плану *(test plan)*

Тест-план є важливою складовою будь-якого грамотно організованого процесу тестування, оскільки містить в собі всю необхідну інформацію, що описує даний процес. Незважаючи на те, що в багатьох компаніях тест-план є формальністю, вміле його створення та оновлення приносить масу користі [77]:

- узгодження обсягів та стратегії тестування різних складових тестованого ПЗ з іншими учасниками проектної команди;
- пріоритезація завдань з тестування;
- своєчасне планування ресурсозатрат на тестування;
- облік ресурсів (ПЗ, обладнання), необхідних для тестування;
- завчасний облік ризиків, які можуть виникнути в процесі реалізації плану, і впровадження попереджувальної стратегії.

11.2 Рекомендації з написання тест-плану *(test plan)*

Хороший тест-план повинен, як мінімум, описувати наступне [76]:

1. ***Що потрібно тестувати?*** – Опис об'єкта тестування: системи, додатки, обладнання.
2. ***Що будете тестувати?*** – Список функцій та опис тестованої системи та окремо її компоненти.
3. ***Як будете тестувати?*** – Стратегія тестування, а саме: види тестування та їх застосування по відношенню до об'єкта тестування.
4. ***Коли будете тестувати?*** – Послідовність проведення робіт: підготовка *(Test Preparation)*, тестування *(Testing)*, аналіз результатів *(Test Result Analisys)* у розрізі запланованих фаз розробки.
5. ***Критерії початку тестування:***
   - готовність тестової платформи *(тестового стенду);*
   - завершеність розробки необхідного функціоналу;
   - наявність всієї необхідної документації.
6. ***Критерії завершення тестування*** – результати тестування задовольняють критерії якості продукту:

- вимоги до кількості відкритих багів виконані;
- витримка певного періоду без зміни початкового коду додатка Code Freeze (*CF);*
- витримка певного періоду без відкриття нових багів Zero Bug Bounce (ZBB).

Відповівши в тест-плані на перераховані вище питання, можна вважати, що в наявності є непогана чернетка документу з планування тестування, яку ***рекомендується доповнити наступними пунктами:***

- оточення тестованої системи *(опис програмно-апаратних засобів);*
- необхідне для тестування обладнання та програмні засоби (тестовий стенд та конфігурація, програми для автоматизованого тестування та ін.);
- ризики та шляхи вирішення.

11.3 Види тест-планів *(test plan)*

Найчастіше на практиці доводиться стикатися з наступними видами тест-планів [76]:

- **майстер тест-план** *(Master Plan or Master Test Plan)*, який створюється у двох випадках [77]:
  - якщо продукт має безліч релізів або ітерацій, між якими зберігається загальна інформація, яку немає сенсу повторювати;
  - якщо різні тестові команди працюють над одним продуктом, виконуючи різні завдання, які необхідно об'єднати в рамках одного документа.

При цьому в майстері тест-плані міститься інформація [77]:

  - загальна інформація про продукт, посилання на документацію, баг-трекер та інші проектні ресурси;
  - загальні правила тестування: вимоги до дефектів, які заводяться, умови прийняття збірки на тестування;
  - критерії готовності продукту до випуску, метрики якості;
  - інструменти та техніки, які були використані.

- **тест-план** *(Test Plan)*, або детальний тест-план, який створюється до кожного релізу / ітерації або для кожної команди в рамках проекту.

Основна мета якого – коротко та доступно відобразити завдання тестування. ***При цьому в детальному тест-плані міститься інформація [77]:***

  - перелік областей тестування з пріоритетами;

- стратегія тестування;
- проектні ризики;
- ресурси, необхідні для виконання завдань;
- проектний план *(терміни готовності основних завдань).*

- **план приймального тестування** *(Product Acceptance Plan)* – документ, що описує набір дій, пов'язаних з приймальним тестуванням (*стратегія, дата проведення, відповідальні працівники та ін.)*

У випадку з невеликим проектом створюється один тест-план, що містить у собі зведену інформацію з майстер тест-плану та детального тест-плану.

Очевидна відмінність майстра тест-плану від просто тест-плану в тому, що майстер тест-плану є більш статичним за рахунок того, що містить у собі високорівневу *(High Level)* інформацію, яка не схильна до частих змін у процесі тестування та перегляду вимог. Сам же детальний тест-план, який містить більш конкретну інформацію про стратегії, види тестування, розклад виконання робіт, є "живим" документом, який постійно зазнає змін, що відображають реальний стан речей на проекті [76].

У повсякденному житті на проекті може бути один майстер тест-плану та кілька детальних тест-планів, що описують окремі модулі однієї програми.

11.4 Стандарти тест-планів *(test plan)*

Кожна методологія або процес пропонує свої формати представлення планів тестування**.**

Існує два міжнародних стандарти опису планів тестування:

- стандарт *IEEE 829 (Institute of Electrical and Electronics Engineers)*;
- стандарт *RUP (Rational Unified Process)*.

11.4.1 Стандарт *IEEE 829*

Розглянемо структуру плану тестування за даним шаблоном [78]:

1. ***Ідентифікатор тест-плану (Test Plan Identifier)*** – унікальний номер для ідентифікації документа всередині компанії. Цей номер може вказувати на те, чи є документ майстер тест-планом або детальним тест–планом, а також він вказує і на ітерацію (етап) тестування, на якому застосовується. Це дозволяє полегшити та систематизувати організацію процесу тестування.

2. ***Посилання (Preferences)*** – список усіх документів, які підтримують тест-план. Цей розділ призначений для того, щоб не копіювати частини тексту з перерахованих документів та не створювати надлишок інформації в тест-плані. Можуть бути згадані такі документи:
   - план проекту;
   - технічні вимоги;
   - дизайн-документ високого рівня;
   - деталізований дизайн-документ;
   - стандарти розробки і тестування;
   - принципи та методології тестування;
   - корпоративні стандарти і принципи.

3. ***Вступ (Introduction)*** – містить опис мети тест-плану та його рівня (майстер тест-план або детальний тест-план). По суті, це узагальнювальна частина всього документу, в якій у загальних рисах описується, що саме буде протестовано й які методи будуть для цього використані.

4. ***Об'єкти тестування (Test Items)*** – ті речі, які будуть протестовані в рамках даного тест-плану. По суті, це список того, що необхідно протестувати. Такий список можна скласти, виходячи з компонентів програмного продукту або ж з документації. Інформація даного розділу також містить номери версій тестованого продукту та вимоги до конфігурації *(якщо необхідно)*.

5. ***Ризики в програмному забезпеченні (Software Risk Issues).*** У даному пункті необхідно визначити критичні аспекти у тестуванні даного програмного забезпечення, такі як:
   - надання продукта третьою стороною;
   - нова версія пов'язаного програмного забезпечення;
   - можливість використання (розуміння) нових інструментів;
   - надзвичайно складні функції;
   - зміни в компонентах, які раніше містили критичні помилки;
   - безпека;
   - кілька інтерфейсів;
   - вплив на клієнта;
   - державне регулювання та правила.

   Ще однією ключовою галуззю ризику є нерозуміння оригінальних вимог. Це може статися на рівні управління, користувачів та розробників. Завжди слід уточнювати неконкретизовані або незрозумілі вимоги.

6. ***Функції, які будуть протестовані*** – список того, що буде перевірено, що з точки зору користувача виконує система. Це не технічний опис програмного забезпечення, а опис тестованих функцій з боку кінцевого користувача системи. Для кожної функції слід встановити рівень ризику (використовуючи, наприклад, просту шкалу High, Medium, Low). При формуванні тест-плану необхідно бути готовим пояснити, чому саме цей рівень ризику був обраний для конкретної функції.

Слід зазначити, що розділ 4 та розділ 6 дуже схожі. Відмінність полягає в точці зору. Розділ 4 – технічний опис, який включає номери версій ПЗ та іншу технічну інформацію. Розділ 6 складається з погляду користувачів, які не розуміють технічної термінології та сприймають процеси й функції залежно від того, як ці функції пов'язані з їх роботою.

7. ***Функції, які не будуть протестовані*** – список того, що не буде протестовано як з точки зору користувачів, так й з точки зору управління конфігураціями. Аналогічно розділу 6, це не є технічним описом функцій.

Необхідно визначити, чому ті чи інші функції виключені з тестування – цьому може ***бути кілька причин:***

- функція не буде включена до поточного релізу ПЗ;
- низький рівень ризику, функція була протестована раніше та вважається стабільною;
- функція увійде до релізу, але не буде задокументована як частина релізу нової версії ПЗ.

8. ***Підхід (Стратегія)*** – загальна стратегія тестування тест-плану, яка повинна відповідати рівню (виду) плану (майстер, приймальний та ін.) й повинна бути узгоджена з усіма вищими та нижчими рівнями плану.

Повинні бути зазначені загальні правила й процеси:

- Чи будуть використовуватися спеціальні інструменти й які саме?
- Чи потребують інструменти додаткового вивчення?
- Які метрики (показники) будуть зібрані в ході тестування?
- На якому з рівнів буде зібраний кожен показник?
- Як буде здійснюватися управління конфігурацією?
- Яка кількість конфігурацій буде протестована?
- Апаратне забезпечення.
- Програмне забезпечення.
- Комбінації програмного та апаратного забезпечення.
- Які рівні регресійного тестування будуть проведені й в якій кількості?

- Регресійне тестування буде базуватися на серйозності чи на знайдених дефектах?
- Яким чином слід обробляти елементи вимог та дизайну у випадку, якщо їх неможливо протестувати?

При складанні майстертест-плану також повинна бути позначена загальна стратегія тестування всього проекту та покриття вимог. Вкажіть, чи є особливі вимоги до тестування:

- будуть протестовані тільки цілі компоненти;
- зазначена група елементів (компонентів) повинна бути протестована разом.

9. ***Критерій «Пройдено / Провалено» (Item Pass/Fail Criteria)*** – кожен тест-план повинен чітко визначати критерії завершення тестування:
   - на рівні компонентного тестування *(unit test level)* ***критерії можуть бути наступними:***
     - всі тест-кейси виконані;
     - вказану кількість *(у відсотках)* тест-кейсів виконано;
     - вказану кількість *(у відсотках)* тест-кейсів виконано із зазначеним допустимим відсотком дрібних (незначних) дефектів;
     - інструмент покриття коду *(code coverage tool)* визначає, що весь код покритий.

   - на рівні майстер тест-плану ***критерії можуть бути наступними:***
     - всі плани нижніх рівнів виконані;
     - вказану кількість планів виконано без помилок й певний відсоток – з дрібними дефектами.

Також може бути створений окремий рівень критеріїв, або критерії можуть бути описані у вимогах для планів вищих рівнів.

Дефект та збій в рамках тест-плану можна визначити наступним чином:

- дефектом є те, що може викликати збій й може бути прийнятним, щоб залишатися в додатку.
- збій є результатом дефекту, видимим для користувача.

10. ***Критерії припинення та поновлення*** тестування, коли необхідно вказати, за яких умов та в яких ситуаціях слід призупинити й продовжити виконання тестування. Найчастіше таким критерієм може бути загальна кількість дефектів (або кількість дефектів певного типу), після досягнення якого, подальше тестування не має сенсу.

Таким чином, необхідно визначити певну допустиму кількість дефектів, незважаючи на яку, тестування слід продовжувати.

Продовження тестування після виникнення критичної помилки може створити умови, які можуть бути ідентифіковані як дефекти, які насправді є наслідком проігнорованої раніше помилки.

11. ***Що надається в ході тестування (Test deliverables)***, а саме: «що буде надано як частина тест-плану?»:
    - документ тест-плану;
    - тест-кейси;
    - специфікації тест-дизайну;
    - інструменти та результат роботи;
    - емулятори;
    - статичні та динамічні генератори;
    - логи помилок та логи виконання програм;
    - звіти про проблеми та коригувальні дії.

Єдина річ, яка не надається в ході тестування, – це саме програмне забезпечення, оскільки воно надається в ході розробки.

12. ***Майбутні тестові завдання***, якщо процес тестування проходить у кілька етапів або розробка продукту ведеться шляхом інкрементного додавання версій, тест-план може не охоплювати деякі частини програмного продукту. Ці частини повинні бути описані в тест-плані, щоб уникнути тестування ще неготових частин програми, й, як наслідок, заведення дефектів по функціоналу, який ще не готовий до тестування.

Якщо програмне забезпечення розробляється третьою стороною, у данному розділі може міститися опис завдань як для зовнішніх, так і для внутрішніх груп тестувальників.

13. ***Потреби середовища.*** У цьому розділі перераховуються специфічні вимоги для даного тест-плану, такі як:
    - спеціальне програмне забезпечення (симулятори, статичні генератори та ін);
    - яким чином надаються тестові дані та чи повинні надаватися специфічні набори даних або специфічний діапазон значень?
    - який обсяг тестування буде виконаний для кожного компонента системи?
    - спеціальні вимоги до потужності;
    - специфічні версії допоміжного програмного забезпечення;
    - обмежене використання системи під час тестування.

14. ***Потреби у персоналі та навчанні.***
    - Навчання роботі з програмою або системою.
    - Навчання роботі з інструментами для тестування.
    - Хто несе відповідальність за навчання персоналу.

15. ***Відповідальність.*** Хто за що відповідає? Цей розділ включає всі аспекти тест-плану, наприклад:
    - оцінка ризиків;
    - вибір функцій, які будуть і не будуть протестовані;
    - вибір загальної стратегії тестування для тест-плану даного рівня;
    - перевірка наявності всіх необхідних для тестування елементів;
    - регулювання конфліктів у розкладі, особливо, коли тестування виконується на виробничій системі;
    - Хто забезпечує необхідне навчання?
    - Хто приймає критичні рішення по об'єктах, які не покриває даний тест-план?

16. ***Розклад***, формування якого повинно ґрунтуватися на реальних та затверджених термінах. Неточна оцінка часу тестування може містити терміни виконання загального плану розробки програмного забезпечення, оскільки тестування є його невід'ємною частиною.

Рекомендується вказувати терміни проведення тестування з прив'язкою до дати закінчення етапу розробки ПЗ. Це захистить команду тестувальників від сприйняття їх як винуватців несвоєчасного випуску ПЗ у тому випадку, якщо затримка сталася з вини розробників. Наприклад, якщо тестування системи починається після надання фінальної збірки, тестування розпочнеться наступного дня після її надання. Якщо фінальна збірка надана пізніше визначеного терміну, тестування починається на наступний день після того, як вона була отримана тестувальниками, а не в конкретно зазначений день. Це називається відносним або залежним плануванням.

17. ***Планування ризиків та непередбачених обставин.***

Які загальні ризики на проекті з акцентом на процес тестування:
- відсутність кадрових ресурсів перед початком тестування;
- відсутність доступу до необхідного апаратного або програмного забезпечення, даних або інструментів;
- несвоєчасне надання програмного, апаратного забезпечення або інструментів;
- затримки в навчанні роботі з додатком і / або інструментами;
- зміна оригінальних вимог або дизайну.

Визначте, що необхідно зробити у разі виникнення різних ситуацій в процесі тестування, наприклад:

- визначення вимог буде завершено 1.01.20XX, й, якщо вимоги будуть змінені після цієї дати, будуть виконані наступні дії:
  - розклад тестування та розклад розробки буде зміщено на відповідну кількість днів – відбувається рідко, оскільки більшість проектів має фіксовані терміни випуску;
  - кількість проведених тестів буде зменшено – може знизити загальний рівень якості продукту, що випускається;
  - кількість допустимих дефектів буде збільшено – може знизити загальний рівень якості продукту, що випускається;
- до команди тестувальників будуть додані допоміжні ресурси;
- команда тестувальників буде працювати понаднормово;
- мета плану може бути змінена;
- може бути проведена оптимізація ресурсів.

Слід зауважити, що, якщо взагалі нічого не вжити з наведеного списку, то тестування програмного продукту буде скорочено або взагалі опущено, що, звичайно ж, негативно позначиться на якості виробленого ПЗ.

18. ***Затвердження***: хто має право підтвердити, що процес тестування завершений, й дозволити перехід на наступний рівень у розробці проекту (залежно від рівня плану).

На рівні майстер тест-плану це можуть бути всі зацікавлені сторони. При визначенні процесу затвердження небхідно враховувати, ***хто входить до цільової аудиторії:***

- аудиторія плану рівня модульного тестування відрізняється від плану рівня інтеграційного, системного тестування та рівня майстер тест-плану;
- рівні та типи знань також будуть відрізнятися на різних рівнях тест-планів;
- програмісти дуже технічні, але часто можуть не мати чіткого уявлення про бізнес-процес та управління проектом;
- люди можуть мати різний рівень ділової хватки та дуже низький технічний рівень;
- необхідно бути обережними з користувачами, які заявляють про високий рівень технічної підготовки й з програмістами, які заявляють про повне розуміння бізнес-процесу. Такі особистості можуть принести більше шкоди ніж користі, якщо насправді не володіють тими навичками, про які заявляють.

19. ***Глоссарій:*** використовується для визначення термінів та скорочень, що використовуються у документі й тестуванні в цілому, для того, щоб не допустити плутанини та сприяти полегшенню комунікацій на проекті.

11.4.2 Стандарт *RUP*

Розглянемо структуру плану тестування за даним шаблоном [79]:

1. ***Вступ.***
   - 1.1. ***Мета.***
     Опишіть існуючу інформацію про проект та програмні компоненти, які повинні бути перевірені, рекомендовані вимоги для тестування й загальну стратегію тестування. Також повинні бути визначені необхідні ресурси й надані елементи для проекту.
   - 1.2. ***Передумови (Background)***.
     У цьому розділі необхідно дуже коротко описати основний функціонал та завдання. Необхідно включити таку інформацію, як головні функції системи, її архітектуру та історію проекту.
   - 1.3. ***Тестова область (Test Scope).***
     - опишіть стадії тестування (*компонентне, інтеграційне, системне*) та типи тестування *(наприклад, функціональне, навантажувальне та ін.)*;
     - наведіть короткий список тих функцій системи, що будуть / не будуть протестовані;
     - перерахуйте будь-які пропозиції, що виникли в процесі написання документа;
     - перерахуйте ризики та непередбачені обставини, які можуть виникнути в ході розробки та реалізації тестів;
     - перерахуйте обмеження, які можуть вплинути на проектування, розробку та реалізацію тестів.
   - 1.4. ***Ідентифікація проекту.***
     Необхідно перерахувати наявну документацію по проекту та доступність на момент написання даного тест-плану:
     - специфікація вимог;
     - специфікація функціоналу;
     - звіти про використання;
     - план проекту;
     - специфікація дизайну;
     - прототип;

- посібники користувача;
- бізнес-модель;
- бізнес функції та правила;
- оцінювання ризиків.

2. ***Вимоги для тестування –*** список, який представляє перелік того, що необхідно протестуват, в якому необхідно описати функціональні та нефункціональні вимоги, які були позначені, як цілі тестування.

3. ***Стратегія тестування –*** рекомендований підхід до тестування. У попередньому розділі пояснено, що буде протестовано, а у цьому розділі – якими методами тестування буде виконуватися.

Для кожного типу тестування необхідно навести докладний опис й позначити, для чого даний вид тестування виконується. Основними складовими стратегії є техніки тестування, які будуть використовуватися, й критерії завершення тестування.

3.1.1 *Типи тестування.*

У кожному з пунктів необхідно визначити об'єкт тестування, техніку (методи), за допомогою яких буде проводитися даний вид тестування, критерій проходження тесту. Також опис може включати інформацію про особливості проведення даного виду тестування (якщо такі є).

3.1.2 *Тестування цілісності даних та баз даних.*

Бази даних та процеси мають бути протестовані, як підсистема у рамках проекту без взаємодії з інтерфейсом користувача. Додаткові дослідження в системі управління базами даних необхідні для того, щоб визначити засоби та методи, за допомогою яких буде виконуватися даний вид тестування.

3.1.3 *Тестування бізнес-циклу.*

Даний вид тестування передбачає імітацію діяльності, здійснюваної на проекті протягом довгого часу. Повинен бути визначений період, наприклад, один рік, й всі транзакції та дії, які виконуються над проектом протягом даного проміжку часу, повинні бути виконані.

3.1.4 *Тестування інтерфейсу користувача (UI Testing).*

На даному етапі виконується перевірка взаємодії користувача з програмним забезпеченням. Мета – переконатися, що

інтерфейс надає користувачеві необхідний доступ та функції навігації в системі. Крім того, тестування інтерфейсу користувача гарантує, що всі об'єкти працюють згідно з очікуваннями та відповідають корпоративним стандартам та стандартам галузі.
Таким чином, перевіряється зміст та зовнішній вигляд кожного вікна програмного продукту на відповідність дизайну та вимогам, а також правильність взаємодії між вікнами програми.

3.1.5 ***Профілювання продуктивності.*** Являє собою тестування продуктивності, при якому вимірюються часово-залежні вимоги такі, як час відгуку, швидкість виконання операцій та ін. Метою даного виду тестування є підтвердження того, що всі вимоги продуктивності виконані.
Профілювання продуктивності реалізується та виконується для окремих профілів з налаштовуваною поведінкою та умовами середовища такими, як конфігурація устаткування й робоче навантаження.

3.1.6 ***Навантажувальне тестування*** – тест продуктивності, який піддає систему різним навантаженням для оцінки продуктивності та здатності системи продовжувати коректну роботу під даними навантаженнями.
Мета такого тестування полягає в тому, щоб визначити та гарантувати коректну роботу системи при навантаженні понад максимально очікуваного рівня.

3.1.7 ***Стрес-тестування*** – тестування продуктивності, яке реалізується та виконується з метою виявлення помилок, пов'язаних з недостатністю ресурсів. Недостатня кількість пам'яті або місця на жорсткому диску може спричинити виникнення помилок, які не проявляються в умовах, коли системних ресурсів достатньо. Також дефекти можуть виникати при використанні загальних ресурсів таких, як бази даних або пропускна здатність мережі. Стресове тестування може також проводитися з метою визначення рівеня максимального навантаження, при якому система продовжує виконувати встановленні функції.

3.1.8 ***Об'ємне тестування:*** завдання якого полягає в отриманні оцінки продуктивності при збільшенні обсягів даних у базі даних програми.

При цьому відбувається:

- вимірювання часу виконання обраних операцій при певних інтенсивностях виконання цих операцій;
- може здійснюватися визначення кількості користувачів, що одночасно працюють з додатком.

3.1.9 ***Тестування безпеки та контролю доступу*** зосереджено на двох основних сферах безпеки:

- безпека рівня додатків, включаючи доступ до даних або бізнес функцій;
- безпека рівня системи, включаючи авторизацію або віддалений вхід у систему.

3.1.10 ***Тестування відмовостійкості та відновлення***. У ході проведення даного виду тестування перевіряється стійкість системи до програмних та апаратних збоїв та здатність до відновлення після таких збоїв без втрати інформації. Прикладами таких збоїв можуть бути:

- відключення живлення клієнта;
- відключення живлення сервера;
- порушення мережевого з'єднання;
- пошкоджений елемент у базі даних та ін.

3.1.11 ***Тестування конфігурації.*** Перевіряє роботу системи, яка підлягає тестованню на різних конфігураціях програмного та апаратного забезпечення.

3.1.12 ***Тестування установки.*** По-перше, даний вид тестування виконується для того, щоб переконатися, що ПЗ коректно встановлюється за різних умов, наприклад, нова установка, оновлення, повна або вибіркова установка. Також перевіряється установка ПЗ в умовах недостатності системних ресурсів (наприклад, місця на жорсткому диску). По-друге, необхідно переконатися, що одного разу встановлене ПЗ коректно функціонує при наступних запусках.

3.1.13 ***Інструменти.*** У цьому розділі необхідно перелічити всі інструменти, за допомогою яких буде виконуватися тестування, наприклад:

- система адміністративного управління тестуванням;
- система відстеження дефектів;
- інструменти для функціонального тестування;

- інструменти для тестування продуктивності;
- засіб моніторингу тестового покриття;
- засіб профілювання (профайлер);
- інструменти СУБД.

4. ***Ресурси.***
У цьому розділі представлені рекомендовані ресурси для проекту, основні обов'язки персоналу і набір знань та навичок.

4.1 ***Ролі.*** Цей пункт містить перелік посад усіх, хто задіяний на проекті, та основні обов'язки. Найчастіше список персоналу містить наступні посади:
- тест-менеджер – керівництво процесом тестування, комунікації;
- тест-дизайнер – створення тест-плану, моделі тестування;
- тестувальник – виконання тест-кейсів, логування, написання звітів про помилки, запити на зміну документації;
- адміністратор тестової системи – управління системою тестування, встановлення та доступ до тестової системи;
- адміністратор баз даних – управління тестовими даними (базою даних);
- дизайнер – створення тест-кейсів та тест-ранів.

4.2 ***Система.*** Даний розділ містить опис системних ресурсів для тестування та може містити наступні елементи:
- сервер бази даних;
- тестові ПК клієнта;
- тестовий репозиторій;
- тестові ПК розробників.

5. ***Основні етапи проекту*** містять основні етапи тестування проекту та терміни виконання, як правило, виділяють наступні етапи:
- планування тестування;
- тест-дизайн;
- реалізація тестів;
- виконання тестів;
- оцінка тестів.

6. ***Надаване у процесі тестування*** описує різні документи, інструменти та звіти, які будуть створені, та саме ким, для кого й коли будуть створені.

- ***модель тестування*** описує звіти та інші артефакти, які будуть створені та розподілені в рамках тестової моделі;
- ***тестові журнали (логи)*** описують методи та інструменти, які використовуються для запису та створення звітів;
- ***звіти про помилки*** описують методи та інструменти для написання звітів про помилки, управління та моніторингу статусу.

11.5 Приклад тест-плану

Тест-план для проекту top-page.ru [80]:

Зміст

✓ ***Зміни в документі***

Дата	Автор	Вид змін
01.09.2011	Дмитриева І.В.	Створення
02.09.2011	Дмитриева І.В.	Створення
06.09.2011	Пономаренко Е.В.	Редагування

✓ ***Вступ***

- Призначення документа

  Метою даного тест-плану є опис процесу тестування сайту Top-page.ru *(повна адреса* http://top-page.ru*)*.

  Даний документ дозволяє отримати уявлення про планові роботи, терміни, а також ціни за послуги тестування.

  У даному документі не передбачається опис тест-кейсів, знайдених дефектів, а також їх аналіз.

- ***Терміни***
  - ***Проект*** – Проект «Top-page.ru» служить для створення закладок улюблених адрес Інтернету по типу стартових сторінок Google Chrome, Opera.
  - ***Тестування*** – процес, спрямований на виявлення дефектів та помилок у програмному продукті шляхом пошуку невідповідностей між очікуваним та отриманим результатом. Процес тестування не припускає аналізу отриманих проблем.
  - ***Функціональне тестування*** – тестування функцій програми на відповідність вимогам.
  - ***Стрес-тестування*** – оцінка надійності та стійкості системи в умовах перевищення меж нормального функціонування.
  - ***Тестове середовище*** – набір програмного забезпечення для відтворення дій користувача, максимально наближених до реальних.
  - ***ТЗ (Технічне завдання)*** – документ, що описує набір технічних та функціональних вимог до програмного продукту.
  - ***Юзер-сторі*** – покрокова інструкція, яка відтворює дії користувача.

- ***Мета тестування***

  Метою тестування Проекту є перевірка всіх функціональних можливостей на різних версіях браузерів, при різних дозволах монітора, а також проведення серії стрес-тестів для виявлення вузьких місць та вразливостей Проекту.

Підсумковими документами процесу тестування будуть:

- звіт про результати тестування, який включає опис тестових середовищ й знайдених дефектів та недоліків;
- висновок тестувальників про загальний стан Проекту, що представляє собою графік співвідношення критичних дефектів до загального їх числа;
- тестування буде проводитись у ручному режимі, без використання автоматизованих систем.

- ***Версія проекту***

  Top-page.ru реліз 10.004, Ітерація 3.

✓ ***Стратегія процесу тестування***

Плануються три етапи проведення процесу тестування. Перший етап полягає в аналізі ТЗ, формуванні критичного чек-листу, формуванні тест-плану, а також частковому прогоні функціональних тестів. Другий етап буде присвячений деталізації функціонального чек-листа та детальному

прогоні функціональних тестів з виявленням та описом дефектів. На третьому етапі буде проведене стрес-тестування з описом знайдених дефектів. Таким чином, досягається максимальна деталізація глибини тестування, що, у свою чергу, дозволяє точніше визначити ресурси, що витрачаються, й також дозволяє розробникам Проекту почати виправляти дефекти на ранніх етапах.

Зважаючи на відмову від занесення дефектів у багтрекер, усі виявлені дефекти будуть передаватися менеджерам Проекту письмово електронною поштою.

На першому етапі буде застосований смоук-тестінг, при якому будуть уточнюватися вимоги, визначатися й конфігуруватися тестові середовища. До початку другого етапу буде сформований критичний чек-лист, а також чек-лист по функціональному тестуванню та юзер-сторі. На другому етапі проводиться детальне тестування функціоналу Проекту, збираються й описуються дефекти. Кожен чек-лист проганяється для кожного браузера. Третій етап завершує роботи з тестування. На даному єтапі проводиться встановлений набір тестів для виявлення вразливостей. Такий вид тестування досить витратний за часом, тому необхідний набір тест-кейсів розробляється спільно з розробниками Проекту.

Браузери, затверджені до перевірки:

Internet Explorer 6
Internet Explorer 7
Internet Explorer 8
Internet Explorer 9
Firefox 4
Firefox 6
Google Chrome 12
Opera 10

- ***Методи тестування***

- ***Смоук-тестування***
  Мета: Окреслити скелет чек-листів для функціонального тестування та стрес-тестування. Даний метод застосовується з мінімальним набором тестів й мінімальним ТЗ. Метою даного тестування не є виявлення помилок, хоча, якщо на даному етапі будуть знайдені явні дефекти, то вони будуть зафіксовані тестувальником.

- ***Типи тестування***

- ***Функціональне тестування***

Мета: Виявлення функціональних помилок, невідповідностей ТЗ очікуванням користувача шляхом реалізації стандартних, а також нетривіальних тестових сценаріїв.

Класифікація функцій:

1. ***Реєстрація / Авторизація***
   a. Реєстрація користувача.
   b. Авторизація користувача.
   c. Анонімний користувач.
   d. Відновлення паролю.
   e. Редагування облікового запису.
2. ***Робота с закладками***
   a. Додавання закладки.
   b. Видалення закладки.
   c. Редагування закладки.
   d. Переміщення закладки.
   e. Перехід до обраного сайту.
3. ***Геотаргетинг***
   a. Вибір міста.
   b. Автоматичне визначення міста користувача.
4. ***Пошук***
   a. Пошук з використанням Google.
   b. Пошук з використанням Yandex.
   c. Пошук з використанням Wiki.
5. ***Сервіси***
   a. Перехід до вибраного сервісу.
   b. Нетривіальні сценарії.
   b. Використання сервісу на різних вкладках браузера.
   c. Використання сервісу від різних користувачів у різних браузерах.

- ***Тестування в певному середовищі***
  Мета: Перевірити коректність роботи та дизайн Проекту в різних браузерах й при різних дозволах монітора.

Опис конфігурацій:

Браузер/Розширення	1024*768	1280*800	1680*1050
Internet Explorer 6	Так	Так	Так
Internet Explorer 7	Так	Так	Так
Internet Explorer 8	Так	Так	Так
Internet Explorer 9	Так	Так	Так
Firefox 4	Так	Так	Так
Firefox 6	Так	Так	Так
Google Chrome 12	Так	Так	Так
Opera 10	Так	Так	Так

Окремі конфігурації для перевірки кожного браузера з відключеним JS та відключеними зображеннями.

- ***Стрес-тестування***
  Мета: Виявити вразливості в роботі Проекту шляхом використання масивних даних, довгих посилань, некоректних даних тощо.
  Опис процесу:
  Даний вид тестування та тест-кейси обумовлюються та описуються окремо разом з розробниками Проекту.

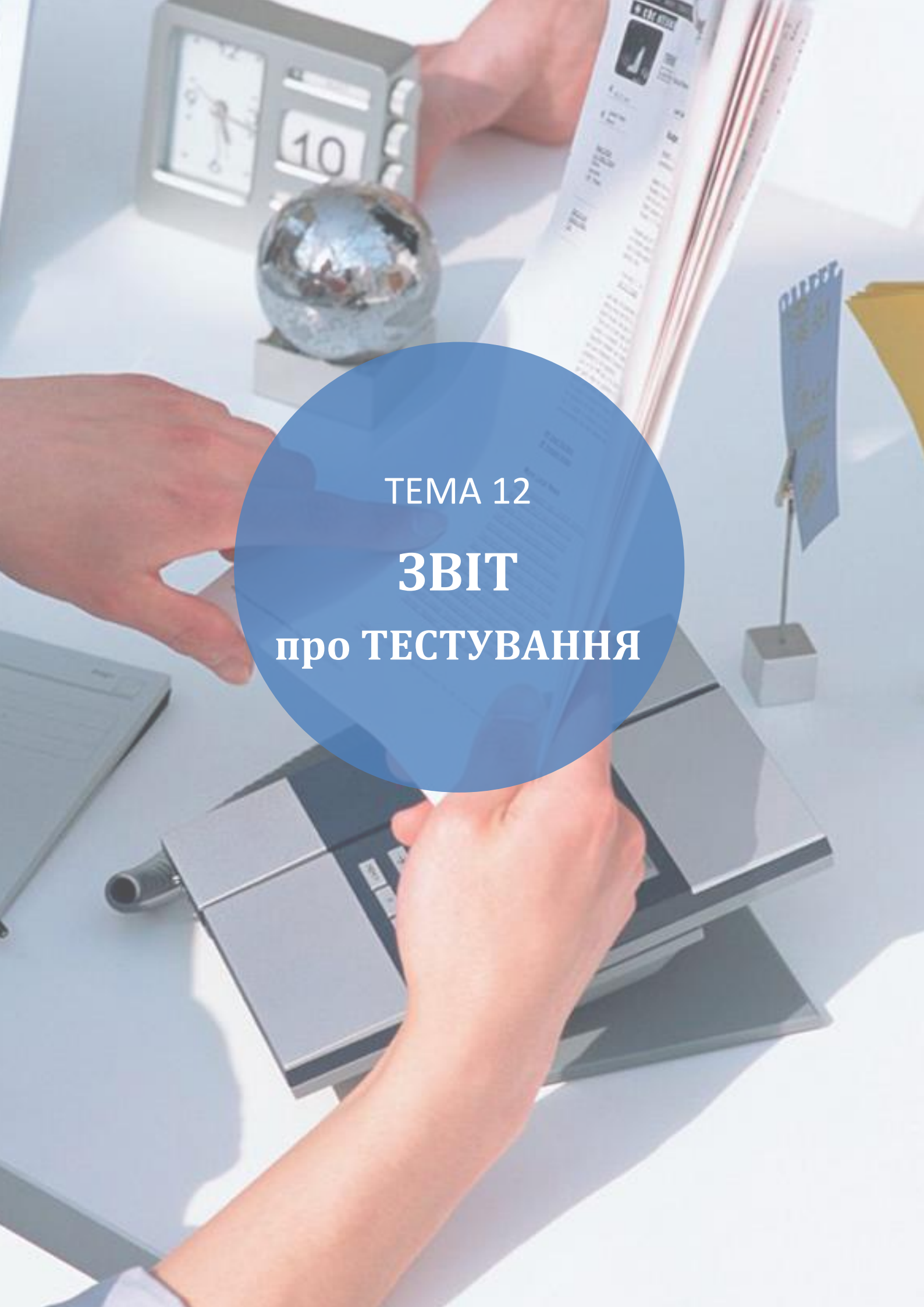
✓ ***План робіт***

Завдання	Час	Дата початку	Дата завершення
Складання тест-плану та чек-листа	4 год.	01.09.2011	02.09.2011
Корегування тест-плану та чек-листа	1 год.		
Виконання тестів другого етапу	IE 6 – 2 год. IE 7 – 2 год. IE 8 – 2 год. IE 9 – 2 год. FF 4 – 2 год. FF 6 – 2 год. Chrome12 – 2 год. Opera10 – 2 год.	09.09.2011	10.09.2011
Виконання тестів третього етапу	IE 6 – 1,5 год. IE 7 – 1,5 год. IE 8 – 1,5 год. IE 9 – 1,5 год. FF 4 – 1,5 год. FF 6 – 1,5 год. Chrome12 – 1,5 год. Opera10 – 1,5 год.	10.09.2011	11.10.2011

Ставка спеціаліста складає 350 руб / год. Виходячи із загальної кількості годин, які будуть використані для процесу тестування, підсумкова сума становить 11550 рублів.

✓ ***Підсумки***

Кінцевим підсумком проведення тестування повинен стати оформлений кінцевий результат процесу тестування з описаними дефектами, а також рекомендаціями щодо поліпшення продукту з точки зору кінцевого користувача.



ТЕМА 12

ЗВІТ
про ТЕСТУВАННЯ

12 ЗВІТ ПРО ТЕСТУВАННЯ

Відомо, що проведення будь-якого тестування, не залежно від виду тестування, має на меті отримання певної інформації, яка служить основою для оцінки поточної версії ПЗ чи окремих функціональностей. Розробник, передаючи в тестування якийсь продукт, не важливо на якій стадії розробки він знаходиться, переслідує мету – отримати інформацію про стан цього продукту.

Тестувальник володіє всією необхідною інформацією щодо стану продукту, яку отримує під час тестування, й надає розробнику вичерпну й в той же час неперенавантажену зайвими даними інформацію, надає об'єктивну оцінку, показує прогрес та ін. Для цього після закінчення тестування (чи певного етапу тестування) складається звіт про виконану роботу. І саме від того, наскільки правильно буде складено цей звіт, залежить, на скільки правильно та швидко будуть прийняті рішення щодо подальших дій відносно продукту, що безпосередньо впливає на результативність тестування.

Звіт – документ, який містить інформацію про виконані дії, результати про виконану роботу. Зазвичай, включає таблиці, графіки, списки, просто опис інформації у вигляді тексту. Зміст та пропорція визначають корисність та зрозумілість звіту.

12.1 Правильне розуміння призначення звіту.

Мета звіту – дати розробнику інформацію про поточний стан програмного продукту. Повна інформація про стан продукту є тільки у команди, що приймала участь у процесі тестування, яка "прощупала" всі функції додатку, володіє інформацією про кількість дефектів та динаміку зміни кількості, наприклад, порівняно з попереднім білдом.

Звіт про результати тестування в основному потрібен:

- менеджеру проекту;
- лідеру команди розробників;
- лідеру команди тестувальників;
- замовнику.

Менеджер проекту на основі короткого опису та статистики знайдених багів робить висновки про продуктивність роботи проектної команди та поточну якість проекту. Графік робіт дозволяє визначити завантаженість окремих співробітників у проектній команді. Список нових

дефектів дозволяє швидко оцінити необхідність прийняття певних заходів з коригування розвитку проекту. Рекомендації показують ще одну точку зору на розвиток проекту.

Таким чином, інформація зі звіту про результати тестування ***дозволяє менеджеру проекту:***

- покращити процес управління, розробки та тестування;
- перерозподілити ресурси проекту;
- сформувати власний звіт для керівництва та замовника.

Лідер команди розробників на основі рекомендацій тестувальників, що міститься у звіті про результати тестування, переглядає деякі питання з підвищення якості створюваного програмного коду та прийняття рішення про можливий перерозподіл обов'язків всередині команди.

Лідер команди тестувальників у процесі збору інформації та складанні звіту про результати тестування більш деталізовано заглиблюється у процеси, що відбувається на проекті та всередині команди, що дозволяє побачити поточну картину деталізовано й обширно одночасно. Такий погляд дозволяє виявити пропущені моменти й прийняти відповідні рішення, спрямовані на підвищення якості проекту.

Замовника завжди цікавить відповідь на питання, куди йдуть гроші, тому докладний звіт про результати тестування дозволяє оцінити, наскільки інвестиції виявилися виправданими. У випадку, якщо безпосередній замовник виступає як підрядчик, звіт про результати тестування дозволяє відзвітувати перед безпосереднім замовником. У випадку, якщо проект планується приймати та впроваджувати по частинах, замовник може зрозуміти, які частини проекту вже готові до прийняття та впровадження.

Фінальний звіт про результати тестування – звіт про результати тестування, який створюється в кінці роботи з проектом, як додаток до вже розглянутих розділів; такий звіт включає опис та аналіз проблем, які зустрічались на проекті, та знайдені ефективні рішення їх вирішення. Даний звіт обговорюється на загальних зборах проектної команди, де за результатом обговорення формуються та документуються висновки, спрямовані на уникнення в майбутньому проблем, що виникли на даному проекті, а також спрямовані на накопичення позитивного досвіду з метою застосування в майбутніх або виконуваних паралельно проектах.

12.2 Складові, які враховують при написанні хорошого звіту

Для того, щоб написати дійсно хороший звіт, який буде містити всю необхідну інформацію та в той самий час буде лаконічним та зрозумілим, необхідно ***враховувати наступні моменти:***

- яке було поставлено завдання?
- для кого пишеться звіт ?
- за який період часу складається звіт?

1. Яке завдання? Перше, що повинні врахувати – це результат, який очікує отримати розробник. Якщо виконувалося тестування певної частини програмного продукту або ж якісь конкретні види тестування, то у звіт слід у мінімальному обсязі включати інформацію, яка не відноситься до поставленої задачі. Наприклад, якщо проводилося функціональне тестування, крім приведення функціональних дефектів, повинні підсумувати стан об'єкта тестування, спираючись на результати функціональних тестів; у той час, як іншу інформацію, котра виходить за рамки завдання, винести на задній план та привести в кінці звіту. Крім того, всі дефекти, які не відносяться до даного етапу тестування, винести окремо як додаток.

Якщо ж протягом певного етапу проводилося декілька видів тестування, то оцінку програмного продукту слід робити загальною, спираючись на результати, отримані зі всіх видів тестування.

2. Яка цільова аудиторія? Слід визначитися з форматом надання інформації. Відповівши на питання "Для кого пишеться звіт?", зрозумієте, яку інформацію слід включити до звіту. ***Виділяють наступні цільові аудиторії:***

- ***технічні фахівці*** володіють переважно технічною стороною продукту, а також приймають рішення, ґрунтуючись на конкретних технічних даних. Для даної цільової аудиторії звіт повинен містити максимальну кількість інформації, наведено зміст проблем всіх частин програмного продукту, а список дефектів можна надати з необхідними коментарями.

- ***продюсерів (менеджерів) проектів*** цікавлять загальні дані без будь-яких подробиць. При написанні звіту для даної аудиторії слід сфокусуватися на цифрову статистику (цифрові та порівняльні метрики), сроки та висновки з отриманих результатів.

- ***бізнес-аудиторія*** приймає рішення про завершення тестування. Для них, у першу чергу, важливий кінцевий результат у максимально короткому та зрозумілому форматі *(так / ні),* візуалізація інформації

(*графіки, діаграми*), експертна думка про можливість випуску продукту в промислове середовище та ін. без заглиблення в деталі.

Написати звіт, який влаштує всі аудиторії – не можливо. Перш ніж писати звіт, обов'язково визначте цільову аудиторію. Залежно від аудиторії, зміст буде сильно відрізнятись структурою та містити різноманітні деталі, необхідні конкретній аудиторії.

3. За який період часу складається звіт? За даним критерієм можна виділити наступні види звітів:

- ***проміжний звіт:*** варто пам'ятати, що чим менша ітерація тестування, тим менше інформації слід надавати у звіті. Наприклад, якщо тестування проводилося декілька днів – це може бути короткий звіт з коментарем щодо перевірених функціональностей та переліком знайдених дефектів. Якщо це один-два тижні – варто написати звіт з оцінкою готовності продукту, а також надати інформацію про прогрес у порівнянні з попереднім етапом тестування, ґрунтуючись на певних метриках, які індивідуальні для кожного проекту. Метриками можуть бути кількість нових дефектів, критичність, результати проходження тест-кейсів та ін.;
- ***версійний звіт*** відрізняється від попереднього лише тим, що у ньому розглядається стан поточної версії ПЗ, а для візуалізації прогресу він порівнюється з попередніми версіями. У даному звіті відображається матриця тестових наборів та версій, які перевірялися, із зазначенням результату проходження тестів. Такий звіт використовується для контролю за повнотою тестування та змінами в якості функціоналу, які можуть виникати від версії до версії;
- ***фінальний звіт*** узагальнює всі отримані дані за весь час тестування та дає оцінку продукту в цілому на даний момент, а також надає дані по проблемах, які залишилися невиправлені, та можливу кількість невиявлених дефектів у зв'язку із часовими обмеженнями. Необхідно резюмувати всю виконану роботу, спираючись на визначені метрики.

12.3 Найкращий вигляд даних у поданому звіті

Якщо звіт пишеться для розробника або менеджера, то ніяких запитань щодо надання звіту не виникає – необхідна термінологія та опис проблем будуть легко сприйматися людиною, яка буде читати представленний звіт. Але якщо стоїть мета написати для людей, які незнайомі зі специфікою та технічною стороною програмного забезпечення, то з'являється задача, для вирішення якої краще використовувати графічну форму надання інформації: графіки та діаграми.

Це може бути, наприклад, графік пройдених тест-кейсів по кількості дефектів (рис. 12.1).

Рис.12.1 Графік пройдених тест-кейсів по кількості дефектів

Також це може бути, наприклад, графік пройдених тест-кейсів по модулям, який наочно покаже, який обсяг роботи в кожному з модулів вже пророблений та допоможе виявити проблеми (рис. 12.2).

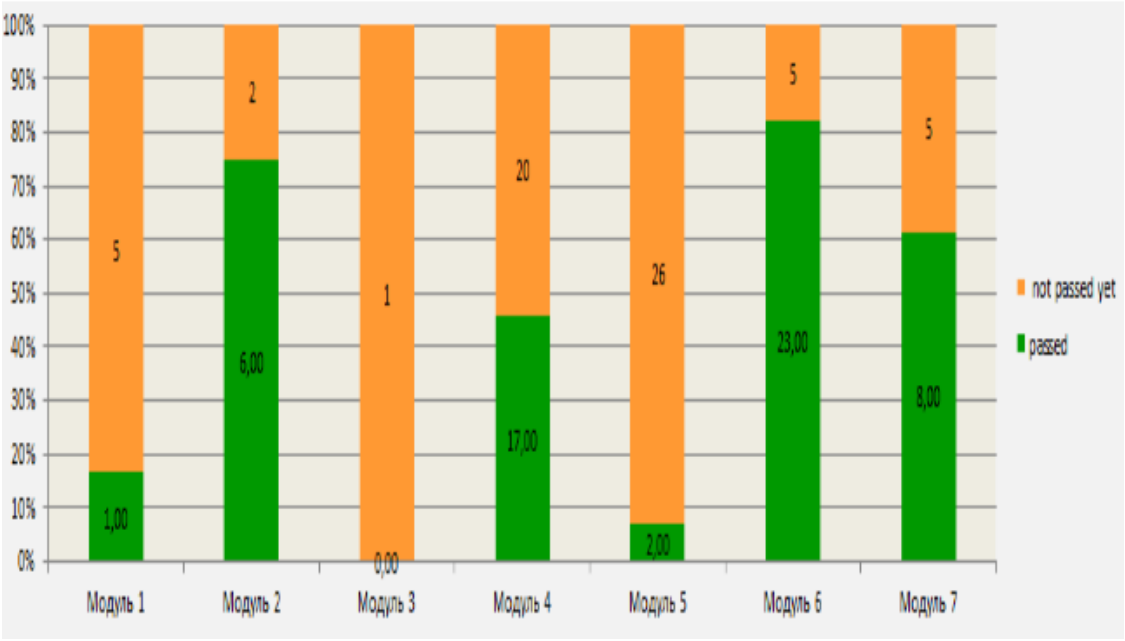


Рис.12.2. Графік пройдених тест-кейсів по модулям

На додаток до діаграми або графіку необхідно надати зведену таблицю з усіма даними (табл.12.1).

Таблиця 12.1. Зведена таблиця

Модуль	121	Прохождение	
	Общее количество ТК по модулю	passed	not passed yet
Модуль 1	6	1	5
Модуль 2	8	6	2
Модуль 3	1	0	1
Модуль 4	37	17	20
Модуль 5	28	2	26
Модуль 6	28	23	5
Модуль 7	13	8	5

Також, наприклад, наводиться зведена таблиця з інформацією про помилки, з якими команді тестувальників доводилося мати справу за весь час роботи з проектом. Таку таблицю можна зобразити у вигляді графіка (рис. 12.3).

Рис. 12.3. Графік з інформацією про помилки

У звіт можна додати графік, котрий показує, яка кількість тестових випадків перейшла в стан автоматизованих за певний проміжок часу (рис. 12.4).

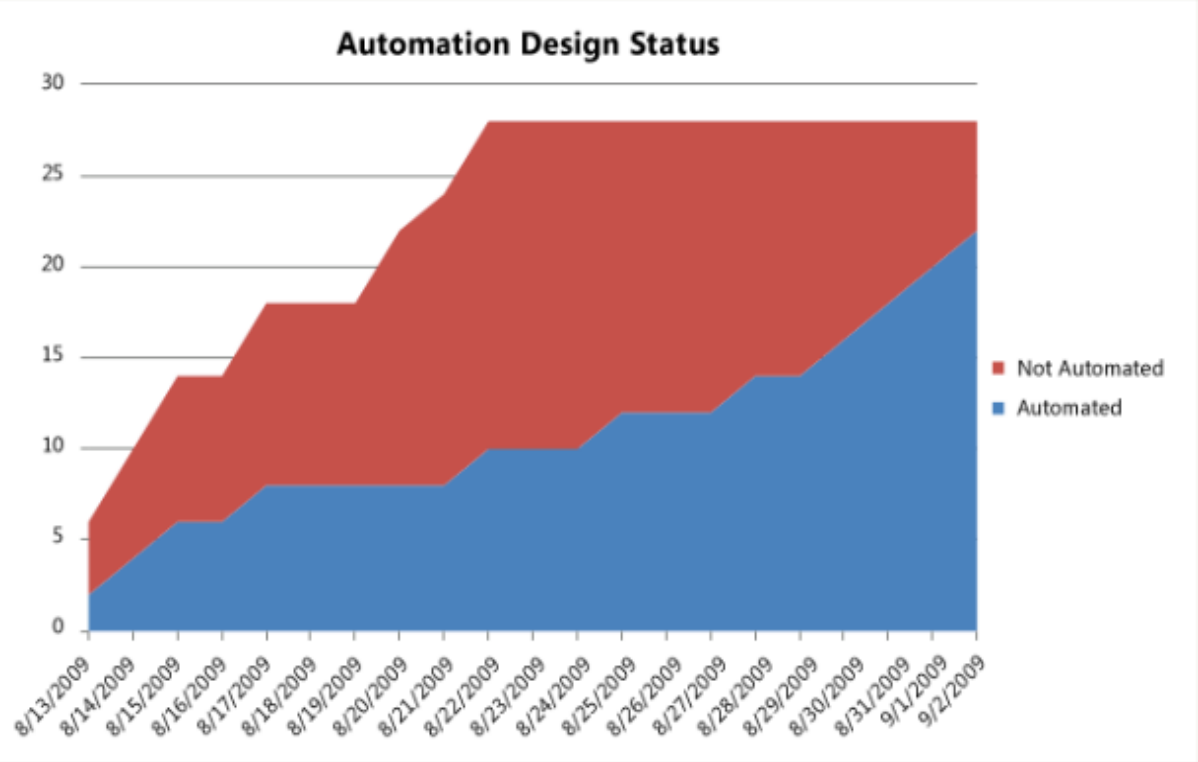


Рис. 12.4. Кількість тестових випадків, яка перейшла в стан автоматизованих за певний проміжок часу.

При створенні звіту обов'язково необхідно вказувати:

1. Склад команди.
2. Терміни виконання, за які складається звіт.
3. Опис процесів тестування.
4. Зміни тестової моделі, доповнення ТК.
5. Відсоток пройдених ТК.
6. Критичні та блокуючі проблеми та вжиті заходи по їх усуненню.
7. Результати регресу (акцент на невирішених проблемах).
8. План на наступну ітерацію / тиждень / місяць.

Пункти 3, 4, 6 та 8 варто писати з урахуванням цільової аудиторії звіту, у пункті 7 варто вказувати, коли проводилося «регрес-тестування», зазвичай цей пункт фігурує у «версійних» звітах. Пункт 8 з підсумкового звіту виключається.

Таким чином, для створення зрозумілого документу необхідно визначити цільову аудиторію та період, за який буде написаний звіт, а також зміст та блоки звіту. При цьому звіт необхідно складати деталізовано, грамотно та із задоволенням, адже хороший звіт – це, як мінімум, третина роботи, а також єдина її частина, яка надається іншим учасникам проекту.

Необхідно відзначити, що до звітів про тестування відносять ***баг- репорти та чек-листи:***

- ***баг-репорт*** – спеціальний документ, який описує ситуацію або послідовність дій, яка призвела до некоректної роботи об'єкта тестування, із зазначенням причин та очікуваного результату, та ***складається з наступних елементів:***
  - короткий опис помилки;
  - кроки для відтворення бага;
  - фактичний результат;
  - очікуваний результат;
  - інша релевантна інформація (скріншоти, логи).

- ***чек-лист тестування*** – документ, який описує, які функції повинні бути перевірені, та є важливим інструментом для тестування веб-додатків та програмних продуктів, при цьоиу може мати досить різні рівні деталізації. ***Чек-лист тестування складається з двох основних частин:***
  - перелік функцій конкретного продукту, виконання яких повинно бути перевірено;
  - список можливих помилок, які можуть існувати в різних проектах.

Чек-лист тестування програмного забезпечення використовується для поділу завдань за рівнем кваліфікації, а також для підтримки звітності та результатів тестування.

Чек-лист тестування повинен включати наступні пункти:

- докладний перелік перевірок;
- статус перевірки;
- результати тестування.

- ***графік перевідкритих дефектів*** використовується для контролю за щоденною кількістю перевідкритих дефектів. Якщо тренд збільшення зберігається, то необхідно з'ясувати причини, наприклад, пов'язані з поганою якістю вихідного коду, недостатнім тестуванням розробниками, неоднозначним трактуванням вимог та ін.

- ***графік збіжності дефектів*** використовується для контролю за збіжністю процесу стабілізації функціональності. Після того, як активна фаза розробки завершується, і починається активна фаза стабілізації функціональності та тестування, необхідно стежити за тим, щоб кількість відкритих дефектів постійно знижувалася.

- ***графік числа помилок, виявлених тестувальниками***, використовується для оцінки розподілу дефектів по авторам – тестувальникам, які їх виявили. Даний звіт доступний в загальному списку звітів, доступному з меню "Звіти" у головному меню "Проект".

12.4 Шляхи поліпшення структури звіту про тестування

У процесі роботи компанія з тестування постійно взаємодіє з клієнтами. Після того, як закінчиться будь-який вид тестування *(тестування веб-сайтів, мобільне тестування та ін.),* інженери з тестування складають звіт для замовника, тим самим постійно підтримуючи комунікацію. Це цілком природно, що вони отримають відгуки не тільки про виявлені під час тестування помилки, а й про представлені звіти.

Необхідно задати замовнику декілька питань, наприклад, відносно представленого звіту:

1. Чи отримав замовник всю необхідну інформацію зі звіту?
2. Чи потрібні замовнику додаткові дані?
3. Чи влаштовує замовника структура звіту?

Обговорюйте всі деталі до тих пір, поки не створите таку структуру звіту, яка буде підходити замовнику й яка не буде складати особливих труднощів. Надалі вдосконалений звіт можна використовувати як основу для всіх наступних, щоб заощадити час на тому, що вже й так добре зробили раніше, а також можна додати елементи наочності у звіт, щоб зробити його більш логічним та легким для розуміння. Завдяки цьому, у виграші залишиться не тільки той, хто складає звіт, а й компанія з тестування, на яку працює відповідальний за складання звіту.

12.5 Приклад звіту

У даному прикладі розглянуто лише два недоліки з перерахованих.

Вступна частина

Умови тестування

Ми вивчали роботу інтернет-сайту «www.dp.ru» на комп’ютерах під управлінням операційної системи Microsoft Windows XP. На комп'ютерах був встановлений Macromedia Flash 7, включені cookies та JavaScript.

Сайт тестували в браузерах:

- Google Chrome 40
- Opera 12
- Mozilla Firefox 35

Ми працювали в різних умовах з'єднання з Інтернет:

- з'єднання по виділеній лінії;
- з'єднання по модему;
- мобільний інтернет (GPRS)

При тестуванні у зазначених вище умовах сайт показав свою працездатність. Відмінності при тестуванні в декількох браузерах були незначні. Завантаження сторінок проходило повільно в порівнянні з аналогічними сайтами.

Цілі проекту DP.RU:

- Позиціонування dp.ru як окремий від газети проект
- Отримання прибутку від проекту dp.ru

Завдання проекту DP.RU

- Розміщення банерної та текстової (прес-релізи, рекламні статті) реклами.
- Ефективне представлення інформації.
- Надання максимуму корисною для бізнес-аудиторії інформації.
- Розвиток сервісів з метою отримання прибутку від них.
- Підвищення продажів електронного доступу до архівів газети.

Наші коментарі: Аудиторія будь-якого сайту складається з постійної частини та нерегулярних відвідувачів. Постійна аудиторія забезпечує стабільну відвідуваність і популярність, що важливо для рекламодавців. Отже, необхідно збільшувати число постійних відвідувачів. Для формування постійної аудиторії ключове значення мають два фактори: зручність роботи і довіра відвідувачів. Останнім часом все більшого значення набуває також можливість спілкування, створення спільноти активних відвідувачів.

Найчастіші типи активності відвідувачів інформаційних сайтів:

- регулярний перегляд новин;
- пошук інформації про поточні події;
- пошук інформації про попередні події;
- обмін думками, реакція на матеріали сайту.

Вимоги до інформаційних сайтів:

- Internet Explorer 8.0

Сайти новин та інформаційні сайти створені для подачі великого обсягу різноманітного матеріалу: новин, статей за різними тематиками, реклами, прес-релізів. Це істотне навантаження на відвідувача, у якого є свої завдання і обмежений час *(аудиторія – ділові люди)*.

Звідси випливають ключові вимоги:

- зручність читання тексту;
- зручність орієнтації в масиві інформації;
- очевидна логічна організація матеріалу;
- швидкість завантаження сторінок;
- легкий і надійний пошук потрібного матеріалу;
- зручна робота з архівами.

Огляд недоліків сайту DP.RU

Виходячи з перерахованих вимог до інформаційних сайтів та існуючих на сьогоднішній день критеріїв оцінки юзабіліті, ми визначили основні недоліки та ***об'єднали їх в групи:***

- візуальне та інформаційне навантаження;
- розподіл екранного простору;
- недоліки системи навігації;
- недоліки системи перехресних посилань;
- недоліки службових сторінок і повідомлень;
- труднощі роботи з архівом;
- неефективна робота пошуку;
- проблеми дизайну та єдності стилю;
- недоліки форм введення;
- інтеграція з паралельними проектами;
- технічні недоліки;
- інше.

Деякі проблеми входять одночасно у дві групи (наприклад, розміщення заголовків повідомлень форуму у верхній частині сторінки у вузькій колонці не є оптимальним по компонуванню і одночасно є візуальним шумом).

Візуальне та інформаційне навантаження.

На сторінках сайту одночасно присутня велика кількість підзаголовків, 2-3 анімованих рекламних блоки, «біжучий рядок», додаткові посилання на форум та інші сторінки, інформер погоди, інформер курсу валют, кілька різних меню. Відмовитися від розміщення

текстів і реклами неможливо, тому навантаження на відвідувача слід знижувати за рахунок інших елементів.

Доведено, що анімовані об'єкти погіршують сприйняття тексту та є візуальними подразниками. Зменшити навантаження можна знизивши інтенсивність анімації банерів, а також відмовившись від «біжучого рядка».

Аргументи проти «біжучого рядка»:

- Він помітний відвідувачеві тільки на самому початку перегляду сторінки. Якщо людина прийшла на сторінку для того, щоб прочитати матеріал *(аудиторія – ділові люди),* то він з меншою ймовірністю покине її заради випадково повідомлення.
- Повідомлення «біжучого рядка» дуже численні, що ще більше обмежує час контакту конкретного повідомлення з відвідувачем.
- Він збільшує обсяг завантажених файлів *(це скрипт)* і знижує швидкість завантаження.
- Він розташований у безпосередній близькості від анімованого рекламного банера. Існує явище «банерної сліпоти», коли об'єкти, що знаходяться в області розміщення банера, залишаються без уваги. Це також знижує ефективність «біжучого рядка».

Рекомендації:

Ми пропонуємо відмовитися від «біжучого рядка», замінивши його при необхідності ротацією якісно виконаного неанімованого текстового блоку, який рекламує матеріали сайту.

Ми рекомендуємо також переглянути набір та кількість інформаційних блоків:

- Інформери погоди і курсу валют. Безсумнівно, ця інформація для когось буде корисна, але навряд чи саме вона приваблює цільову аудиторію. Ці інформери є на «Яндексі», існують спеціалізовані сайти.
- Деякі банери, наприклад, «З Новим роком ДП», самі по собі являються малопривабливими для цільової аудиторії (у них на сайті інші завдання). При цьому вони займають місце і є візуальним подразником.

Частково дубльовані меню паралельних проектів ДП (у верхньому колонтитулі та у лівій колонці знизу повторюється каталог сайтів та фото з вечірок). Статистика показує невисоку частку переходу на ці проекти з сайту ДП. Отже, навіть така презентація проектів малоефективна. Це меню займає місце і є додатковим навантаженням. Присутні елементи, що

повторюють більш звичні й зручні функції, наприклад, відображення дати. Користувачі звикли дивитися дату, навівши мишку на годинник у системній панелі.

Розподіл екранного простору

- Праворуч постійно присутній великий анімований банер.
- Він не є релевантним об'єктом (тобто не відповідає задачам відвідувача), тим не менш, постійно займає 1/6 частину екрану. На більшості інформаційних сайтів банери не займають місце, а прокручуються з усією сторінкою.
- Ще 1/6 частина екрану зайнята порожньою смугою, яка розташована під інформерами у вузькій правій колонці. Таким чином, третину корисної площі (особливо головної сторінки) витрачено даремно. Другий момент – частину цього банеру не видно, якщо висота екрану менша, ніж висота банера.
- Жорстка верстка сторінок призводить до втрат екранного простору і появи білого поля праворуч від банера. Більшість користувачів DP.RU, згідно статистики, працюють при ширині екрану 1024 і більше точок. Крім того, присутня тенденція до збільшення ширини екрану користувачів. Це означає, що все більше і більше людей будуть бачити пусте поле.
- У шапці праворуч від логотипу розташований ще один банер. Таким чином, постійна наявність банера цього формату стає обов'язковим. Прибрати банер або поставити інший формат без шкоди для дизайну не вийде.
- Частково продубльовано меню паралельних проектів ДП. Меню в шапці сторінок набрано дуже дрібним шрифтом, що погіршує його читабельність і при швидкому перегляді перетворює його в сіру смужку.
- Список анонсів публікацій при цьому стає витягнутим і поширюється далеко за межі першого екрану, що знижує вірогідність їх перегляду, отже, час і детальність перегляду сайтів.

Рекомендації:

Ми рекомендуємо змінити вигляд сторінок таким чином, щоб найбільш релевантні (відповідають запитам користувачів) об'єкти перебували якомога вище, а список анонсів став більш компактним. Ми рекомендуємо також відмовитися від «Банера, що не прокручується» праворуч, можливо, змінити його формат.

Паралельні проекти ДП можна розмістити у верхньому колонтитулі, у формі чітко позначеної навігаційної смуги (горизонтального меню), що впритул прилягає до верхнього краю сторінки.

На сторінці зі статтями у вузькій правій колонці розташовані автоматично обрізані повідомлення форуму (оформлені ненайкращим чином). У даному місці повідомлення форуму не цілком прийнятні і є інформаційним шумом. Як правило, серйозні люди приєднуються до дискусії вже після того, як прочитали матеріал статті.

Повідомлення форуму краще поміщати під статтею, не відволікаючи уваги читача від того, за чим він прийшов на цю сторінку. Оформляти повідомлення у вигляді блоку нормальної ширини, а не вузької колонки. У тому ж блоці вказувати кількість повідомлень і посилання «Прокоментувати».

ТЕМА 13

ВИДИ ТЕСТУВАННЯ

13 ВИДИ ТЕСТУВАННЯ

Існуючі на сьогодні методи тестування програмного забезпечення не дозволяють однозначно та повністю виявити всі дефекти й встановити коректність функціонування аналізованої програми, тому всі існуючі методи тестування діють в рамках формального процесу перевірки досліджуваного або розроблюваного програмного забезпечення.

Такий процес формальної перевірки, або верифікації, може довести, що дефекти відсутні з точки зору використовуваного методу. Тобто немає ніякої можливості точно встановити чи гарантувати відсутність дефектів у програмному продукті з урахуванням людського чинника, присутнього на всіх етапах життєвого циклу програмного забезпечення.

Існує безліч підходів до вирішення завдання тестування та верифікації програмного забезпечення, але ефективне тестування складних програмних продуктів – це процес надзвичайно творчий, що не зводиться до слідування строгим та чітким процедурам або створення таких.

Якість програмного забезпечення можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових:

- надійність;
- супроводжуваність;
- практичність;
- ефективність;
- мобільність;
- функціональність.

Склад та зміст документації, супутньої процесу тестування, визначається стандартом *IEEE 829-1998*.

Існує кілька ознак, за якими прийнято класифікувати види тестування, а саме:

- ✓ ***за об'єктом тестування:***
  - функціональне тестування;
  - тестування продуктивності:
    - навантажувальне тестування;
    - стрес-тестування;
    - тестування стабільності;
  - конфігураційне тестування;
  - юзабіліті-тестування;
  - тестування інтерфейсу користувача;
  - тестування безпеки;
  - тестування локалізації;
  - тестування сумісності;

- ✓ ***за знанням системи:***
  - тестування чорного ящика;
  - тестування білого ящика;
  - тестування сірого ящика;

- ✓ ***за ступенем автоматизації:***
  - ручне тестування;
  - автоматизоване тестування;
  - напівавтоматизоване тестування;

- ✓ ***за ступенем ізольованості компонентів:***
  - модульне тестування;
  - інтеграційне тестування;
  - системне тестування;

- ✓ ***за часом проведення тестування:***https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B8_%D0%BD%D0%B0_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8
  - альфа-тестування:
    - димове тестування (*smoke testing*);
    - тестування нової функції (*new feature testing*);
    - підтверджуюче тестування;
    - регресійне тестування;
    - приймальне тестування;
  - бета-тестування;

- ✓ ***за ознакою позитивності сценаріїв***
  - позитивне тестування;
  - негативне тестування;

- ✓ ***за ступенем підготовленості до тестування:***
  - тестування за документацією *(формальне тестування);*
  - інтуїтивне тестування (*ad hoc testing*).

На рис. 13.1. у вигляді схеми представлені види тестування.

Рис.13.1. Види тестування

13.1 Рівні тестування

13.1.1 Компонентне (модульне) тестуванн

Компонентне (модульне) тестування перевіряє функціональність та шукає дефекти у частинах додатку, які доступні й можуть бути протестовані окремо (модулі програм, об'єкти, класи, функції та ін.). Зазвичай компонентне (модульне) тестування проводиться шляхом виклику коду, який необхідно перевірити, і за підтримки середовищ розробки, таких як фреймворки (frameworks – каркаси) для модульного тестування або інструменти для налагодження. Усі знайдені дефекти, як

правило, виправляються в коді без формального їх опису в системі менеджменту 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 напочатку» ***(test first approach).*** При цьому підході створюються та інтегруються невеликі частини коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір, поки всі тести не будуть успішно пройдені.

13.1.2 Інтеграційне тестування

Інтеграційне тестування призначене для перевірки зв'язку між компонентами, а також взаємодії з різними частинами системи (операційною системою, обладнанням 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 вгору*** (*Bottom Up Integration*): усі низькорівневі модулі, процедури або функції збираються разом й потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування. Даний підхід вважається корисним, якщо всі або практично всі модулі розроблюваного рівня готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності додатку;

- ***згори вниз*** (*Top Down Integration*): спочатку тестуються всі високорівневі модулі й поступово, один за одним додаються низькорівневі. Усі модулі нижчого рівня симулюються заглушками з аналогічною функціональністю, потім у міру готовності вони замінюються реальними активними компонентами. Таким чином ми проводимо тестування згори вниз;

- ***великий вибух*** (*"Big Bang" Integration*): всі або практично всі розроблені модулі збираються разом у вигляді закінченої системи

або її основної частини, а потім проводиться інтеграційне тестування. Такий підхід застосовується для збереження часу. Однак, якщо тест-кейси та їх результати записані невірно, то сам процес інтеграції дуже ускладниться, що стане перешкодою для команди тестування при досягненні основної мети інтеграційного тестування.

13.2 Системне тестування

Системне тестування – перевіряється інтегрована система на відповідність вимогам. Основним завданням 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 користувача рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання та ін. Для мінімізації ризиків, пов'язаних з особливостями поведінки системи в тому чи іншому середовищі, під час тестування рекомендується використовувати оточення максимально 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 уявлення про способи використання продукту створюються випадки використання системи ***(Use Cases)***. За конкретним випадком використання можна визначити один або більше сценаріїв. Для перевірки кожного сценарію пишуться **тест-кейси *(Test Cases)***, які повинні бути протестовані.

13.2.1 Приймальне тестування або приймально-здавальне випробування (Acceptance Testing)

Формальний процес тестування, який перевіряє відповідність системи вимогам та ***проводиться з метою:***

- визначення, чи задовольняє система приймальні критерії;
- винесення рішення замовником або іншою уповноваженою особою, чи приймається додаток.

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

Рішення про проведення приймального тестування приймається у наступних випадках:

- продукт досяг необхідного рівня якості;
- замовник ознайомлений з планом приймальних робіт ***(product acceptance plan)*** або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні особи та ін.

Фаза приймального тестування триває доти, поки замовник не виносить рішення про відправлення додатку на доопрацювання або видачу додатку.

13.3 Статичне та динамічне тестування

Описані нижче техніки – тестування білого ящика та тестування чорного ящика – припускають, що код виконується, й різниця полягає лише в тій інформації, якою володіє тестувальник. В обох випадках це ***динамічне тестування.***

При ***статичному тестуванні*** програмний код не виконується – аналіз програми відбувається на основі вихідного коду, який вираховується вручну або аналізується спеціальними інструментами. У деяких випадках аналізується не вихідний, а проміжний код *(такий як байт-код або код на MSIL).*

Також до статичного тестування відносять ***тестування вимог, специфікацій, документації.***

13.4 Регресійне тестування

Вид тестування, спрямований на перевірку змін, зроблених у додатку або в оточуючому середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб-сервер або сервер додатку), для підтвердження того факту, що існуюча функціональність працює як і раніше. Регресійними можуть бути тести як функціональні, так і не функціональні.

Зазвичай для регресійного тестування використовуються ***тест-кейси***, написані на ранніх стадіях розробки та тестування. Це дає гарантію того, що зміни у новій версії додатку не пошкодили вже існуючу

функціональність. Рекомендується робити автоматизацію регресійних тестів для прискорення подальшого процесу тестування та виявлення дефектів на ранніх стадіях розробки програмного забезпечення.

Термін "Регресійне тестування", залежно від контексту використання, може мати різний зміст. Сем Канер описує три основних типи регресійного тестування:

- ***регресія багів*** *(bug regression)* – спроба довести, що виправлена помилка насправді не виправлена;

- ***регресія старих багів*** *(old bugs regression)* – спроба довести, що нещодавня зміна коду або даних зламала виправлення старих помилок, тобто старі баги знову стали відтворюватися;

- ***регресія побічного ефекту*** *(side effect regression)* – спроба довести, що нещодавня зміна коду або даних зламала інші частини розроблюваного додатку.

13.5 Тестові сценарії

Тестувальники використовують тестові сценарії на різних рівнях: як у модульному, так і в інтеграційному та системному тестуванні. Тестові сценарії, як правило, пишуться для перевірки компонентів, в яких найбільш висока ймовірність появи відмов, або вчасно не знайдена помилка може дуже дорого кошутвати.

Тестовий сценарій – це послідовність дій, що включає тестові дані та очікувані результати для кожної дії, яка може бути виконана для перевірки фунціональності програмного забезпечення. Зазвичай тестові сценарії сформульовані природною мовою у вигляді текстового опису, за винятком автоматизованих тестових сценаріїв, які можу бути сформульовані безпосередньо мовою програмування. Надалі мова йде про неавтоматизовані тестові сценарії, які виконуються вручну.

Представлено перелік випадків, для яких необхідне написання тестових сценаріїв:

- ***поділ праці фахівців різної кваліфікації*** – у цьому випадку тестові сценарії є інтерфейсом між розробником тестових сценаріїв та тестувальником, що виконує ці тестові сценарії;

- ***регресійне тестування*** – у цьому випадку тестові сценарії дозволяють забезпечити повторюваність ітерацій регресійного тестування, що дає можливість більш точного планування та стабільність результатів тестування;

- ***повторне виконання тестових сценаріїв*** – у цьому випадку стає можливим виконання тестових сценаріїв, які виявили дефект, повторно в тій послідовності дій, яка планувалася спочатку. На жаль, повторні ітерації без написаних тестових сценаріїв через людський фактор можуть бути виконані неповністю або неточно: стає нецікаво, знижується уважність;

- ***звітність за результатами роботи (за прогресом та за якістю)*** – у цьому випадку стає можливою кількісна оцінка ступеня виконання роботи з точністю до тестового сценарію, після завершення роботи можна оцінити причини пропущених дефектів, так як результати виконаних тестувальником у програмному забезпеченні дій (фактичні результати) можуть бути задокументовані для кожного кроку тестового сценарію.

- ***відстеження тестового покриття за вимогами до програмного забезпечення*** – у цьому випадку наявність раніше написаних тестових сценаріїв дозволяє переконатися, що певні вимоги будуть перевірені, й за необхідності доповнити набір тестових сценаріїв. Тобто, для відстеження тестового покриття послідовність дій, що приводиться у сценарії, не обов'язкова. Для цих цілей достатньо опису перевірки, яка повинна бути виконана;

- ***поділ праці фахівців однакової кваліфікації*** – у цьому випадку тестові сценарії є засобом вимірювання обсягу роботи фахівців та планування робіт, тобто кожному тестувальнику може бути призначений свій комплект тестових сценаріїв, які він повинен провести за певний термін. Слід зазначити, що подібний поділ праці можна здійснити без тестових сценаріїв, наприклад, за вимогами до програмного забезпечення.

Перераховані випадки не завжди є актуальними. Наприклад, якщо на певному етапі розвитку програмного забезпечення потрібно виявити тільки найбільш критичні дефекти, так як інші дефекти будуть виявлені та виправлені в ході супроводу, то розробка тестових сценаріїв може виявитися неефективною.

13.6 Тестування білого ящика та чорного ящика

Залежно від доступу розробника тестів до вихідного коду тестованої програми, розрізняють тестування білого ящика та тестування чорного ящика.

При тестуванні ***білого ящика*** *(прозорого ящика)* розробник тесту має доступ до вихідного коду програм та може писати код, який пов'язаний з бібліотеками тестованого програмного забезпечення. Це типово для модульного тестування, при якому тестуються лише окремі частини системи. Такий вид тестування забезпечується тим, що компоненти конструкції – працездатні та стійкі, до певної міри. При тестуванні білого ящика використовуються метрики ***покриття коду або мутаційне тестування.***

При тестуванні ***чорного ящика***, тестувальник має доступ до програми тільки через ті ж ***інтерфейси***, що й замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування. Наприклад, тестуючий модуль може віртуально натискати клавіші або кнопки миші у програмі, що тестується, за допомогою механізму взаємодії процесів з упевненістю в тому, що все йде правильно, що ці події викликають той самий відгук, що й реальні натискання клавіш та кнопок миші. Як правило, тестування чорного ящика ведеться з використанням специфікацій або інших документів, що описують вимоги до системи. Зазвичай в даному виді тестування ***критерій покриття*** складається з покриття структури вхідних даних, покриття вимог та покриття моделі (у тестуванні на основі моделей).

При тестуванні ***сірого ящика*** розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібен.

Якщо «альфа-» і «бета-тестування» відносяться до стадій випуску продукту *(а також, неявно, до обсягу спільноти тестувальників та до обмежень на методи тестування),* тестування «білого ящика» та «чорного ящика» має відношення до способів, якими тестувальник досягає мети.

Бета-тестування в цілому обмежене технікою чорного ящика, хоча постійна частина тестувальників зазвичай продовжує тестування білого ящика паралельно з бета-тестуванням.

Таким чином, термін «бета-тестування» може вказувати на стан програми (ближче до випуску ніж «альфа») або може вказувати на деяку

групу тестувальників та процес, що виконується цією групою. Тобто, тестувальник може продовжувати роботу з тестування білого ящика, хоча програма вже на «бета-стадії», але в цьому випадку не є частиною «бета-тестування».

13.7 Покриття коду

Покриття коду за своєю суттю є тестуванням методом білого ящика. Тестоване програмне забезпечення збирається із спеціальними налаштуваннями або бібліотеками або запускається в особливому оточенні, у результаті чого для кожної використовуваної (виконуваної) функції програми визначається місцезнаходження цієї функції у вихідному коді. Цей процес дозволяє розробникам та фахівцям із забезпечення якості визначити частини системи, які при нормальній роботі використовуються дуже рідко або ніколи не використовуються (такі як код обробки помилок та ін). Це дозволяє зорієнтувати тестувальників на тестування найбільш важливих режимів.

Тестувальники можуть використовувати результати тесту покриття коду для розробки тестів або тестових даних, які розширять покриття коду на важливі функції.

Зазвичай інструменти та бібліотеки, що використовуються для отримання покриття коду, вимагають значних витрат продуктивності та/або пам'яті, неприпустимих при нормальному функціонуванні ПЗ. Тому можуть використовуватися тільки в лабораторних умовах.

ТЕМА 14

МОБІЛЬНЕ ТЕСТУВАННЯ

14 МОБІЛЬНЕ ТЕСТУВАННЯ

Згідно зі звітом Clearwater Technology Team у 2015 році доходи мобільної галузі наближуються до 330 млрд. дол. Проте стрімке зростання попиту на мобільні пристрої (особливо на смартфони та планшети), а також на спеціально призначені для них програми супроводжується збільшенням потреби в адекватних якісних інструментах тестування.

У сучасних реаліях, коли ринок переповнений безліччю додатків, лише ті з них, які протягом усього процесу тестування демонструють високі показники, можуть розглядатися як ті, що пройшли ретельну перевірку. Тестування – один з обов'язкових пунктів маркетингової стратегії, що відноситься до мобільних додатків.

Дослідження роботи мобільних додатків істотно відрізняється від веб-тестування на десктопі, де єдина складність – наявність декількох різних операційних систем. Стосовно мобільного тестування необхідно враховувати значно більше аспектів. Адже існує досить багато різних платформ, пристроїв та навіть ринків.

Мобільний користувач очікує, що програми, які встановлюються, прості, інтуїтивно зрозумілі, працюють завжди та скрізь без збоїв. Якщо очікування не виправдовуються, то користувач просто-напросто встановлює аналогічний додаток від іншого учасника, яких у сфері мобільних розробок завжди достатньо. Тому якість програми є одним з головних факторів популярності.

14.1 Поняття мобільного тестування

Тестування мобільних додатків – процес, під час якого програмна частина додатку для мобільних пристроїв тестується на функціональність, зручність та змістовність. Таке тестування може бути ручним або автоматизованим. Мобільні додатки можуть бути попередньо (до покупки пристрою) чи пізніше встановлені за допомогою магазину додатків або інших платформ.

Основні відмінності мобільних та десктопних додатків:

- екран;
- датчики та пристрої вводу;
- телефонні функції;
- енергоспоживання;
- мережа;
- особливості платформи;
- вузька спеціалізація;

- норми та рекомендації.

Мобільний web-сайт – спеціалізований сайт, адаптований для перегляду та функціонування на мобільному пристрої.

Тестування веб-додатків має дати оцінку якості мобільних веб-додатків при використанні різних браузерів на різних мобільних пристроях. На відміну від мобільних додатків, веб-додатки зазвичай дозволяють звертатися до функцій базового серверу через тонкий мобільний клієнт. Таким чином, крім аналізу функціональності та поведінки, виконання вимог до QoS, зручності використання, безпеки та конфіденційності, тестування мобільних веб-додатків повинно враховувати ще й зв'язність.

Мобільний додаток – це спеціально розроблений додаток під конкретну мобільну платформу.

Спеціально призначені для тестування програми встановлюються та запускаються на мобільних пристроях і, як правило, звертаються до спеціальних інтерфейсів API (наприклад, до API GPS) та апаратних компонентів (наприклад, до камери). Веб-додатки складаються з сервера додатків та клієнтського ПЗ, виконуваного в середовищі браузерів, через які користувачі отримують доступ до сервісів.

Тестування додатків, спеціально призначених для мобільних пристроїв, дає змогу оцінити якість мобільних програм, які завантажуються та виконуються на різних мобільних пристроях обраної мобільної платформи. Тестування спрямоване на аналіз функціональності та поведінки (у тому числі підтримки специфічних для пристрою функцій – наприклад, управління за допомогою жестів), виконання вимог до QoS, зручності використання, безпеки та конфіденційності.

14.2 Види тестування мобільних додатків

Щоб зрозуміти особливості тестування мобільних додатків, слід брати до уваги моменти, які принципово відрізняють мобільні додатки від десктопних: специфічність ОС для мобільних платформ, різні компанії-виробники пристроїв та конфігурації комплектуючих, функціональність пристроїв як комунікаторів та ін.

У зв'язку з цими особливостями підхід дорозробки додатків, зокрема тестування на мобільних пристроях, досить сильно відрізняється від десктопного. Виникає безліч додаткових важливих нюансів та вимог, які необхідно протестувати. До основних відмінностей у деяких видах тестування відносять:

1. ***Тестування оновлень.*** Часті оновлення системи призводять до необхідності оновлювати додаток. Слід виконувати тестування

оновлень програми з боку сервера, перевіряти шляхи установки програми (WiFi, шнур, Bluetooth, SD).

2. ***Тестування інтернаціоналізації.*** Дозволяє на ранньому етапі процесу розробки мобільного додатку переконатися у підтримці інтернаціоналізації – головним чином, у мовній підтримці. Може виникнути багато проблем, пов'язаних з нестачею вільного простору на екрані, а також такі проблеми, як, наприклад, форматування дат.

3. ***Тестування зручності користування (usability).*** Дозволяє виявити частини програми, які недостатньо привабливі або викликають труднощі в навігації чи використанні, переконатися, що модель споживання ресурсів додатком відповідає цільовій аудиторії. Наприклад, офісні додатки без необхідності не повинні викликати надмірне споживання енергії. Найчастіше це тестування проводиться у вигляді бета-тестування.

4. ***Тестування навантаження.*** Передбачає спостереження за використанням пам'яті та системних ресурсів; крім того, тестування навантаження дозволяє виявити «вузькі» місця в додатку, пов'язані з продуктивністю, а також виявити небезпечні витоки пам'яті.

5. ***Випадкове тестування (monkey testing, стрес-тест).*** Додаток повинен коректно реагувати на виникнення випадкових та непередбачуваних подій. Мобільні пристрої досить часто потрапляють в умови, в яких отримують хаотичну інформацію (наприклад, незаблокований комунікатор у кишені), тому додаток повинен адекватно реагувати на подібні потоки даних.

6. ***Мультиплатформене та мультидевайсове тестування.*** Додаток повинен правильно працювати на всіх конфігураціях пристроїв та на всіх пристроях, для яких він розроблявся. Завдання тестування всіх доступних пристроїв, на яких використовуються різні версії інструментарію і які мають екрани з різними розмірами, різний функціонал, архітектуру ОС та апаратне забезпечення, настільки ж важливе, наскільки і складне для виконання.

7. ***Лабораторне тестування.*** Проводиться імітація реальних умов якості зв'язку.

8. ***Атестаційне тестування.*** Використовується для підтвердження відповідності додатку стандартам, ліцензійним угодам та умовам використання.

14.3 Основні вимоги та підходи до тестування

Тестування мобільних додатків відрізняється від тестування звичайного ПЗ наявністю ряду унікальних вимог. Насамперед, мобільні додатки повинні правильно відкриватися у будь-який час і у будь-якому місці. Необхідно, щоб вони коректно функціонували на різних платформах, які відрізняються операційними системами, розмірами екрану, обчислювальними ресурсами та тривалістю безперервної роботи від батарей. Мобільні додатки повинні підтримувати безліч каналів введення *(клавіатура, голос, жести та ін.)*, мультимедійні технології та володіти іншими особливостями, що підвищують зручність їх використання. Для утримання низьких цін на обладнання необхідно широко використовувати засоби моделювання та віртуалізації. І нарешті, оскільки більшість мобільних сервісів підтримують досить широкий спектр бездротових мереж *(2G, 3G, 4G, Wi-Fi, WiMax)*, мобільні додатки повинні нормально функціонувати в неоднорідному мережевому середовищі.

Можна виділити чотири популярні підходи до тестування мобільних додатків, які засновані на використанні базової клієнт-серверної інфраструктури (рис. 14.1).

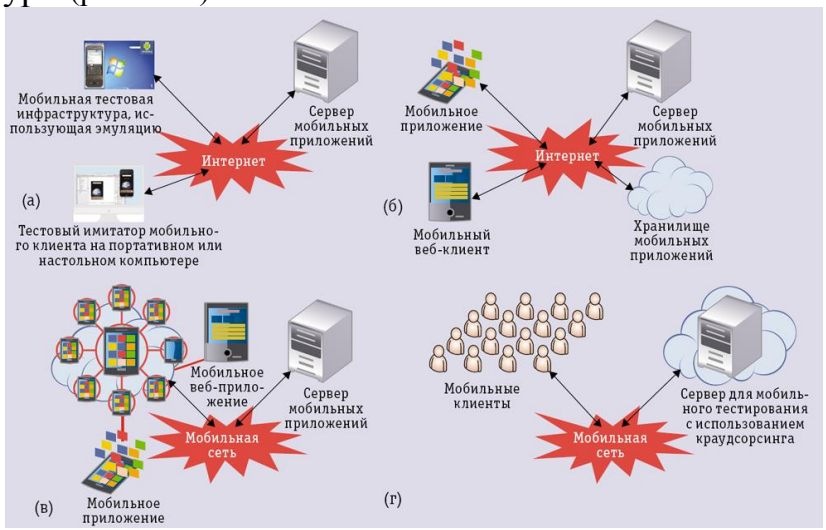


Рис. 14.1. Інфраструктура мобільного тестування:
а) емуляція, б) хмара, в) пристрої, г) краудсорсинг

1. Тестування на основі емуляції. Цей спосіб тестування передбачає використання емулятора мобільного пристрою, що імітує поведінку у віртуальній машині. Досить часто такі емулятори бувають включені до складу комплекту інструментів для розробника, прикладеного до мобільної платформи (наприклад, до складу SDK Android). Ціна такого рішення відносно невелика, тому що в цьому випадку немає необхідності у створенні тестової лабораторії, покупці або оренді фізичних пристроїв.

Однак емуляцію можна використовувати тільки для оцінки функціональності системи в дуже обмеженому контексті. Такий підхід недорогий, але він має ряд істотних недоліків. Наприклад, перевірити обробку всіх жестів у повному обсязі досить важко, тому що більшість емуляторів підтримує лише обмежений набір жестів і лише специфічні для конкретного пристрою функції. Інший недолік пов'язаний з обмеженими масштабами тестування QoS. Для того, щоб подолати ці недоліки, необхідний тестовий імітатор, що моделює обробку різних мобільних операцій (у тому числі різноманітні жести) й підтримує більше одного мобільного клієнта. Але навіть у цьому випадку перевірити специфічні для конкретних пристроїв функції мобільних сервісів досить важко. Крім того, неможливо протестувати роботу на безлічі різних пристроїв та браузерів, тому що емулятори зазвичай створюються на базі конкретного пристрою або платформи.

2. ***Тестування на базі пристроїв***. Для такого тестування потрібні створення тестової лабораторії та покупка реальних мобільних пристроїв, тому й коштує це набагато дорожче. Але в цьому випадку можна перевірити функції та поведінку конкретних пристроїв, а також параметри QoS. Крім того, можна оцінити можливості базових мобільних мереж, вибираючи та налаштовуючи конфігурацію в тестовому середовищі. Одна з головних проблем такого підходу полягає в тому, що мобільні пристрої та платформи змінюються дуже швидко. Труднощі обумовлені також обмеженими можливостями перевірки QoS, оскільки для неї потрібно багато мобільних пристроїв.

3. ***Тестування у хмарі.*** При такому підході зазвичай використовується хмара – наприклад, NTT Data. Основна ідея підходу цієї компанії до хмарного тестування на базі пристроїв полягає в тому, щоб побудувати хмару з мобільних пристроїв, яка підтримувала б сервіси тестування на великомасштабній основі. У цьому випадку значне зростання попиту на надання сервісів мобільного тестування задовольняється за рахунок бізнес-моделі, що передбачає внесення оплати в міру фактичного споживання. Мобільні користувачі отримують необхідне тестове середовище на умовах оренди. Цей підхід відрізняється від інших більш високою ефективністю при масштабному використанні додатків, а також при проведенні різноманітних тестових заходів на мобільних пристроях.

4. ***Тестування з використанням краудсорсингу.*** Краудсорсинговий підхід передбачає залучення фрілансерів, інженерів, що працюють за контрактом, чи спільнот кінцевих користувачів – зразок uTest (www.utest.com), а також формування краудсорсингової тестової

інфраструктури та встановлення сервера управління сервісами для підтримки неоднорідних користувачів. Сьогодні провайдери сервісів по реалізації такого підходу пропонують базове управління тестами, сервіс тестування та видачу звітів про помилки. Однак більшість тестових мобільних операцій управляються інструментами автоматизації мобільного тестування з дуже обмеженими можливостями – тестувальники отримують переваги стихійного тестування, що не потребує проведення інвестицій в лабораторію та купівлю чи оренду пристроїв, але виникає ризик низької якості тестування та невизначеності строків проведення.

14.4 Сучасні мобільні платформи

Android – одна з провідних мобільних програмних платформ на світовому ринку. ОС Android заснована на модифікованій версії ядра Linux. Спочатку розроблялася компанією Android Inc., яку потім купила Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз займається підтримкою та подальшим розвитком платформи. Android дозволяє створювати Java-додатки, керує пристроєм через розроблені Google бібліотеки. У 75% смартфонів, проданих у третьому кварталі 2012 року, була встановлена операційна система Android.

iOS – мобільна операційна система, що розробляється та випускається американською компанією Apple. Була випущена у 2007 році; спочатку – для iPhone і iPod touch, пізніше – для таких пристроїв, як iPad та Apple TV. На відміну від Windows Phone та Google Android, випускається тільки для пристроїв, вироблених фірмою Apple.

Maemo – операційна система та набір додатків, що працює в основному на мобільних веб-планшетах виробництва фірми Nokia.

Windows Phone 8.1 – ОС для мобільних пристроїв, розроблена компанією Microsoft.

Symbian – ОС для смартфонів та комунікаторів, розробляється консорціумом Symbian з 1998 року. Для розробки додатків під Symbian використовують мови C ++ та Java.

BlackBerry OS –операційна система з основним набором додатків для смартфонів та комунікаторів, що випускаються компанією Research In Motion Limited (RIM).

14.5 Основні операції тестування

14.5.1 Установка мобільних додатків

Тестування мобільного додатку починається з установки дистрибутива.

Android-додаток може бути отримано від розробника у вигляді * .apk файлу (у такому випадку для установки знадобиться файловий менеджер). Для установки потрібно підключити пристрій до ПК та скопіювати * .apk файл на даний пристрій. Потім запускається файловий менеджер, в ньому – інсталяційний файл * .apk. Якщо ж додаток вже знаходиться в магазині, то необхідно просто завантажити його.

Для iOS-додатків існує кілька варіантів установки, а саме: встановлення * .ipa файлу через iTunes або iFunbox або ж за допомогою програми для розробників Testflight, перейшовши за отриманим посиланням. Для створення файлів .mobileprovision необхідно надати розробникам UDID своїх пристроїв (UDID – унікальний ідентифікатор пристрою, який складається з 40 символів: малих літер та цифр).

Щоб дізнатися свій UDID, необхідно запустити програму iTunes, вибрати пристрій, відкрити вкладку «Огляд», натиснути «Серійний номер».

Щоб встановити додаток на iOS–девайс, потрібно запустити програму iTunes, потім вибрати папку, в якій знаходиться * .ipa файл, двічі натиснути, після чого він додасться в iTunes. В iTunes зайти в потрібний девайс та вибрати вкладку «Програми». У вкладці «Програми» вибрати потрібний додаток та натиснути кнопку «Встановити». Потім натиснути кнопку «Застосувати» – почнеться установка програми. Дочекатися установки програми і можна запускати її на пристрої.

14.5.2 Зняття скріншотів на мобільних пристроях

На iOS-пристроях: одночасне затискання кнопок Home + Power. Скріншот зберігається в Галереї. З Галереї можна дістати, підключивши до комп'ютера й скопіювавши файли як з зовнішнього носія або відправивши на пошту. Для відео потрібна спеціальна програма – Jing, Screencapture.

На Android-пристроях: для 2.3 та пізніших версій – довге натискання на кнопку «Недавні програми». Для 4.0 й вище – кнопка зменшення гучності + Вимкнення.

Скріншоти так само зберігаються в Галереї, звідки можуть бути скопійовані на комп'ютер або відправлені на пошту.

14.5.3 Запис відео-опису помилки на пристрої

Важливою особливістю тестування додатків на мобільних пристроях є складність повторення, здавалося б, добре описаних помилок. Навіть якщо з точністю до дрібниць опишете помилку, програміст, можливо, не зможе повторити її. Тут на допомогу прийде запис відео з пристрою. Починаючи з Android KitKat (4.4), можна записувати відео за допомогою Android Studio and SDK Tools.

14.5.4 Вилучення креш-логів

Для iOS-пристроїв:

- ***OS X:*** ~ / Library / Logs / CrashReporter / MobileDevice / <your iPhone's name> /
- ***Windows XP:*** C: \ Documents and Settings \\ Application Data \ Apple computer \ Logs \ CrashReporter \ <your iPhone's name> \
- ***Windows Vista / 7:*** C: \ Users \\ AppData \ Roaming \ Apple Computer \ Logs \ CrashReporter \ MobileDevice \ <your iPhone's name> \

Для Android-пристроїв:

- для версій нижче 4.1 – Logcollector, alogcat, sendlog.
- для версій вище 4.1 – Android SDK, який необхідно встановити на комп'ютер. При підключенні девайса у списку пристроїв у вікні з'являються консоль-логи.

14.6 Перелік перевірок у мобільному тестуванні

1. ***Розмір екрану та touch-інтерфейс:***

- усі елементи повинні бути такого розміру, щоб користувач міг точно обирати один із них;
- відсутність порожніх екранів у додатку – користувач не повинен опинятися у ситуації, в якій не очевидно, що зараз відбувається й що робити;
- слід перевіряти багаторазове швидке натискання кнопки – часто при цьому може статися падіння програми. Так само слід перевіряти мультитач – натискання кількох кнопок одночасно;
- слід перевіряти наявність або відсутність «нативних» жестів (pinch-to-zoom, doubletap) – якщо, наприклад, підтримується зум частини програми, то повинен використовуватися жест за замовчуванням. А якщо немає необхідності виділяти картинку, то по даблтапу вона не повинна виділятися.

2. ***Ресурси пристроїв***:

- витоки пам'яті особливо можуть проявлятися на вікнах з великою кількістю інформації *(приклад, довгі списки)*, під час завдань з тривалим workflow (коли користувач довго не виходить з програми), при некоректно працюючому кешуванні зображень;
- обробка ситуацій браку пам'яті для функціонування ОС, коли програма активна або працює у фоні;
- нестача місця для установки або роботи програми;
- відсутність у деяких пристроях підтримуваних додатком функцій (*3G, SD-карта та ін.);*
- установка або перенесення програми на карту SD.

3. ***Різні розміри екрану та версії ОС:***

- ретина та звичайні екрани. На ретина-екранах елементи інтерфейсу та текст відображаються дрібніше. Картинки для ретина-екрану можуть потрапити в неретина версію і тоді будуть дуже великими;
- адаптація додатків до портретної та альбомної орієнтації пристрою;
- версії ОС. Додаток не має встановлюватися на непідтримувані пристрої. Обов'язкова перевірка на всіх доступних з підтримуваних девайсів;
- підтримка необхідних медіа-файлів даною моделлю та ОС, тому що окремі розробники можуть урізати підтримку роботи з деякими форматами;
- відповідність використовуваних у додатку view, смисловому призначенню та концепціям платформи. Проектні рішення, які мають сенс для однієї платформи, можуть виглядати та бути недоречними в контексті іншої платформи.

4. ***Реакція програми на зовнішні переривання:***

- вхідні та вихідні SMS, MMS, дзвінки, сповіщення інших додатків;
- вимкнення пристрою, вилучення акумулятора, розрядка пристрою;
- перехід у режим очікування *(у тому числі і з захистом паролем)*. Зміна орієнтації пристрою в режимі очікування;
- відключення та підключення проводу;
- відключення та включення мережі, Bluetooth, авіарежиму, GPS;
- втрата зв'язку з сервером або проксі *(підключення є, але пакети не доходять);*
- відключення та підключення SD-карти, додаткових пристроїв на зразок фізичної клавіатури або гарнітури;
- зарядка пристрою;
- робота з акселерометром;

- робота з фізичною клавіатурою *(якщо у списку підтримуваних моделей є такі).*

5. ***Платний контент всередині програми:***

- відповідність ціни та вмісту, заявленого в додатку, та того, що потрапляє до користувача;
- відновлення покупки (оновлення додатку).

6. ***Інтернаціоналізація (перевіряти і в портретному, і в ландшафтному режимі):***

- перевірка коректності перекладу;
- перевірка того, що всі написи входять у відповідні форми, кнопки та ін.;
- перевірка форматів дат, роздільників у числах, специфічних особливостей локалізації (наприклад, пробіл перед знаком питання у французькій мові, верхні індекси «о» та «а» у порядкових числівниках в іспанській мові та іншінетривіальні моменти).

7. ***Постійний зворотний зв'язок з користувачем:***

- у всіх натиснених елементів має бути натиснутий стан (відгук на дію) – завдяки цьому користувач завжди буде бачити, чи дійсно натискання сталося. В Android-додатках у елементів може бути ще один стан – focused;
- реакція кнопок на натискання. Швидкість відгуку елементів повинна бути досить високою. Бажано використовувати для перевірки цього пункту найслабші пристрої серед підтримуваних;
- повідомлення при завантаженні контенту або прогрес-бар.
- повідомлення при помилці доступу до мережі, BT, GPS;
- наявність зрозумілих повідомлень при спробі видалити важливу інформацію;
- наявність екрану або повідомлення при закінченні процесу або гри;
- наявність та синхронність звуків або вібрації з повідомленнями та іншими подіями на екрані.

8. ***Оновлення:***

- переконатися, що підтримуються ті ж версії ОС, що й попередня (якщо нова версія додатку використовує нові можливості ОС, то для старих підтримуваних версій ОС необхідне створення урізаної версії додатку);
- перевірка адекватного відновлення *(зберігаються всі дані користувача та ін.).*

14.7 Розміри та дозволи екранів мобільних пристроїв

В iPhone і iPad використовується поняття точок – apple points. Роздільність дисплею iPhone аж до 5 моделі – 320 на 480 точок. Що таке Retina Display? Retina Display використовує подвійну щільність пікселів, тобто на одну точкку припадає 4 фізичних пікселя. Так, у iPhone 4 при фізичному дозволі екрану 640 на 960 пікселів все ж 320 на 480 точок. Але при цьому графіка більш деталізована за рахунок використання вдвічі більших зображень у додатках. Для iPhone 5 значення в точках збільшилося з 480 до 568. У iPad усіх версій, включаючи Mini, воно незмінне – 1024x768.

За умовчанням у масштабі 1 до 1 стандартна графіка в web (наприклад, background-image) буде відображена у збільшеному вдвічі вигляді, на 4 пікселя фізичного екрану буде розтягнутий один піксель растрового зображення. Як використовувати графіку високої роздільної здатності в такій ситуації, ми розглянемо в наступному розділі. Головне, що потрібно пам'ятати про дозволи iOS пристроїв – не треба думати, що сторінки на ретині і неретині мають різну ширину. Різниця тільки в щільності пікселів, пропорції абсолютно ті ж.

В Android ситуація набагато складніша, точніше, різноманітніша. В Android використовується поняття different screen densities (DPI). Суть його в тому, що при різних фізичних розмірах дисплея і роздільній здатності, модель поведінки по точках відрізняється в залежності від призначення пристрою. Аналогом Points служить вищезгаданий DIP.

Щільність екрану – 1. У цю категорію входить безліч пристроїв, такі як телефони 320x480 (HTC Hero, LG Optimus One), 7-дюймові планшети 1024x600 (Samsung Galaxy Tab, Kindle Fire), 10-дюймові планшети (Asus Transformer, Acer A500, Galaxy Tab 10'1).

Щільність екрану 1,33 використовується в 7-ми дюймових планшетах таких, як Nexus 7. Фізичний дозвіл – 1280x800, в DIP – 960x600.

Щільність екрану 1,5 – мобільні телефони високої цінової категорії 2011 і середньої 2012 року, 10-ти дюймові планшети з FullHD. Наприклад, Nexus One 480x800 пікселів, 360x533 * DIP, HTC One S – 540x960 пікселів, 360x640 DIP і Asus Transformer Pad Infinity 1920x1200 пікселів, 1280x800 DIP.

* На Android Developers зустрічається 534, але вимір ширини браузера через JS показує саме 533, відповідно і в медіазапитах варто орієнтуватися на цю цифру.

Щільність екрану 2 – телефони високої цінової категорії 2012 року і буквально пара планшетів. Наприклад, HTC One X, Samsung Galaxy S3 – дозвіл екрана 1280x720, розміри в DIP 360x640. Планшет Google Nexus 10 – дозвіл 2560x1600 пікселів, 1280x800 DIP.

Щільність екрану 3 – флагмани 2013 з FullHD-дисплеєм. При вирішенні в 1920x1080 пікселів у них все ті ж 360x640 DIP.

При всьому цьому вибивається з колії Google Nexus 4 з екраном в 1280x768 пікселів з щільністю 2 *(384x640)*.

Таким чином, є три дозволи для портрета – 320, 460 і рідше 384, і чотири з половиною ландшафтних – 480, 533, 568, 640 *(598 з софт-клавішами)* .У планшетів це портретні 600 і 800 і ландшафтні 960, 1024 і 1280 (рис. 14.2).

Portrait / landscape	320	360	384	600	768	800
480	320x480 (1), 640x960 (2) iPhone 1-4s, HTC Hero			Created: june 2013 Top numbers: Portrait points/DIP, aviable for Web/App Left numbers: landcape points/DIP, aviable for Web/App Cells: A x B (C) A - Portrait side physical pixels B - landscape side physical pixels C - device pixel ratio		
533	480x800 (1.5) Nexus One, Galaxy S2, HTC Desire					
568	640x1136 (2) iPhone 5					
598		720x1196 (2), 1080x1794(3) Galaxy Nexus, Moto X (?) Sony Xperia Z	768x1196 (2) Nexus 4			
640		540x960 (1.5), 720x1280 (2), 1080x1920 (3), HTC One S, Galaxy S3, Galaxy S4, HTC One				
960				800x1280 (1.33), 1200x1960 (2) Nexus 7* *1st and 2nd generation		
1024				600x1024 (1) Galaxy Tab, Galaxy tab 7.0	768x1024 (1), 1536x2048 (2) iPad 1-4, iPad Mini	
1280						800x1280 (1), 1200x1920 (1.5), 1600x2460 (2), Galaxy Tab 10.1, Asus Transformer Prime, Nexus 10

Рис. 14.2. Типові розширення популярних продуктів

14.8 Проблеми в тестуванні мобільних додатків

Мобільні пристрої працюють від акумуляторів і тому змушені автоматично переходити в режим очікування через пару хвилин бездіяльності. Це означає, що вам доведеться включати телефон перед кожним тестуванням, що при одночасному тестуванні декількох телефонів, займає пристойну кількість часу. Звичайно, на багатьох пристроях можна відключити автоматичне блокування (або хоча б зробити час відключення досить великим), але бажано все ж працювати з найпоширенішими серед користувачів налаштуваннями ОС.

Перехід у режим очікування особливо неприємний при використанні таких систем тестування, як Browserscope, що вимагає деякий час для завершення. Можливо, що телефон перейде в режим очікування прямо посередині тесту.

Потім з'являється проблема власне набору певного тексту (наприклад, адреси тестованої сторінки). Набір може зайняти досить багато часу на телефоні з цифровою клавіатурою, але навіть на телефонах з алфавітною (програмною чи апаратною) клавіатурою доведеться акуратно вводити довгі тексти на декілька пристроїв з різними інтерфейсами.

Для тестування в умовах вхідних дзвінків, смс вам доведеться переставляти SIM-карту з одного пристрою в інший, для цього найчастіше треба виймати акумулятор. Особливо гостро ця проблема стоїть при тестуванні особливостей і послуг мобільних операторів на нестандартних і дорогих тарифах. Крім того, вставивши SIM-карту, доведеться почекати, поки телефон увімкнеться.

Часто доводиться передавати тестовані віджети по Bluetooth, що так само досить втомливо в умовах розмаїтості інтерфейсів.

Зазвичай всі тести проводяться на одному пристрої, потім – на іншому и т.д. Це не так оптимально, як тестування на всіх пристроях одночасно (не дає можливості просто порівняти висновок) і виконання тестів на двох пристроях, проте це найбільш економічний за часом спосіб.

Створення скріншотів на мобільних пристроях так само часто нетривіальна робота, особливо якщо тестується телефон, відключений від комп'ютера за умовами тесту або з якихось інших причин.

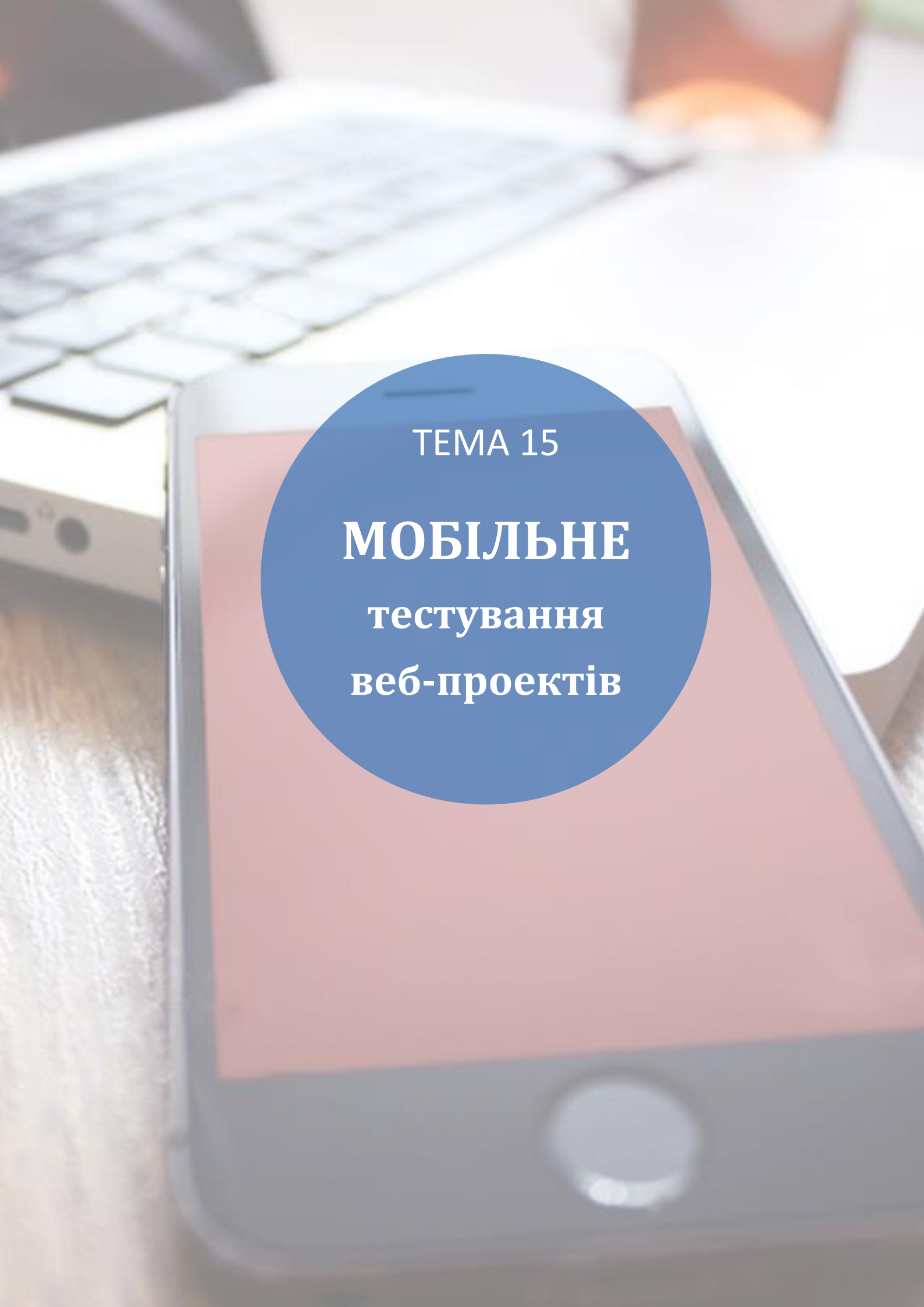
14.9 Особливості тестування на різних пристроях

На прикладі GoPro-камери тестуються:

- кріплення камери, щільність кріплень, фіксація камери при русі;
- фокусування камери при швидкому / повільному русі;.
- якість зйомки в денний і нічний час, в умовах поганої видимості *(дощ, туман)*;
- пило- , волого- захищеність корпусу;
- кріплення та зйомка з швидкого об'єкта *(автомобіля, мотоцикла)*;
- для тестування на gps-навігаторах для андроїд зручно використовувати додаток типу Fake gps для емуляції переміщень.

На прикладі відеореєстратора тестується:

- якість запису / відтворення відео;
- підтримка зв'язку з комп'ютером;
- підтримка карт пам'яті;
- фокусування на об'єкті;
- кріплення та фіксація відеореєстратора.



ТЕМА 15

МОБІЛЬНЕ тестування веб-проектів

15 МОБІЛЬНЕ ТЕСТУВАННЯ ВЕБ-ПРОЕКТІВ

Щоб зрозуміти особливості тестування мобільних веб-додатків, слід брати до уваги чинники, що відрізняють мобільні додатки від десктопних – специфічні та різноманітні ОС для мобільних платформ, різні виробники та конфігурації комплектуючих, функціональність пристроїв як комунікаторів і т.п.

У зв'язку з цими особливостями підхід до розробки додатків і, зокрема, тестування на мобільних пристроях досить сильно відрізняється від десктопного. Виникає безліч додаткових важливих нюансів і вимог, які необхідно протестувати.

15.1 *Тестування з використанням мобільних пристроїв*

Існує поняття ***Responsive web design*** *(«адаптивний дизайн»)*. Даний термін російською звучить як «адаптивний веб-дизайн», звідси походить назва виду верстки – адаптивна верстка, яка забезпечується за допомогою використання css media-запитів.

Адаптивна верстка – це основне, що перевіряється при мобільному тестуванні веб-проектів. Для успішної верстки необхідно перевірити виконану роботу адаптації. Далі розглянемо доступні на сьогодні способи такої перевірки.

15.1.1 Тестування з використанням мобільних пристроїв без емуляторів

Тестування з використанням мобільних пристроїв – це найбільш якісний і в той же час найбільш дорогий підхід до тестування, оскільки необхідна велика кількість пристроїв.

Багато в чому мобільне тестування веб-проектів нічим не відрізняється від десктопного тестування. Однак є кілька ключових відмінностей, пов'язаних з конструктивними особливостями мобільних пристроїв.

Перше – це розміри екрана. Мобільні пристрої мають значно менші розміри екрану, ніж монітор комп'ютера. Сторінка, яку неможливо масштабувати, як правило повинна міститися по ширині екрану пристрою, горизонтальної прокрутки не повинно бути. Весь контент повинен добре читатися і бути доступним для перегляду.

Друге – це навігація по сторінках сайту і взаємодія з активними елементами на сайті за допомогою сенсорного екрану, а не мишки і клавіатури.

Область тапа на мобільному повинна бути такою, щоб користувач мав можливість управляти всіма активними елементами на сайті. При подвійному тапі на активні елементи вони повинні автоматично збільшуватися. При тапі на текстові поля вони також автоматично збільшуються, і з'являється клавіатура. При подвійному тапі на текстовий блок текст повинен вирівнюватися щодо меж екрану і масштабуватися для комфортного читання.

Третє – можливість повороту на мобільному пристрої з портретного режиму в альбомний і навпаки. **Основні проблеми при повороті пристрою:**

1) можуть зникати відкриті вікна, спливаючі підказки і деякі інші активні елементи;
2) якщо на момент повороту курсор знаходиться у текстовому полі, іноді зникає клавіатура, або фокус переноситься на інший елемент;
3) при повороті пристрою може не змінитися ширина клавіатури;
4) при одночасному введенні тексту і повороті пристою введений текст може зникати.

Четверте – особливості роботи з мультимедіа. Зокрема відтворення відео через Adobe flash player. Тут проблеми можуть виникнути у зв'язку з тим, що корпорація Adobe припинила підтримку флеш на Android, тому не на всіх сучасних пристроях на операційній системі Android встановлений Adobe Flash Player. Розробники замінюють його іншими технологіями відтворення відео – наприклад, HTML5.

Особливу увагу слід звернути на розмітку сторінки. У різних режимах перегляду розмітка сторінки і взаємне розташування різних блоків може відрізнятися.

Використання реальних пристроїв для тестування особливо актуально при тестуванні функціоналу сайту, хоча в першому наближенні для цих цілей можна використовувати спеціально розроблені програми-симулятори.

15.1.2 Тестування з використанням мобільних пристроїв у поєднанні з емуляторами браузерів

З виходом 6-ї версії Apple представила інструмент для налагодження мобільного пристрою через Safari Web Inspector. Розглянемо цей інструмент більш докладно.

Для початку необхідно дозволити віддалене налагодження на пристрої. «Settings» → «Safari» → «Advanced» і дозволити «Web Inspector» (рис. 15.1).

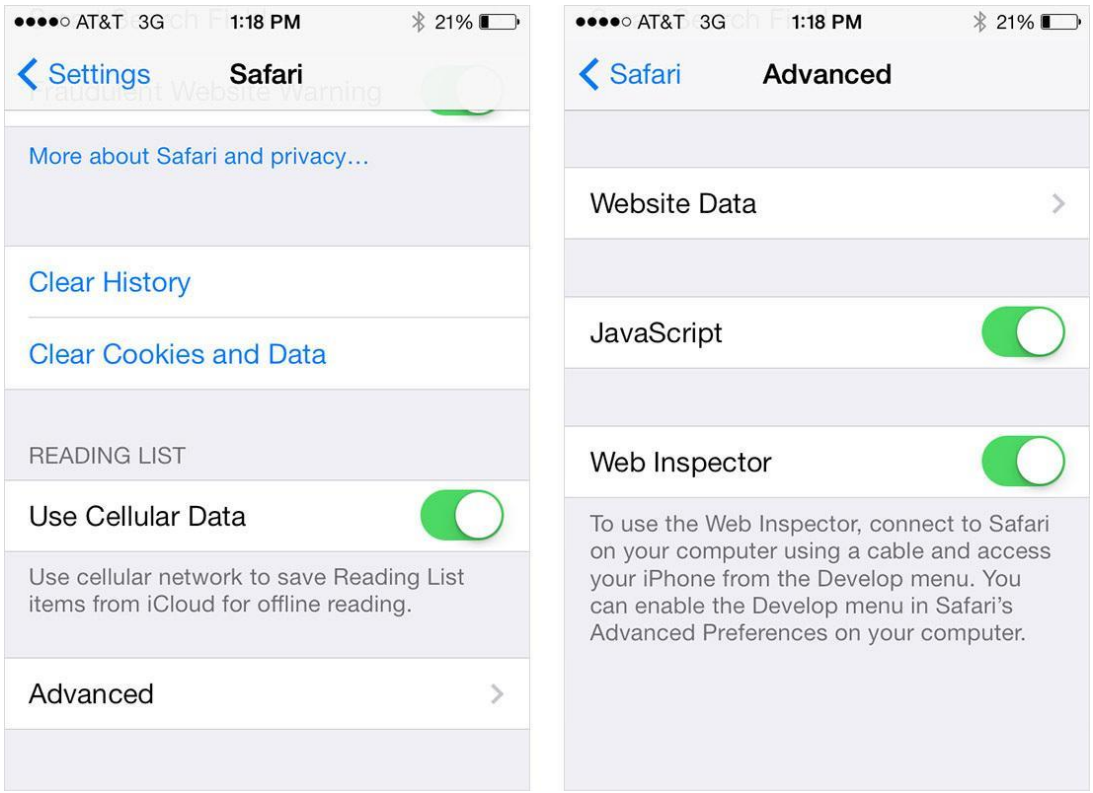


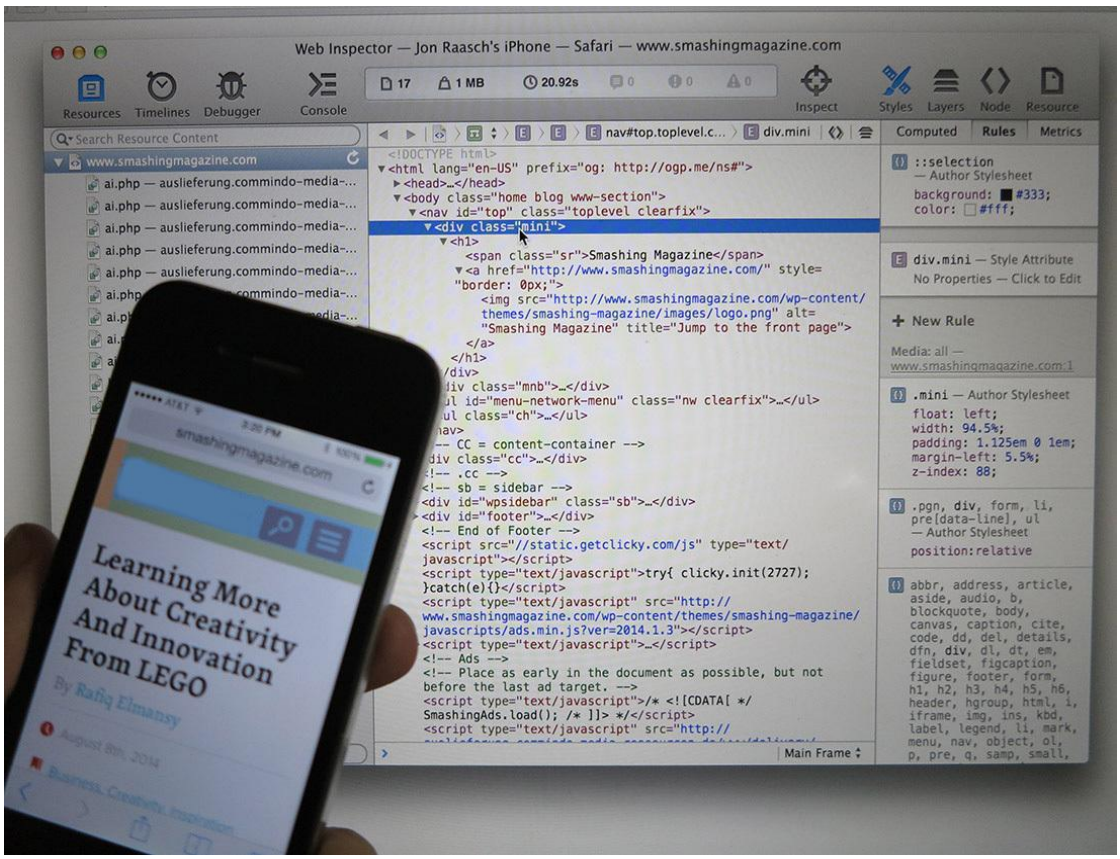
Рис 15.1. Налаштування віддаленого налагодження

Потім слід підключити пристрій до комп'ютера через USB і відкрити Safari *(версії не нижче 6)*. У «Preferences» → «Advanced» виберіть «Show Develop menu in menu bar» (рис. 15.2).

У цьому меню з'являється пристрій і кілька сторінок з налаштуваннями.

Рис 15.2. Вікно з налаштуваннями

На кожній зі сторінок представлено багато інструментів, DOM Inspector (рис. 15.3).



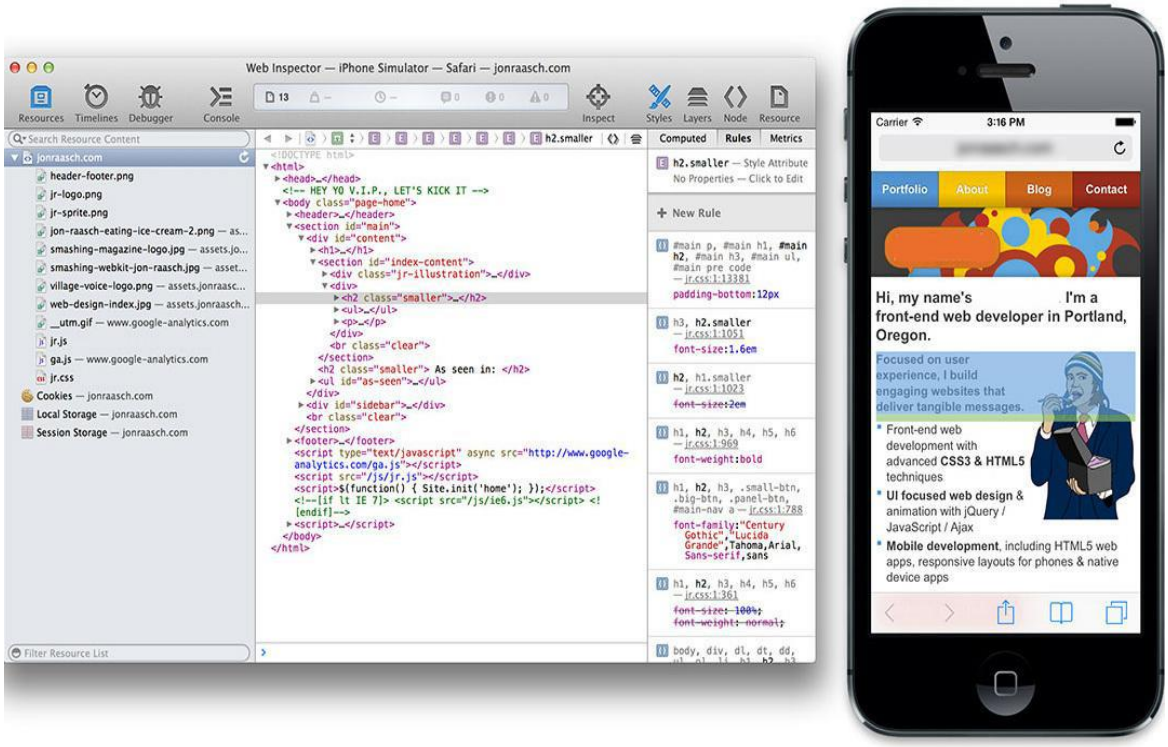


Рис 15.3. Інструменти, DOM Inspector

Крім DOM Inspector, iOS можна також використовувати:

- ✓ часовий графік для перегляду запитів, розкладки і рендерингу сторінок, роботи JavaScript;
- ✓ відладчик з можливістю установки точок зупинення;
- ✓ консоль JavaScript.

Детально все це описано в «Safari Web Inspector Guide».

Як і у випадку з симулятором iOS, віддаленим налагодженням можна займатися тільки з MAC OC.

У випадку з Android інструменти віддаленого налагодження дозволяють працювати з десктопного ПК за допомогою Chrome's Developer Tools. Ці інструменти крос-платформенні.

Спочатку на телефоні необхідно перейти в «Settings» → «About Phone» (для Android 4.4+), або «Settings» → «About Tablet». Потім потрібно натиснути «Build Number» сім разів. Після цього з'явиться додаткове меню з налаштуваннями розробника, де слід встановити чекбокс «USB debugging».

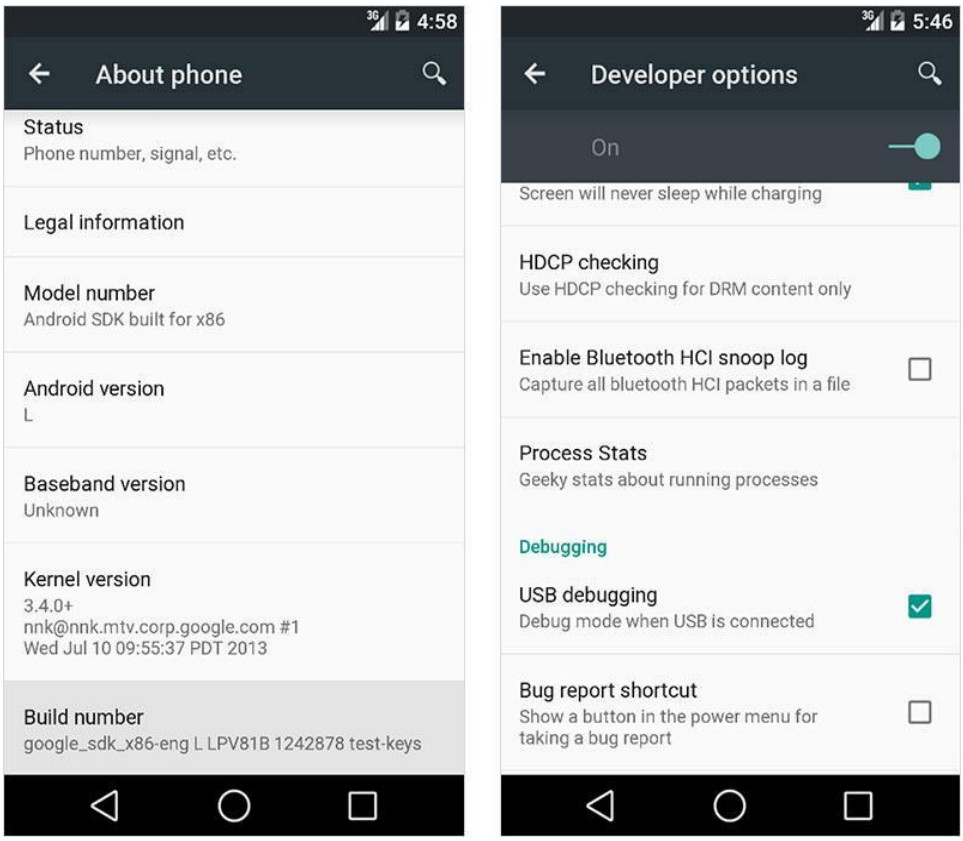


Рис 15.4. Налаштування розробника

В адресному рядку браузера Chrome необхідно набрати about:inspect. Дозволити “Discover USB devices”, після чого в меню з’явиться мобільний пристрій (рис.15.5).

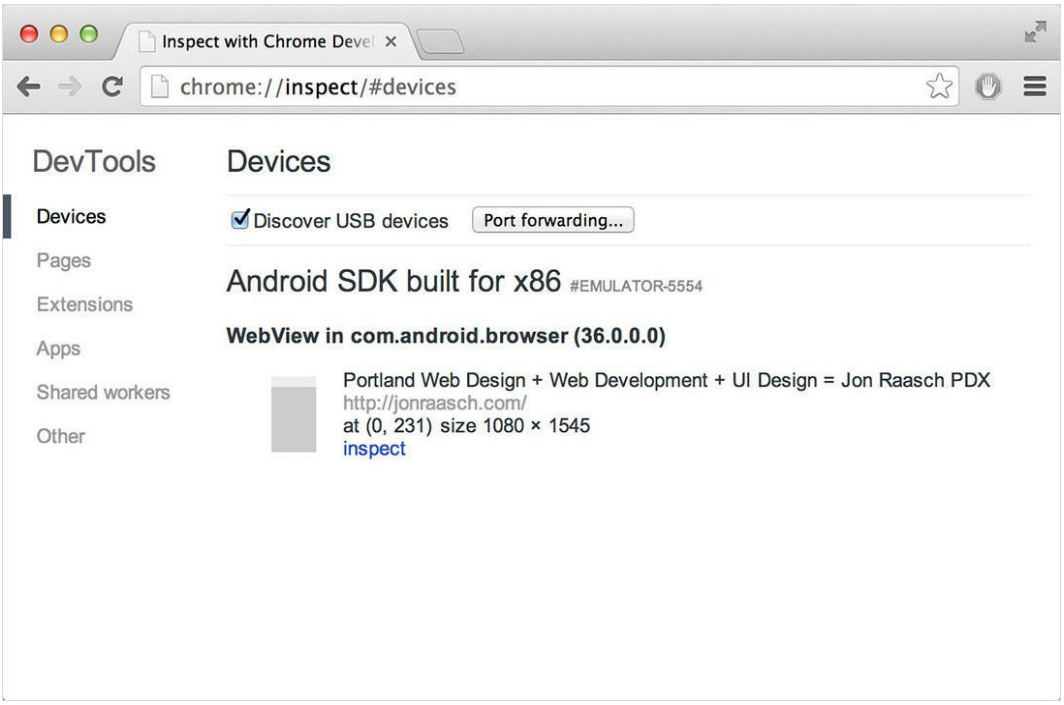


Рис 15.5. Налаштування Google Chrome

Також з'являться відкриті закладки в мобільному браузері. Тут необхідно вибрати одну з них.

При цьому будуть доступні:

- DOM Inspector;
- мережева панель із зовнішніми ресурсами;
- панель вихідного коду для налагодження JavaScript;
- консоль JavaScript.

Подробиці можна знайти в інструкції «Introduction to Chrome Developer Tools, Part One.»

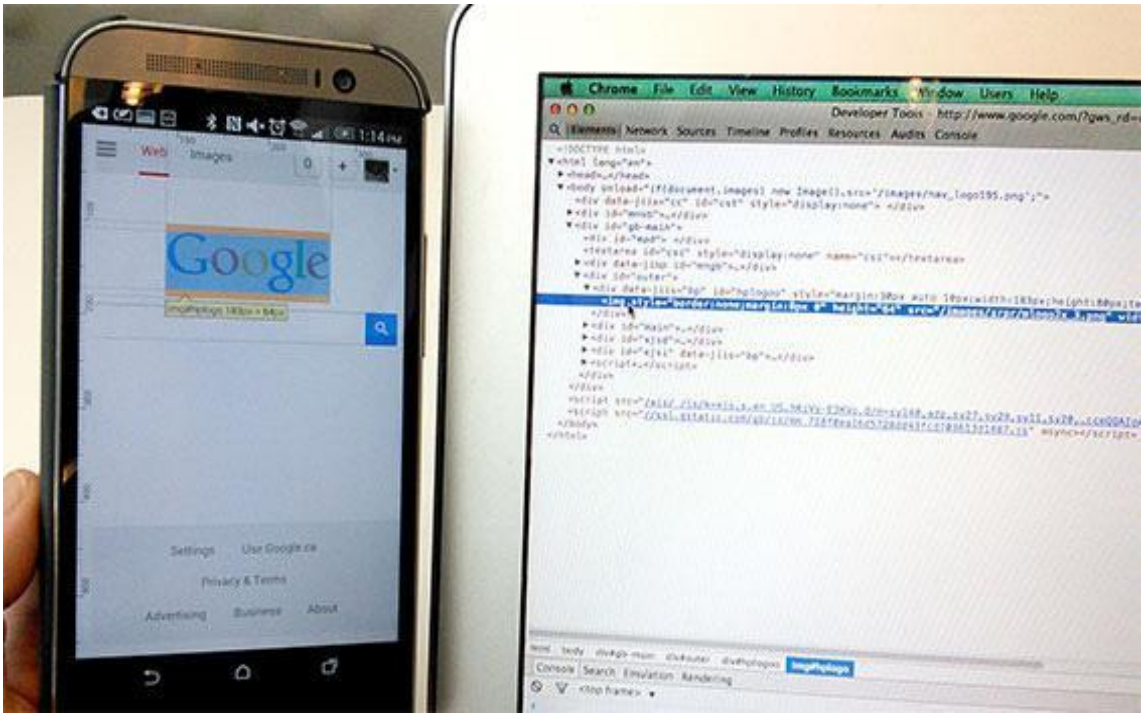


Рис 15.6. Відкриті закладки в мобільному браузері

Також є можливість займатися віддаленим налагодженням в емуляторі Android.

При необхідності налагоджувати пристрій з iOS на Windows або Linux або налагоджувати пристрій, що працює під Windows Phone або BlackBerry – можна використовувати додаток Weinre (web inspector remote).

Налаштування Weinre досить складне, тому його треба встановлювати і на сервері, і на сторінці. Спочатку нам знадобиться встановити Node, а потім встановити модуль Weinre:

npm install –g weinre

Потім слід запустити сервер налагодження (підставити ip-адресу комп'ютера):

weinre --boundHost 10.0.0.1

Проходимо на localhost: 8080 і копіюємо вміст тега <script>. Його потрібно буде вставити на відладжувану сторінку.

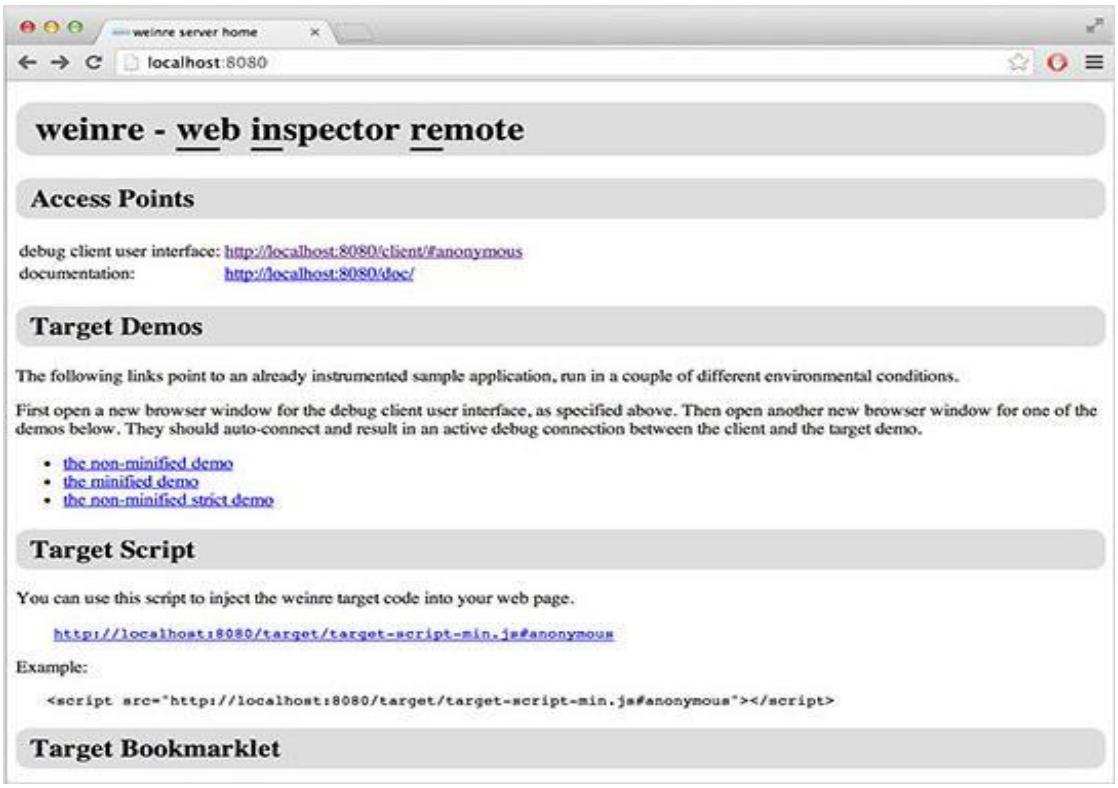


Рис 15.7. Сервер відладки

Натисніть посилання вгорі user interface for debugging clients (http: // localhost: 8080 / client / # anonymous). Тепер, коли ви відкриєте сторінку на пристрої, її можна буде побачити у списку цілей (targets).

Рис 15.8. Список цілей

Після цього можна використовувати інструменти для відладки (рис 15.9).

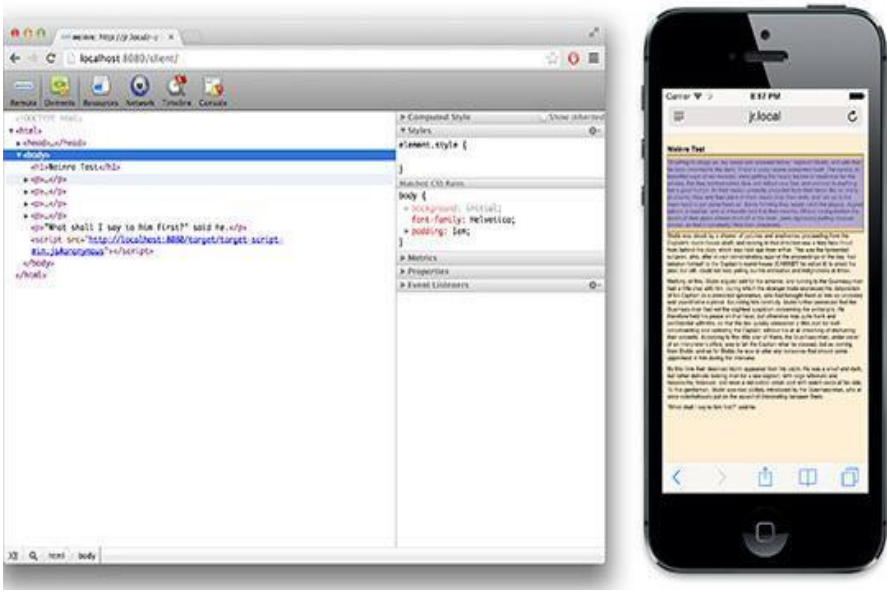


Рис 15.9. Інструменти для відладки в роботі

Weinre дозволяє тестувати будь-який пристрій, але в цьому додатку менше можливостей, ніж у рідних рішеннях для iOS і Android. Наприклад, не можна покроково налагоджувати JavaScript.

Ще один варіант віддаленого тестування називається Ghostlab.

15.2 Тестування з використанням різних емуляторів

За відсутності реальних пристроїв часто користуються спеціально розробленими для цього емуляторами. Однак, з точки зору забезпечення якості – це не переважний підхід до тестування.

Тестування на основі емуляції – це спосіб тестування, що передбачає використання емулятора мобільного пристрою, що імітує його поведінку у віртуальній машині. Досить часто такі емулятори бувають включені до складу комплекту інструментів для розробника, прикладеного до мобільної платформи. Ціна такого рішення відносно невелика, тому що в цьому випадку немає необхідності у створенні тестової лабораторії, покупці або оренді фізичних пристроїв. Однак емуляцію можна використовувати тільки для оцінки функціональності системи в дуже обмеженому контексті. Такий підхід недорогий, але у нього є ряд істотних недоліків. Наприклад, перевірити обробку всіх жестів у повному обсязі досить важко, тому що більшість емуляторів підтримують лише

обмежений набір жестів і лише специфічні для конкретного пристрою функції.

15.2.1 Вбудовані засоби десктопних веб-браузерів

У найбільш популярних веб-браузерах існують вбудовані засоби для мобільного тестування. До таких браузерів відносяться: Google Chrome, Mozilla Firefox, Opera.

Найпростіший спосіб – це зміна ширини вікна браузера. Це можна робити як вручну, так і за допомогою спеціальних плагінів. Крім того, існує безліч плагінів для браузерів для мобільного тестування. Для браузера Google Chrome – це плагіни: Responsive Viewer, Resolution Test, Window Resizer, Screen Resolution Tester, Viewport Resizer, Mobile View, Browser Resize. Для браузера Opera – плагіни: Resize Me, Responsive Web Design Tester. Однак можна знайти й інші плагіни на будь-який смак.

15.2.2 Засоби тестування через віддалений доступ

Емулятори та симулятори корисні, але не на 100% достовірні. Завжди потрібно намагатися проводити тестування на максимальній кількості реальних пристроїв.

Але це не означає, що вам потрібно їх всі купувати. Можна скористатися послугами сервісів віддаленого тестування, які пропонують веб-інтерфейс для тестування на віддалених пристроях. Можна буде взаємодіяти з телефоном і бачити його екран.

Для тестування пристроїв від Samsung, таких як Galaxy S5 можна безкоштовно скористатися послугами Samsung Remote Test Lab – вони надають для тестування великий вибір своїх пристроїв.

Також можна користуватися ресурсами Keynote Mobile Testing. Вони недешеві, але кількість доступних пристроїв вражає, і деякими можна користуватися безкоштовно.

Якщо вам потрібні фізичні пристрої, можна звернутися в Open Device Lab, де є список найближчих доступних лабораторій тестування.

Віддалене тестування виявляє безліч труднощів.

Інструменти віддаленого тестування надають інтерфейс для підключення з десктопу. Таким чином, ми працюємо з даними з реальних пристроїв на потужному десктопном комп'ютері.

Для тестування адаптивної верстки можна використовувати ресурс http://responsivepx.com. Для цього необхідно ввести свою адресу сайту, вибрати потрібні параметри: ширину і висоту та натиснути кнопку Open (див. Рис. 10).

http://www.gtalk.kz/wp-content/uploads/2012/10/responsivepx.jpg

Рис 15.9. Панель властивостей http://responsivepx.comhttp://responsivepx.com/ http://responsivepx.com/

Другий сайт зі схожим функціоналом – http://resizemybrowser.com/ (див. Рис. 11).

З назви стає зрозумілим, що даний ресурс дозволяє змінювати браузер під потрібні розміри. По кліку відкривається нове вікно з потрібними розмірами.

Нічого зайвого на сайті http://quirktools.com/screenfly/. Потрібно просто ввести адресу сайту і відразу ж можна подивитися, як ваш проект виглядає. Багато варіантів гаджетів і телефонів. Можна навіть вибрати телевізор.

Цікавий ресурс для перевірки онлайн вашого сайту на різних планшетах і смартфонах – http://screenqueri.es/. При чому дуже великий вибір: продукція від Apple, Samsung, Htc, Blackberry, LG, Nokia та ін. Є також можливість ручної зміни розмірів екрану.

Цікавий варіант з використанням iframe'ів є на github. Суть: редагуємо html-файл з потрібними значеннями ширини екрану і закидаємо в директорію з сайтом, завантажуємо наш файл, і у фреймах повинен з'явитися потрібний нам сайт у варіантах на девайсах.

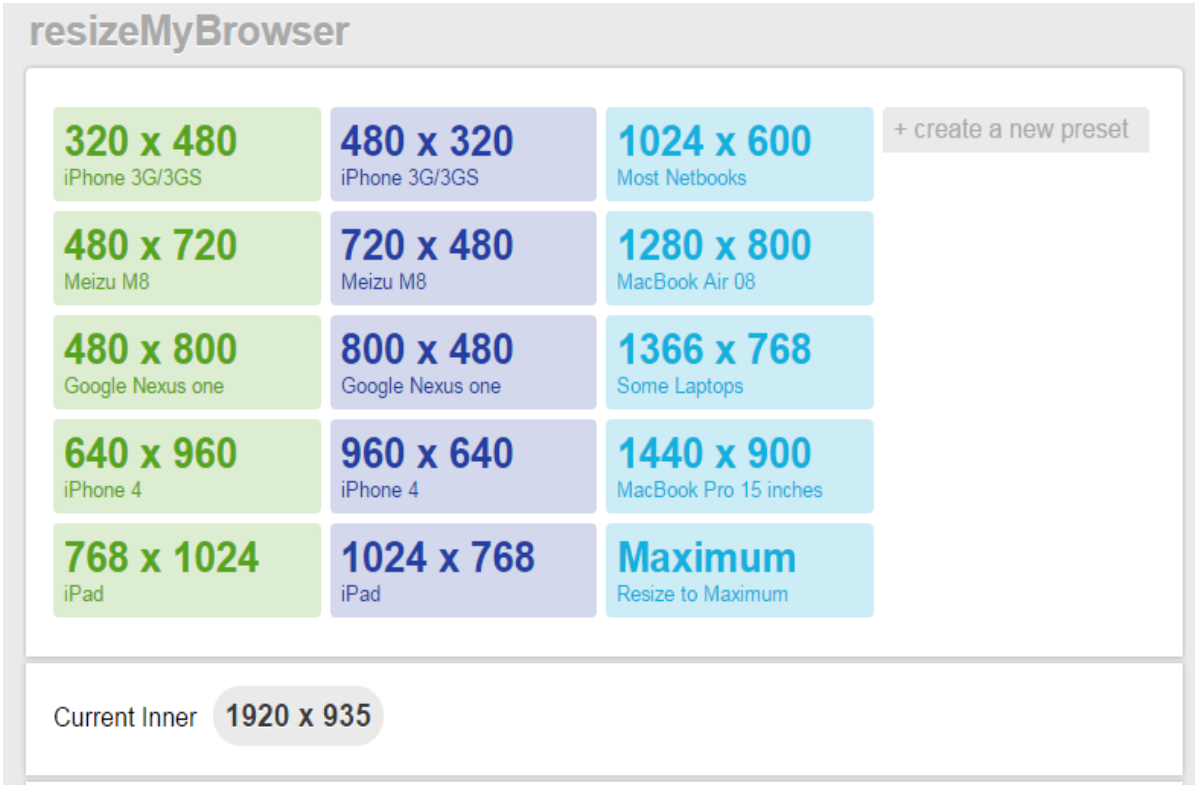


Рис 15.11. Панель налаштувань – http://resizemybrowser.com/

Подібних інтернет-ресурсів дуже велика кількість. Серед них є як безкоштовні, так і платні.

15.3 Тестування на програмах-симуляторах

Для тестування на iPhone- і iPad-пристроїв є кілька варіантів. Перший серед них – офіційний Apple iOS Simulator, що входить у поставку Xcode. Дозволяє тестувати різні комбінації софта і заліза, але тільки на Mac.

Рис 15.12. Зовнішній вигляд симулятора для iPhone

Для цього необхідно встановити і запустити Xcode. Далі слід вибрати «Show Package Contents» → "Contents" → "Applications" → "iPhone Simulator" (рис. 15.13).

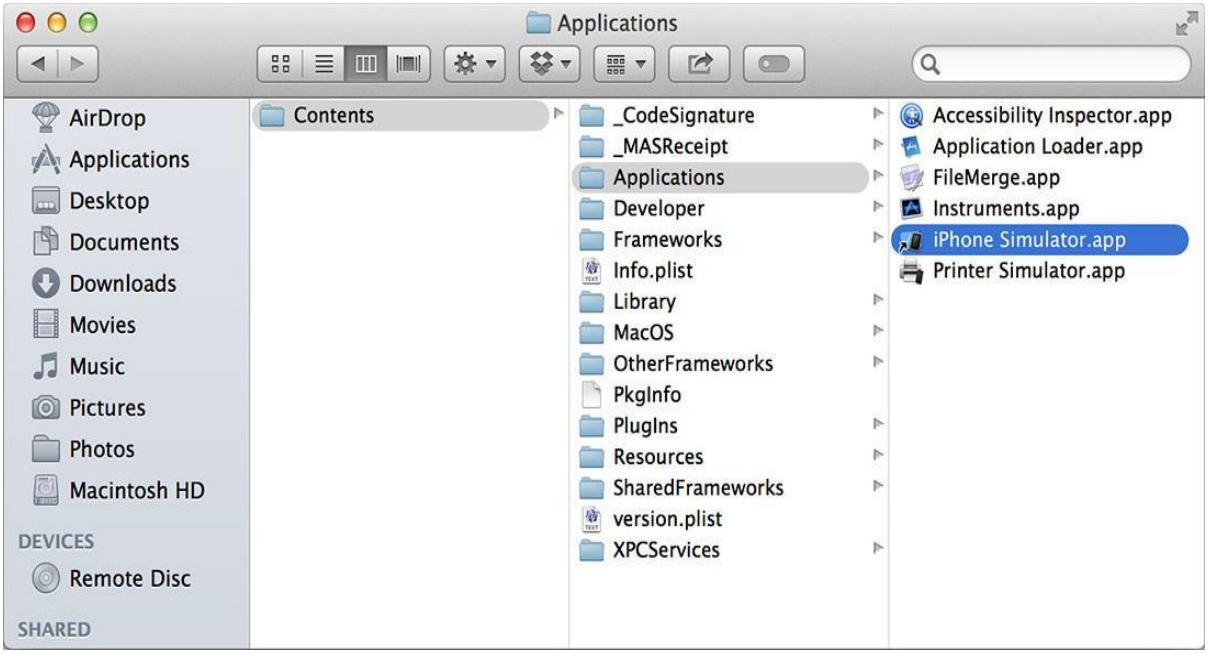


Рис 15.13. Місцезнаходження додатку iPhone Simulator

Хоча знайти його нелегко, використовувати симулятор просто. Для цього необхідно відкрити Safari в симуляторі і можна тестувати свій сайт. Можна перемикатися між різними пристроями iPhone і iPad, міняти версію iOS, повертати пристрій і т.п.

Якщо у вас немає Mac OC, можна взяти симулятор iPad для Windows. Крім нього існує ще кілька варіантів, зокрема – онлайн.

У ОС Android емулятор крос-платформний. На жаль, він досить складний в налаштуванні.

Для початку необхідно завантажити добірку, що включає Android Development Tools (ADT) для Eclipse і Android software development kit (SDK). Потім потрібно слідувати інструкціям по установці. Слід також не забувати про те, що потрібно по замовчуванню встановити і "Intel x86 Emulator Accelerator (HAXM installer)". Також знадобиться встановити HAXM (IntelHaxm.dmg на Макі та IntelHaxm.exe на PC).

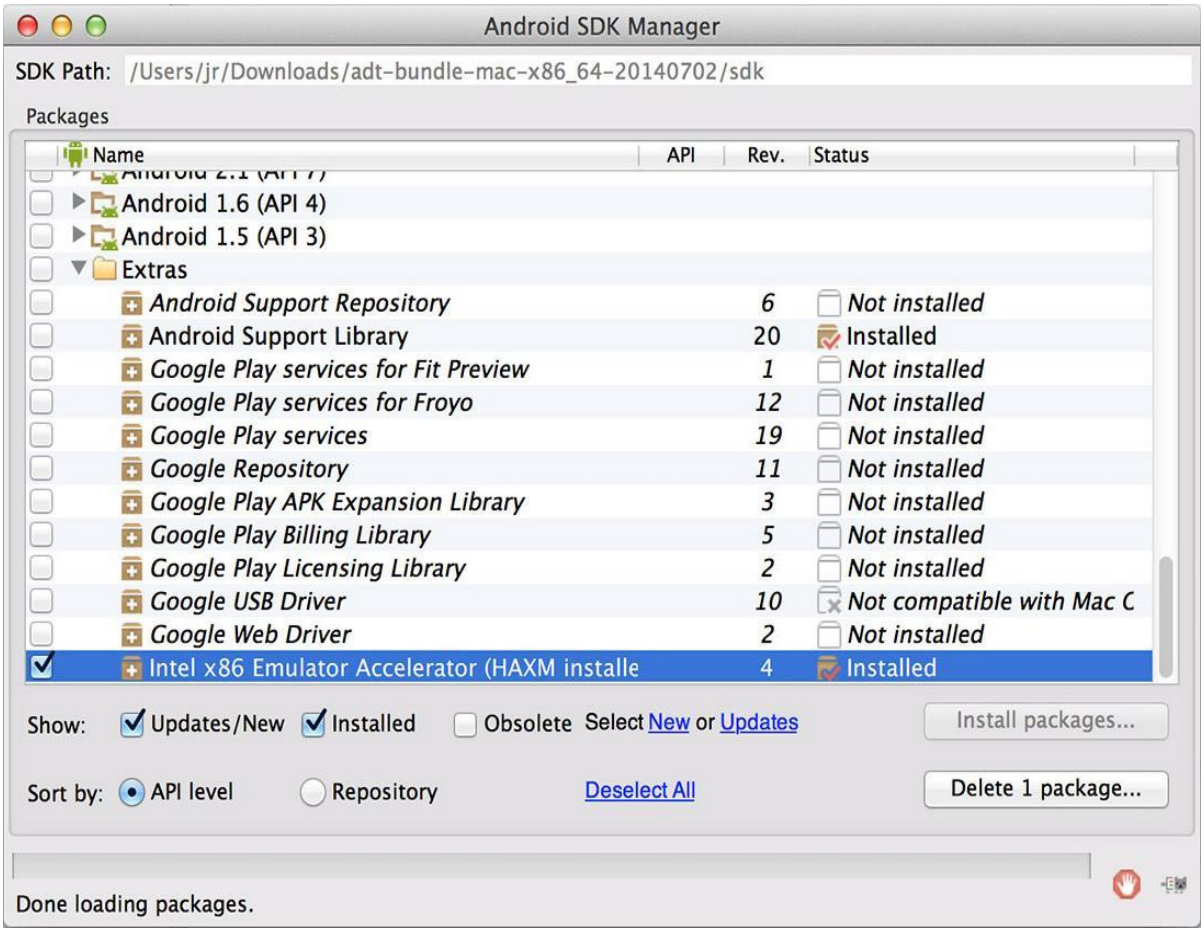


Рис 15.14. Android SDK Manager

Потім необхідно створити Android virtual device (AVD) для тестованого пристрою. У менеджері AVD є список готових пристроїв у "Device Definitions". Для початку потрібно вибрати один з них і натиснути "Create AVD" (рис. 15.15).

Виберіть будь-який CPU і поставте "No skin" і "Use host GPU". Тепер можна запускати віртуальний пристрій і використовувати браузер Android для тестування (рис. 15.16).

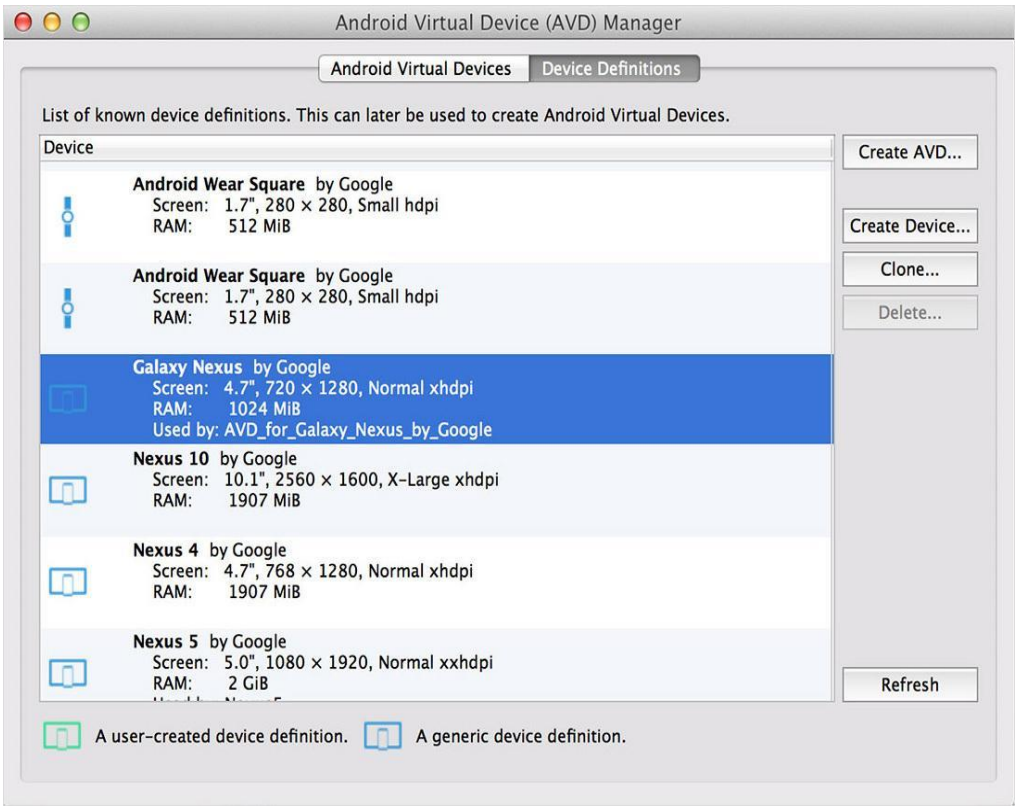


Рис 15.15. Android Virtual Device

Рис 15.16. Віртуальний пристрій

Не завадить підучити клавіатурні команди для більш зручної взаємодії з емулятором (рис.15.17).

Emulated Device Key	Keyboard Key
Home	HOME
Menu (left softkey)	F2 *or* Page-up button
Star (right softkey)	Shift-F2 *or* Page Down
Back	ESC
Call/dial button	F3
Hangup/end call button	F4
Search	F5
Power button	F7
Audio volume up button	KEYPAD_PLUS, Ctrl-F5
Audio volume down button	KEYPAD_MINUS, Ctrl-F6
Camera button	Ctrl-KEYPAD_5, Ctrl-F3
Switch to previous layout orientation (for example, portrait, landscape)	KEYPAD_7, Ctrl-F11
Switch to next layout orientation (for example, portrait, landscape)	KEYPAD_9, Ctrl-F12
Toggle cell networking on/off	F8
Toggle code profiling	F9 (only with `-trace` startup option)
Toggle fullscreen mode	Alt-Enter
Toggle trackball mode	F6
Enter trackball mode temporarily (while key is pressed)	Delete
DPad left/up/right/down	KEYPAD_4/8/6/2
DPad center click	KEYPAD_5
Onion alpha increase/decrease	KEYPAD_MULTIPLY(*) / KEYPAD_DIVIDE(/)

Рис 15.17. Клавіатурні команди

15.4 Встановлення та налаштування мобільних веб-браузерів

Процес установки веб-браузерів на мобільні пристрої не відрізняється від установки будь-яких інших додатків.

Як приклад установки браузерів на ОС Android розглянемо варіант установки браузера Google Chrome.

Кроки для установки браузера наступні:

1) Запустити Play Маркет та знайти в ньому Chrome для Android.
2) Натиснути **Встановити**.
3) Натиснути **Скачати**.

Також як приклад установки браузерів на ОС iOS розглянемо процес установки браузера Google Chrome.

Кроки для установки браузера наступні:

1) Відкрити на пристрої магазин додатків Apple App Store.
2) Натиснути **Безкоштовно**.
3) Натиснути **Встановити**.
4) Ввести пароль Apple ID та нанатиснути **ОК**.

Висновки: розглянуто процес встановлення тестової лабораторії за допомогою комбінації з реальних пристроїв, емуляторів, симуляторів та інструментів віддаленого тестування. З їх допомогою можна тестувати сайти та додатки на різних мобільних пристроях.

Також ознайомилися з різними інструментами віддаленого налагодження, які відкривають доступ до налагоджувальних даних мобільних пристроїв, без яких налагодження дуже ускладнюється.

ТЕМА 16

ІНСТРУМЕНТИ

Тестування IOS-, Android-, Windows phone-додатків

16 ІНСТРУМЕНТИ ТЕСТУВАННЯ IOS-, ANDROID-, WINDOWS PHONE-ДОДАТКІВ

16.1 *iOS*

16.1.1 Xcode (for MAC)

Xcode – інтегроване середовище розробки програмного забезпечення під OS X і iOS, розроблена корпорацією Apple.

Має широкий функціонал:

- Встановлення програм
- Видалення програм
- Зняття креш-логів
- Замір витрати пам'яті
- Завантаження скріншотів / фотографій
- Перегляд логів в real-time з пристрою

16.1.1.1 Інструкція по установці *Xcode*

1. Відкрити AppStore і знайти Xcode. *(Для установки необхідно буде зареєструватися і отримати Apple ID).*

2. Натиснути кнопку **Free**.
3. Натиснути кнопку **Install App**.
4. Після інсталяції – запускаємо програму і погоджуємося.

5. Програма запитує пароль від Операційної системи і встановлюється остаточно.

6. Програма встановлена.

16.1.1.2 Підготовка iOS пристрою до установки самопідписаних додатків

Підключення та налаштування iOS-пристрою до Xcode:

1. Запустити Xcode.
2. Підключити iOS-пристрій до USB.
3. Відкрити вікно Devices (Window \ Devices).

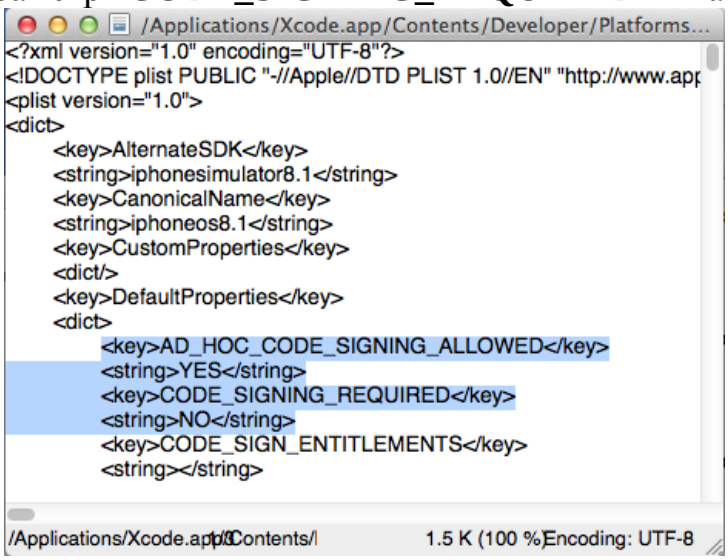
4. Побачити наш підключений пристрій.

Налаштування Xcode для використання самопідписаного сертифіката:

1. Закрити Xcode, якщо він відкритий.
2. Відкрити Terminal

```
cd /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS.sdk
# create copy of SDKSettings.plist
sudo cp -p SDKSettings.plist SDKSettings.plist.orig
# convert to editable xml format
sudo plutil -convert xml1 SDKSettings.plist
```

3. Відкрити **SDKSettings.plist** текстовим редактором і замінити значення параметра "**AD_HOC_CODE_SIGNING_ALLOWED**" на "**YES**", а параметр "**CODE_SIGNING_REQUIRED**" – на "**NO**".



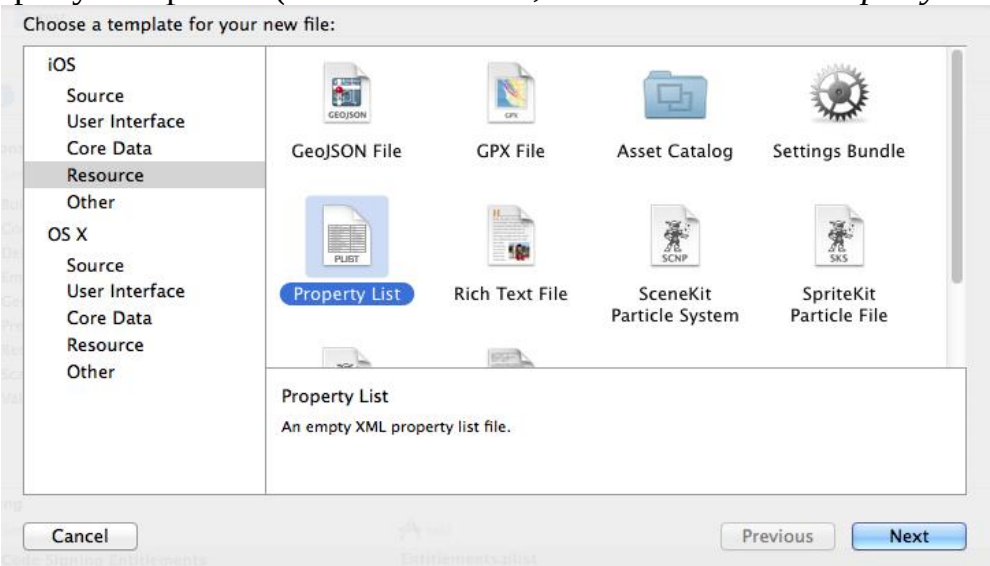
4. Відкрити Xcode
5. Відкрити або створити проект і зайти в налаштування проекту, у закладку "**Build settings**". Встановити в полі "**Code Signing Identity**" значення "**Ad Hoc Code Sign**". Встановити значення потрібно і у "**Target**", і у "**Project**".

6. Біля кнопки «Run» обрати потрібний підключений iOS-пристрій.
7. Натиснути «Run», і Xcode транслює вихідні коди у виконуваний файл і завантажує його у пристрій.
8. Далі Xcode показує нам наступну помилку:

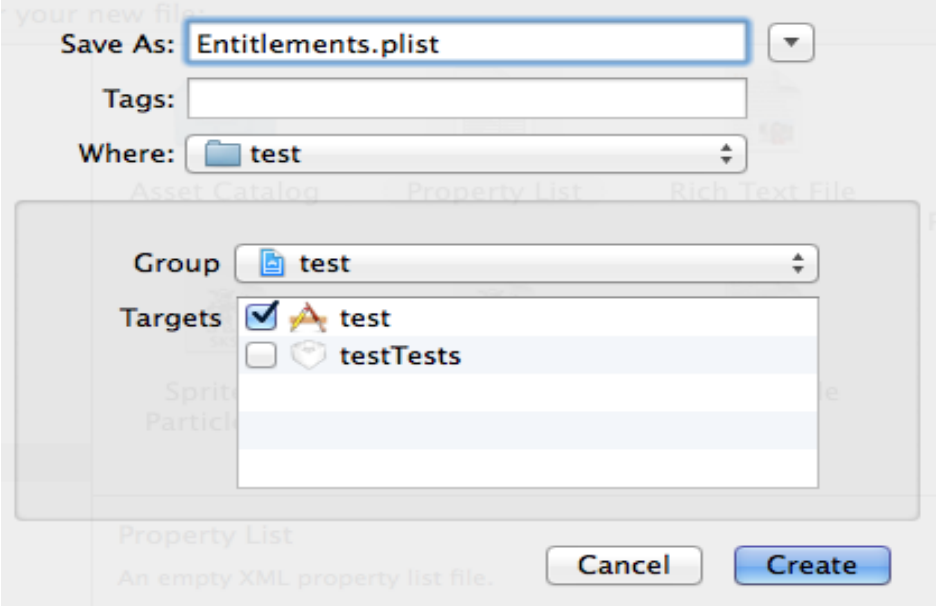
Так і повинно бути, тому що відладчик ще не налаштований, але сам додаток можна вже запускати на пристрої.

Налаштування Xcode для налагодження програми на цільовому пристрої:

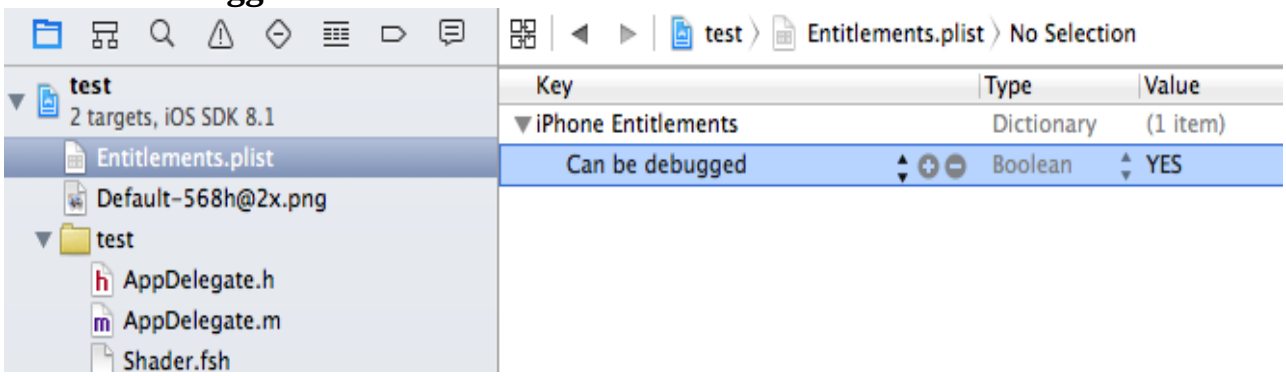
1. При відкритому проекті, який тестується, створюємо новий PropertyList файл – *(File \ New \ File, iOS \ Resource - Property List).*



2. Вказати ім'я створюваного файлу як **Entitlements.plist**



3. Відкрити щойно створений файл і додати туди параметр "**Can be debugged**" зі значенням "**YES**".



4. Зайти в налаштування проекту, у закладку "**Build settings**". У поле "**Code Signing Entitlements**" встановити значення "**Entitlements.plist**". Встановити значення потрібно і у "**Target**", і у "**Project**".

Code Signing Entitlements	Entitlements.plist
Debug	Entitlements.plist
Release	Entitlements.plist

5. Зібрати додаток і запустити на цільовому iOS-пристрої.

16.1.1.3 Заміри пам'яті

Заміри пам'яті проводяться для зменшення ймовірності падіння додатку через недостачу оперативної пам'яті мобільного пристрою; у процес розробки ігор введений процес виміру пам'яті на iOS-додатках. Замір споживаної додатком пам'яті здійснюється тестувальником проекту на прохання продюсера, програміста, дизайнера або в ході перевірки чек-листа.

Замір здійснюється тестувальником, який вже знайомий з проектом, за відсутності такого – опорним тестувальником проекту.

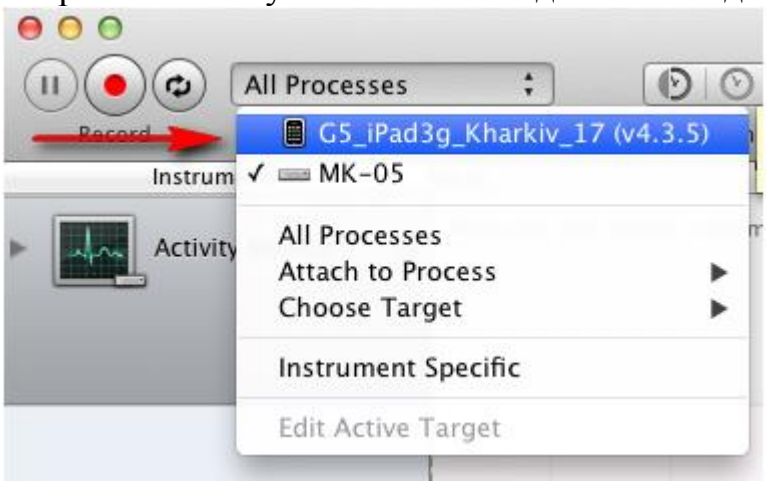
Заміри здійснюються таким чином:

1. Встановити на девайс додаток.
2. Обов'язково перевірити системну мову девайса та мову запуску гри – вона повинна бути **англійською**.
3. Переконатися, що в наявності є комп'ютер під управлінням Mac OSX з відносно новою ОС і зі встановленим додатком Xcode нової версії.
4. Переконатися, що шнур для підключення девайса до PC робочий і не розвалюється.

5. Підключити девайс через шнур до Mac.
6. На Mac запустити додаток Instruments *(він може перебувати вже в Dock Bar або на робочому столі, в іншому випадку – відкрити Finder і через пошук знайти додаток Instruments).*

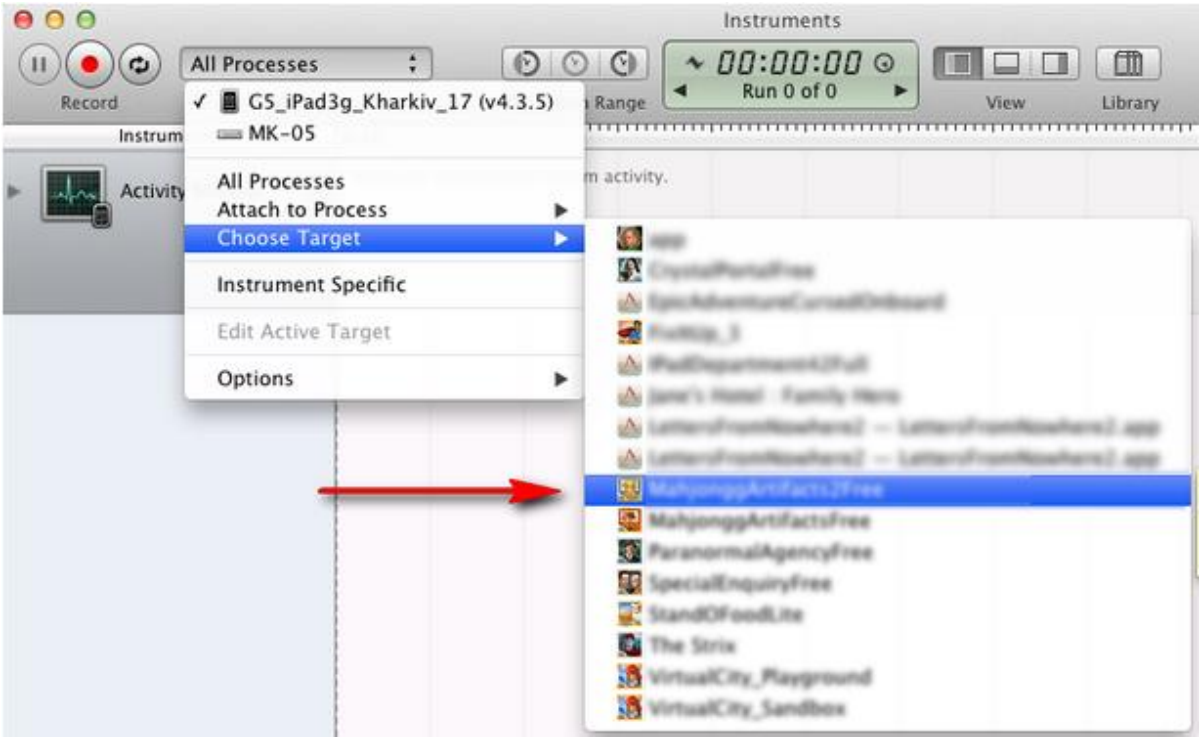
7. У Instruments вибирати пункт Activity Monitor.

8. У вікні вибирати зі списку All Processes підключений девайс.



9. Знову викликати даний список, який з’являється, і навести курсор на Choose Target, щоб відкрити список програм на девайсі.

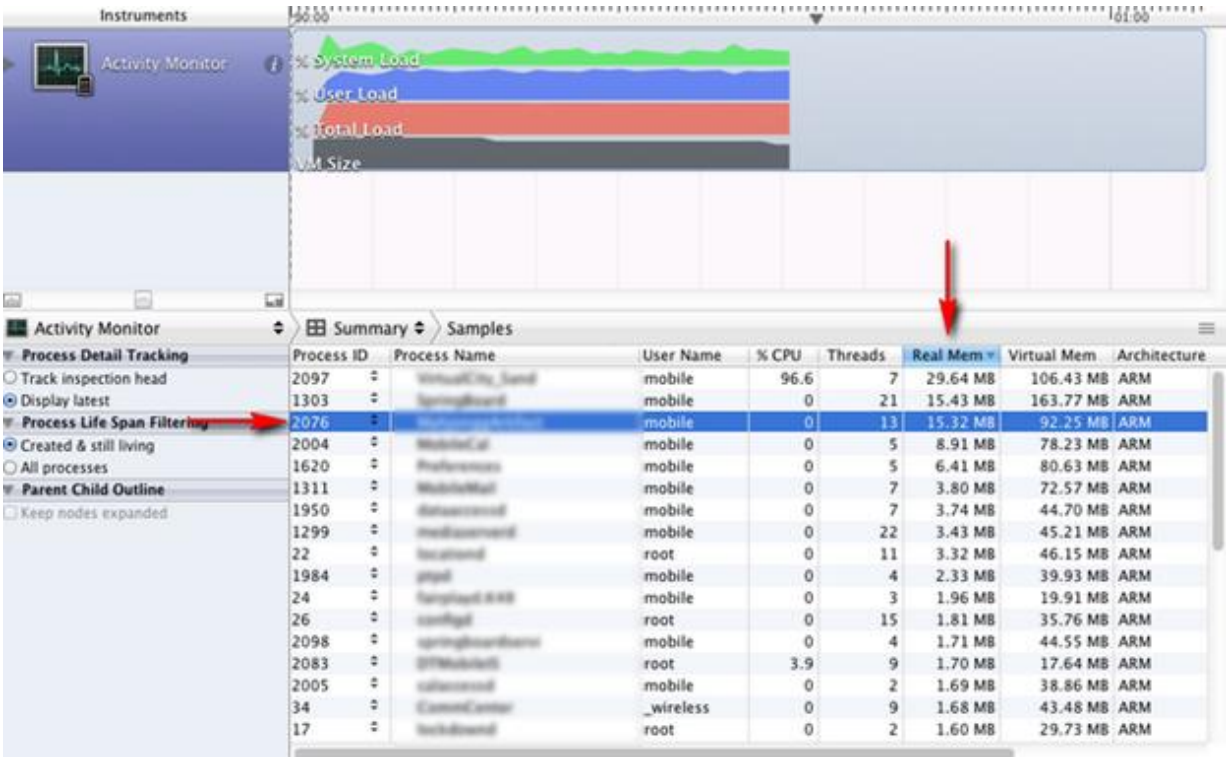
10. Вибрати зі списку додаток, для якого буде замірюватись пам'ять



Важливо: перед початком виміру переконайтеся, що в девайсі виставлена англійська мова системи, а в налаштуваннях гри стоїть вибір мови AUTO.

11. Натиснути кнопку запису

12. Бачимо, що на девайсі запустилася гра, а в Instruments з'явилися дані.
13. Знаходимо нашу гру в Instruments. Дані по пам'яті дивимося у колонці Real Mem.



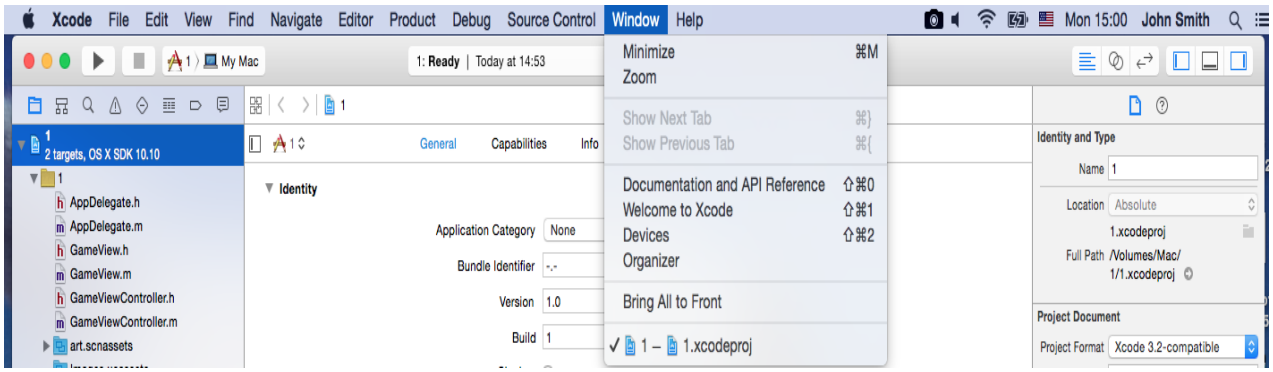
14. Проходимо лінійно гру (бажано з мінімальним використанням читів), записуючи в таблицю споживану пам'ять програми.

Важливо: при проходженні гри – крос-промо та вікно Facebook не відкривати, тому що при їх відкритті показники по пам'яті збільшуються, і подальші виміри є некоректними.

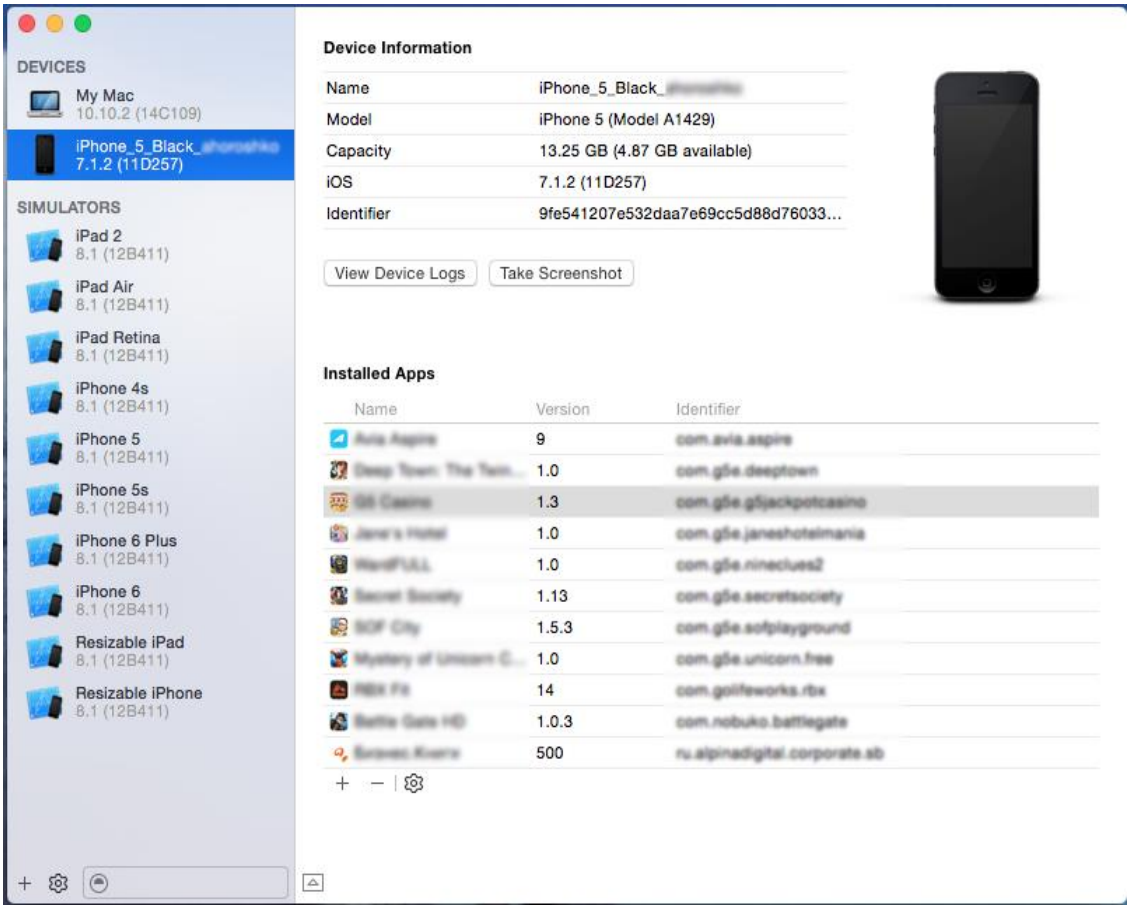
Важливо: при замірах пам'яті з гри не виходити, тому що при наступному запуску гри показники пам'яті нижче, ніж при першому запуску, отже, вони не підходять.

16.1.1.4 Перегляд crash логів в XCode

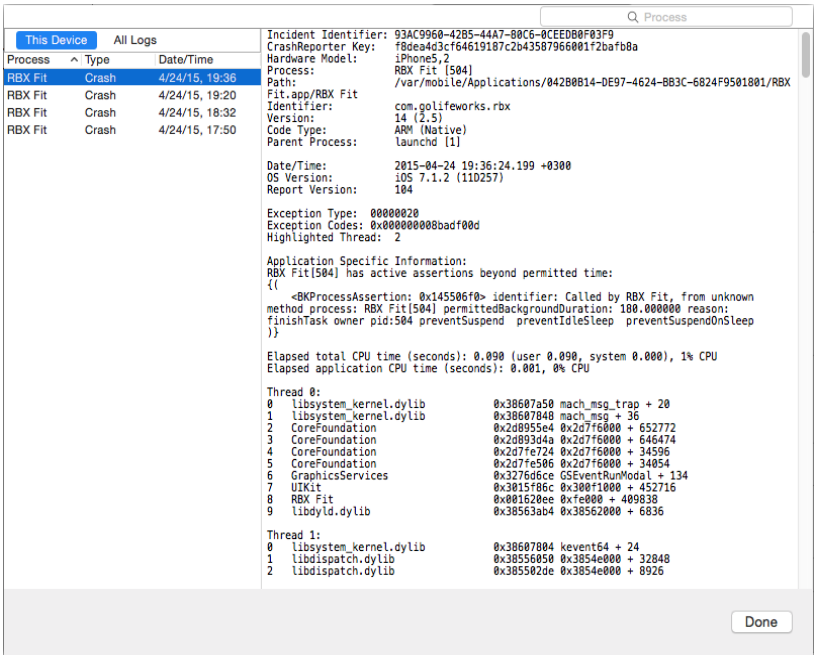
1. Запускаємо XCode
2. Вибираємо вкладку Window і натискаємо Devices *(CMD + Shift + 2)*



1. Вибираємо підключений девайс



2. Натискаємо кнопку "View Device Logs", відкриється вікно з crash-логами

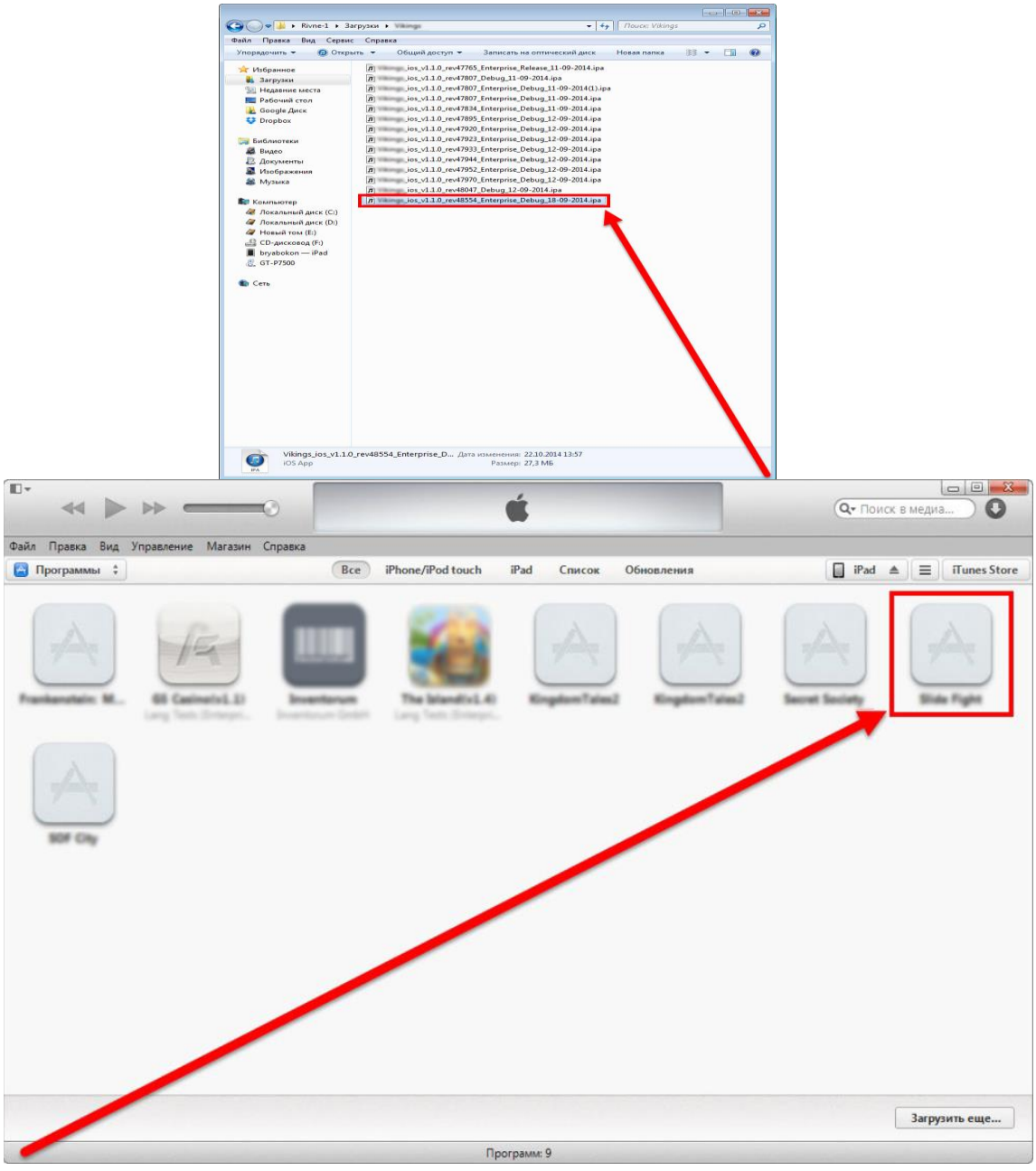


3. Вибираємо потрібний crash-лог для перегляду.

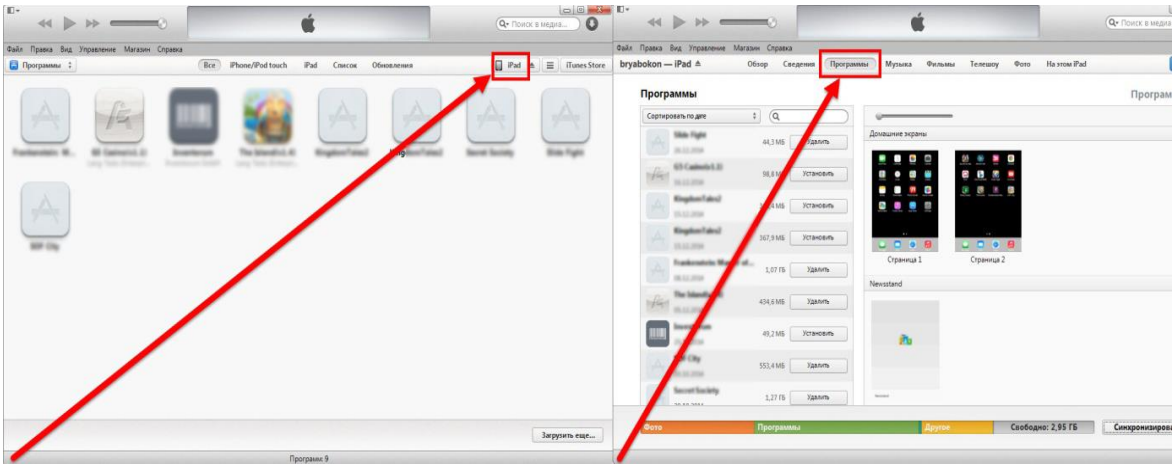
16.2 iTunes

16.2.1 Установка .ipa-додатків

1. Підключити девайс до ПК.
2. Автоматично відкриється iTunes.
3. Двічі натиснути на .ipa-файл, і він додається в iTunes.



4. У iTunes зайти в потрібний девайс і вибрати вкладку "Програми".

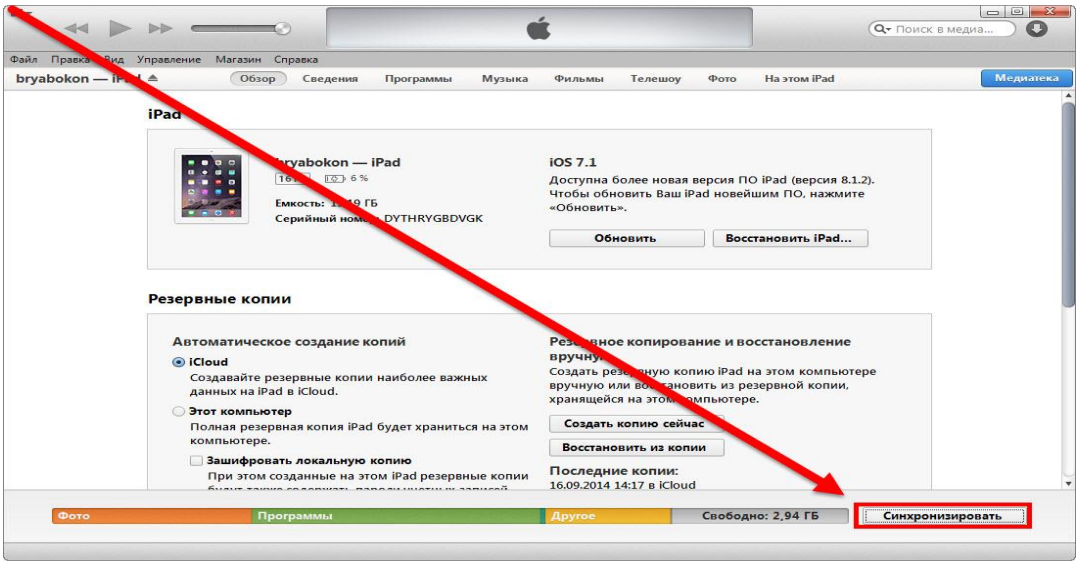


5. У вкладці "Програми" вибрати потрібний додаток і натиснути кнопку "Встановити".
6. Натиснути кнопку "Застосувати", і почнеться установка програми.
7. Після установки програми його можна запускати на девайсі.

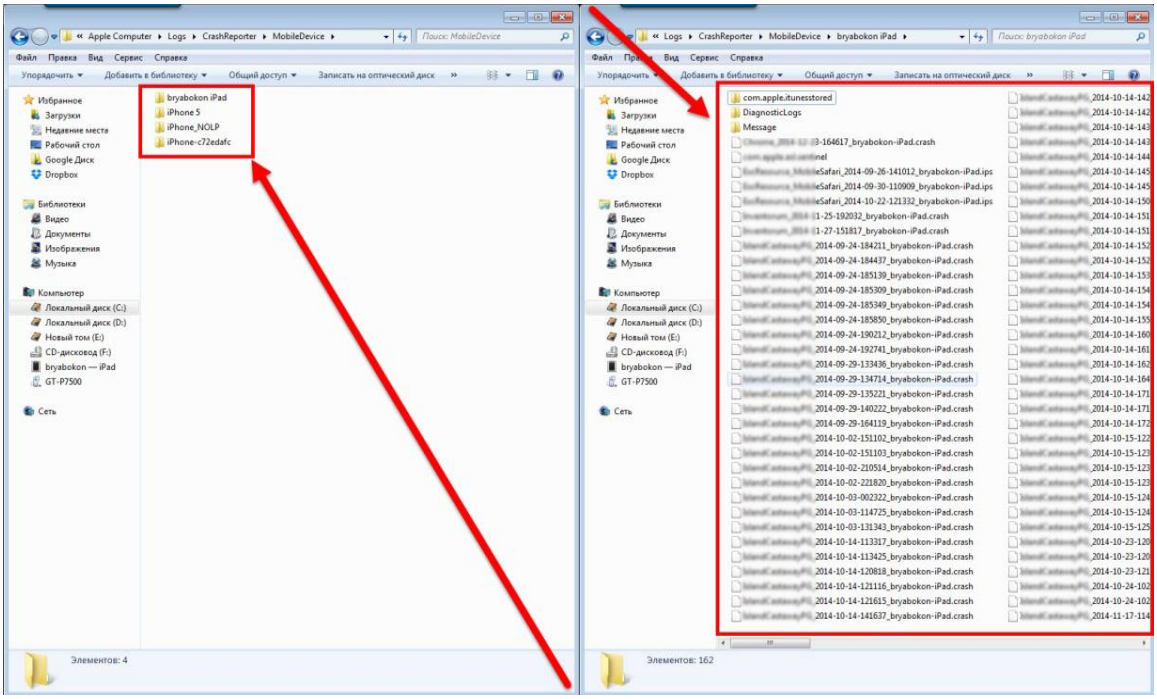
16.2.2 Копіювання crash-логів з iOS на Windows 7, XP

1. Підключити девайс до ПК.
2. Автоматично відкриється iTunes.
3. У iTunes зайти в потрібний девайс.

4. Синхронізувати наш пристрій з ПК, для цього нам потрібно натиснути кнопку "Синхронізувати".



5. Чекаємо закінчення синхронізації.
6. Тепер все. Список з пристрою записаний в папку:
   *c: \ Users \ [ім'я користувача] \ AppData \ Roaming \ AppleComputer \ Log \ CrashReporter *
   Де [ім'я користувача] – ім'я поточного користувача системи.
7. Далі вибрати папку з назвою потрібного девайса і скопіювати потрібний лог. В імені кожного з логів міститься час створення.



16.3 AirServer

Програма дозволяє передавати зображення з iPhone 4S / 5, iPad 2/3/4 / mini, iPod touch 5 на PC або Mac за допомогою функції AirPlay Mirroring *(Відеоповтор)*.

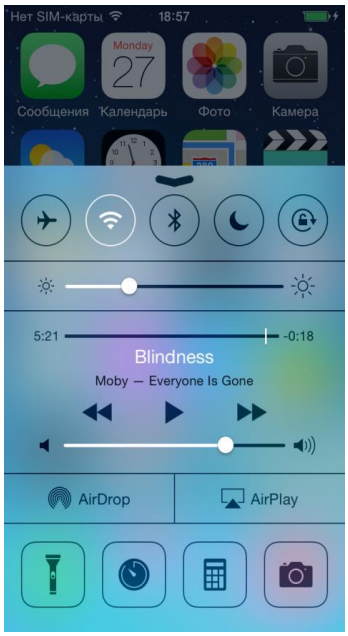
Так само дозволяє знімати відео з екрану iOS пристроїв через AirPlay.

Вивід і запис відео з екрану iPhone або iPad на OS X або Windows

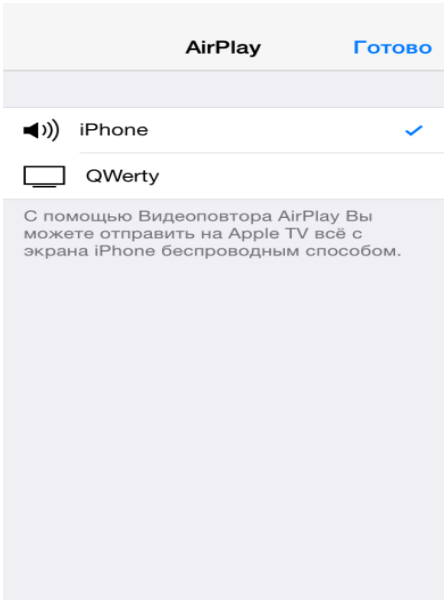
Якщо Ви використовуєте iOS 7, підключити AirServer з вашого iOS пристрою можна наступним чином:

1. Підключіть iOS-пристрій до комп'ютера з працюючим AirServer. Пристрої повинні бути в одній і тій же мережі WiFi.

2. Проведіть знизу вгору по екрану, щоб потрапити в Пункт Управління.

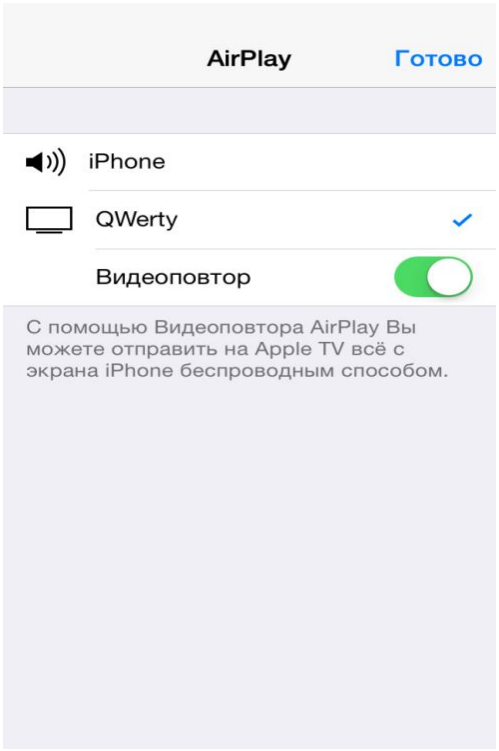


3. Натисніть іконку AirPlay. Ви побачите список приймачів у Вашій мережі, сумісних з AirPlay.

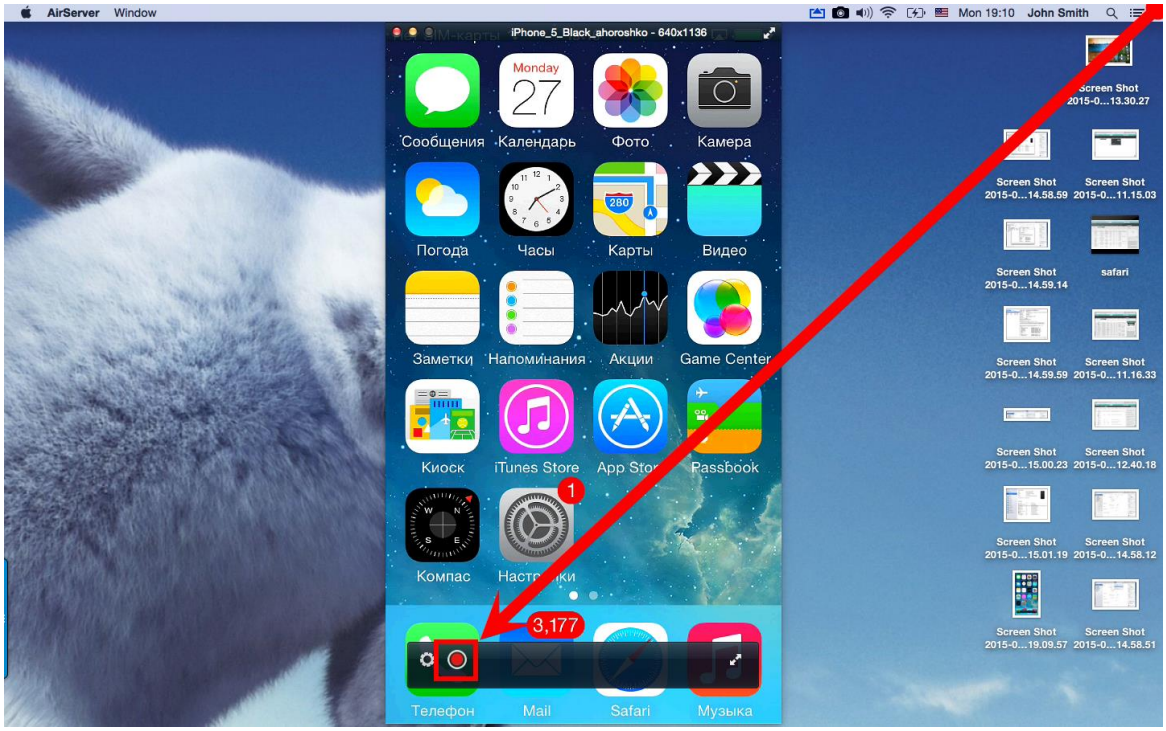


4. Натисніть на назву приймача AirPlay, на який потрібно почати передачу. Це буде назва комп'ютера.

5. Щоб почати транслювання, перемістіть покажчик у положення Вкл.



6. Для початку запису потрібно натиснути кнопку "Record" на комп'ютері.



Пристрої iOS 6 і старше можна підключити до AirServer наступним чином:

1. Увімкніть iOS-пристрій і натисніть двічі кнопку Home. Унизу екрану з'явиться sliding-меню.

2. Прокрутіть вліво, поки не з'явиться кнопка AirPlay. Натисніть іконку, щоб побачити список пристроїв.

3. У списку з'явиться комп'ютер, на який було встановлено AirPlay. Щоб підключити, просто натисніть на його назву.

4. Якщо Ваш пристрій підтримує транслювання, перемістіть покажчик у положення Вкл.

5. Якщо Ви включили додаток, який підтримує AirPlay, наприклад, YouTube, з'явиться ікона AirPlay. Просто натисніть на неї, щоб побачити доступні пристрої. У списку з'явиться комп'ютер, на який було встановлено AirPlay. Щоб підключити, натисніть на його назву.

16.4 *Safari Web Inspector*

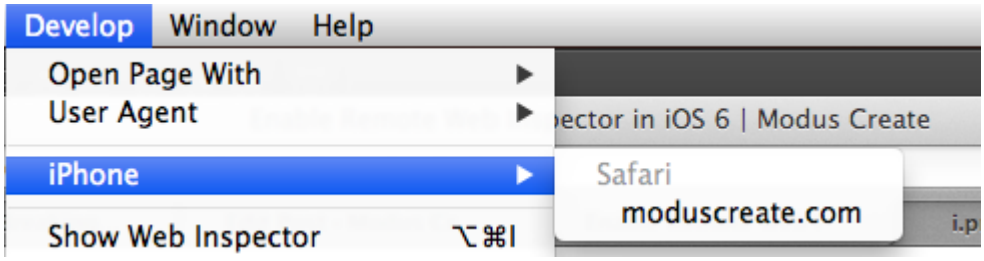
Почавши працювати з Inspector, ви можете:

- ✓ Бачити і вносити зміни в свій HTML і CSS
- ✓ Отримувати доступ до сховищ і переглядати куки, локальну, а також сесійний і SQL-бази даних
- ✓ Здійснювати профільну перевірку продуктивності свого веб-додатку, включаючи тестування мережевих запитів, а також перевірку шарів, візуалізації і JavaScript-подій. Це безумовно великий крок у розвитку інструментів для оптимізації продуктивності
- ✓ Проводити пошук по DOM
- ✓ Одночасно бачити всі попередження і повідомлення про помилки
- ✓ Управляти Web Workers (потоками)
- ✓ Управляти зупинками виконання JavaScript і Uncaught, а також задавати винятки для них.
- ✓ Отримувати доступ до консолі і виконувати JavaScript
- ✓ Займатися налагодженням свого JavaScript-коду
- ✓ Перевіряти на дотик: за допомогою маленької іконки у вигляді долоні ви можете перевіряти механіку сенсорного введення на пристрої і налагоджувати DOM-елементи.

1. Підключити iOS-пристрій до Mac
2. Запустити Safari на iPad / iPhone
3. Перейти в налаштування Settings> Safari> Advanced.

4. Включити Web Inspector.
5. Запустити Safari на Mac.

6. Відкрити налаштування браузера> Preferences> Advanced.
7. Включити налаштування Show Develop menu in menu bar.
8. Вибрати Develop menu.
9. Знайти у списку підключене iOS-пристрій, потім вибрати веб-додаток, який тестується.



16.5 iTools

Установка iTools і можливі проблеми.

Встановлювати iTools не потрібно – необхідно скачати файл програми (в архіві Zip, він усього кілька мегабайт) з офіційного сайту програми. Витягнути його. При запуску файлу можлива ось така помилка:

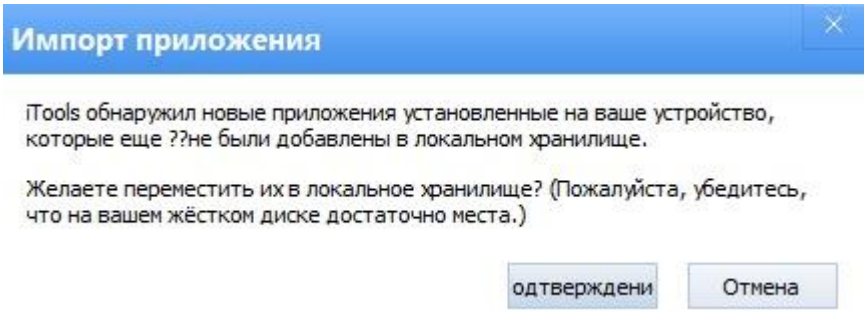
Вирішується все просто. Кладемо файл з програмою в папку:
C: \ Program Files \ Common Files \ Apple \ Apple Application Support або C: \ Program Files (x86) \ Common Files \ Apple \ Apple Application Support і звідти вже запускаємо...

Важливо! iTools вимагає наявності iTunes на комп'ютері. Наявність iTools не скасовує того факту, що iTunes повинен бути хоча б встановлений і спочатку налаштований. Запускати iTunes для роботи iTools не потрібно.

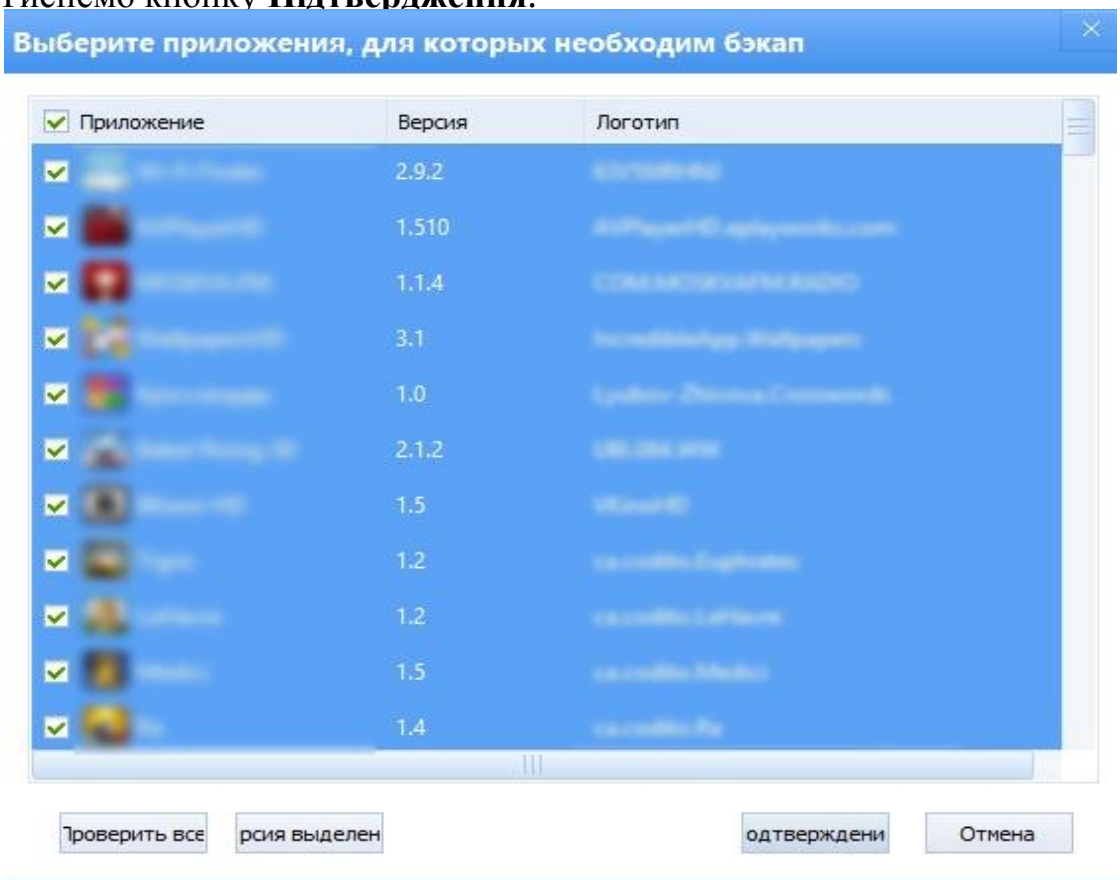
Додатки

Це медіатека iTools. Навіть без підключеного iPad / iPhone видно додатки, які ми маємо в даний момент на комп'ютері.

Якщо ж ми підключаємо iPad / iPhone, то вибір дій збільшується. Вимагає окремого пояснення «кнопка резервного копіювання». При її натисканні з'являється таке вікно.



Тиснемо кнопку **Підтвердження**:



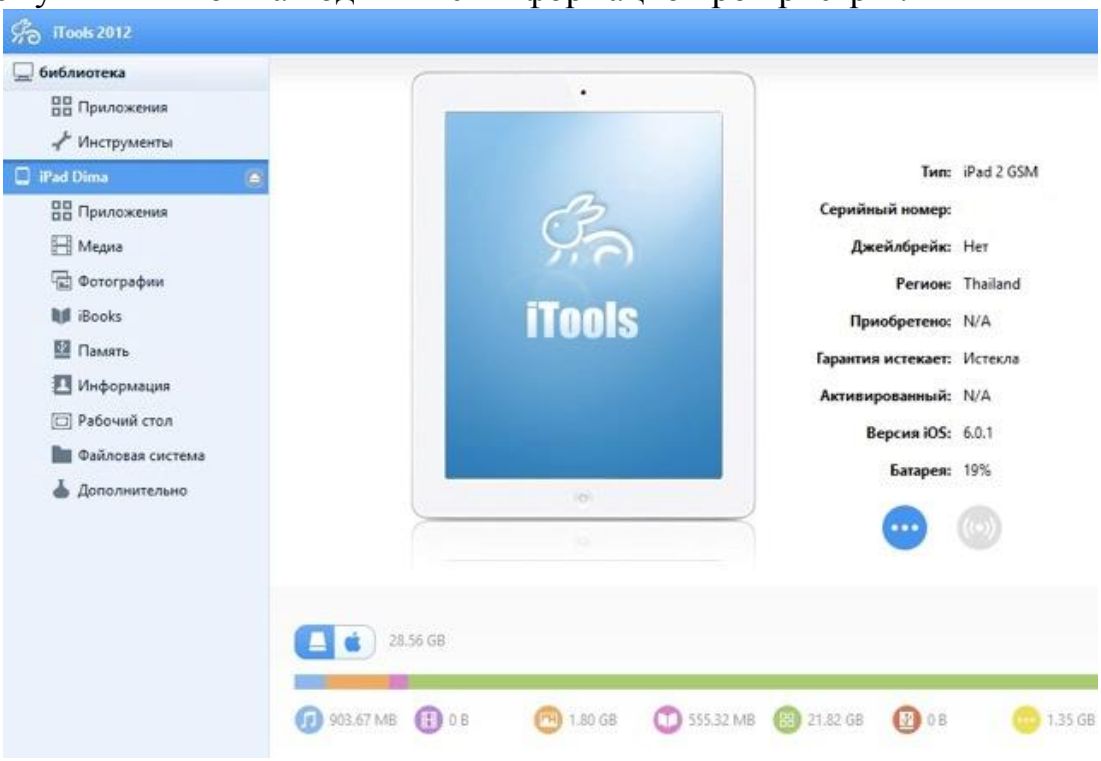
Це, як ви вже зрозуміли по скріншотам, бекап додатків з iPad / iPhone на комп'ютер. Далі ми можемо відстежувати, які програми вимагають резервного копіювання або які можуть бути оновлені. Мова про зв'язок з App Store не йде. Оновлювати додатки в iTools безпосередньо з App Store (як це можна робити в iTunes) не можна.

Інструменти

Тут знаходяться додаткові можливості iTools.

- ***Асоціювання з ipa*** – файли з розширенням ipa (додатки з App Store) будуть за замовчуванням додаватися в медіатеку в iTools. Якщо функція вимкнена, то ipa асоціюється за замовчуванням з iTunes.
- ***Резервне копіювання*** – те, що в iTunes робиться за замовчуванням автоматично, в iTools робиться за запитом користувача. Робота з резервними копіями.
- ***Автоматична резервна копія*** – ця функція дозволяє відключити / включити автоматичне резервне копіювання в iTunes.
- ***PXL to IPA*** – конвертація файлу PXL в ipa, так як ipa-файл безпечніший.
- ***Зробити звуковий сигнал*** – зручний інструментарій створення рінгтона для iPhone.
- ***iTools і iPad***

Підключаємо iPad до комп'ютера, і iTools відразу визначає його. У першому ж вікні можна подивитися інформацію про пристрій.



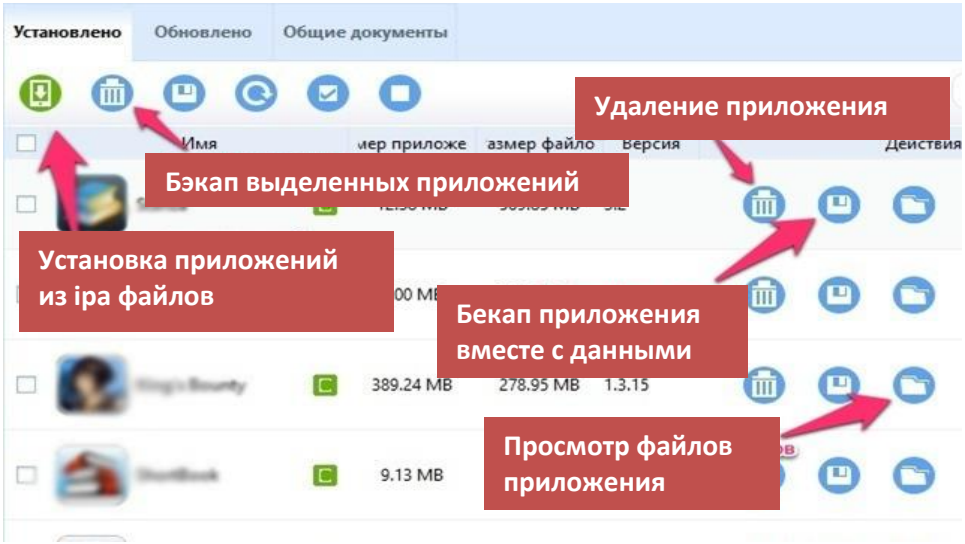
Під параметрами бачимо дві кнопки:

- При натисканні кнопки з трикрапкою з'являється ще більш детальна інформація у вигляді списку. Зокрема, UDID пристрою знаходиться саме там.
- Програма підтримує Wi-Fi-синхронізацію – для цього потрібна друга кнопка.

Для iTools не вимагається Jailbreak, програма підтримує як зламані, так і незламані пристрої. Функціонал для зламаних і незламаних пристроїв не дуже розрізняється.

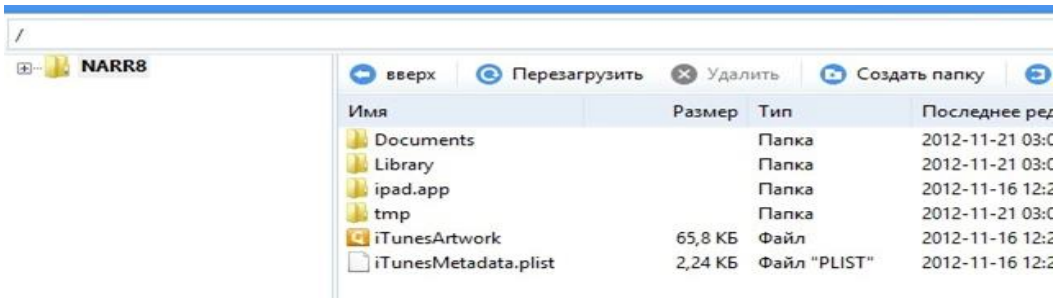
- ✓ ***Applications – Додатки***
- ✓ ***Вкладка Встановлено***

Управління додатками, які встановлені на iPad.

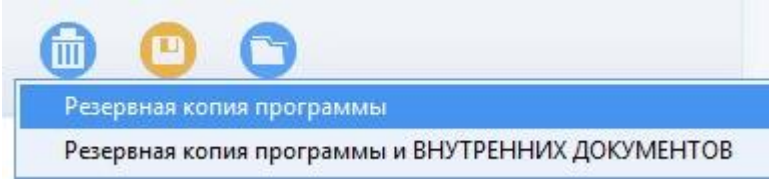


Найпростіше – це скачування програми з ipa-файлів. Якщо у вас джейлбрейкнутий пристрій, то можна скачувати будь-які ipa-файли. Якщо iPad не зламати, то ви можете скачувати тільки ipa-файли, куплені під вашим обліковим записом.

При натисканні «перегляд файлів додатки» ми побачимо файлову структуру програми.



Перенесення програми з усіма даними з одного пристрою на інший виглядає так. Підключаємо iPad / iPhone та натискаємо кнопку з дискетою навпроти потрібного додатку.



Вибираємо «Резервна копія програми і ВНУТРІШНІХ ДОКУМЕНТІВ». Зберігаємо на жорсткий диск.

Для збереження програми на жорсткий диск відбувається архівування додатку, тому на iPad / iPhone має бути вільне місце для цієї операції, рівне приблизно розміру гри (врахуйте це).

Підключаємо інший iPad / iPhone і закачуємо туди цей файл.

- ***Media – медіатека***
  Тут міститься вся медіатека нашого iPad / iPhone.

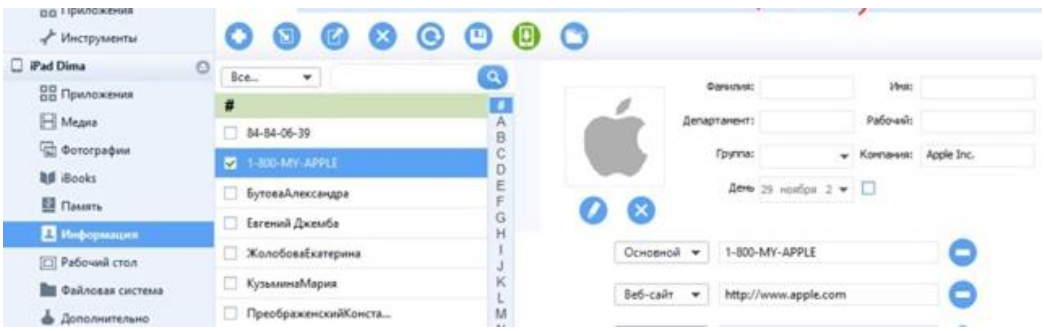
- ***Photos – Фотографії***
  У iTools є окрема вкладка для фотографій. У ній відображаються скріншоти, фотографії зроблені самим iPad та альбоми, скачені в самому iTools.

- ***iBooks – синхронізація книг***
  Імпорт / експорт книг в iTools також простий, як і фотографій або музики. Формати розуміються різні – epub і PDF. Відразу після нехитрої операції імпорту можемо відкрити iBooks, і книги там вже будуть.

- ***Storage – Пам'ять (Сховище)***
  Пам'ять – це можливість використовувати iPad / iPhone як флешку. Також легко можна видалити файли.

- ***Information – Інформація***
  Управління контактами, повідомленнями з iMessage, Замітками, Закладками, Календарем та історіями дзвінків *(ця функція для iPhone).*

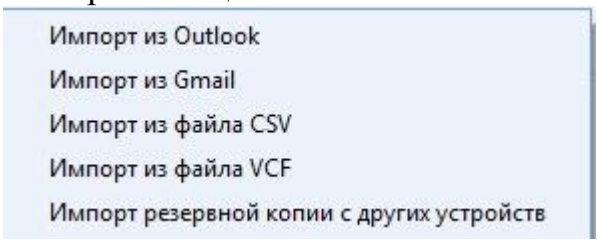
Дані беруться з медіатеки ...



Функціонал усіх вкладок (Контакти, Повідомлення, Замітки, Закладки, Календар) приблизно однаковий. Тому подивимося можливості на прикладі Контактів.

Користувач може:

- Додавати і редагувати контакт. Видалити його.
- Імпортувати та експортувати контакти. Причому розробник iTools надав різні способи зробити це.



- Створити резервну копію контактів і відновити її з резервної копії.

✓ ***Desktop – Робочий стіл***

Ще один корисний розділ, що складається з двох вкладок. Перша вкладка **Іконки** – тут ми можемо керувати іконками робочих столів iPad у реальному часі. Створювати папки, перетягувати їх з екрану на екран, перейменовувати папки. Можна зробити бекап розташування іконок (кнопка Резервна копія), а потім відновити в разі чого (кнопка Імпорт).

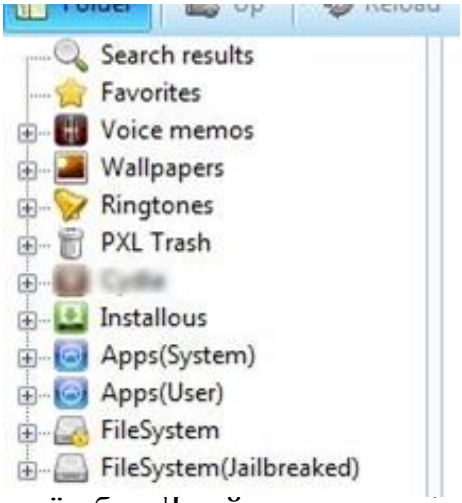
✓ Також є кнопка розумної класифікації – ***Smart classify.***

✓ Вкладка ***Live Desktop:*** ви бачите у прямому ефірі, що діється на екрані iPad. І навіть можете зробити скріншот або записати відео. Спробував записати на iPad 2, відео пишеться ривками і не особливо якісно... Та й саме зображення в прямому ефірі передається низької якості.

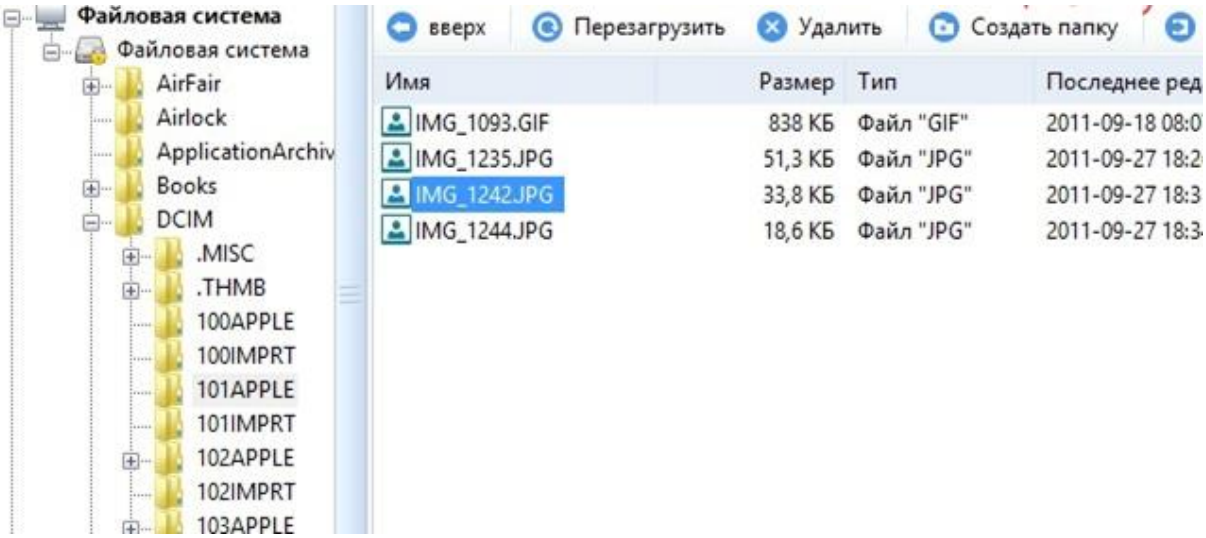
✓ ***FileSystem – Файлова система***

Тут доступ до файлової системи iPad. Розділ для досвідчених користувачів.

У користувачів зламаних пристроїв буде доступна абсолютно вся файлова система.

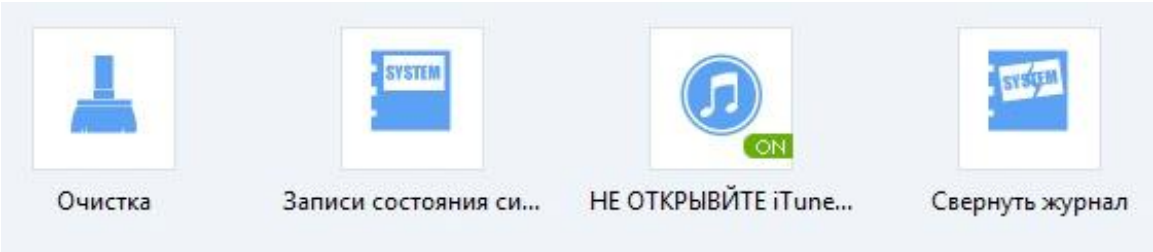


Але навіть для пристроїв без Джейла можна дістати цілком відчутну та цінну інформацію з доступної частини системи. Наприклад, з легкістю можна скопіювати фотографії з фотопотоку.



✓ ***Advanced – Додатково***

Додатковий функціонал, який розробники iTools не знали, куди включити.



✓ ***Очищення*** – звільнення iPad від всякого непотрібного мотлоху. Видаляє, наприклад, кеш-мініатюр.

✓ ***Записи стану системи*** – перегляд логів у реальному часі.

НЕ ВІДКРИВАЙТЕ iTunes – зручний перемикач. Переведіть його у позицію OFF, і iTunes автоматично не буде відкриватися при приєднанні iPad до комп'ютера.

- ✓ ***Журнал*** – тут, мабуть, неправильно перекладено назву. Це різноманітні логи системи.

16.6 Android

16.6.1 Програма ADB. Встановлення та налаштування

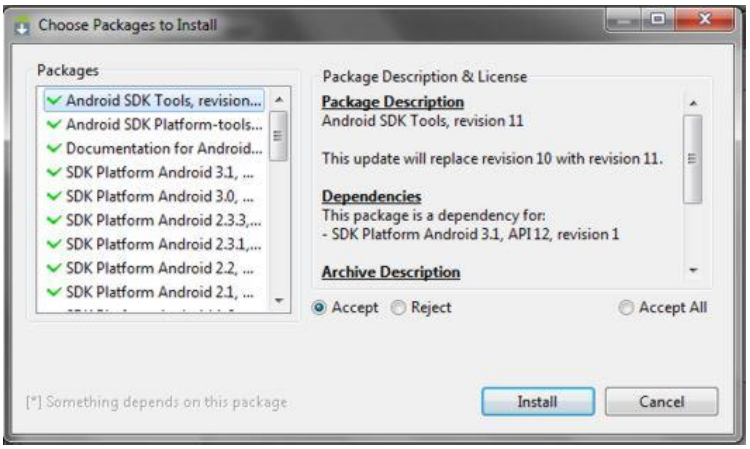
Абревіатура **ADB** розшифровується як **Android Debug Bridge** (відлагоджувальний міст Андроїд). ADB є складовою частиною **Android SDK**.

Так як операційна система **Android** є різновидом Linux, для її налаштування часто виникає необхідність роботи через командний рядок. Звичайно, існують програми – емулятори терміналу, які дозволяють виконувати команди прямо на пристрої але, по-перше, на маленькому екрані телефону робити це незручно, а по-друге, іноді потрібен доступ до пристрою через комп'ютер, і в цих та багатьох інших випадках програма adb просто незамінна. Програма **adb** встановлює зв'язок між пристроєм і комп'ютером і дозволяє на комп'ютері виконувати різні маніпуляції з системою **Android**.

Покрокова інструкція по установці ADB:

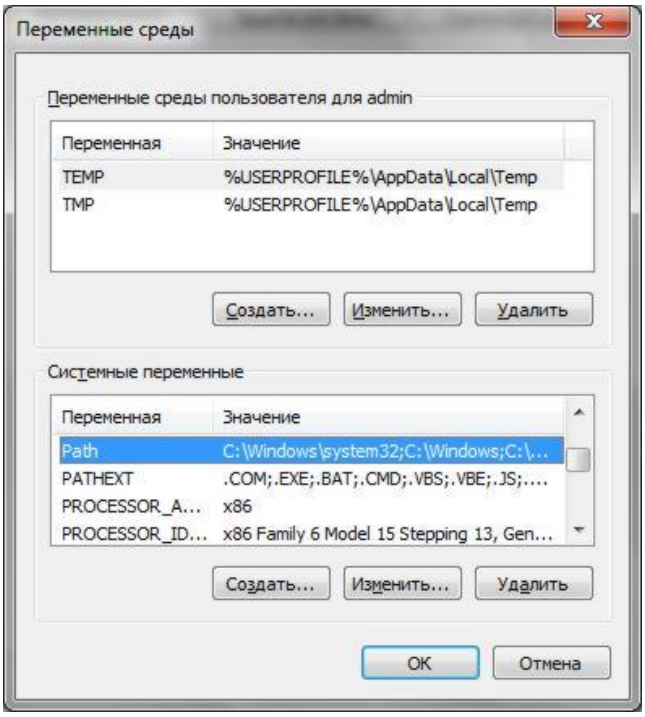
1. Завантажити **Android SDK**, його можна знайти за посиланням http://developer.android.com/sdk/. Є декілька різновидів **SDK** для **Microsoft Windows**, **Mac OS** і **Linux**. Є два варіанти – завантажити установник або zip-архів з Android SDK. Краще всього використовувати zip-архів.

2. Встановити **SDK Platform Tools**. Переконатися, що комп'ютер підключений до Інтернету і запустити програму **SDK Manager**, яка знаходиться в папці **android-sdk-windows**. Після запуску програми з'явиться вікно:



3. Необхідно завантажити і встановити **Android SDK Platform-tools** і **Android SDK Tools**. За допомогою подвійного кліка по пункту або натисканням на «**Accept**», відзначити ці два пункти у списку і зняти відмітку з усіх інших пунктів, як показано на наведеному вище скріншоті. Потім натиснути «**Install**» і дочекатися завантаження та встановлення потрібних компонентів. Установка ADB завершена, але для роботи з мобільним пристроєм Android необхідно встановити драйвера і для подальшої зручності роботи з програмою необхідно прописати шлях до неї та інших компонентів у систему Windows.

4. Необхідно відредагувати системну змінну PATH, щоб кожного разу при запуску програми і введенні команд не набирати шлях до програми, який виглядає так: **C:\android-sdk-windows\platform-tools\ adb**. Якщо ви жодного разу не редагували системні змінні, створіть точку відновлення системи, щоб потім можна було повернути її в первинний стан. На OS **Windows 7**: натискаємо правою клавішею миші ярлик «**Комп'ютер**», вибираємо «**властивості**» і у вікні вибираємо «**Додаткові параметри системи**». На OS **Windows XP**: натискаємо правою кнопкою миші «**Мій комп'ютер**» і потім «**Властивості**». У наступному вікні на вкладці «**Додатково**» натискаємо кнопку «**Змінні середовища**». У списку «**Системні змінні**» вибираємо змінну «**path**» і натискаємо кнопку «**Змінити...**». Відкриється вікно редагування змінної, і в пункті «**значення змінної**» у самий кінець рядка, після крапки з комою додаємо шлях до папки **tools** і папки **platform-tools**:; c:\android-sdk-windows\tools; c:\android-sdk -windows\platform-tools;



(якщо в кінці рядка не було крапки з комою, додайте її – кожен шлях у цьому рядку повинен відокремлюватися від іншого крапкою з комою)

Якщо ви встановили **Android SDK** в іншу папку, пропишіть у кінці рядка ваш шлях до папок **tools** і **platform-tools**.

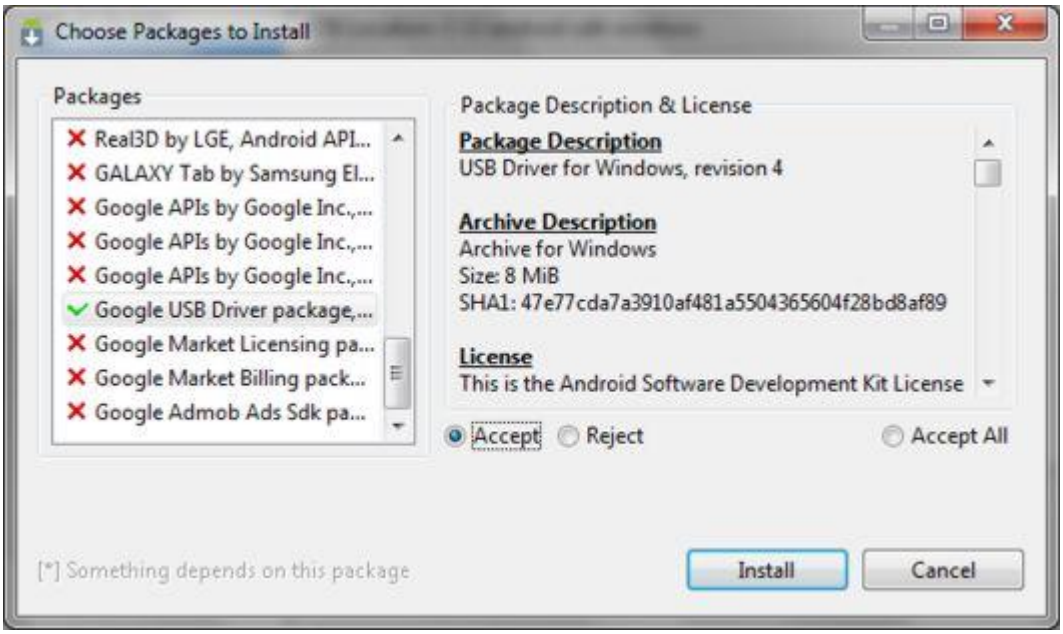
Натискаємо "**ОК**", щоб зберегти зміни. Після цього, щоб зміни вступили в силу, потрібно буде перезавантажити комп'ютер.

16.6.2 Установка драйверів пристрою

Деякі пристрої, наприклад, телефони та планшети компанії **Samsung** мають власне програмне забезпечення для синхронізації з комп'ютером, і якщо воно у вас встановлено на комп'ютері, то драйвер пристрою вже встановлений у вас у системі.

Але для таких пристроїв, як **Nexus One**, які поставляються без будь-яких додаткових програм і драйверів, для роботи з **Android SDK** необхідно встановити драйвера.

Для цього йдемо в папку, в яку ми встановлювали **SDK** і запускаємо **SDK Manager**.



Так само, як були встановлені **Android SDK Platform-tools** і **Android SDK Tools**, знайти і вибрати у списку «**Google Usb Driver package**». Натиснути «**Install**» і дочекатися завершення завантаження драйверів. Драйвери для 32 і 64 розрядної **Windows** будуть завантажені в наступну папку: **\android-sdk-windows\extras\google\usb_driver**.

Тепер можна встановити драйвери для певного пристрою. Для цього в меню налаштувань пристрою необхідно включити пункт "**Налагодження по USB**".

Підключити пристрій до комп'ютера. Комп'ютер виявить нове обладнання і запропонує встановити драйвер. Встановлюємо драйвера з папки, куди вони були завантажені раніше.

Після установки драйверів у диспетчері пристроїв з'явиться новий пристрій «**ADB Interface**», і ми можемо переконатися у цьому відкривши його, натиснувши правою клавішею миші іконку «**Комп'ютер**» -> «**Властивості**» -> «**Диспетчер пристроїв**».

Крім того, ви можете спробувати встановити на комп'ютер **універсальний ADB-драйвер**.

16.6.3 Запуск ADB

Працювати з програмою **adb** найкраще через командний рядок **Windows**. Для виклику командного рядка на комп'ютері з Windows XP натискаємо «Пуск» і в полі введення «Виконати» набираємо cmd і натискаємо «Enter».

На комп'ютері з **Windows 7** натискаємо «**Пуск**» і в полі введення «Знайти програми та файли» набираємо **cmd** і натискаємо «Enter».

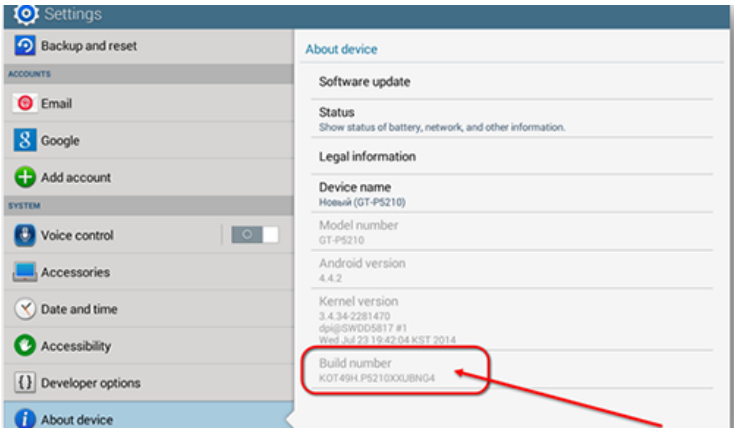
Відкриється вікно командного рядка, і для того щоб, наприклад, подивитися, які пристрої у нас підключені до комп'ютера, набираємо в ньому команду:

adb devices.

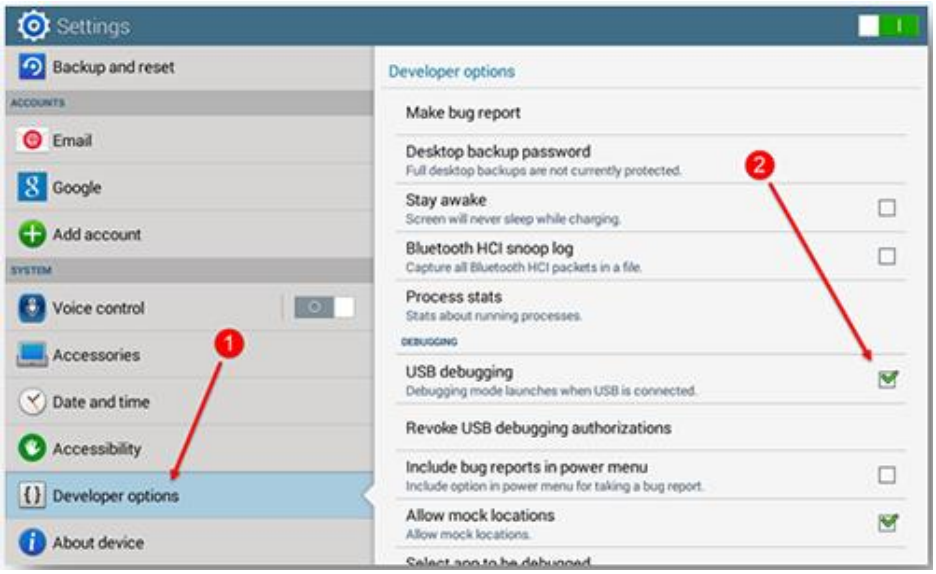
Програма **adb** відобразить список пристроїв, підключених у даний момент до комп'ютера.

16.6.4 Зняття логів з Android пристроїв за допомогою LogCat (Android Device Monitor)

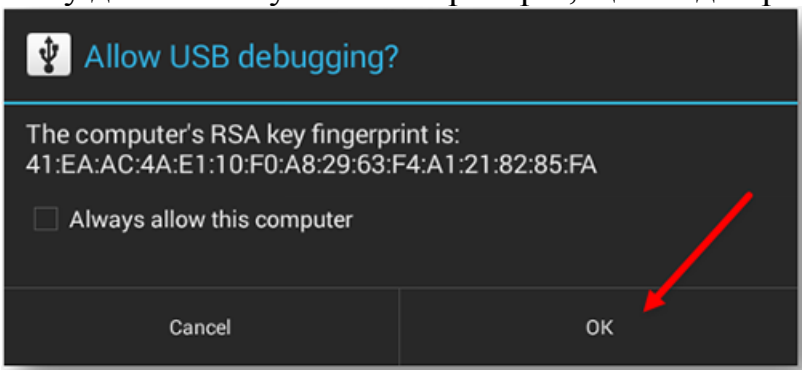
1. Встановити ADB
2. Зайти в налаштування на телефоні або планшеті.
3. У кінці списку вибрати "About device".
4. Натискати номер білда до тих пір, поки не ввімкнеться режим розробника:



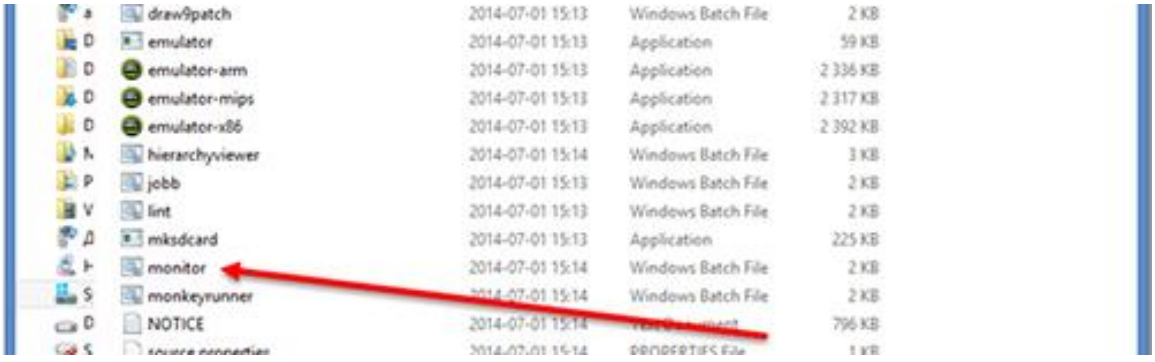
5. Перейти в розділ "Developer options".
6. Відзначити чек-бокс "USB debugging":



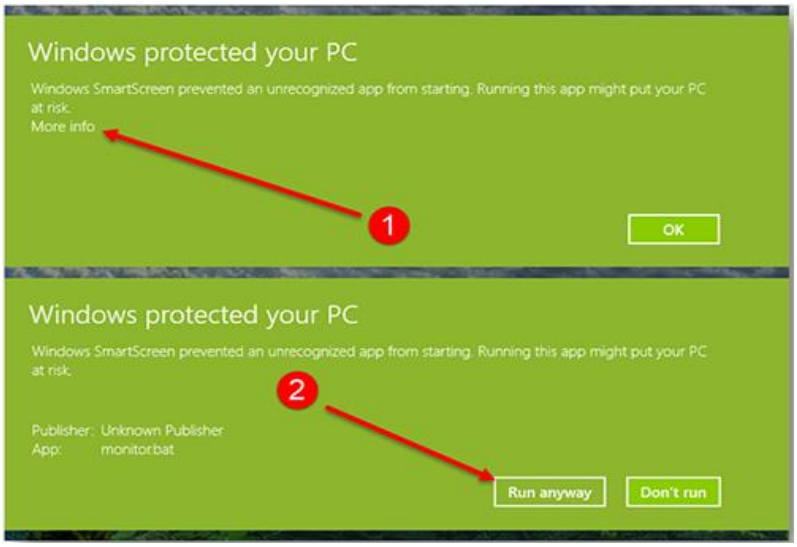
7. Підключити телефон або планшет до ПК.
8. Підтвердити у діалоговому вікні на пристрої, що ми довіряємо ПК:



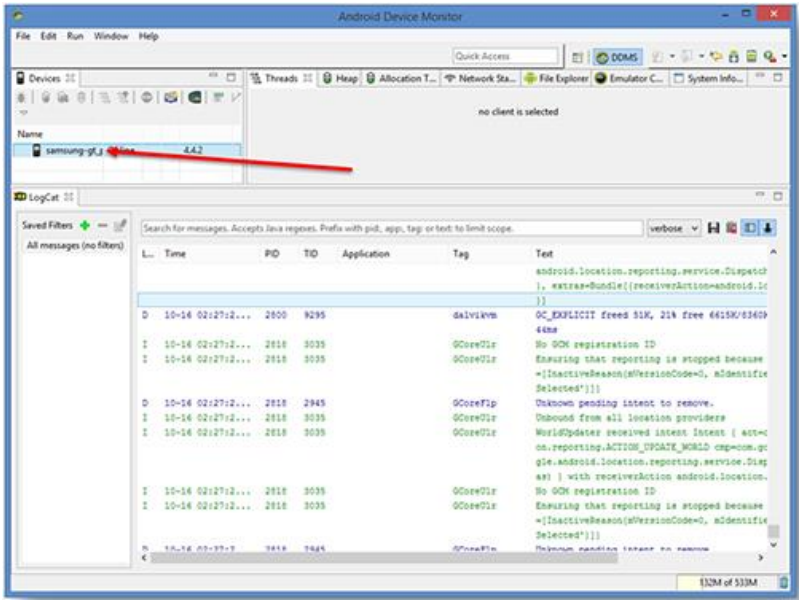
9. Запустити файл "monitor.bat", який знаходиться в папці з інструментами. У нашому випадку адреса буде: c: \ adt \ sdk \ tools \ monitor.bat.



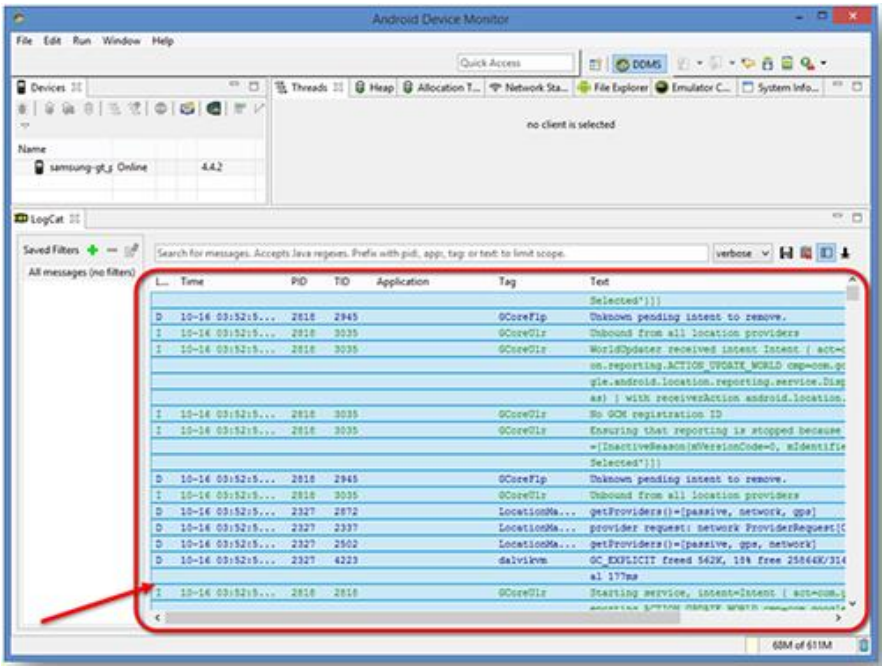
10. У разі, якщо операційна система хоче "захистити" Вас від запуску даного файлу, потрібно вибрати "More info >> Run anyway":



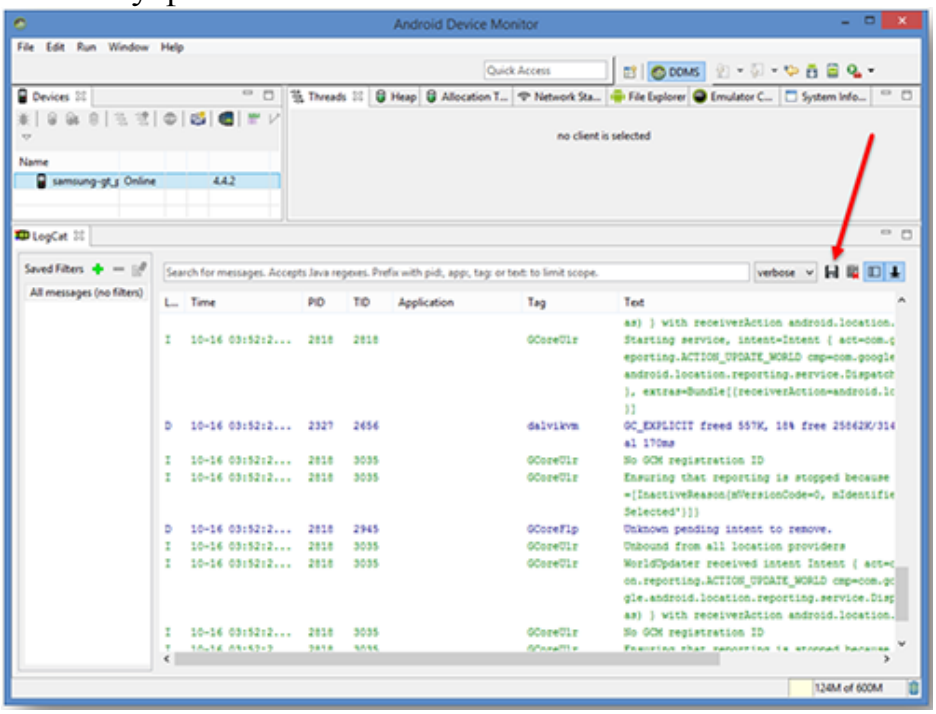
11. **Мобільне тестування** показує, що перший запуск може зайняти до 15-20 секунд.
12. У вікні необхідно вибрати пристрій, з якого буде проводитися логуваня:



13. Після виконання дій, які повинні бути залоговані, необхідно вибрати потрібну ділянку, використовуючи мишу і клавішу SHIFT, або виділити все поєднанням CTRL + A:



14. Зберегти лог у файл:

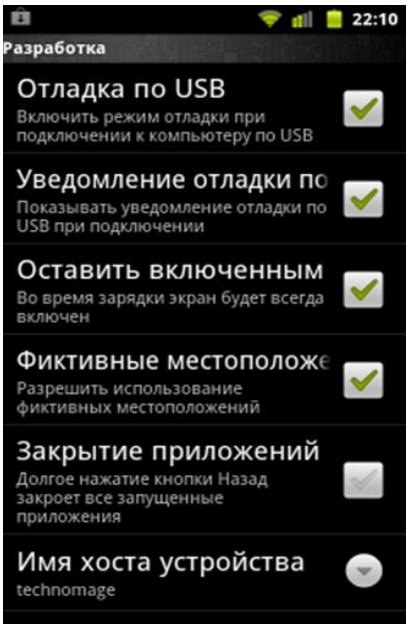


16.6.5 Заміри пам'яті

Підготовка для роботи з Android SDK

1. Встановити USB-драйвер для вашого пристрою.
2. Упевнитися, що на пристрої увімкнено налагодження по USB

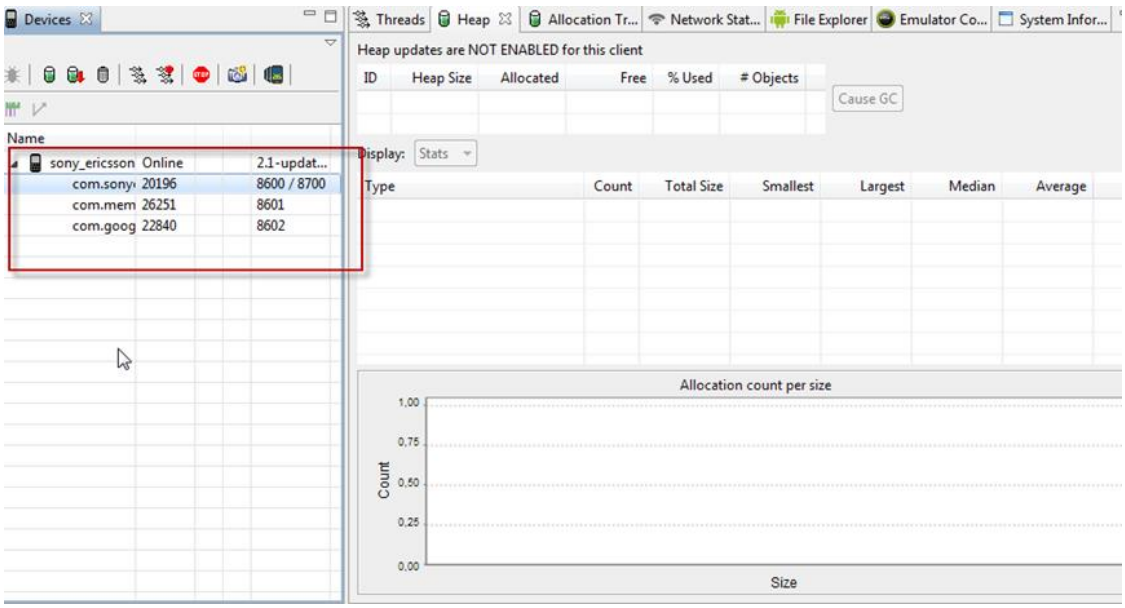
Це можна зробити тут: *Настройки -> Програми -> Розробка -> Налагодження по USB* (встановити прапор).



Після активування **режиму налагодження по USB** ви зможете працювати з пристроєм за допомогою утилітів ADB.
ADB – Android Debug Bridge, дослівно «Налагоджувальний Міст для Андроїд», українською – «Інтерфейс для налагодження Андроїд».

Після цього стануть доступними такі можливості, як: перегляд налагоджувальних повідомлень, установка і видалення програм за допомогою ADB та інші корисні речі. Також деякі способи отримання прав root вимагають увімкнути налагодження по USB.

3. Скачати останній Android SDK.
4. Розпакувати zip-файл з Android SDK у C: \ AndroidSDK.
5. Підключити пристрій до комп'ютера через USB-кабель. Повинна початися установка драйверів, якщо це Windows.
6. Запустити командний рядок (у пуску ввести cmd у рядку пошуку).
   Ввести наступне в командний рядок (натискаємо enter після закінчення кожного рядка):
   cd
   cd AndroidSDK \ tools
   adb devices
7. Відкрити tools \ monitor.bat
8. Переконатися, що пристрій визначено.



Для перегляду використання потоку процесів:

- На вкладці Пристрої виберіть процес, за яким ви хочете виконати дослідження потоку.
- Натисніть кнопку Update Heap, щоб активувати збір інформації для процесу.
- У Heap вкладці натисніть Cause GC, щоб здійснити збір сміття. Коли операція буде завершена, ви побачите групи типів об'єктів і пам'ять, яка була виділена для кожного типу. Ви можете натиснути Cause GC ще раз, щоб оновити дані.
- Натисніть на тип об'єкту у списку, щоб побачити діаграму, яка показує кількість об'єктів, виділених для конкретного розміру пам'яті в байтах.

16.7 Windows Phone

16.7.1 Windows Phone SDK

Windows Phone SDK 8 – це набір інструментів для створення ігор і додатків на базі Windows Phone 8 та для встановлення сторонніх додатків, які знаходяться у стадії розробки на WP8 смартфон.

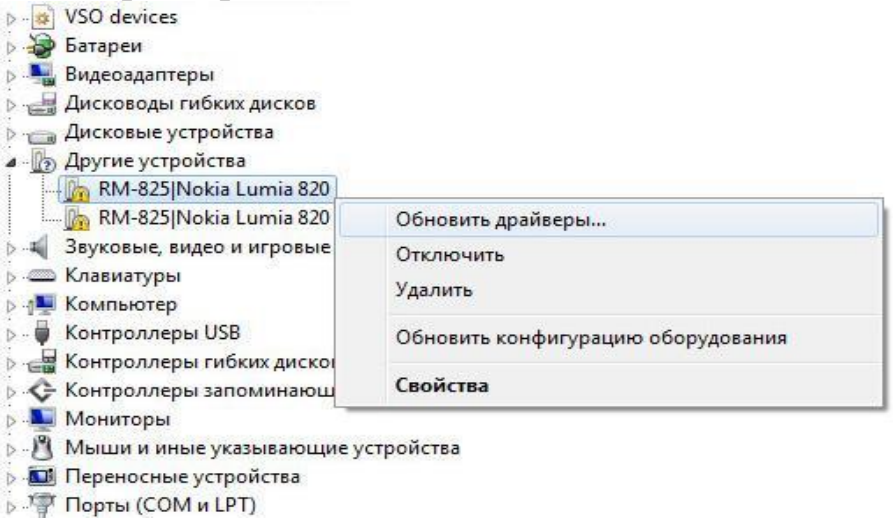
Покрокова інструкція по установці Windows Phone SDK 8.0 на Windows 7:

1. Переконатися, що на комп'ютері не встановлений старий пакет SDK 7.1, так як з ним буде конфлікт програми. У такому разі потрібно

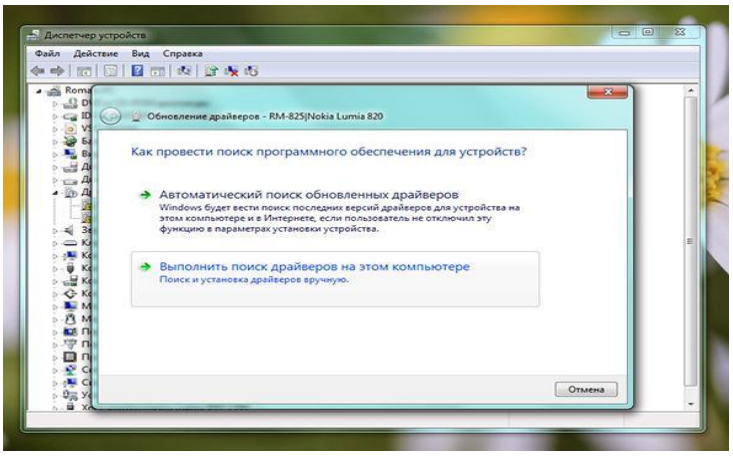
очистити реєстр і всі папки від даного пакета. В ідеалі потрібна чиста Windows 7.

2. Завантажити та встановити **Microsoft .NET Framework 4.5**
3. Завантажити архів з установочними файлами **Windows Phone SDK 8.0** і розпакувати.
4. У папці **SDK 8** відкрити файл **wpsdk_en.msi** і встановити. Під час установки відображається діалогове вікно з повідомленням про помилку – необхідно два рази натиснути кнопку "ОК".

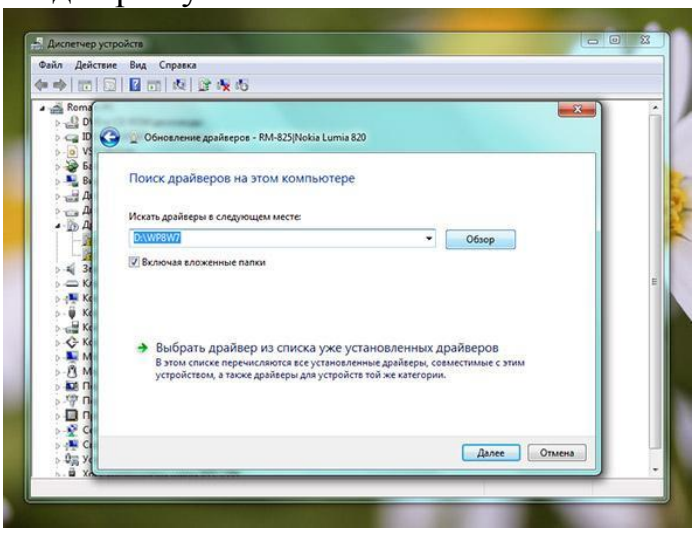
5. Потім у папці **WP Images** запустити додаток **mobiletools_wpimages.msi** і встановити його.
6. У папці **MobileTools Profiler ARM** запустити додаток **MobileTools_ProfilerARM.msi** і встановити його.
7. Після установки всіх компонентів пакету SDK 8.0 необхідно буде підключити смартфон до комп'ютера через USB-кабель.
8. Після підключення пристрою необхідно встановити драйвера. Для цього необхідно відкрити "**Диспетчер пристроїв**", натиснувши зліва пункт "Диспетчер пристроїв". У списку пристроїв знайти смартфон, натиснути лівою кнопкою миші, у контекстному меню вибрати "**Оновити драйвера**".



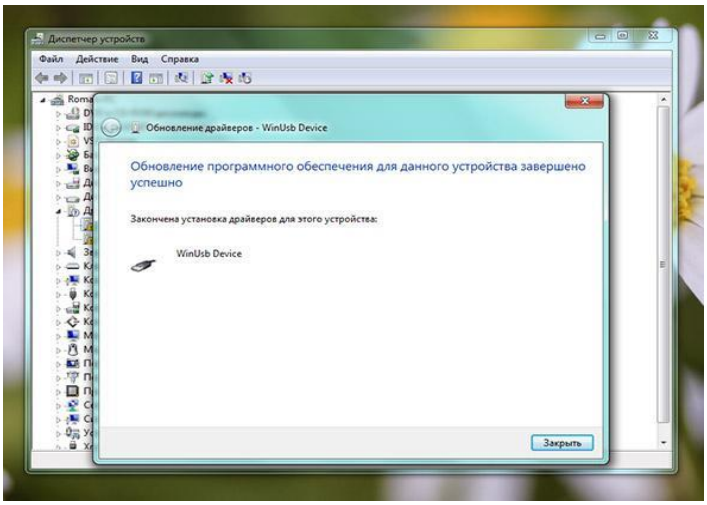
9. У вікні оновлення драйверів вибрати пункт "**Виконати пошук драйверів на цьому комп'ютері**".



10. Вказати шлях до архіву з Windows Phone SDK.



11. Встановлення всіх компонентів Windows Phone SDK успішно завершено.



16.7.2 Установка XAP-файлів на Windows Phone.

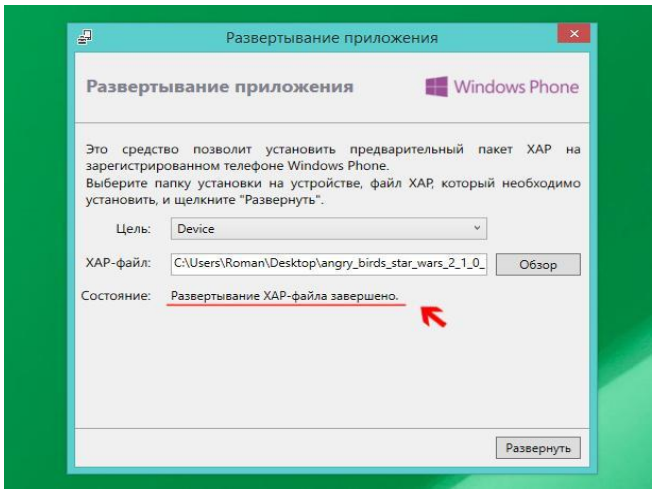
.XAP – формат файлу, який використовується в Windows Phone для установки програм. Являє собою формат файлів для архівів із стисненням.

Усередині знаходиться додаток, супутні папки і кілька xml-файлів, що відповідають за безпеку і порядок доступу до бібліотек програми.

Покрокова інструкція по установці XAP файлів:

1. На ПК має бути встановлений **Windows Phone SDK**.
2. Підключити Windows Phone смартфон до комп'ютера за допомогою USB-кабелю.
3. Запустити **Application Deployement**, який встановлюється разом з **Windows Phone SDK**.

4. У програмі **Application Deployment** вибрати мету "**Device**" (це смартфон, який підключений по USB).
5. У рядку "**XAP-файл**" натискаємо кнопку "**Перегляд**" і вибираємо завантажений вже раніше файл гри (ігри під операційну систему Windows Phone мають розширення * .xap).
6. Натиснути кнопку "**Розгорнути**".
7. Почекати завершення установки.

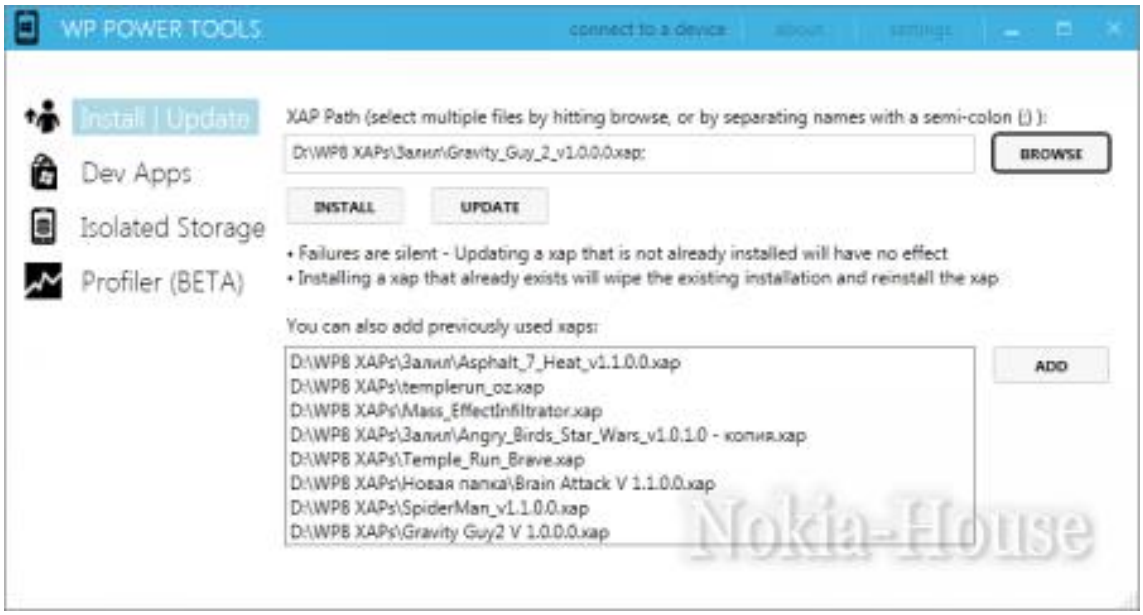


16.7.3 Windows Phone Power Tools

Windows Phone Power Tools – спеціальний інструмент для розробників додатків, але буде корисним цей інструмент не тільки для них, а й для звичайних користувачів. Це програмне забезпечення має функціонал, який, на жаль, не доступний на **Windows Phone SDK**, що робить його кращим.

Windows Phone Power Tools має такі можливості як:

- Установка, оновлення, видалення XAP-додатків розробника – дуже корисна функція, яку оцінить кожен з користувачів;
- Завантаження файлів, а також візуальний браузер з IsolatedStorage;
- Можливість отримання детальної інформації про ваш пристрій;
- Спеціальна можливість контролю додатків розробника, а саме запуск програми або його зупинка.



ТЕМА 17

Тестування ІГОР (GAME TESTING)

17 ТЕСТУВАННЯ ІГОР *(game testing)*

У сучасному світі ігрова індустрія зайняла стабільну нішу у сфері світової економіки. Якщо на початковому етапі розвитку цієї сфери вартість розробки гри була не дуже високою, то сьогодні для створення ігор потрібен великий штат розробників, тестувальників, художників та ін.

Стабільний попит на комп'ютерні ігри стимулює інвесторів робити вкладення в розвиток даної сфери. Однак не варто забувати про те, що інвестування в ігри може бути не настільки успішним, як інвестування у спеціальне програмне забезпечення. Розробивши ПЗ, яке повністю задовольняє вимоги користувача та перевершує своїх конкурентів на ринку, можна з значною часткою ймовірності очікувати, що це буде економічно вигідне вкладення.

Ігри ж не вирішують бізнес-завдання. Основна функція ігор – розважальна. Кінцевий користувач очікує одержати не чітку реалізацію специфікації чи іншої документації, а захоплюючий ігровий процес. Ступінь привабливості гри найчастіше оцінюється протягом перших годин знайомства з грою. Якщо перше враження було негативним, гравець навряд чи придбає наступні версії гри, наскільки б успішними вони не були.

Тому тестувальник програмного забезпечення повинен провести тестування ігор до того, як вони надійдуть у продаж. Успіх гри може залежати від наступних факторів: механіка гри (властивості програмної моделі, втілені за допомогою анімації та програмування), ігровий інтерфейс (сукупність елементів та методів, за допомогою яких користувач управляє грою) й ігровий процес (різні аспекти гри: графіка, звук, дії, які повинен виконати користувач для досягнення мети гри та ін.).

17.1 Види тестування ігор

1. ***Функціональне тестування.*** Мета – виявити відхилення від функціональних вимог. Зводиться до багаторазового проходження гри, виявлення неполадок й умов, за яких їх можна виправити.

2. ***Навантажувальне тестування.*** При тестуванні ігор доцільно створити ситуації, які вимагають великого обчислювального навантаження. Таким чином, тестувальник може перевірити продуктивність системи у стресовій ситуації. Так легше помітити й вчасно виправити потенційно ненадійні ділянки коду.

3. ***Тестування локалізації.*** Ігри часто перекладають мовами тих країн, в яких їх припускають випускати на ринок. У деяких випадках перекладачі не можуть надати абсолютно точний переклад, який би повністю відповідав подіям гри. Навіть правильно перекладена фраза може

не відповідати ситуації та не сприйматися носієм мови. Тому після локалізації корисно провести тестування гри мешканцям тих країн, де передбачається реалізація кінцевого продукту.

4. ***Тестування на сумісність.*** Найчастіше програмування ігор проводять на персональних комп'ютерах або ноутбуках. Однак багато ігор можуть бути призначені для інших пристроїв: ігрові приставки, мобільні телефони, комунікатори та ін. Розробка здійснюється на симуляторах даних пристроїв, однак вони можуть значно відрізнятися від оригіналу. Тому надалі можуть виникнути труднощі при запуску гри на оригінальному пристрої. Крім того, слід звернути особливу увагу на ліцензування програмного забезпечення.

При будь-яких відхиленнях гру можуть повернути на доопрацювання, що вимагає додаткового часу та втрату грошових коштів.

Отже, дуже важливо перевірити гру на відповідність заявленим параметрам апаратного забезпечення.

17.2 Ігрові механіки *(game mechanics)*

Ігрові механіки – внутрішньо-ігрові механізми, які допомагають гравцям досягти мети гри, іншими словами, це визначений набір правил та петель зворотного зв'язку, які призначені для створення сценаріїв комп'ютерної гри ***(gameplay)***, які приносять задоволення від ігрового процесу та є будівельними блоками, які можуть застосовуватися та комбінуватися з ігровим й не ігровим контекстом.

Наблизити камеру / Віддалити камеру (zoom in/zoom out)

Максимальний рівень збільшення повинен встановлюватися таким чином, щоб при цьому ігрове поле відповідало розміру на персональному комп’ютері. Управління рівнем збільшення здійснюється щипком або зворотнім щипком на екрані пристрою, забезпечуючи плавність зміни розміру. За час переміщення пальців від центру екрану до країв – картинка повинна збільшитися до максимального значення. За час переміщення пальців від країв дисплея до центру – картинка повинна зменшитися до мінімального значення.

Автоматичне віддалення камери

Необхідно реалізувати повне зменшення ігрового поля до мінімального значення при відображенні діалогу. У деяких випадках, коли відбувається подія в іншій частині ігрового поля, так само необхідно робити повне віддалення камери, щоб гравець не пропустив важливі зміни на ігровому полі. Це може бути відтворення будь-якої анімації (відкриття

дверей, поява нової активної зони на полі) або зміна поточного стану ігрового поля.

При використанні гравцем підказки (натискання кнопки «HINT») має відбуватися автоматичне зменшення ігрового поля та зміщення камери до моменту, поки на один екран не будуть поміщатися:

- предмет, на який вказує підказка;
- у разі якщо в підказці бере участь кілька предметів, то камера повинна виставляти такий рівень зменшення та положення, щоб всі необхідні об'єкти відображалися на екрані.

Зменшення ігрового поля має відбуватися плавно, але час, який витрачається на перехід з поточного розміру поля до необхідного розміру, завжди має бути однаковим, але не перевищувати 1 секунду.

Автоматичне наближення рухомого об'єкту в полі зору камери (smart zoom)

У випадку, якщо у грі представлений об'єкт типу «контейнер», при натисканні на який, відкривається додаткове вікно з предметами (скрині, сумки, ящики та ін.), повинні відбуватися наступні дії:

- плавна установка необхідного рівня збільшення/зменшення до того моменту, поки вікно контейнера не займатиме максимальну площу екрану (без урахування GUI);
- центрування камери на вікні контейнера.

При закритті вікна контейнера (гравець взяв предмет з контейнера або примусово закрив його кнопкою «CLOSE») рівень збільшення та положення камери повинні повертатися до моменту відкриття вікна контейнера.

Мінімальна активна область

У будь-якій грі мінімальна активна область об'єктів та інтерфейсів (на яку припадає натискання гравця) не повинна бути менше ніж 40x40 px.

Зони предметів

Не рекомендується обробляти активні зони предметів по контуру об'єкту, що переміщуються по екрану ***(sprite)***, які варто визначати обмежуючими паралелепіпедами ***(bounding box)***. Особливістю пристроїв з сенсорним дисплеєм є те, що гравець перекриває пальцем зону, на яку хоче натиснути, тому існує ймовірність потрапити пальцем між границями об'єкту, що переміщується по екрану; це може бути причиною того, що предмет не добереться до цілі, й гравець не буде розуміти чому. Так само рекомендується налаштовувати збільшені зони для предметів, які видно на ігровому полі не повністю або які приховані за іншими предметами. Щоб гравцю не доводилося цілиться в кілька пікселів.

Smart Tap

У випадку, якщо палець гравця при торканні екрану «накриває» більше однієї активної області, то оброблятися повинна та, чий центр ближче до місця дотику.

Обробка натискань

Натиснення ***(tap)*** предметів повинне оброблятися за подією завершення. На рух пальця по екрану павине бути задане певне скидання ***(falloff)*** (10 пікселів), після подолання якого гра вважає цю дію пролистуванням ***(swipe)*** та не обробляє натискання ні при торканні пальця, ні при його відриві від екрану.

Управління предметами на ігровому полі

У разі якщо у процесі гри, гравець може маніпулювати об'єктами на ігровому полі (переміщати їх, з’єднувати один об'єкт з іншим), необхідно реалізовувати подвійну механіку управління предметами "point and tap" та "drag and drop".

Механіка "point and tap "

Предмет має два стани:

- активний;
- неактивний;
- при натисканні на предмет, він переходить в активний стан;
- поки предмет активний, натискання гравця на ігрове поле сприймається, як спроба застосувати предмет до точки натискання;
- повторне натискання на предмет переводить його в неактивний стан;
- у разі якщо предмет застосовується в неправильному місці, то він так само переходить в неактивний стан.

Механіка "drag and drop"

Використовуючи механіку "drag and drop", предмет має також два стани:

- активний;
- неактивний;
- здійснювати "drag&drop" з неактивним предметом неможливо;
- щоб активізувати предмет, гравцеві необхідно натиснути його;
- утримуючи активізований предмет, його можна переміщати по ігровому полю за допомогою механіки "drag&drop";
- наблизивши предмет до краю екрану, екран повинен почати прокручуватися ***(scroll)***;
- швидкість прокрутки ***(scroll)*** повинна збільшуватися залежно від того, наскільки близько гравець підвів предмет до краю екрана;
- предмет застосовується по контуру об'єкта ***(sprite)***, а не по точці під пальцем;

- якщо гравець відпустить предмет, відбувається подія завершення ***(release)***, над зоною скомпонованого об'єкту, частиною якого є активованим предметом, предмет вважається зібраним;
- якщо гравець відпустить предмет над зоною скомпонованого об'єкта, частиною якого не є предмет, то предмет залишається на тому ж місці, куди його перемістив гравець.

Управління предметами з інвентарем

У разі якщо у процесі гри, гравець може маніпулювати об'єктами з інвентарем, необхідно реалізовувати подвійну механіку управління предметами "point and tap" та "drag and drop".

Механіка "point and tap "

- при натисненні на предмет він переходить в активний стан;
- поки предмет активний, натиснення гравця на ігрове поле сприймається, як спроба застосувати предмет до точки натискання;
- повторне натискання на предмет переводить його в неактивний стан;
- у разі якщо предмет застосовується в неправильному місці, то він так само переходить в неактивний стан;
- у випадку, якщо гра передбачає механіку закриття інвентарю, то при активації предмета інвентар повинен закритися;
- у разі якщо гравець виконує прокрутку ***(scroll)*** ігрового поля, поки предмет активний, предмет не повинен застосовуватися.

Механіка "drag and drop"

- утримуючи пальцем предмет, його можна переміщати по ігровому полю за допомогою механіки "drag&drop";
- наблизивши предмет до краю екрану, екран повинен почати прокручуватися ***(scroll)***;
- швидкість прокрутки ***(scroll)*** повинна збільшуватися залежно від того, наскільки близько гравець підвів предмет до краю екрану;
- предмет застосовується по контуру об'єкта ***(sprite)***, а не по точці під пальцем.
- предмет «застосовується» у місці, де відбулася подія завершення ***(release)***;
- у разі якщо подія завершення ***(release)*** відбулася не на активній області ігрового поля, то предмет залишається на цьому ж місці, й гравець може продовжити маніпулювати ним;
- щоб покласти предмет назад в інвентар, гравець може натиснути набудь-яке місце на екрані;

Панель завдань (список предметів для пошуку)

При адаптації панелі завдань рекомендується реалізувати наступне:

- одночасно на панелі завдань може відображатися тільки N кількість предметів, зазвичай це 6 або 8 назв предметів;
- список предметів прокручується посторінково, індикатор сторінок вказує, яка сторінка відкрита в даний момент;
- у випадку, коли гравець знаходить предмет з даного списку, список предметів повинен автоматично прокручуватися на необхідну сторінку, та повина програтися анімація викреслювання знайденого предмету;
- у разі якщо на сторінці не залишилося предметів для пошуку, система автоматично повинна відкрити сторінку з активними предметами.

Адаптація міні-ігор

Активна область міні-гри повинна займати максимальну площу на екрані. Допускається обрізати нефункціональні області. Приклад збільшення ігрової області, можна побачити на концепті нижче.

Вірно розташовані об'єкти рекомендується блокувати. У кожній міні-грі рекомендується використовувати наступні кнопки:

- *Menu* – викликає Pause Menu.
- *Skip Puzzle* – пропускає поточну міні-гру. При натисканні кнопки з'являється діалогове вікно з підтвердженням.
- *Hint* – виводить оригінальну підказку \ правила міні ігри.

Рекомендації міні-ігор з "ліхтариком"

- монокль / ліхтарик можна схопити за будь-яку його частину;
- піднятий монокль / ліхтарик не повинен центруватися на пальці гравця;
- наблизивши монокль / ліхтарик до краю екрану, екран починає прокручуватися. Швидкість прокрутки буде збільшуватися залежно від того, наскільки близько гравець підвів монокль до краю екрану;
- прокрутка повинна починатися тоді, коли палець з моноклем / ліхтариком досягає краю екрана, а не коли край монокля / ліхтарика досягне краю екрану. Коли гравець прибирає палець від екрану, прокрутка не повинна продовжуватися;
- якщо гравець натиснув на екран за межами ліхтаря \ лупи, то ліхтар \ лупа повинні переміщатися до місця дотику з постійною швидкістю, поки найближчий до місця торкання край об'єкта не досягне точки дотику.
- після того як об'єкт досяг пальця гравця, він чіпляється до нього та поводить себе так, якщо б гравець схопився за це місце об'єкта.

Рекомендації щодо Jigsaw Puzzle

У будь-якій міні-грі даного типу повинні бути реалізовані наступні принципи:

- реалізовано функціонал зменшення/збільшення;
- об'єкт можна схопити за будь-яку частину;
- піднятий об'єкт не повинен центруватися на пальці гравця;
- у міні-грі доступно "drag and drop"-управління;
- застосування предмету повинно бути по контуру об'єкта;
- об'єкт можна покласти на будь-яку частину ігрового поля;
- вірно поставлений елемент необхідно блокувати;
- у момент, коли гравець правильно встановлює останній шматочок, повинен відбутися zoom out / in (залежно від поточний режиму) до рівня, поки зібрана картинка міні-гри не займатиме всю область екрану.

17.3 Заміри пам'яті

Для зменшення ймовірності падіння додатку через недостатній об’єм оперативної пам'яті мобільного пристрою у процесі розробки ігор здійснюють вимір пам'яті на iOS-додатках. Замір споживаної додатком пам'яті здійснюється тестувальником проекту на прохання продюсера, програміста, дизайнера або в ході перевірки чек-листа. Замір здійснюється тестувальником, знайомим з проектом, за відсутності такого – опорним тестувальником проекту[1].

17.4 Тестування локалізації *(localization testing)*

Міжнародний ринок зробив істотний внесок у доходи компаній, що створюють комп'ютерні ігри, тому можливість локалізації гри, як правило, закладається ще на етапі її розробки.

Тестування локалізації – процес тестування локалізованої версії програмного продукту або сайту.

Локалізація програмного забезпечення – процес адаптації до культурних особливостей тієї чи іншої країни: переклад документації, елементів інтерфейсу користувача, допоміжних матеріалів з однієї мови іншою.

Локк (Localization kit) – документ, що містить переклади абсолютно всіх ігрових текстів однієї або декількох мов, які підтримуються додатком.

[1] Тема 16 «Інструменти тестування iOS, Android, Windows phone додатків»

Процес тестування локалізації включає наступні кроки:

- визначення та вивчення списку мов, що підтримуються;
- перевірка відповідності перекладу тематиці даного сайту або програми;
- перевірка правильності перекладу елементів інтерфейсу користувача;
- перевірка коректності перекладу системних повідомлень та помилок;
- перевірка перекладу розділу «Допомога» та супровідної документації.

До основних багів локалізації відносять:

- нечитабельні символи;
- неоднорідні шрифти (жирні / стандартні літери);
- текст, який виходить за межі плашки / робочої області;
- плашка виходить за межі робочої області;
- текст накладається на яку-небудь область або інший текст;
- невідцентрований текст;
- знаки відображаються не по середині висоти рядка (китайською, японською, корейською);
- пропущені розділові знаки;
- знаки пунктуації відображаються на другому рядку (наприклад, крапка відображається одна на наступному рядку);
- текст відображається не повністю;
- частина тексту відсутня;
- не виділена кольором частина тексту (певні умови);
- маленький міжрядковий інтервал;
- некоректне перенесення;
- відсутні або зайві пробіли в тексті;
- зайві символи в тексті;
- розрив слів при перенесенні;
- неправильний текст у діалогах (стать персонажа);
- граматичні помилки як у локка, так і у грі;
- текст відсутній в локка;
- текст не перекладено мовою локалізації (графіка);
- текст не перекладено мовою локалізації;
- текст не відповідає локкіту;
- некоректно відображаються спец. символи.

17.5 Тестування One time offer

One time offer (далі спец. пропозиція) – спеціальні акційні пропозиції, які спрямовані на поліпшення монетизації гравців, а також для мотивації продовжувати грати гравців, які потенційного готові піти з гри.

Місця показу пропозицій

Для показу пропозицій вибираються кілька місць у грі, в яких можуть бути відображені спец. пропозиції, наприклад:

- після проходження міні-гри;
- після проходження локації та ін.

Точний список місць, в яких повинні відображатися спец. пропозиції, знаходиться в проектно-специфічній документації з спец. пропозицій.

Типи пропозицій

Для угоди на пільгових умовах ***(special deal)*** використовуються такі типи пропозицій:

- пропозиція за реальні гроші: гравцеві необхідно успішно здійснити платіж у реальній валюті, щоб отримати предмети з спец. пропозиції;
- пропозиція за (кристали, інвест-поінти, марки тощо);
- пропозиція за монети;
- подарунок гравцеві;
- перехід за посиланням.

Для участі у пропозиціях доступні всі предмети з гри, які доступні гравцеві в магазині гри.

Параметри, що впливають на появу спец. пропозицій

На появу спец. пропозицій можуть вплинути деякі параметри, які можуть бути як внутрішньо-ігровими, так й зовнішніми.

До внутрішньо-ігрових умов можна віднести:

- рівень гравця;
- версію гри;
- чи здійснював гравець покупки за реальну валюту (гравці, що проводять та не проводять платіж).

До зовнішніх умов відноситься, наприклад, дата на пристрої.

Тест-кейси для перевірки коректності спец. пропозицій:

- перевірка на появу чи відсутність спец. пропозиції;
- перевірка коректності здійснення покупки;

- перевірка наявності ідентифікатора покупки у файлі конфігурації ***(config)*** крос-промо;
- перевірка коректного логування зробленої покупки;
- перевірка додавання куплених ресурсів гравцеві;
- перевірка коректності параметрів спец. пропозиції (ціна, кількість ресурсу) у вікні спец. пропозиції.

Приклад спец. пропозиції, яка має наступні умови:

- дата: 28.11.2013-01.12.2013;
- ціна: $ 1.99;
- ресурс: 200 кристалів;
- рівень гравця: 10+;
- гравці, які не платять;
- для версії гри 1.6.

17.6 Зняття креш-логів на різних типах пристроїв

Для MacOS :

- відкрити додаток Finder;
- креш-логи зберігаються в папку → User / Library / Containers / <Назва програми> / dala / log.

Для Win 8.1 x86&RT

Зняття логів з пристроїв на x86 архітектурі:

1. Встановити Visual Studio Express 2013 for Windows with Update3 (перед установкою переконатися, що в Windows вимкнені автоматичні оновлення);
2. Встановити додаток на пристрій, який підлягає тестуванню;
3. Запустити Visual Studio Express 2013;
4. Зайти в меню Debug \ Other Debug Targets \ Debug Installed App Package (рис. 17.1);

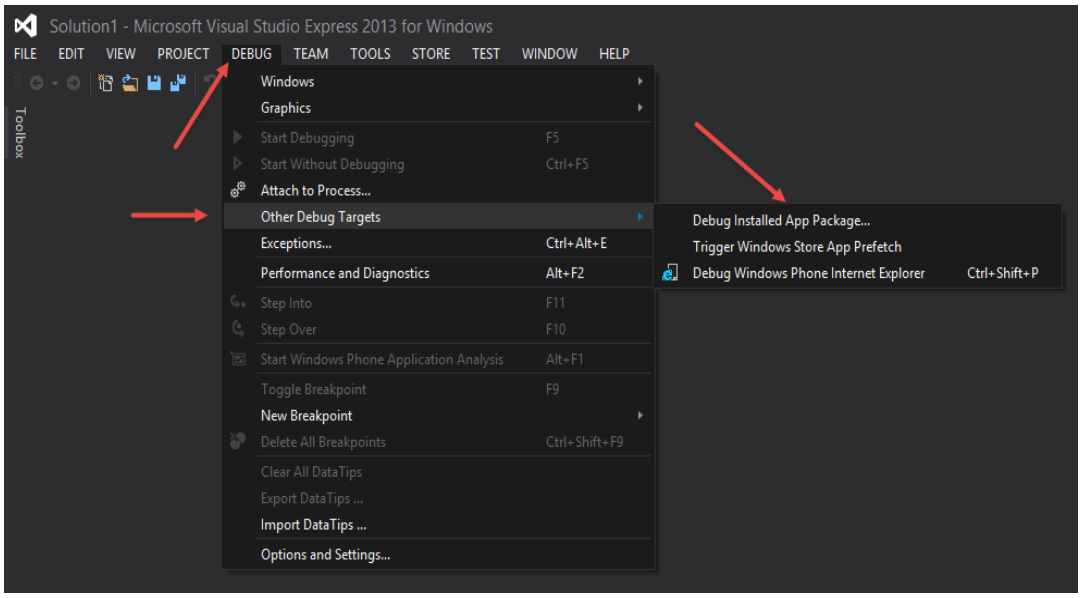


Рис.17.1.

5. У вікні Debug Installed App Package обрати “Local Machine” (рис. 17.2);
6. Знайти додаток у списку встановлених програм, що підлягає тестуванню;

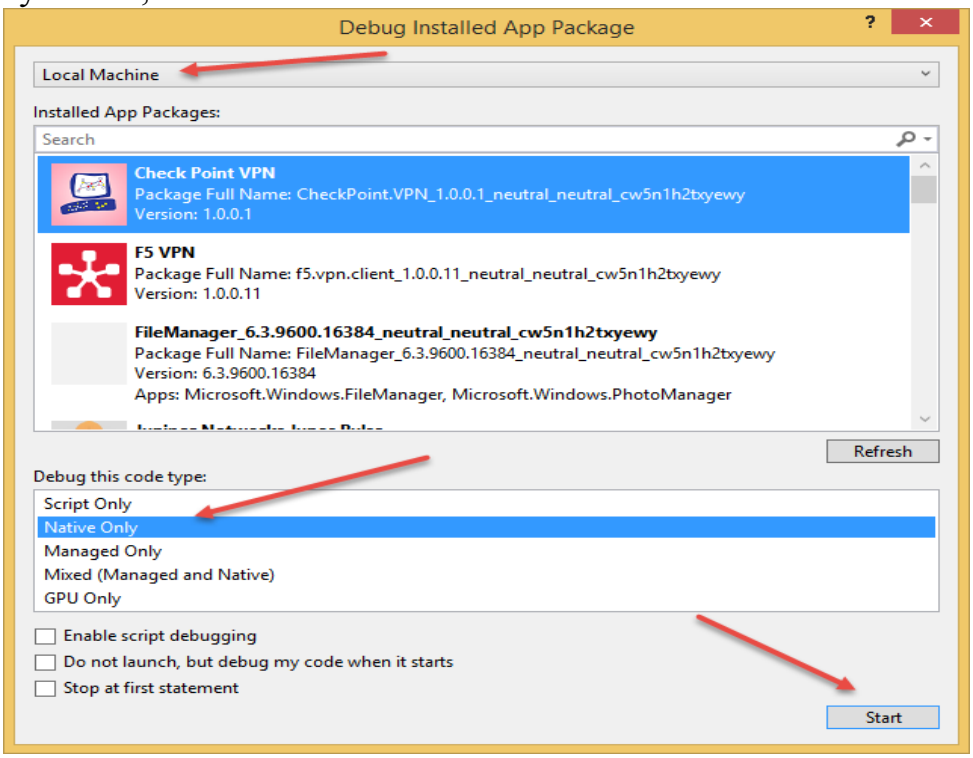


Рис. 17.2.

7. Натиснути кнопку “Start” (запуститься додаток);
8. Переключитися на Visual Studio, у вікні “Output” будуть відображатися логи з пристрою (рис. 17.3).

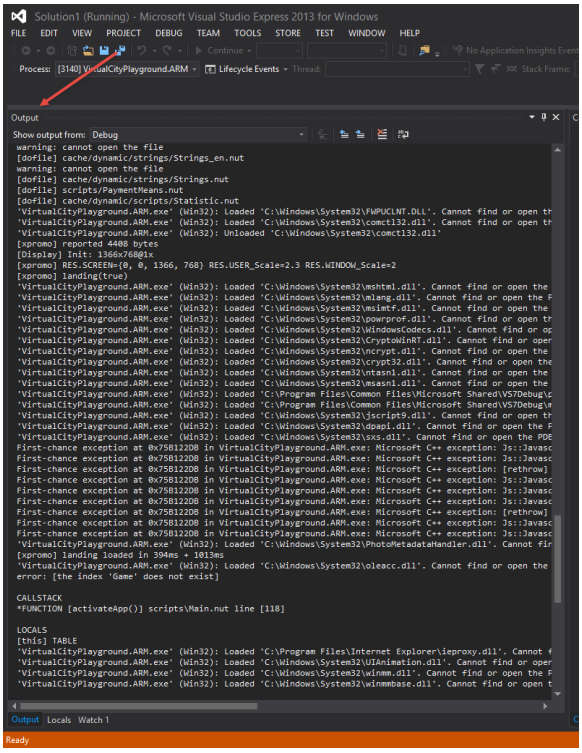


Рис. 17.3.

Зняття логів з ARM-пристроїв

Для зняття логів з ARM-пристрою необхідно виконати дві умови:

- комп'ютер, на якому стоїть Visual Studio, й сам пристрій повинні бути в одній підмережі;
- на ARM-пристрої повинен бути Remote Tools for Visual Studio 2013.

Необхідно виконати наступні дії:

1. Встановити Visual Studio Express 2013 for Windows with Update3;
2. Встановити додаток на пристрій, що підлягає тестуванню;
3. Запустити Visual Studio;
4. Зайти в меню Debug \ Other Debug Targets \ Debug Installed App Package.
5. У вікні Debug Installed App Package вибрати Remote Machine;
6. Переконатися, що ARM пристрій та ПК знаходяться в одній підмережі;
7. Запустити Remote Debugger з пакету Remote Tools for Visual Studio 2013 на ARM-пристрої;
8. Пристрій повинен з'явитися в Remote Debugger Connections;
9. Натиснути кнопку “Select” (рис. 17.4, 17.5);
10. На ПК ввести логін та пароль локального користувача з ARM-пристрою.

Подальші дії збігаються зі зняттям логів на x86 пристроях[2].

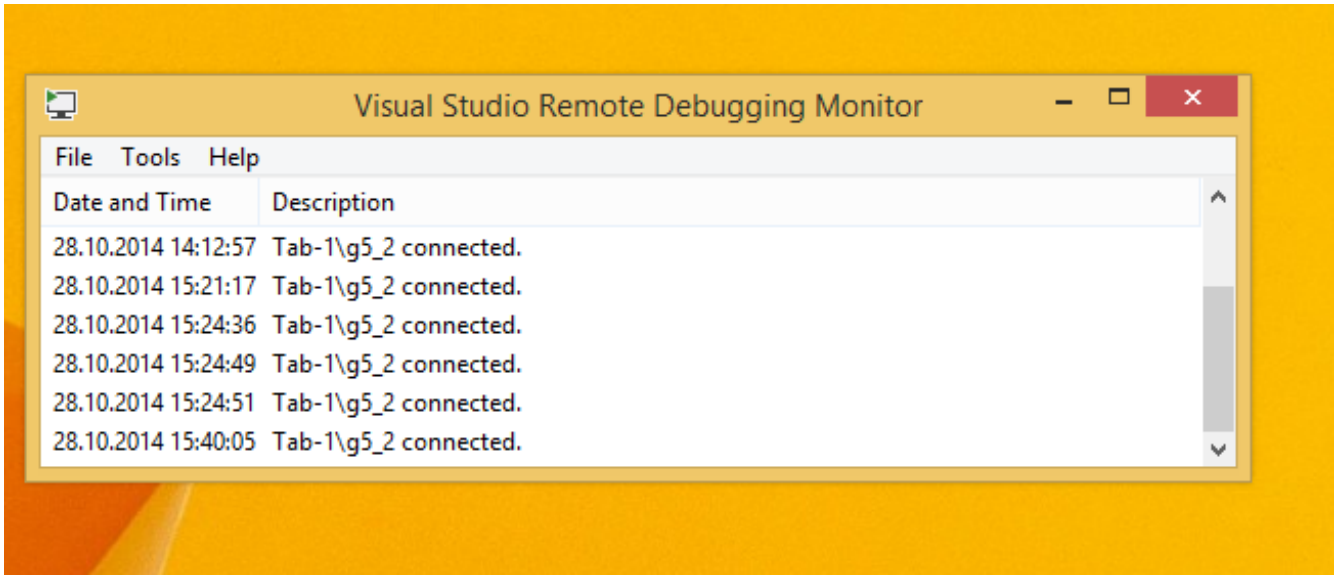


Рис. 17.4.

[2] Тема 16 «Інструменти тестування iOS-, Android-, Windows phone-додатків»

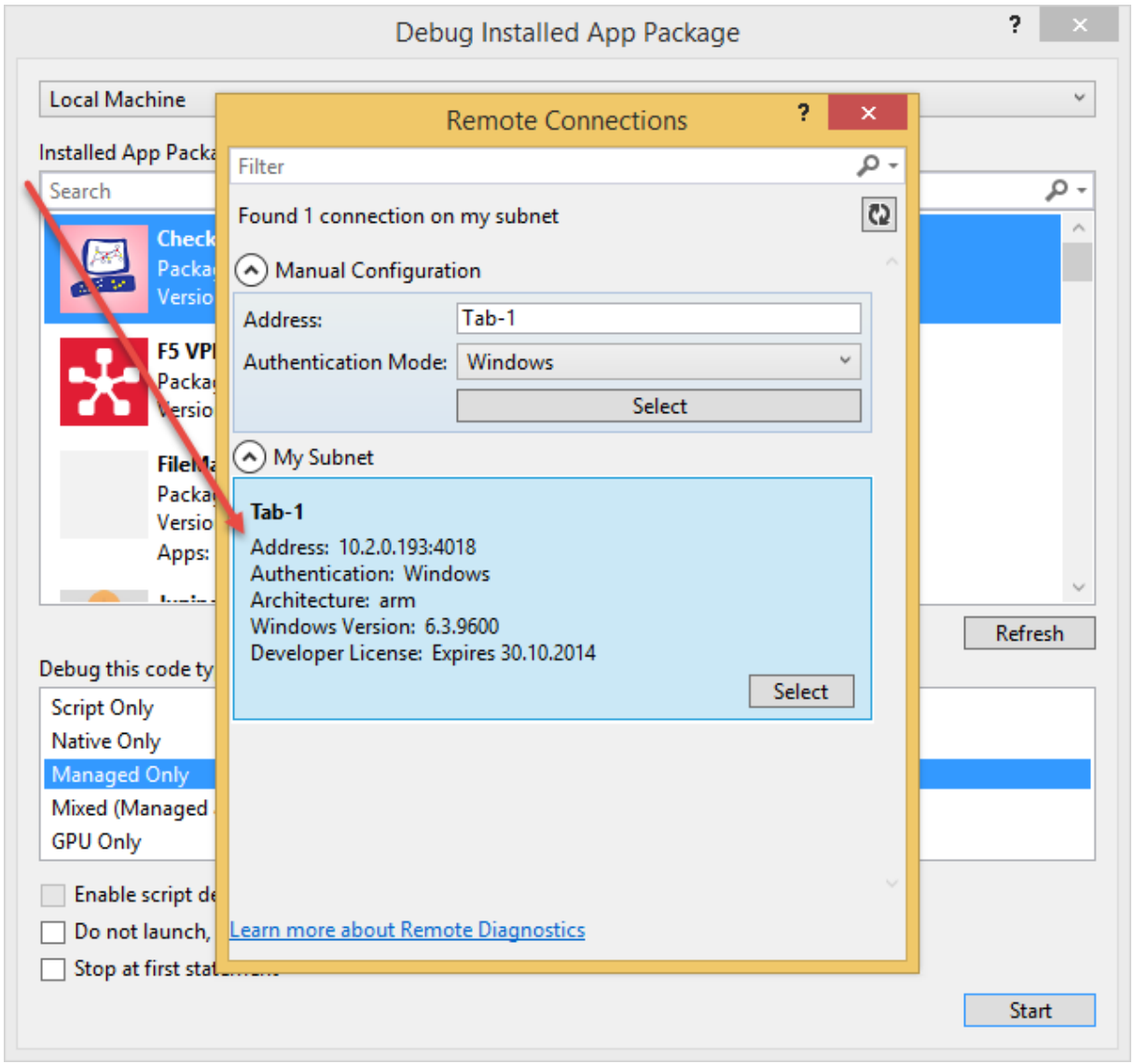


Рис. 17.5.

17.7 Тестування ігрового інтерфейсу та процесу по записах реакцій гравця

Анкетування та обговорення дозволяє виявити якісні проблеми ігор, проте будь-яка оцінка гри суб'єктивна. Крім того, добровільні тестери часто не здатні пояснити, що саме їм не подобається у процесі гри, а професіонали обходяться дорого. Далі будь-яке тестування, засноване на фіксації думки гравця, здатне вирішити проблему лише якісно. Конкретні чисельні метрики доводиться знаходити експериментально, що збільшує час та вартість роботи. Тому прийнято записувати реакції гравця й, аналізуючи їх, коригувати чисельні параметри гри.

Як приклад застосування фіксування інформації, яку можна отримати з його допомогою, розглянемо такий клас ігор, як головоломки.

Суть таких ігор зводиться до вирішення послідовності завдань на логіку, увагу, пам'ять, іноді необхідно вирішити завдання за певний час. Класичними головоломками є пазли, шахові та шашкові завдання, судоку, кросворди та ін.

Очевидно, потрібно отримати гру, яка буде захоплювати користувача з перших хвилин та яка буде викликати інтерес, який протягом усього часу гри не буде згасати. З поступовим збільшенням складності, від рівня до рівня, якісним звуковим й музичним супроводом. Імовірно, у грі реалізовано автоматичне збереження, можливість налаштування елементів управління, керування звуком та музикою, створення рівнів для користувача.

Важливим параметром ігор-головоломок є еталонне проходження рівня, яке виконується досвідченим гравцем, який добре знає правила. В основному здійснюється кілька проходжень. З них вибирається найбільш близьке за сукупністю параметрів до середньої величини.

У таблиці 17.1 наведено параметри, що підлягають запису та дані, які можна отримати в процесі їх аналізу.

Таблиці 17.1. Параметри, що підлягають запису, та дані, які можна отримати в процесі їх аналізу

Характеристика	***Параметри, що підлягають запису***
Якість гри	Час, проведений користувачем за грою.
Зрозумілість правил	Швидкість проходження першого рівня по відношенню до еталонного проходження. Кількість звернень за допомогою.
Складність рівня	Час проходження рівня по відношенню до еталонного часу. Кількість програних ігор або рівнів.
Коректність обраних інтервалів автоматичного збереження гри	Кількість збережень зроблених користувачем. Наявність великої кількості збережень на певній ділянці гри говорить про те, що в цьому місці необхідно зберігати дані автоматично.
Складність рівня по відношенню до інших	Порівняння параметрів «складність» для кожного рівня
Залучення користувачів у гру	Кількість користувачів, які пройшли більше певного числа рівнів. Кількість користувачів, які пройшли останній рівень. Кількість користувальницьких рівнів, які були створені.
Якість звукового та музичного супроводу	Кількість користувачів, які відключили звук або музику

Зручність управління, відповідність стандартам індустрії	Кількість користувачів, що змінили управління
Оптимальність кількості елементів управління	Середньоквадратичне відхилення процентних співвідношень використання конкретних елементів управління

Використовуючи такі характеристики, можна судити про слабкі місця гри та своєчасно змінити їх. Очевидно, що крім представлених даних, дуже цікавими представляються знання про користувача: вік, стать, соціальний статус, професія та ін.

17.8 Приклади багів популярних ігор

Нижче приведений короткий опис багів популярних ігор:

1. ***Баг гри "Mafia II"[3]:*** найчастіше розробники масштабних ігор допускають помилки взаємодії 3D-об'єктів. Приклад такого бага можна зустріти в багатьох іграх. У даному випадку при зіткненні з конструкцією будівлі автомобіль не зупиняється, а продовжує спроби руху, в той час як гра намагається відтворити логічну поведінку (зупинка автомобіля з відскоком від будівлі). Подібна поведінка продовжується й після того як герой покинув автомобіль.

2. ***Баг гри "FIFA15"[4]:*** ще один приклад поганої реалізації взаємодії між 3D-об'єктами, у наслідок чого порушується фізика анімації. Зверніть увагу на неприродний рух футболіста в даному прикладі.

3. ***Баг гри "Warface"[5]:*** так само хорошим прикладом є порушення доступності переміщення героя (наприклад, біг по воді / повітрю або по інших об'єктах). У даному прикладі баг полягає в можливості переміщення героя по натягнутому тросу моста.

4. ***Баг гри "GTA: San Andreas"[6]:*** досить цікавий баг був помічений у грі GTA: San Andreas. Збільшивши приціл, дивлячись у дзеркало, в будівлі відображається об'єкт, який перебуває за ним.

[3] https://www.youtube.com/watch?v=DxRAuBEnqy4
[4] https://www.youtube.com/watch?v=DqNIGCxxWj4
[5] https://www.youtube.com/watch?v=wjIoBeY9_V4
[6] https://www.youtube.com/watch?v=3CnJ52riKmg

5. ***Баг гри "GTA V"[7]:*** з виходом нової версії гри деякі баги продовжують відтворюватися. Знаменитий баг серії ігор GTA все так само присутній й в останній версії гри. При спробі піднятися на будинок герой проходить крізь стіну, а потім потрапляє за межі віртуального 3D-світу, продовжуючи взаємодіяти за сценарієм (відкривається парашут, герой продовжує падіння)

6. ***Баг гри "Euro Truck Simulator 2"[8]:*** також часті випадки з багами, коли текстури й 3D-об'єкти завантажуються із затримкою. У підсумку це може нашкодити проходженню гри. У даному випадку частина віртуального світу (відрізок дороги) завантажилася з затримкою, а відобразилася за мить до аварії.

7. ***Баг гри "Driver: San Francisco"[9]:*** досить популярний приклад бага з порушенням фізики руху 3D-об'єктів. Зверніть увагу на рух вантажівки після чергового зіткнення з автомобілем героя. Після ковзання по трасі вантажівка перевертається, неприродно переміщується у повітрі з прискоренням й уповільненням руху, частково "провалюється" крізь дорогу.

[7] https://www.youtube.com/watch?v=IgOaoIsOisQ
[8] https://www.youtube.com/watch?v=z1MZqkemgM8
[9] https://www.youtube.com/watch?v=KEeTbZ5MpPc

ТЕМА 18

Ролі у процесі розробки ПЗ. Комунікації у сфері тестування

18 РОЛІ У ПРОЦЕСІ РОЗРОБКИ ПЗ. КОМУНІКАЦІЇ У СФЕРІ ТЕСТУВАННЯ

Розробка програмного забезпечення *(software development)* – це різновид діяльності *(професія)* і процес, спрямований на створення і підтримку працездатності, якості і надійності програмного забезпечення, використовуючи технології, методологію і практики з інформатики, управління проектами, математики, інженерії та інших сфер знань.

Процес розробки програмного забезпечення *(англ. software development process, software process)* – структура, згідно з якою побудована розробка програмного забезпечення *(ПЗ)*.

Існує кілька моделей такого процесу, кожна з яких описує свій підхід у вигляді задач і/або діяльності, які мають місце під час процесу.

18.1 Обов'язки серед команди під час розробки ПЗ

Найтиповіша рольова структура проекту запропонована Центром об'єктно-орієнтованої технології компанії IBM. Ця структура охоплює досить повний перелік типових ролей, погоджений із багатьма реальними дисциплінами розвитку програмних проектів. У той же час вона представляє ролі розробників в організаційному контексті, тобто розглядає не тільки розробників, а й тих, хто, не беручи участі у проекті в якості виконавців, впливає на постановку задач проекту, на виділення ресурсів і забезпечення здійсненності розвитку робіт.

У наведеному переліку характеристика кожної ролі, по суті, задає коло споріднених організаційних і виробничих функцій, які поєднуються з метою визначення ролі.

Менеджер Проекту *(PM)* – особа, яка займається питаннями пошуку замовників проектів і виконавців. При цьому, до основних навичок (окрім професійних навичок у конкретній галузі) менеджера проектів насамперед належать навички управління ресурсами, планування робіт, а також управління ризиками.

Як правило, менеджер проекту відповідає за розвиток проекту в цілому, гарантує, що розподіл завдань і ресурсів дозволяє виконати проект, що роботи і надання результатів ідуть за графіком, що результати відповідають вимогам. У межах цих функцій менеджер проекту взаємодіє із замовником і планувальником ресурсів.

На менеджера проекту покладаються обов'язки з:

- організації процесу розробки;
- координації і контролю усіх видів діяльності у проекті;
- розробки плану проекту (Project Plan);
- проведення регулярних статус-мітингів у проектій групі;
- контролю готовності делівері і нового білду для QA;
- надання замовнику документації і проміжних версій для перегляду і затвердження або коментування;
- регулярного спілкування із замовником, з'ясування вимог;
- надання звітів про статус проекту.

Бізнес-аналітик *(Business Analyst)* – спеціаліст, який використовує методи бізнес-аналізу для аналітики потреб діяльності організацій з метою визначення проблем бізнесу і пропозиції їх вирішення.
Бізнес-аналітик працює над:

- з'ясуванням і аналізом усіх вимог замовника;
- фіксуванням усіх вимог замовника *(у багтрекінговій системі та в функціональних специфікаціях),* відстеженням усіх змін у вимогах;
- написанням і підтримкою специфікацій.

Системний аналітик *(Technical Leader)* – широко: спеціаліст з вирішення складних організаційно-технічних проблем, що мають міждисциплінарну природу, який використовує принципи загальної теорії систем і методи системного аналізу.

У вузькому значенні у сфері інформаційних технологій цей термін використовується для позначення професійної ролі і професії, відповідальної за аналіз інтересів зацікавлених осіб на предмет можливості їхнього задоволення технічними властивостями IT-системи, яка створюється.

Як правило, системний аналітик відповідальний за:

- ✓ координацію і контроль діяльності з дизайну, архітектури і кодування;
- ✓ підтримку контролю версій;
- ✓ налаштування скрипту для авто-білдера і своєчасну зборку версій.

Одну з основних ролей у процесі розробки ПЗ виконує QA-менеджер (QA manager), адже саме він відповідає за:

- організацію і контроль процесу тестування на проекті;
- планування тестування;

- участь в адаптації процесу розробки під проект, аналіз його якості;
- аналіз результатів тестування і якості продукту;
- участь в управлінні вимогами;
- участь у налаштуванні багтрекінгової системи, повне відстеження багів;
- контроль готовності делівері і нового білду для QA.

Іншу важливу роль у процесі розробки ПЗ виконує QA-аналітик (QA Analyst), який у більшості випадків відповідальний за:

- ✓ підготовку тест-дизайну;
- ✓ написання тест-кейс специфікації;
- ✓ проведення тестування;
- ✓ реєстрацію багів;
- ✓ відстеження і перевірку багів;
- ✓ написання документації користувача.

Розробник (Developer), він же Програміст – це спеціаліст, який займається написанням і коригуванням програм для комп'ютерів (будь-яких обчислювальних приладів), тобто програмуванням.

Саме девелопери у процесі розробки ПЗ займаються:

- розробкою якісного коду;
- проведенням модульного тестування;
- підтримкою контролю версій;
- написанням документації для користувача, яка стосується інсталяції й адміністрування.

Замовник – широко: особа (фізична або юридична), зацікавлена у виконанні робіт, наданні послуг або придбанні у продавця певного продукту. Інакше кажучи, це реальний (в організації, якій підпорядкована команда, або поза нею) ініціатор розробки або хтось інший, уповноважений приймати результати розробки (як поточні, так і остаточні).

Замовник або customer повинен:

- своєчасно переглядати специфікації та інші документи, які надсилаються *(з метою затвердження документа, надання коментарів, виправлення неточностей тощо);*
- вносити зауваження і побажання до багтрекінгової системи.

Серед інших ролей у процесі розробки ПЗ варто відзначити:

Планувальник ресурсів (Planner) – висуває і координує вимоги до проектів в організації, яка здійснює дану розробку, а також розвиває і спрямовує план виконання проекту з погляду організації.

Керівник команди (Team Leader) – виконує технічне управління проекту. Для великих проектів можливе залучення кількох керівників від команд, які відповідають за вирішення окремих завдань.

Архітектор (Architect) – відповідає за проектування архітектури системи, узгоджує розвиток робіт, які пов'язані з проектом.

Проектувальник підсистеми (Designer) – відповідає за проектування підсистеми або категорії класів, визначає реалізацію та інтерфейси з іншими підсистемами.

Експерт предметної сфери (Domain Expert) – відповідає за вивчення сфери додатку, підтримує спрямованість проекту на вирішення завдань цієї сфери.

Розробник інформаційної підтримки (Information Developer) – створює документацію, яка супроводжує продукт, коли випускається версія. Інсталляційні матеріали, які до неї входять, так само як і відсилкові та навчальні, а також матеріали допомоги, надаються на паперових і машинних носіях. Для складних проектів можливий розподіл цих завдань між кількома розробниками інформаційної підтримки.

Спеціаліст з інтерфейсу користувача (Human Factors Engineer) – відповідає за зручність використання системи. Працює із замовником, щоб переконатися, що інтерфейс користувача задовольняє вимоги.

Бібліотекар (Librarian) – відповідає за створення і ведення загальної бібліотеки проекту, яка містить усі проектні робочі продукти, а також за відповідність робочих продуктів стандартам.

18.2 Коротко про суміщення ролей членів команди у процесі розробки ПЗ

Залежно від проекту та умов його виконання ролі учасників проекту можуть суміщатися. ***Граничний випадок*** – програміст розробляє проект для себе *(на власне замовлення),* сам планує розподіл ресурсів *(строки виконання роботи, використання обчислювальної техніки тощо),* сам

приймає проектні рішення *(керує і управляє собою)* і сам же займається розробкою, експертизою й обслуговуванням. Але навіть тут корисно для себе чітко розуміти, чиє *(у розумінні розподілу ролей)* завдання вирішується у кожний конкретний момент. Навіть тут варто відмовлятися, наприклад, від безглуздої документації – не із загальних міркувань, а в результаті мотиваційного співставлення витримок *(часу підготовки)* і надбань *(фіксація отриманих результатів на майбутнє, додатковий досвід тощо)*.

Зрозуміло, що суміщення ролей не може бути довільним. Одні суміщення небажані (різною мірою), другі нейтральні, треті корисні для проекту.

Загальний регламент для суміщення визначають наступні принципи:

- Не варто допускати суміщення ролей, які мають конфліктні або суперечні інтереси. Якщо таке суміщення все-таки використовується, то необхідно передбачити механізми, які демпфуватимуть суперечності.

- Надання ролей з конфліктними інтересами різним людям забезпечує рівновагу суперечних поглядів.

- Вдаватися до суміщення ролей для учасників проекту, основною роллю яких є розробка, означає завідоме збільшення строків виконання відповідних робіт. З цієї причини для них допустимі тільки такі суміщення, які справді необхідні в даний момент розвитку проекту (наприклад, у деяких авральних ситуаціях) і які не потребують постійної діяльності протягом тривалого терміну.

- Якщо розробнику доручається кілька ролей, то він завжди повинен знати, яку з них він виконує у даний момент.

18.3 Роль тестувальника у процесі розробки ПЗ

Перед тим, як перейти до питання про роль тестувальників у процесі розробки ПЗ, варто дещо вивчити теорію про моделі розробки ПЗ.

Каскадна модель *(англ. waterfall model, іноді перекладають як модель «Водоспад»)* – модель процесу розробки програмного забезпечення, у якій процес розробки має вигляд потоку, який послідовно проходить фази аналізу вимог, проектування, реалізації, тестування,

інтеграції і підтримки. Як джерело назви часто вказують статтю, опубліковану У.У. Ройсом *(W. W. Royce)* у 1970 році.

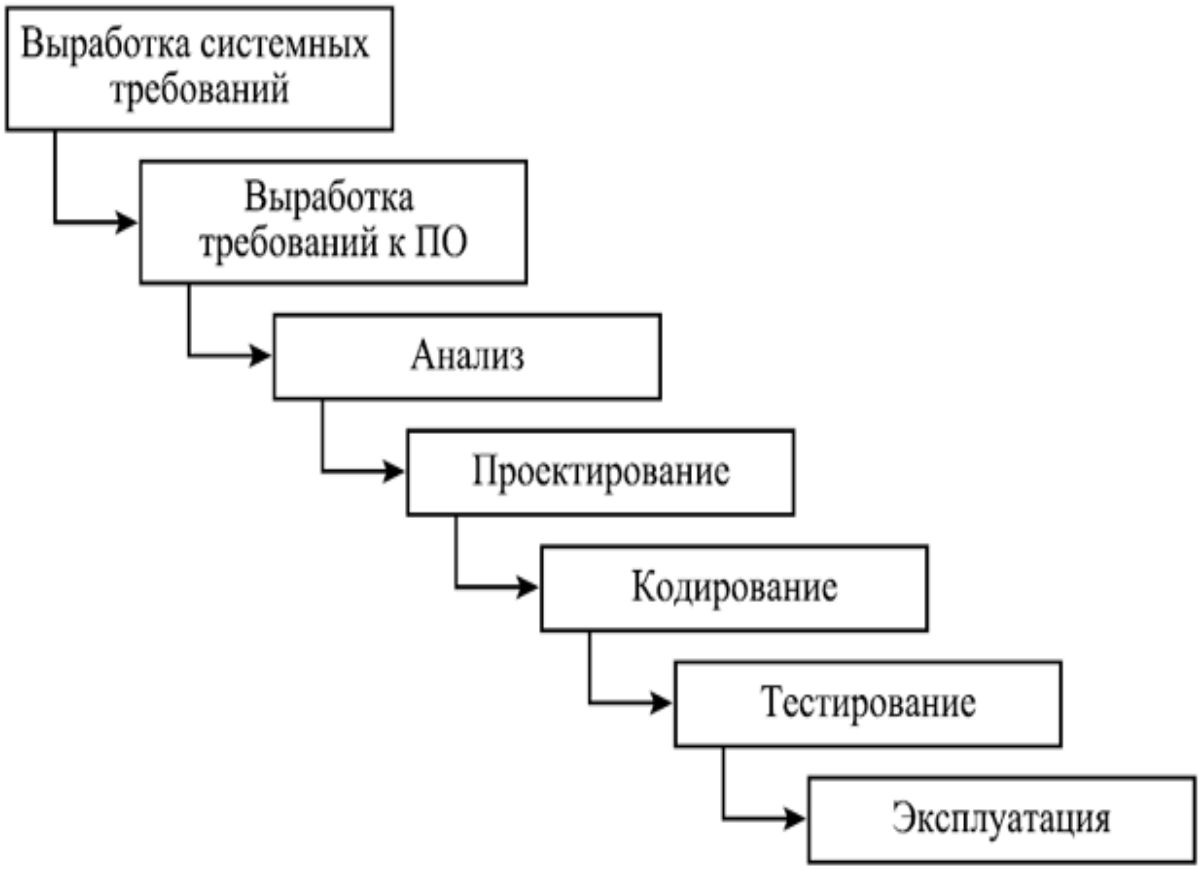


Рис. 18.1. Каскадна модель.

Ітеративна модель *(англ. iteration, «повторення»)* у розробці програмного забезпечення – це виконання робіт паралельно з безперервним аналізом отриманих результатів і коригуванням попередніх етапів роботи. Проект за цим підходом у кожній фазі розвитку проходить повторюваний цикл PDCA: Планування – Реалізація – Перевірка – Оцінка *(англ. plan-do-check-act cycle).*

Рис. 18.2. Ітеративна модель.

Гнучка методологія розробки (*англ. Agile software development, agile-методи)* – серія підходів до розробки програмного забезпечення, орієнтованих на використання ітеративної розробки, динамічне формування вимог ітеративної розробки, динамічне формування вимог і забезпечення їх реалізації в результаті постійної взаємодії всередині само організованих робочих груп, які складаються із спеціалістів різного профілю.

Рис. 18.3. Гнучка методологія розробки.

Спіральна модель, запропонована Баррі Боемом у 1986 році, стала істотним ривком у розумінні природи розробки ПЗ. Вона являє собою процес розробки програмного забезпечення, який поєднує у собі як проектування, так і постадійне прототипування з метою поєднання переваг висхідної та низхідної концепцій і робить наголос на початкових етапах життєвого циклу: аналізі і проектуванні. Відмінною ознакою цієї моделі є спеціальна увага до ризиків, які впливають на організацію життєвого циклу.

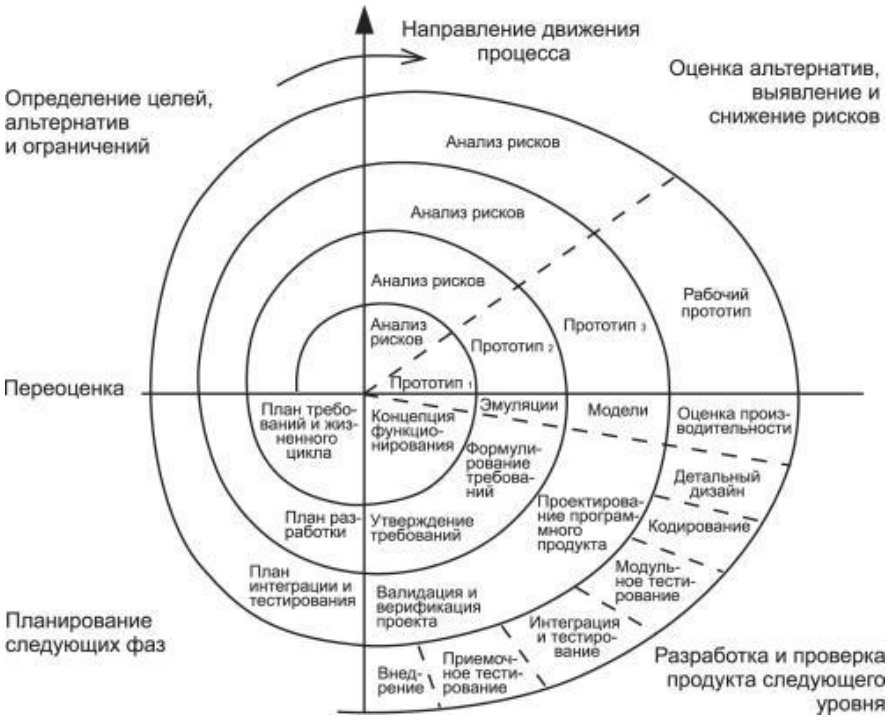


Рис. 18.4. Спіральна модель.

І, ***V-Model*** *(або VEE модель)* – модель розробки інформаційних систем (ІС), спрямована на спрощення розуміння труднощів, пов'язаних із розробкою систем. Вона використовується для визначення єдиної процедури розробки програмних продуктів, апаратного забезпечення і людино-машинних інтерфейсів.

Рис. 18.5. V-Model.

У V-моделі розробки ПЗ наочно видно, що тестувальник бере участь на всіх етапах життєвого циклу.

Подібність розробника і тестувальника, розмитість меж областей за наявності належної кваліфікації – головний показник інтересу у роботі тестувальника. Звичайно, можна стверджувати, що причетність до продукту і його якості у випадку тестування «чорного ящика» є мотиваційним фактором, але, як правило, людина, яка прийшла в технічну, інженерну галузь, зробила це свідомо з бажанням розвиватися і працювати над собою. Тому можливість розширювати свій кругозір, впливати на якість продукту *(хай навіть невеликих речей),* виявляти активність, а також накладати вето на випуск продукту у тестувальника, на нашу думку, більш вагомі аргументи за професію. У такому разі хороший тестувальник – не мавпа, а інженер-досліджувач, знавець, який бере участь на майже всіх етапах життєвого циклу ПЗ *(на відміну від ролей аналітиків, дизайнерів, розробників і технічних письменників)*, відповідальний за якість продукту *(не процесу, це до QA; під «якістю продукту» розумітимемо відповідність заявленим і затвердженим вимогам і стандартам, чек-листам тощо).* Отож, для підвищення якості продукту і розвитку себе як спеціаліста тест-

інженер повинен виявляти активну позицію. Він знає, як покращити продукт, знає, що таке «досить добре», і як його, продукт або процес, таким зробити, ініціювати рух як інженера-розробника, так й інших команд та навіть компаній.

Наприклад, у будь-яких проектах бувають моменти простоювання, який може допускатися з боку керівництва. Зазвичай у такий час співробітники вирішують особисті проблеми або займаються самоосвітою. Відносно роботи розробник може зайнятися рефакторингом, вивчати алгоритми і перенести їх на свої проекти. А що робити тестувальнику? Освоювати нові засоби тестування? Адже він нічого не робить без початкового пакету від розробників/аналітиків. Насправді він може зайнятися автоматизацією своєї діяльності або покращенням продуктів, які ним використовуються, навіть якщо він далекий від інфраструктури і розробки.

18.4 Актуальні проблеми розробки ПЗ

Під час розробки програмного забезпечення, як правило, виникають різного роду проблеми, які необхідно вирішити максимально швидко.

Найпоширенішими проблемами під час розробки ПЗ є:

- ✓ ***Недолік прозорості.*** У будь-який момент часу важко сказати, у якому стані перебуває проект і який відсоток його завершення. Ця проблема виникає при недостатньому плануванні структури *(або архітектури)* майбутнього програмного продукту. Під час планування слід врахувати, скільки часу займе розробка, які етапи, чи можна якісь етапи виключити або зекономити – наслідком цього процесу буде те, що етап проектування скорочується.

- ✓ ***Недостатній контроль.*** Без точної оцінки процесу розробки зриваються графіки виконання робіт і перевищуються встановлені бюджети. Складно оцінити обсяг виконаної роботи і роботи, що залишилася.

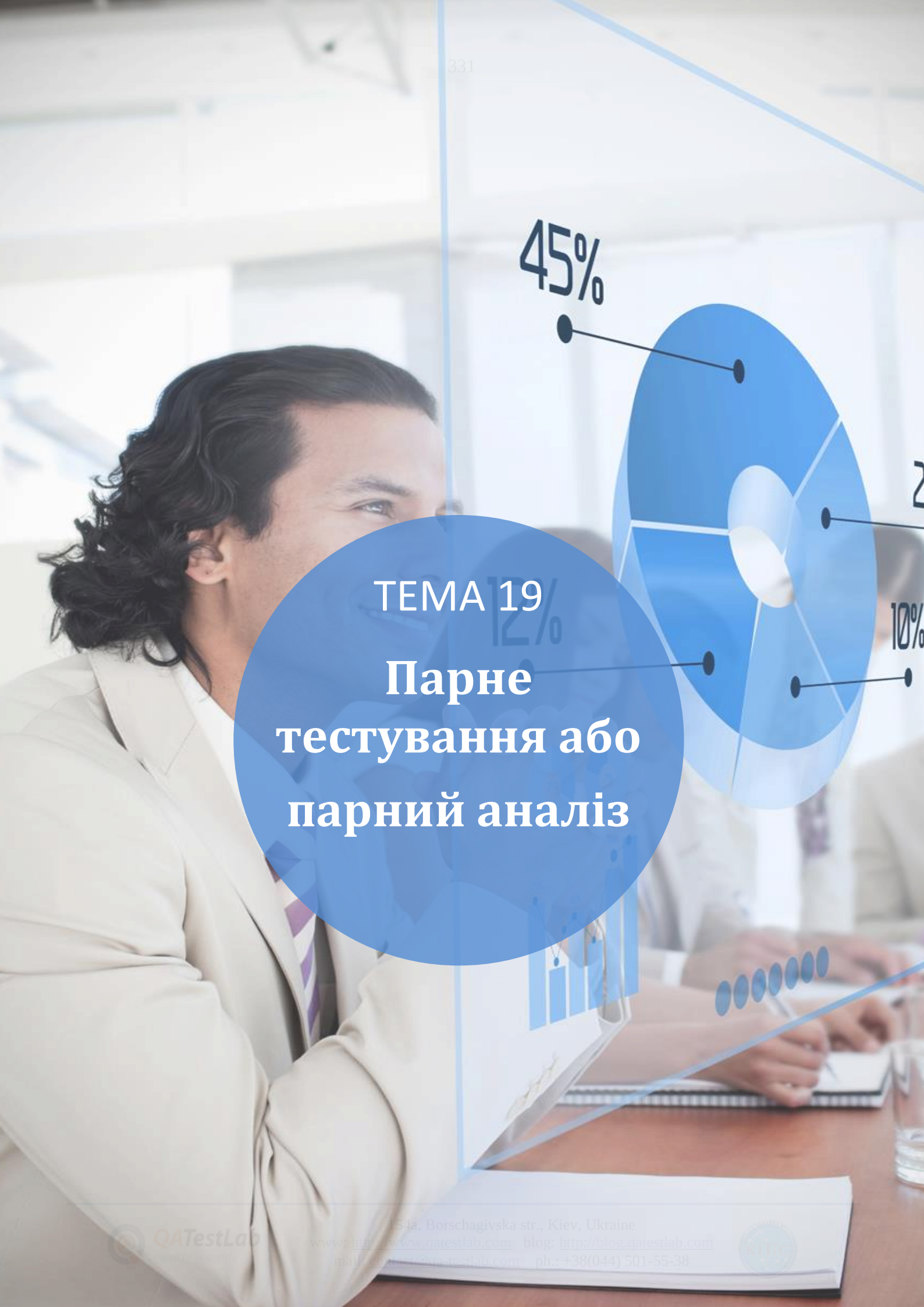
- ✓ ***Недостача моніторингу.*** Неможливість спостерігати хід розвитку проекту не дозволяє контролювати хід розробки у реальному часі.

- ✓ ***Неконтрольовані зміни.*** У замовника постійно виникають нові ідеї стосовно програмного забезпечення, яке розроблюється. Вплив змін може бути істотним для успіху проекту, тому важливо оцінювати

пропоновані зміни і реалізовувати тільки ті, які затверджені виконавцем. Ця проблема виникає внаслідок того, що замовник не продумав добре на етапі проектування програми її функціональне призначення *(операції, які вона виконуватиме тощо)*.

- ✓ ***Відладка програми***. Найскладніший етап – пошук і виправлення помилок у програмах, які обов'язково є. Ця проблема виникає з боку виконавця – у випадку допущених помилок під час розробки програми.

Висновки: Вивчення і повне усвідомлення того, яку роботу повинен виконати кожний учасник процесу розробки ПЗ у певний момент часу, позбавляє від можливих конфліктів під час реалізації проекту. Усі ролі описуються у плані розробки проекту (Software Development Plan). Цей документ дає розуміння учасникам проекту (проектній групі і замовнику) про те, яким чином ітиме робота у рамках проекту, які правила тощо. Таким чином, визначивши ролі в команді, план матиме успіх!



TEMA 19

Парне тестування або парний аналіз

19 ПАРНЕ ТЕСТУВАННЯ АБО ПАРНИЙ АНАЛІЗ (PAIRWISE TESTING)

У тестуванні програм дуже часто постає завдання перевірки комбінацій вхідних параметрів, від яких залежить підсумковий результат програми. Типовий приклад – діалогове вікно друку файлу: воно має безліч налаштувань, полів введення, різних взаємозалежних опцій, від включення або виключення яких підсумковий результат може дуже відрізнятися. Якби навіть всі опції мали тільки два режими роботи (вкл / викл), а всього опцій було б 10, то це вже дає $2^{10} = 1024$ їх комбінацій.

Для того, щоб переконатися у працездатності програми, в ідеалі потрібно перевірити всі тестові набори, що складаються з усіх можливих комбінацій параметрів, так як для однієї з них вона може працювати некоректно. Але, по-перше, таких тестових наборів може вийти досить багато, і процес перевірки їх всіх буде трудомістким. По-друге, при тестуванні зазвичай бажають одержати не поєднання всіх параметрів з усіма, адже в цьому випадку буде важче локалізувати дефект і відтворити проблему, а перевірити окремі пари значень параметрів, які можуть призвести до проблеми. Для спрощення підбору таких пар використовують методику Pairwise testing, яка дозволяє виділити комбінації унікальних пар, перевіряючи значення, й одночасно зменшити число тестових наборів у порівнянні з повним перебором [143].

19.1 Основні поняття та визначення

Існує кілька визначень терміну ***«попарне тестування» (pairwise testing або all-pairs analysis – попарний аналіз, аналіз всіх пар комбінацій):***

- техніка формування наборів тестових даних. Полягає в наступному: формуються такі набори даних, в яких кожне тестове значення кожного з перевірених параметрів хоча б раз поєднується з кожним тестовим значенням всіх інших параметрів [144];
- сучасна та ефективна методика тестування, грунтується на тому припущенні, що більшість дефектів виникає при взаємодії не більше двох факторів. Тестові набори генеруються при використанні даної методики, охоплюють всі унікальні пари комбінацій факторів, що вважається достатнім для виявлення більшого числа дефектів [142,143];
- методика оптимізації тестових даних з повного набору вхідних даних у системі, яка дозволяє істотно (на порядки) скоротити кількість тестів з імовірністю знаходження всіх дефектів – 97% (цифра взята з дослідження IBM) [141].

Парне тестування (pairwise testing) починається з вибору значень вхідних змінних, при цьому окремі значення вибираються шляхом розбиття області таким чином, щоб перекрити всі парні значення [145]. Не завжди зручно здійснити таке розбиття в ручну, на практиці для попарного тестування (pairwise testing) використовують спеціальні алгоритми та інструментарії [149], засновані на:

- ортогональній побудові масивів, яка спирається на теоретичні дослідження в області комбінаторних алгоритмів, алгоритмів дискретної математики, зокрема, латинських квадратів (див., наприклад, М. Хол "Комбінаторика", М .: Мир, 1970, глава 13, стор. 261);
- алгоритмі економного просування *(greedy algorithms)*;
- методах, запозичених з розробки статистичних даних [145-148].

Ортогональний масив (ортогональна таблиця) – це таблиця $L_m(k^n)$, де m – число рядків, n – число стовпців (по числу вхідних параметрів), k – кількість варіантів для значень елементів таблиці, які володіють наступними властивостями:

Будь-які два стовпця таблиці містять всі комбінації значень цих стовпців.

Якщо яка-небудь пара значень двох стовпців зустрічається кілька разів, то всі можливі парні комбінації значень цих стовпців повинні зустрітися стільки ж разів.

У ортогональних масивах не обов'язково всі стовпці повинні мати однакову кількість значень. Існують так звані змішані (mixed) ортогональні масиви. Наприклад:

$L_4(2^3)$ – ортогональний масив з чотирма рядками, трьома стовпцями (за кількістю змінних), 2 означає, що всі змінні приймають тільки два значення – 1 та 2.

$L_{18}(2^1 3^7)$ – змішаний ортогональний масив з вісімнадцятьма рядками, у якого один стовпець зі значеннями 1 та 2 й сім стовпців зі значеннями 1, 2, 3.

Для тестування з використанням ортогональних масивів виконують наступні кроки:

1. Визначають змінні для вхідних даних у комбінаціях. Наприклад, це можуть бути назви опцій, параметрів налаштування, допустимих конфігурацій устаткування та ін.
2. Визначають значення, які можуть приймати змінні. Наприклад, конкретні назви пунктів меню, числові значення, назви операційних систем або баз даних та ін.

3. Будують ортогональний масив, який має стовпець для кожної змінної. Зазвичай для цього використовується будь-яка програма, наприклад, STATISTICA.

Кожен рядок побудованого масиву інтерпретується як єдина комбінація значень змінних для одного тестового випадку.

All-Pairs Algorithm (алгоритм всіх пар) – це комбінаторна методика, яка була спеціально створена для попарного тестування. В її основі лежить вибір можливих комбінацій значень всіх змінних, в яких містяться всі можливі значення для кожної пари змінних. Виходячи з визначення, при цьому буде отримано менше число комбінацій, ніж при використанні ортогональних масивів.

Для тестування з використанням All-Pairs алгоритму виконують наступні кроки:

1. Аналогічно, як для ортогональних масивів, визначають таблицю всіх змінних та їх значень.
2. Залишають у таблиці лише всі можливі унікальні комбінації пар значень змінних [143].

Випадки, коли попарне тестування (pairwise testing) може бути невдалим [145]:

- коли були обрані неправильні парні значення для тестування;
- коли відсутня достатня кількість передбачених значень;
- коли високо ймовірні комбінації одержують занадто мало уваги;
- коли не знаєте, як взаємодіють змінні.

Необхідно пам'ятати, що попарне тестування (pairwise testing) використовується лише на пізніх етапах розробки, або в доповненні з основними функціональними тестами. Наприклад, якщо проводити конфігураційне тестування, то перш ніж використовувати парне тестування слід переконатися, що основний сценарій функціонує на всіх операційних системах з параметрами за замовчуванням (BVT). Це значно полегшить локалізацію майбутніх багів, адже при парному тестуванні в одному тесті фігурує безліч параметрів зі значеннями не за замовчуванням, кожен з яких може стати причиною збою, та його локалізація в цьому випадку досить важка. А в разі провалу BVT слід відмовитися від використання методу парного тестування, так як багато тестів будуть провальними, а виключення навіть одного тесту тягне за собою втрату, як правило, кількох пар, та сенс використання методу втрачається. Тому метод слід використовувати лише на стабільному функціоналі, коли поточні тести вже втрачають свою ефективність.

19.2 Приклади застосування попарного тестування (pairwise testing)

19.2.1 Попарне тестування та тестування ортогональними масивами

Розглянемо наступну ситуацію:

Сайт повинен працювати в 8 браузерах – Internet Explorer 6, 7, і 8, Netscape 6.0 і 7.0, Mozilla 2 і 3, і Opera 9; використовуючи плагіни – RealPlayer, MediaPlayer, без плагінів; на ОС – Windows NT, 2003, XP, Vista, 2008 і Seven; на різних веб серверах – IIS, Apache, і WebLogic; запущених на сервері з різними ОС – Windows 2003, 2008 і Linux.

- 8 браузерів;
- 3 плагіна;
- 6 ОС клієнта;
- 3 веб-сервера;
- 3 ОС сервера.

Разом: 1,296 можливих комбінацій.

Розглянемо суть основних кроків при побудові ортогонального масиву:

1. Визначити змінні (кіл-ть вхідних даних)

Важливо: необхідно вибирати вхідні дані, будь-які комбінації значень яких, можуть логічно існувати. Наприклад, якщо потрібно перевірити у браузерах: IE та FF й операційних системах Linux й Windows, то явно не зможемо перевірити роботу сайту під Linux в IE (ну або це позбавлено логічного сенсу). У цьому випадку необхідно зробити 2 ортогональних масива з браузерами для Linux та Windows.

2. Визначити кількість значень, що може приймати кожна змінна

Важливо: До моменту визначення кількості значень вже повинні застосувати всілякі техніки тест-дизайну для зменшення кількості значень – класи еквівалентності, граничні значення та ін. Наприклад, якщо змінна може приймати значення від 1 до 100, то ортогональний масив, в якому 1 стовпець може приймати 100 значень, дуже суттєво розростеться, у порівнянні з ортогональним масивом, де розбили 100 значень на класи еквівалентності, й тепер ця змінна може приймати, скажімо, 5 значень – 5 діапазонів класів еквівалентності.

3. Визначити ортогональний масив, у якого буде стовпець для кожної змінної (кожен стовпець ортогонального масиву має стільки ж варіантів значень, скільки має ваша змінна).

4. Спроектуйте (тар) задачу тестування на ортогональний масив.

Важливо: кроки 3 та 4 успішно роблять різні утиліти, наприклад, як http://www.testersdesk.com/pairwse_testersdesk.html, всього лише потрібно ввести, які вхідні дані можуть бути, наприклад, так:

Journey mode: Return trip, One way
Preferred time: Morning, Afternoon, Evening, Late night
Leaving from: Type1_City,Type2_City,Type3_City
Going to: Type1_City,Type2_City,Type3_City
Booking type: Economy, First class
Customer type: Registered, New
Adults: 1, 2, 3, 4, 5, 6
Children: 0, 1, 2, 3, 4

Все інше утиліта зробить сама, й залишиться перейти до 5 кроку.

5. Побудувати тест-кейси.

Застосуємо кроки для побудови ортогонального масиву до раніше наведеного прикладу:

1. Визначте змінні (кіл-ть вхідних даних)

- браузери;
- плагіни;
- ОС клієнта;
- веб-сервісу;
- ОС сервера.

2. Визначити кількість значень, що може приймати кожна змінна

- 8 браузерів – Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7.
- 3 плагіна – RealPlayer, MediaPlayer, без плагінів.
- 6 ОС клієнта – Windows 95, 98, ME, NT, 2000, and XP.
- 3 веб-сервера – IIS, Apache и WebLogic..
- 3 ОС сервера – Windows NT, 2000, and Linux.

3. Визначити ортогональний масив, у якому буде стовпець для кожної змінної

Який розмір необхідний? Для початку ортогональний масив повинен бути з 5 стовпцями, кожен стовпець для кожної змінної:

- перший стовпець повинен приймати значення від 1 до 8 – браузерів;
- другий стовпець повинен приймати значення від 1 до 3 – плагін;
- третій стовпець повинен приймати значення від 1 до 6 – ОС клієнта;
- четвертий та п'ятий стовпці повинні приймати значення від 1 до 3 – веб-серверів і ОС серверів.

4. Спроектуйте задачу тестування на ортогональний масив.

Значення браузера проектуються на перший стовпець ортогонального масиву:

1-й стовпець – браузери:

- 1 = Internet Explorer 6;
- 2= Internet Explorer 7;
- 3= Internet Explorer 8;
- 4= Netscape 6;
- 5= Netscape 7;
- 6= Mozilla 2;
- 7= Mozilla 3;
- 8= Opera 9.

Таким же чином проектуємо кожну змінну (вхідний параметр).

2-й стовпець – плагіни:

- 1 = None;
- 2= RealPlayer;
- 3= MediaPlayer;
- 4= 4, 4-е значення плагінів відсутнє, тому залишаємо "4";

3-й стовпець – ОС клієнта:

- 1 = Windows 95;
- 2= Windows 98;
- 3= Windows ME;
- 4= Windows NT;
- 5= Windows 2000;
- 6= Windows XP;
- 7= 7, 7-е значення – ОС клієнта відсутня, тому залишаємо "7";
- 8= 8, 8-е значення – ОС клієнта відсутня, тому залишаємо "8";

4-й стовпець – веб-сервери

- 1 = IIS;
- 2= Apache;
- 3= WebLogic;

- 4= 4. 4-е значення – призначення веб-серверів відсутнє, тому залишаємо "4";

5-й стовпець – ОС сервера:

- 1 = Windows 2003;
- 2= Windows 2008;
- 3= Linux;
- 4= 4, 4-е значення – ОС сервера відсутня, тому залишаємо "4".

Результуючий ортогональний масив (табл. 19.1).

Таблиця 19.1.Результуючий ортогональний масив

№	Browser	Plug-in	Client	Server	Server OS
1	IE 5.0	None	Win	IIS	Win NT
2	IE 5.0	4	Win	4	4
3	IE 5.0	4	Win	4	4
4	IE 5.0	None	Win	IIS	Win NT
5	IE 5.0	MediaPlayer	Win	WebLogic	Linux
6	IE 5.0	RealPlayer	7	Apache	Win 2000
7	IE 5.0	RealPlayer	Win	Apache	Win 2000
8	IE 5.0	MediaPlayer	8	WebLogic	Linux
9	IE 6.0	4	Win	WebLogic	Linux
10	IE 6.0	None	Win	Apache	Win 2000
11	IE 6.0	None	Win	Apache	Win 2000
12	IE 6.0	4	Win	WebLogic	Linux
13	IE 6.0	RealPlayer	Win	IIS	Win NT
14	IE 6.0	MediaPlayer	7	4	4
15	IE 6.0	MediaPlayer	Win	4	4
16	IE 6.0	RealPlayer	8	US	Win NT
17	IE 5.5	MediaPlayer	Win	Apache	Win NT
18	IE 5.5	RealPlayer	Win	WebLogic	4
19	IE 5.5	RealPlayer	Win	WebLogic	4
20	IE 5.5	MediaPlayer	Win	Apache	Win NT
21	IE 5.5	None	Win	4	Linux
22	IE 5.5	4	7	IIS	Win 2000
23	IE 5.5	4	Win	IIS	Win 2000
24	IE 5.5	None	8	4	Linux
25	Net 6.0	RealPlayer	Win	4	Linux
26	Net 6.0	MediaPlayer	Win	IIS	Win 2000
27	Net 6.0	MediaPlayer	Win	IIS	Win 2000
28	Net 6.0	RealPlayer	Win	4	Linux
29	Net 6.0	4	Win	Apache	Win NT
30	Net 6.0	None	7	WebLogic	4
31	Net 6.0	None	Win	WebLogic	4
32	Net 6.0	4	8	Apache	Win NT
33	Net 6.1	RealPlayer	Win	4	Win 2000
34	Net 6.1	MediaPlayer	Win	IIS	Linux
35	Net 6.1	MediaPlayer	Win	IIS	Linux
36	Net 6.1	RealPlayer	Win	4	Win 2000

37	Net 6.1	4	Win	Apache	4
38	Net 6.1	None	7	WebLogic	Win NT
39	Net 6.1	None	Win	WebLogic	1 Win NT
40	Net 6.1	4	8	Apache	4
41	Moz 1.1	MediaPlayer	Win	Apache	4
42	Moz 1.1	RealPlayer	Win	WebLogic	Win NT
43	Moz 1.1	RealPlayer	Win	WebLogic	Win NT
44	Moz 1.1	MediaPlayer	Win	Apache	4
45	Moz 1.1	None	Win	4	Win 2000
46	Moz 1.1	4	7	IIS	Linux
47	Moz 1.1	4	Win	IIS	Linux
48	Moz 1.1	None	8	4	Win 2000
49	Net 7.0	4	Win	WebLogic	Win 2000
50	Net 7.0	None	Win	Apache	Linux
51	Net 7.0	None	Win	Apache	Linux
52	Net 7.0	4	Win	WebLogic	Win 2000
53	Net 7.0	RealPlayer	Win	IIS	4
54	Net 7.0	MediaPlayer	7	4	Win NT
55	Net 7.0	MediaPlayer	Win	4	Win NT
56	Net 7.0	RealPlayer	8	IIS	4
57	Opera 7	None	Win	IIS	4
58	Opera 7	4	Win	4	Win 2000
59	Opera 7	4	Win	4	Linux
60	Opera 7	None	Win	IIS	Linux
61	Opera 7	MediaPlayer	Win	WebLogic	Win 2000
62	Opera 7	RealPlayer	7	Apache	Win 2000
63	Opera 7	RealPlayer	Win	Apache	Win 2000
64	Opera 7	MediaPlayer	8	WebLogic	Linux

Для непризначених значень (4 = 4,7 = 7 та ін.) вибираються будь-які валідні значення для змінної та записуються в таблицю. Тепер можна переходити до складання тест-кейсів [150].

19.2.2 Тестування з використанням all-pairs algorithm

Розглянемо наступну ситуацію [151]:

Представлена форма для тестування, яка містить:

- список – може приймати значення від 0 до 9 включно;
- поле введення – може приймати цілочисельні значення від 1 до 99 включно;
- checkbox $_1$ – приймає значення "on" або "off";
- checkbox $_2$ – приймає значення "on" або "off".

У результаті отримуємо $99 \cdot 10 \cdot 2 \cdot 2 = 3\,960$ можливих валідних комбінацій. Так як отримана кількість комбінацій значна, необхідно зменшити кількість потрібних комбінацій. Для цього можемо застосувати

комбінаторний спосіб All-pairs algorithm, суть якого полягає в тому, що здійснюється тестування тільки комбінації значень кожної пари змінних.

Для початку необхідно визначити, яка кількість вхідних параметрів (змінних) є в наявності, та які значення може приймати кожен вхідний параметр.

Таблиця 91.2.

List	TextBox	Checkbox $_1$	Checkbox $_2$
0	1	on	on
1	2	off	off
2	3		
3	4		
4	…		
5	96		
6	97		
7	98		
8	99		
9			

Якщо можна спростити значення змінних, то обов'язково необхідно це зробити. Наприклад, поле введення може приймати значення від 1 до 99. Розуміючи бізнес-логіку додатку, можемо розбити значення 1 до 99 на класи еквівалентності та використовувати їх вже, як можливі значення для комбінаторики. Або можемо використовувати техніку граничних значень. Як би там не було, у розглянутому прикладі для простоти зменшимо кількість можливих значень до 3-х – валідність цілочисельне значення, невалідне цілочисельне значення – символи. Тепер зменшимо кількість можливих значень для списку до 2-х – 0 і будь-яке інше значення. Після зменшимо кількість можливих значень, вхідні дані мають вигляд:

Таблиця 19.3.

List	TextBox	Checkbox $_1$	Checkbox $_2$
0	Valid int	on	on
any other	Invalid int	off	off
	Alpha chars		

Тепер необхідно змінити порядок стовпців таблиці так, щоб змінна з найбільшою кількістю значень була першою, а з найменшою кількістю значень – останньої.

Таблиця 19.4.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)

Кожен рядок таблиці представлятиме унікальний набір вхідних даних для тесту. Необхідно звернути увагу на кількість значень, які може

приймати змінна другого стовпця (список), а саме 2 значення. Таким чином, стільки разів потрібно вставити всі значення першого стовпця.

Таблиця 19.5.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)
Valid Integer			
Valid Integer			
…			
Invalid Integer			
Invalid Integer			
…			
Alpha chars			
Alpha chars			

Було додано 6 рядків. 3 можливих значення поля введення (тобто першого стовпця) значення повторюються двічі. Так само пропустили по рядку після кожного дубля значення.

Виконуємо заповнення другого стовпця: для кожного дубля значень стовпця 1 заповнюємо обидва значення стовпця 2.

Таблиця 19.6.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)
Valid Integer	0		
Valid Integer	any other		
…			
Invalid Integer	0		
Invalid Integer	any other		
…			
Alpha chars	0		
Alpha chars	any other		

Після заповнення табл.19.6 отримали всі можливі комбінації пар значень першої та другої змінної. Переходимо до розгляду третьої змінної, приймає 2 значення – "on" та "off". Для кожного дубля значень стовпця 1 заповнюємо обидва значення стовпця 3 (табл. 19.7).

Таблиця 19.7.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)
Valid Integer	0	on	
Valid Integer	any other	off	
…			
Invalid Integer	0	on	
Invalid Integer	any other	off	
…			
Alpha chars	0	on	
Alpha chars	any other	off	

Необхідно перевірити:

- чи є всі можливі комбінації значень між першим та другим стовпцем – є;
- є всі можливі комбінації значень між другим та третім стовпцем – відсутні комбінації {0, off} і {any other, on} (табл. 19.8).

Таблиця 19.8.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)
Valid Integer	0	on	
Valid Integer	any other	off	
…			
Invalid Integer	0	off	
Invalid Integer	any other	on	
…			
Alpha chars	0	on	
Alpha chars	any other	off	

Перевірити, чи є всі можливі комбінації значень першої, другої та третьої змінної. Необхідно визначитися з четвертою змінною – Checkbox $_2$, яка теж може мати 2 можливих значення. Необхідно підставити в таблицю ці значення так, щоб усі можливі комбінації значень кожної пари змінних існували (табл. 19.9).

Таблиця 19.9.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)
Valid Integer	0	on	Checked
Valid Integer	any other	off	Unchecked
…			
Invalid Integer	0	off	Checked
Invalid Integer	any other	on	Unchecked
…			
Alpha chars	0	on	Unchecked
Alpha chars	any other	off	Checked

Необхідно перевірити:

- є всі можливі комбінації першого і четвертого стовпців – є;
- є всі можливі комбінації другого і четвертого стовпців – є;
- є всі можливі комбінації третього і четвертого столбців – є.

Даний підхід дозволяє створити ***6 тест-кейсів***, коли всього існує ***24 можливі комбінації.***

Однак ще залишилися порожні рядки, для того щоб зрозуміти необхідність у цих рядках, потрібно ускладнити приклад. Додомо ще 2 checkbox (2 бінарних вхідних параметра) – checkbox $_3$ та checkbox $_4$. Продовжимо будувати тестові дані на основі all-pairs algorithm, при цьому додаємо значення для п'ятого стовпця (табл. 19.10).

Таблиця 19.10.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)	Checkbox $_3$(2)	Checkbox $_4$(2)
Valid Integer	0	on	Checked	Yes	
Valid Integer	any other	off	Unchecked	No	
…					
Invalid Integer	0	off	Checked	No	
Invalid Integer	any other	on	Unchecked	Yes	
…					
Alpha chars	0	on	Unchecked	No	
Alpha chars	any other	off	Checked	Yes	

Тепер необхідно перевірити, чи є всі можливі комбінації після заповнення 5-го стовпця. Після перевірки заповнюємо останній стовпець (табл. 19.11).

Таблиця 19.11.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)	Checkbox $_3$(2)	Checkbox $_4$(2)
Valid Integer	0	on	Checked	Yes	1
Valid Integer	any other	off	Unchecked	No	2
…					
Invalid Integer	0	off	Checked	No	2
Invalid Integer	any other	on	Unchecked	Yes	1
…					
Alpha chars	0	on	Unchecked	No	1
Alpha chars	any other	off	Checked	Yes	2

Необхідно перевірити:

- є всі можливі комбінації першого та п'ятого стовпців – є;
- є всі можливі комбінації другого та шостого стовпців – є;
- є всі можливі комбінації третього та шостого стовпців – відсутні комбінації значень {on, 2} та {off, 1};
- є всі можливі комбінації четвертого та шостого стовпців – є;
- є всі можливі комбінації п'ятого та шостого стовпців – є.

У таких випадках й застосовуються зайві залишені рядки, в які можна додати не вистачаючі комбінації третього та шостого стовпців. Значення решти змінних для доданих рядків можна вибрати будь-які, бо всі комбінації значень кожної пари вже є (табл. 19.12).

Таблиця 19.12.

TextBox(3)	List(2)	Checkbox $_1$(2)	Checkbox $_2$(2)	Checkbox $_3$(2)	Checkbox $_4$(2)
Valid Integer	0	on	Checked	Yes	1
Valid Integer	any other	off	Unchecked	No	2
Invalid Integer	any other	***on***	checked	yes	***2***
Invalid Integer	0	off	Checked	No	2
Invalid Integer	any other	on	Unchecked	Yes	1

invalid integer	any other	***off***	unchecked	no	***1***
Alpha chars	0	on	Unchecked	No	1
Alpha chars	any other	off	Checked	Yes	2

Таким чином, всі можливі комбінації були створені, у результаті чого отримали ***8 комбінацій замість 96***.

Для спрощення роботи були розроблені спеціальні інструментарії, так як вручну перебирати всі пари є достатньо трудомістким процесом.

19.3 Програмна реалізація All-Pairs алгоритму для Pairwise testing

Існує безліч інструментів для реалізації техніки попарного аналізу, наприклад, список таких інструментаріїв представлений за посиланням http://www.pairwise.org/tools.asp

У деяких командах автоматизаторів при розробці однієї з систем тестування широко використовується Python. Розглянемо приклад використання у своїх скриптах генератора комбінацій All-Pairs, розробленого MetaCommunications Engineering для Python, що реалізує однойменну методику. Завантажити можна за посиланням http://sourceforge.net/apps/trac/allpairs/

Після стандартної установки бібліотеки All-Pairs можна скопіювати в проект каталог <PYTHON_HOME> \ Lib \ site-packages \ metacomm \ для спрощення імпорту та перенесення проекту на інші комп'ютери.

Розглянемо одну задачу з раніше згаданих, при цьому трохи видозмінимо умову задачі. Нехай потрібно перевірити працездатність веб-сайту в різних конфігураціях браузерів, операційних системах та на платформах з різною розрядністю. Наприклад, задані чотири конкретні операційні системи: Windows XP SP 3, Windows 7 SP 1, Debian 7.1, Ubuntu 12.04 для двох розрядностей: x86, x64, і три браузери: Chrome, Firefox, Safari.

При цьому отримуємо: $4 \cdot 2 \cdot 3 = 24$ – конфігурації для тестових наборів, які потрібно перевірити в загальному випадку. Можна зменшити число тестових наборів за допомогою All-Pairs.

Вхідні дані для модуля All-Pairs можуть бути задані як перелік списків з рядків, значення яких однозначно характеризують параметри для тестових наборів. Для зазначеної задачі це можуть бути просто назви параметрів для конфігурацій:

- список можливих ОС:

Windows XP SP 3, Windows 7 SP 1, Debian 7.1, Ubuntu 12.04;

- список розрядностей:

x86, x64;

- список браузерів:

Chrome, Firefox, Safari.

Завдання вхідних параметрів у Python-коді та використання All-Pairs для побудови їх комбінацій може виглядати як на лістингу 19.1. (передбачається, що каталог ext містить пакет metacomm).

```
# -*- coding: utf-8 -*-
#
# (c) Gilmullin Timur Mansurovich,  2013.

# Using AllPairs by MetaCommunications Engineering example.

from ext.metacomm.combinatorics import all_pairs2 as ap

def Generator(parameters):
    """
    Use generator which work on AllPairs-algorithm.
    Input:
    parameters - list of list of different parameters, that looks like this:
    [ [par1_val_1, par1_val_2, ...],
      [par2_val_1, par2_val_2, ...], ... ]
    Output:
    list of enumerate possible unique combinations, that looks like this:
    [ (0, [par1_val_1, ..., parK_val_K]), ...
      (N, [parX_val_X, ..., parY_val_Y]), ...]
    """
    combinations = list(enumerate(ap.all_pairs2(parameters)))
    return combinations


if __name__ == '__main__':
    # define list of list of different parameters:
    inputData = [
        ['Windows XP SP 3', 'Windows 7 SP 1', 'Debian 7.1', 'Ubuntu 12.04'],
        ['x86', 'x64'],
        ['chrome', 'firefox', 'safari']
        ]
    # get combinations and output it:
    outputData = Generator(inputData)
    with open('output.txt', 'w') as fH:
        for line in outputData:
            stringData = 'Combination {}:\t{}'.format(str(line[0]), str(line[1]))
            print(stringData)
            fH.write(stringData + '\n')
```

Лістинг 19.1 Код скрипта allpairs.py:

У лістенгу кожний новий рядок змінної inputData містить список допустимих значень для окремого параметра в тестових конфігураціях. Після визначення таким чином вхідних даних потрібно просто виконати скрипт командою: python allpairs.py

На виході буде отриманий пронумерований список рядків виду:

Combination 0: ['Windows XP SP 3', 'x86', 'chrome'] ...

Як бачимо, комбінацій всього 12 – у два рази менше, ніж їх повне число в загальному випадку (рис.19.1).

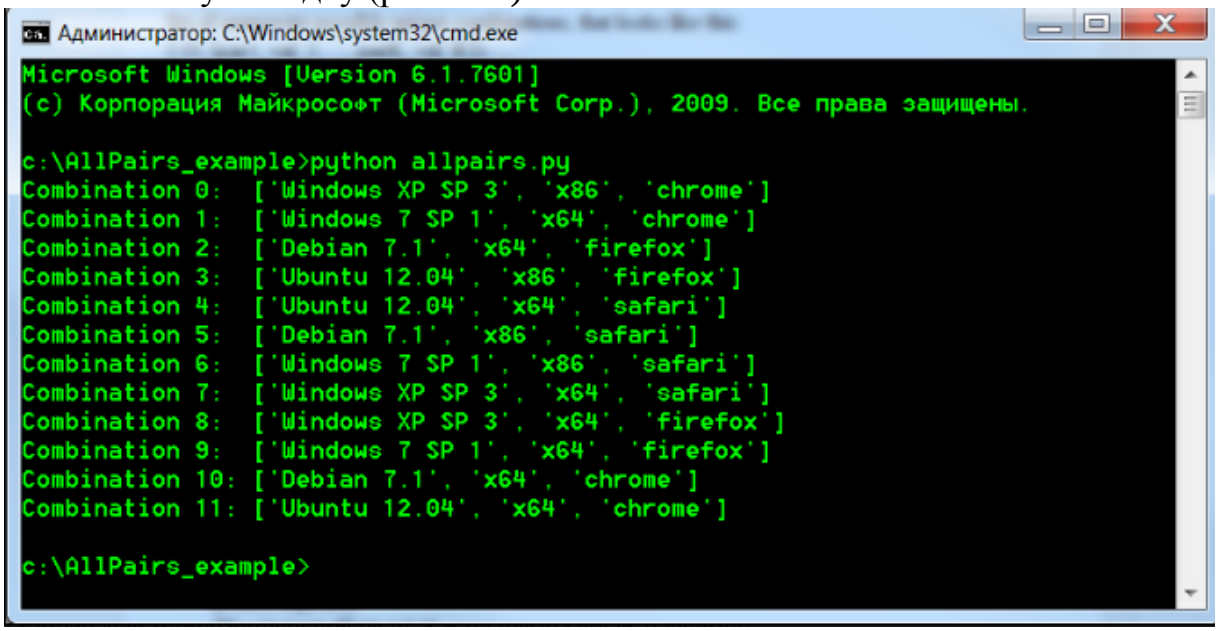


Рис. 19.1.

Потім отримані комбінації можна інтерпретувати як тестові випадки, наприклад:

"Виконати перевірку працездатності веб-сайту для конфігурації:

- ОС: Windows XP SP 3;
- розрядність системи: x86;
- браузер: Chrome."

Завантажити код скрипта allpairs.py для даної задачі можна за посиланням:https://drive.google.com/file/d/0B-1rf8K04ZS5ZHJpdXRRd24zc0E/edit?usp=sharing

код на GitHub:
https://github.com/Tim55667757/AllPairs_example

Прикладений архів включає в себе каталог metacomm з бібліотекою All-Pairs, модуль allpairs.py з прикладом вирішення наведеного вище завдання та виведення результатів у файлі output.txt.

Перевизначаючи у скрипті allpairs.py значення параметра inputData аналогічним чином, після його відпрацювання можна отримати оптимальні комбінації тестових наборів для всіх подібних завдань. А з метою подальшої автоматизації тестування можна використовувати генеруючі дані, наприклад, для запуску Автотест з потрібними параметрами конфігурації [143].

Ще один інструментарій, передбачений для реалізації All-Pairs, доступний за на сайті www.satisfice.com. All-pairs calculator дозволяє побудувати всі пари змінних по all-pairs algotithm.
Для цього необхідно:

1. Побудувати таблицю з усіма змінними та значеннями. (зробити це найлегше в Excel та зберегти документ як txt-файл).

Наприклад (табл. 19.3):

List	TextBox	Checkbox 1	Checkbox 2
0	Valid int	on	on
any other	Invalid int	off	off
	Alpha chars		

2. Запустити all-pairs calculator з командного рядка "allpairs input.txt> output.txt", де input.txt – складена таблиця в Excel, а output.txt – файл, в який хочете зберегти висновок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Портал знань [Електронний ресурс] – Режим доступу: http://www.znannya.org/?view=software-testing-testing.
2. Тестування Дот Ком, або Посібник по жорстокому поводженню з багами в інтернет-стартапах. – М.: Дело, 2007. – 312 с.
3. Система відслідковування помилок [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Система_отслеживания_ошибок.
4. Хроніки детермінованості [Електронний ресурс] – Режим доступу: http://svyatoslav.biz/education/bug-reports-summary/.
5. Bugs Catcher. Thinking about high quality testing [Electronic resource] – Access mode: http://bugscatcher.net/archives/3307.
6. Історія розвитку тестування програмного забезпечення. [Електронний ресурс] – Режим доступу: https://social.msdn.microsoft.com/Forums/ru-RU/531410fb-fc1e-4f1d-bb9c-0804970bb949/-?forum=fordesktopru.
7. Професія тестувальник [Електронний ресурс] – Режим доступу: http://enjoy-job.ru/professions/testirovschik/
8. Web testing [Electronic resource] –Access mode: http://en.wikipedia.org/wiki/Web_testing.
9. Тестування WEB-додатків [Електронний ресурс] – Режим доступу: http://taac.org.ua/files/a2011/proceedings/UA-1-Viktoria%20 Valeriivna%20Gurkivska-173.pdf.
10. Software Testing Help. “Web Testing: Complete guide on testing web applications” [Electronic resource] – Access mode: http://www.softwaretestinghelp.com/web-application-testing/.
11. Web-testing [Electronic resource] – Access mode: http://www.edb.utexas.edu/minliu/multimedia/PDFfolder/WebTestingPadolina.pdf.
12. Software Testing Help. “Entries Tagged 'Cookie Testing. Website Cookie Testing, Test cases for testing web application cookies?” [Electronic resource] – Access mode: http://www.softwaretestinghelp.com /category/cookie-testing/
13. Матеріали для проведення онлайн-тренінгу "Основи тестування програмного забезпечення" компанії QATestLab.
14. CyD Software Labs “Автоматичне тестування безпеки WEB-сайту” [Електронний ресурс] – Режим доступу: http://russia.cydsoft.com/products.php?helpid=118&product=19
15. Огляд рішень для тестування сайтів, різні види тестів для веб-додатків і вживане ПЗ [Електронний ресурс] – Режим доступу: http://hostinfo.ru/articles/442.
16. Дизайнери настрою. "Золотий перетин у веб-дизайні" [Електронний ресурс] – Режим доступу: http://jarlex.com/article/zolo toe-sechenie-v-veb-dizajne/

17. КоЛай. “Функціонали сайту” [Електронний ресурс] – Режим доступу: http://colain.ru/2011-03-17-15-11-14.html
18. UCGuide. “Модулі uCoz та їх завдання” [Електронний ресурс] – Режим доступу: http://uguide.ru/news/moduli_ucoz/2014-07-11-119.
19. TADVISER. “Тестування програмного продукту” [Електронний ресурс] – Режим доступу: http://www.tadviser.ru/index.php/ Статті: Тестирование_программного_продукта.
20. seoprofy.ua “Что такое юзабилити?” [Електронний ресурс] – Режим доступу: http://seoprofy.ua/blog/wiki/chto-takoe-usability
21. protesting.ru “Юзабилити тестирование” [Електронний ресурс] – Режим доступу: http://www.protesting.ru/testing/types/usability.html
22.https://ru.wikipedia.org/wiki/%D0%AE%D0%B7%D0%B0%D0%B1%D0%B8%D0%BB%D0%B8%D1%82%D0%B8-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5
23. sitechco.ru “Что такое чек-лист?” [Електронний ресурс] – Режим доступу: http://sitechco.ru/2011/08/chto-takoe-chek-list/
24. Usabilitylab “Юзабилити тестирование” [Електронний ресурс] – Режим доступу: http://usabilitylab.ru/services/usability-testing/
25. Nielsen Norman Group “Thinking Aloud: The #1 Usability Tool” [Electronic resource] – Access mode: http://www.nngroup.com/ articles/thinking-aloud-the-1-usability-tool/
26. Хабрахабр “Краткая история юзабилити“ [Електронний ресурс] – Режим доступу: http://habrahabr.ru/company/uidesign/blog/172713/
27. Круг С. “Веб-дизайн: книга Стива Круга или «не заставляйте меня думать!»”, 2-е издание. – Пер. с англ. – СПб: Символ Плюс, 2008. – 224 с.: цв. ил.
28. Гаврилюк О. “Как сделать сайт удобным. Юзабилити по методу Стива Круга” [Електронний ресурс] – Режим доступу: http://www.aweb.com.ua/seo-blog/steve-krug-usability-testing/
29. Нестерова Т. “Методы юзабилити-тестирования сайта” [Електронний ресурс] – Режим доступу: http://shikalakula.livejournal.com/3687.html
30. “Основы тестирования юзабилити сайта”[Електронний ресурс] – Режим доступу: http://web-atelier-g12.eu/index.php?option=com_ content&view=article &id=10:2011-09-17-13-42-19&catid=2
31. С. Ф. Сергеев “Методы тестирования и оптимизации интерфейсов информационных систем” [Електронний ресурс] – Режим доступу: http://window.edu.ru/resource/441/80441/files/itmo1363.pdf
32. MSDN “Руководство по Internet Explorer 10 для разработчиков” [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**.

33. Блог Марата Таналина “Internet Explorer 11 (IE11)” [Електронний ресурс] – Режим доступу: http://tanalin.com/blog/2013/11/internet-explorer-11/.
34. MSDN “Новые возможности Internet Explorer 8” [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/ru-ru/library/cc288472(v=vs.85).aspx.
35. MSDN “Примеры и учебники для Windows Internet Explorer 9” [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/ ru-ru/library/gg491737(v=vs.85).aspx
36. Блог Марата Таналина “Internet Explorer 10 (IE10)” [Електронний ресурс] – Режим доступу: http://tanalin.com/blog/2013/03/internet-explorer-10/
37. Блог Марата Таналина “Internet Explorer 9 (IE9)” [Електронний ресурс] – Режим доступу: http://tanalin.com/blog/2011/03/internet-explorer-9/
38. Маковод “Safari: как включить меню «Разработка»” [Електронний ресурс] – Режим доступу: http://macovod.com.ua/safari-how-to-enable-develop-menu/.
39. WayBackMachine, Internet Archive “Why Opera isn’t planning on going Open Source” [Electronic resource] – Access mode: https://web.archive.org/web/20081226012954/http://computerworld.co.nz/news.nsf/tech/982280C3DA7766DFCC257213007BC166
40. SMARTBEAR “Testing Flash and Flex Applications with the Debug Version of Flash Player – Overview” [Electronic resource] – Access mode: http://support.smartbear.com/viewarticle/59634/
41. Adobe “Flash Player Help. Install Flash Player in five easy steps” [Electronic resource] – Access mode: https://helpx.adobe.com/flash-player.html.
42. Портал знаний [Електронний ресурс] – Режим доступу:http://fresh-design.com.ua/blog/technology/website-testing
43. Тестирование. Легкий старт Азарский К. Copyright, 2014. –297 с.
44. Портал знаний [Електронний ресурс] – Режим доступу: http://vikips.blogspot.com/2012/02/blog-post.html
45. Портал знаний [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
46. Тестирование WEB-приложений [Електронний ресурс] – Режим доступу: http://taac.org.ua/files/a2011/proceedings/UA-1-Viktoria%20Valeriivna%20Gurkivska-173.pdf.
47. Software Testing Help. “Web Testing: Complete guide on testing web applications” [Electronic resource] – Access mode:http://www.softwaretestinghelp.com/web-application-testing/.
48. Web-testing [Electronic resource] – Access mode: **Ошибка! Недопустимый объект гиперссылки.**.

49. Software Testing Help. “Entries Tagged 'Cookie Testing. Website Cookie Testing, Test cases for testing web application cookies?” [Electronic resource] – Access mode: http://www.softwaretestinghelp.com /category/cookie-testing/
50. ALQA. "Тестування додатків" [Електронний ресурс] – Режим доступу: http://www.a1qa.ru/service/software_testing/web_appli cation_testing /.
51. Barancev. "Есе про валідації даних" [Електронний ресурс] – Режим доступу: http://habrahabr.ru/post/72796//
52. The programmer's worst friend "Тестуємо поля логін/пароль" [Електронний ресурс] – Режим доступу: http://testitquickly.com/ 2009/09/09/vvodeste-loginu-la-adnaklassni6i/
53.Система управління вмістом [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0% B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2% D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BE%D0 %B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D1%8B%D0%BC
54. Firebug [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**wiki/Firebug
56. Налагодження коду за допомогою Firebug [Електронний ресурс] – Режим доступу: http://htmlbook.ru / samlayout / testirovanie-i-otladka-koda / firebug
57. 10 причин, чому слід використовувати Firebug [Електронний ресурс] – Режим доступу: http://www.quizful.net/post/ten-reasons-using-firebug
58. Firebug – як користуватися кращим плагіном для вебмайстрів [Електронний ресурс] – Режим доступу: http://ktonanovenkogo.ru /vokrug-da-okolo/programs/poleznye-rasshireniya-dlya-brauzerov-firefox-i-opera-chast-1-firebug-dlya-firefox.html
59. Проводим аудит внутрішньої структури сайту програмою Xenu Link Sleuth [Електронний ресурс] – Режим доступу: https://devaka.ru/ articles/xenu-link-sleuth
60. SEO: XENU – БЕЗКОШТОВНИЙ АУДИТ САЙТУ І МЕРТВИХ ПОСИЛАНЬ [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
61. Joomla! [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Joomla!
62. Drupal [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Drupal
63. Портал знаний [Електронний ресурс] – Режим доступу: http://www.znannya.org/?view=software-testing-testing.
64. Профессия тестировщик [Електронний ресурс] – Режим доступу: http://enjoy-job.ru/professions/testirovschik/

65. Web testing [Electronic resource] – Access mode: http://en.wikipedia.org/wiki/Web_testing.
66. Профессия тестировщик [Електронний ресурс] – Режим доступу: http://www.protesting.ru
67. Краткие основы тестирования ПО – Коробейник А.Н. – К.,2012.–126 с.
68. Про тестинг. “Тест Дизайн (Test Design)” [Електронний ресурс] – Режим доступа: http://protesting.ru/testing/testdesign.html.
69. ITfap. «Кухонные принадлежности в тестировании: или что выбрать из онлайн-ресурсов для тестирования продуктов?» [Електронний ресурс] – Режим доступу: http://itfap.ru/read/Kuhonnye_prinadlezhnosti_v_testirovanii_ili_chto_vybratq_iz_onlajn-resursov_dlya_testirovaniya_produktov.
70. TestRail. “Test Suites & Cases. Managing your test case repository in TestRail” [Electronic resource] – Access mode: http://www.gurock.com/testrail/videos/suites-test-cases/.
71. TestRail. “ Introduction to TestRail managing project [Electronic resource] – Access mode: http://www.gurock.com/testrail/videos/introduction-projects/.
72. Техніки тест дизайну [Електронний ресурс] – Режим доступу: http://www.protesting.ru/testing/testdesign_technics.html
73.Тест-дизайн і ручне тестування [Електронний ресурс] – Режим доступу: http://software-testing.ru/forum/index.php?/forum/111-test-dizajn-i-ruchnoe-testirovanie/
74. Ефективне застосування комбінаторних технік тест дизайну [Електронний ресурс] – Режим доступу: http://w1zle.blogspot.ru/
75.Коберн А. Сучасні методи опису функціональних вимог досистем. – М.: Лорі, 2002. – ISBN 0-201-70225-8, ISBN 5-85582-152-8.
76. Тест план (План тестування) ПО [Електронний ресурс] – Режим доступу: http://www.protesting.ru/testing/plan.html
77. Тест план [Електронний ресурс] – Режим доступу: http://wiki.software-testing.ru/Тест-план_(NR)
78. Test Plan Outline (IEEE 829 Format) [Electronic resource] – Access mode:http://gerrardconsulting.com/tkb/guidelines/ieee829/main.html
79. RUP Test Plan Template [Eletcronic resource] – Access mode:http://www.protesting.ru/documentation/test_plan_template_rup.zip
80. kavichki.com [Електронний ресурс] – Режим доступу:http://kavichki.com/documents/Test_plan_example_OOO_Kavichki.pdf
81. Створення зрозумілих звітів по тестуванню [Електронний ресурс] – Режим доступу: http://habrahabr.ru/company/performance_ lab/ blog/ 207512/

82. Структура звіту про результати тестування [Електронний ресурс] – Режим доступу: http://testingworld.ru/struktura-otchyota-o-rezultatax-testirovaniya/
83. Звіт "Команда тестування: хід виконання" в форматі Excel [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
84. Створення зрозумілих звітів по тестуванню Excel [Електронний ресурс] – Режим доступу: http://savepearlharbor.com/?p=207512
85. Кому потрібен звіт про помилки [Електронний ресурс] – Режим доступу: http://testingworld.ru/komu-nuzhen-otchet-ob-oshibkax/
86. Як покращити структуру Вашого звіту по тестуванню? [Електронний ресурс] – Режим доступу: http://blog.i.ua/user/6074047/1297477/
87. Які тести вам потрібні? Частина 2. Матриця видів тестування [Електронний ресурс] – Режим доступу: http://habrahabr.ru/post/ 257159/
88. Виді тестирования [Електронний ресурс] – Режим доступу: http://xmindshare.s3.amazonaws.com/preview/5HiG-QziDBqK-00619.png
89. Гленфорд Майерс, Том Баджетт, Корі Сандлер. Мистецтво тестування програм, 3-е видання = The Art of Software Testing, 3rd Edition. – М.: «Диалектика», 2012. – 272 с. – ISBN 978-5-8459-1796-6.
90. Лайза Кріспін, Джанет Грегорі. Гнучке тестування: практичне керівництво для тестувальників ПЗ і гнучких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. – М.: «Вильямс», 2010. – 464 с. – (Addison-Wesley Signature Series). –1000 экз. – ISBN 978-5-8459-1625-9.
91. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізнес-додатків. – Киев: ДиаСофт, 2001. – 544 с. – ISBN 9667393879.
92. Калбертсон Роберт, Браун Крис, Кобб Гэри. Швидке тестування. — М.: «Вильямс», 2002. – 374 с. – ISBN 5-8459-0336-X.
93. Сініцин С. В., Налютін Н. Ю. Верифікація програмного забезпечення. – М.: БИНОМ, 2008. – 368 с. – ISBN 978-5-94774-825-3.
94. Бейзер Б. Тестування чорного ящика. Технології функціонального тестування програмного забезпечення та систем. – СПб.: Питер, 2004. – 320 с. – ISBN 5-94723-698-2.
95. Тестові сценарії [Електронний ресурс] – Режим доступу: http://software-testing.ru/community/blog/238.html
96. Особенности тестирования приложений на мобильных устройствах [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
97. Тестирование мобильных приложений [Електронний ресурс] – Режим доступу: http://www.osp.ru/os/2014/03/13040836/

98. Особенности тестирования мобильных приложений (iOS, Android) [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
99. Материалы для проведения онлайн-тренинга «Основы тестирования программного обеспечения» компании QATestLab.
100. Особенности тестирования приложений на мобильных устройствах [Електронний ресурс] – Режим доступу: http://ringames.com/ru/testirovanie/154-osobennosti-testirovaniya-prilozhenij-na-mobilnykh-ustrojstvakh.html
101. Тестирование мобильных приложений [[Електронний ресурс] – Режим доступу: http://habrahabr.ru/hub/mobile_testing/
102. Software Quality Characteristics [Electronic resource] – Access mode: http://thetesteye.com/posters/TheTestEye_ SoftwareQuality Characteristics.pdf
103. Что нужно знать дизайнеру о мобильных устройствах [Електронний ресурс] – Режим доступу: http://random1911.net/ that_a_web_designer_should_know_about_mobile_devices
104. Особенности тестирования приложений на мобильных устройствах [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
105. Тестирование для мобильных устройств: эмуляторы, симуляторы и удалённая отладка [Електронний ресурс] – Режим доступу: http://habrahabr.ru/ post/237499/
106. Как протестировать мобильный сайт: проверка верстки и отображения дизайна в мобильных браузерах [Електронний ресурс] – Режим доступу: http://great-world.ru/kak-protestirovat-mobilnyj-sajt/
107. Тестирование мобильных приложений [Електронний ресурс] – Режим доступу: http://www.osp.ru/os/2014/03/13040836/
108. Developer.Android [Електронний ресурс] – Режим доступу: Встановлення та використання ADB: **Ошибка! Недопустимый объект гиперссылки.**
109. Cyagenmod wiki [Електронний ресурс] –Режим доступу: Зняття логів за допомогою LogCat: http://wiki.cyanogenmod.org/w/Doc:_ debugging_with_logcat/ru
110. Wikipedia [Електронний ресурс] – Режим доступу: Windows Phone SDK: https://ru.wikipedia.org/wiki/Windows_Phone_SDK
111. App-s [Електронний ресурс] – Режим доступу: Керівництво по iTools: http://apps.ru/publ/kak_polzovatsja_itools_fajlovyj_menedzher_iphone_ipad_na_russkom/1-1-0-46
112. Використання утиліти Instruments в Xcode. Як користуватися утилітою Instruments в Xcode [Електронний ресурс] – Режим доступу: http://habrahabr.ru/post/168491/

113. Пошук витоку пам'яті в Xcode: [Електронний ресурс] – Режим доступу:http://heximal.ru/blog/coding/ishhem-utechki-pamyati-v-iphone-prilozheniyax/
114. Перегляд crash-логів в XCode: Збір логів і зняття скріншотів з iPhone і iPad [Електронний ресурс] – Режим доступу: http://mobile-testing.ru/how_to_test_mobileapp/logs_and_screens_apple/
115. Перегляд crash-логів в XCode: Перегляд системних логів [Електронний ресурс] – Режим доступу: http://bulkin.me/notes/2884
116. Установка .ipa-додатків: Встановлюємо тестовий додаток на Ваш пристрій [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
117. В.В. Кожаев. Тестирование интереса к игре [Електронний ресурс] – Режим доступу: http://dspace.nbuv.gov.ua/bitstream/handle/ 12345 6789/14703/3%D0%A06%20%D1%81%20452-456.pdf?sequence=1
118. Олейник Ольга. Презентация с SQA Days #11. Тестирование игр на мобильных устройствах и 3D телевизорах [Електронний ресурс] – Режим доступу: http://lib.custis.ru/images/6/6f/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%B8%D0%B3%D1%80_%D0%BD%D0%B0%D0%BC%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0%D1%85_%D0%B83D_%D1%82%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B7%D0%BE%D1%80%D0%B0%D1%85_(%D0%9E%D0%BB%D1%8C%D0%B3%D0%B0_%D0%9E%D0%BB%D0%B5%D0%B9%D0%BD%D0%B8%D0%BA,_SQADays-11).pdf
119. Habrahabr. Юлия Нечаева. О тестировании одной игры с картинками [Електронний ресурс] – Режим доступу:http://habrahabr.ru/post/98203/
120. Леонид Черный. Организация процесса тестирования компьютерной игры [Електронний ресурс] – Режим доступу: http://www.dtf.ru/ articles/read.php?id=1269
121. Дмитрий Анкудинов. CodeFest 2012. О специфике мультиплатформенного тестирования игр (+ Видео) [Електронний ресурс] – Режим доступу: http://2012.codefest.ru/lecture/405
122. Наука гейминга. Как и зачем тестируют игры? [Електронний ресурс] – Режим доступу: http://joystick.ru/nauka-gejminga-kak-i-zachem-testirujut-igry/887/
123. Game Testing All in One by Charles P. Schultz, Robert Bryant and Tim Langdell [Electronic resource] – Access mode: http://computernote.files.wordpress.com/2011/04/course-technology-game-testing-all-in-one-feb20051.pdf

124. Claudio Redavid, Adil Farid. An Overview of Game Testing Techniques [Electronic resource] – Access mode: http://www.idt.mdh.se/kurser/ct3340/ht11/MINICONFERENCE/FinalPapers/ircse11_submission_15.pdf
125. Марк Зальцман. Компьютерные игры: как это делается (Глава 15 - Тестирование) [Електронний ресурс] – Режим доступу: **Ошибка! Недопустимый объект гиперссылки.**
126. Команда проекта и ее роль в процессе разработки крупных [Електронний ресурс] – Режим доступу: http://old.ci.ru/inform20_97/astr1.htm
127. SEOLUX – Роли в команде разработчиков ПО [Електронний ресурс] – Режим доступу: http://www.seolux.com.ua/archives/155
128. Разработка программного обеспечения [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Разработка программного обеспечения
129. Royce, Winston (1970), Managing the Development of Large Software Systems [Electronic resource] – Access mode: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
130. Роль менеджера проектов в разработке ПО обречена на вымирание [Електронний ресурс] – Режим доступу: http://www.e-xecutive.ru/blog/ProjectManagement/13830.php
131. Процесс разработки программного обеспечения [Електронний ресурс] – Режим доступу: http://www.williamspublishing.com/PDF/5-8459-0276-2/part1.pdf
132. Гленфорд Майерс, Том Баджетт, Кори Сандлер. Искусство тестирования программ, 3-е издание = The Art of Software Testing, 3rd Edition. – М.: «Диалектика», 2012. – 272 с. [Електронний ресурс] – Режим доступа: http://www.dialektika.com/books/978-5-8459-1796-6.html
133. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. – М.: «Вильямс», 2010. –464 с. [Електронний ресурс] – Режим доступу: http://www.twirpx.com/file/882498/
134. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. – Киев: ДиаСофт, 2001. – 544 с. [Електронний ресурс] – Режим доступу: http://www.ex.ua/19572007
135. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование. – М.: «Вильямс», 2002. –374 с. [Електронний ресурс] – Режим доступу: http://adm-lib.ru/programmirovanie /kalbertson-braun-kobb-byistroe-testirovanie.html

136. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. – М.: БИНОМ, 2008. – 368 с. [Електронний ресурс] – Режим доступу: http://window.edu.ru/resource/700/41700
137. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб.: Питер, 2004. – 320 с. [Електронний ресурс] – Режим доступу: http://testingbooks.ru
138. Майк Кон. Scrum: гибкая разработка ПО = Succeeding with Agile: Software Development Using Scrum (Addison-Wesley Signature Series). – М.: «Вильямс», 2011. – С. 576. [Електронний ресурс] – Режим доступу: http://diamail.com.ua/book/4914.html
139. Роберт С. Мартин, Джеймс В. Ньюкирк, Роберт С. Косс. Быстрая разработка программ. Принципы, примеры, практика = Agile software development. Principles, Patterns, and Practices. – Вильямс, 2004. –752 с. [Електронний ресурс] – Режим доступу: http://www.ozon.ru/context/detail/id/1573723/
140. James A. Highsmith. Agile Software Development Ecosystems. – Addison-Wesley Professional, 2002. [Electronic resource] – Access mode:http://www.amazon.com/Agile-Software-Development-Ecosystems-Highsmith/dp/0201760436
141. Software testing. Что это? [Электронный ресурс] – Режим доступа: http://testing.biz.ua/pairwise-testing-ili-uzhasnaya-fraza-testirovanie-s-pomoshhyu-ortogonalnyx-massivov/
142. Pairwise testing [Электронный ресурс] – Режим доступа: https://uk.wikipedia.org/wiki/Pairwise_testing
143. Timur Gilmullin Pairwise testing: добиваемся оптимального покрытия различных тестовых комбинаций [Электронный ресурс] – Режим доступа: http://forworktests.blogspot.ru/2013/11/pairwise-testing.html
144. Алексей Федорко Pairwise testing with PICT [Электронный ресурс] – Режим доступа: http://www.software-testing.ru/library/testing/test-analysis/1559-pairwise-testing-with-pict
145. James Bach, Patrick J. Schroeder Pairwise Testing: A Best Practice That Isn’t [Electronic resource] – Access mode: http://testingeducation.org/wtst5/PairwisePNSQC2004.pdf
146. R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," Communication of the ACM, vol. 28, no. 10, pp. 1054-1058, 1985.
147. A. S. Hedayat, N. J. A. Sloane, and J. Stufken, Orthogonal Arrays: Theory and Applications. New York: Springer-Verlag, 1999.
148. D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design,"

IEEE Transactions on Software Engineering, vol. 23, no. 7, pp. 437-444, 1997.
149. Available Tools [Electronic resource] – Access mode: http://pairwise.org/tools.asp
150. Pairwise testing. Part 1 – Orthogonal Arrays [Электронный ресурс] – Режим доступа: http://w1zle.blogspot.ru/2010/11/pairwise-testing-part-1-orthogonal.html
151. Pairwise testing. Part 2 – AllPairs Algorithm [Электронный ресурс] – Режим доступа: http://w1zle.blogspot.ru/2010/11/pairwise-testing-part-1-allpairs.html

Додаток А

Глосарій

Testing (EN)	Тестирование (RU)	Тестування (UA)
Application	Приложение	Додаток
Ad hoc testing	Интуитивное тестирование	Інтуїтивне тестування
Boundary value analysis - BVA	Анализ граничных значений	Аналіз граничних значень
Bounding Box (BBox)	Ограничивающий параллелепипед	Обмежуючий паралелепіпед
Bug regression	Регрессия багов	Регресія багів
Bug Report	Баг/Дефект-Репорт	Баг/Дефект-Репорт
Bug tracking system	Система отслеживания ошибок	Система відслідковування помилок
Config	Файл конфигурации	Файл конфигурації
Debugger	Отладчик	Відладчик
Equivalence partitioning - EP	Эквивалентное разделение	Еквівалентний поділ
Exhaustive testing - ET	Исчерпывающее тестирование	Вичерпне тестування
Falloff	Сброс	Скидання
Footer	Подвал сайта	Підвал сайту
Game mechanics	Игровые механики	Ігрові механіки
Gameplay	Игровой процесс	Ігровий процес
Game testing	Тестирование игр	Тестування ігор
Hyperlink	Гиперссылка	Гіперпосилання
Incident Report	Отчет по инциденту	Звіт щодо інциденту
Improvement Report	Отчет о запросе на изменение	Звіт про запит на зміну
Link	Ссылка	Посилання

Testing (EN)	Тестирование (RU)	Тестування (UA)
Localization kit	Локк (набор ресурсов для локализации)	Локк (набір ресурсів для локалізації)
Localization testing	Тестирование локализации	Тестування локалізації
Old bugs regression	Регрессия старых багов	Регресія старих багів
Page-proofs	Вёрстка	Верстка
Side effect regression	Регрессия побочного эффекта	Регресія побічного ефекту
Sprite	Объект, перемещающийся	Об'єкт, що переміщується
Smart Zoom	Автоматическое приближение движущегося объекта в поле зрения	Автоматичне наближення рухомого об'єкту в полі зору камери
Software Risk Issues	Риски в программном обеспечении	Ризики у програмному забезпеченні
Tester	Тестировщик	Тестувальник
Test Incident	Инцидент	Інцидент
Test Plan	План тестирования	План тестування
Test Case	Тестовый случай	Тестовий випадок
Test Coverage	Тестовое покрытие	Тестове покриття
Test Case Detalization	Детализация тест-кейсов	Деталізація тест-кейсів
Test Case Pass Time	Время прохождения теста	Час проходження тесту
Test Plan Identifier	Идентификатор тест-плана	Ідентифікатор тест-плану
Testing purpose	Цель тестирования	Мета тестування
Test-driven development	Разработка от тестирования	Розробка від тестування
Test Items	Объекты тестирования	Об'єкти тестування
UI Testing	Тестирование интерфейса пользовательского	Тестування інтерфейсу користувача
Verification	Верификация	Верифікація
Validation	Валидация	Валідація
Zoom in	Приблизить камеру	Наблизити камеру

Testing (EN)	Тестирование (RU)	Тестування (UA)
Zoom out	Отдалить камеру	Віддалити камеру
Responsive web design	Адаптивный веб-дизайн	Адаптивний веб-дизайн
Software development	Разработка программного	Розробка програмного
Project Manager	Менеджер проекта	Менеджер проекту
Project	Проект	Проект
Business Analyst	Бизнес-аналитик	Бізнес-аналітик
Technical Leader	Системный аналитик	Системний аналітик
QA Manager	Менеджер контроля	Менеджер контролю
QA Analyst	Аналитик контроля	Аналітик контролю якості
Developer	Разработчик, программист	Розробник, програміст
Customer	Заказчик	Замовник
Planner	Планировщик ресурсов	Планувальник ресурсів
Team Leader	Руководитель команды	Керівник команди
Architect	Архитектор	Архітектор
Designer	Проектировщик	Проектувальник
Domain Expert	Эксперт предметной области	Експерт предметної сфери
Information Developer	Разработчик информационной поддержки	Розробник інформаційної підтримки
Human Factors Engineer	Специалист по пользовательскому интерфейсу	Спеціаліст з інтерфейсу користувача
Librarian	Библиотекарь	Бібліотекар
Waterfall model	“Модель водопада”, каскадная модель	“Модель водоспаду”, каскадна модель
Iteration model	Итеративная модель	Ітеративна модель
Agile software development	Гибкая методология	Гнучка методологія
Life cycle	Жизненный цикл	Життєвий цикл

Black-box testing	Тестирование “черного ящика”	Тестування “чорної скриньки”
White-box testing	Тестирование “белого ящика”	Тестування “білої скриньки”
Grey-box testing	Тестирование “серого ящика”	Тестування “сірої скриньки”
Software Development Plan	План разработки проекта	План розробки проекту

Додаток Б

Глосарій до теми 18: “РОЛІ У ПРОЦЕСІ РОЗРОБКИ ПЗ. КОМУНІКАЦІЇ У СФЕРІ ТЕСТУВАННЯ”

Термін UA:	Термін EN:	Опис:
АйОС	**iPhone (iPad) Operation System**	iOS – мобільна операційна система, розроблена американською компанією Apple на основі Mac OS X спочатку для iPhone, а потім розширена для підтримки таких мобільних пристроїв, як Apple iPod Touch, iPad і Apple TV. Apple не ліцензує iOS для установки на стороннє обладнання.
АйСпай	**iSpy**	Режим пошуку предметів, список яких відображається на панелі пошуку. Користувач збирає заховані предмети за допомогою тапів.
Аккаунт	**Account**	Обліковий запис – запис, що містить відомості, які користувач повідомляє про себе деякій комп'ютерній системі. Як синоніми в побуті можуть використовуватися розм. учетка та сленгові варваризми акк та аккаунт, від англ. account - обліковий запис, особистий рахунок.
Альфа	**Alpha**	Етап розробки, портирування (адаптація) гри. Передбачає впровадження в додаток нової задокументованої функціональності, а також перевірку програми на основну працездатність: - доведення додатку до стабільного стану, тобто відсутність раптових вимикань (краш), ситуацій при якій відсутня можливість повного проходження (шоустопов); - перевірка базової та нової функціональності на коректність роботи по документах та відповідність вимогам. Більш детальна інформація на Вікі: http://wiki.g5e.com/index.php/Porting_Project_Management_Process
Андроїд	**Android**	Android – портативна (мережева) операційна система для комунікаторів, планшетних комп'ютерів, цифрових програвачів, наручних годинників, нетбуків та смартбуків, заснована на ядрі Linux. Спочатку розроблялася компанією

		Android Inc., яку потім купила Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз і займається підтримкою та подальшим розвитком платформи. Android дозволяє створювати Java-додатки, керувати пристроєм через розроблені Google бібліотеки.
Термін UA:	**Термін EN:**	**Опис:**
Апдейт	**Update**	Апдейт (update, (англ.) модернізація; коригування; оновлення інформації, даних) – оновлення програмного забезпечення, випуск нової версії з виправленням та коригуванням попередньої версії. Читай "покращення", "оновлення".
Аппрув	**Approve**	Аппрув (від англо-американського to approve "схвалити, санкціонувати, підтримати") – "добро", схвалення.
Апсел скрін	**Upsell screen**	Спеціальне вікно покупки, що виникає перед користувачем після проходження безкоштовної частини гри. Запускається з метою підняття продажі додатку. Апсель скрін виникає на цікавому моменті гри, спонукаючи користувача заплатити гроші, щоб дізнатися, що ж буде далі.
АСАП	**As Soon As Possible**	Акронім від англ. As Soon As Possible – відповідає російському кліше «як найшвидше»
Аттачмент	**Attachment**	Приєднання (вкладення, вкладиш, додаток) [до листа, багу] якого-небудь файлу.
Бабл	**Bubble**	Вікно, оформлене у вигляді непрямокутної (зазвичай у вигляді хмари) області, що містить ігровий текст і не містить кнопок.
Баг	**Bug**	(англ. Bug – жук; глюк) – жаргонне слово, зазвичай позначає помилку у програмі або системі, яка видає несподіваний або неправильний результат. Більшість багів виникають через помилки, які були допущені розробниками програми в її вихідному коді або в її дизайні. Також деякі баги виникають через неправильну роботу компілятора, який виробляє некоректний код. Програму, яка містить велику кількість багів, та / або баги, які серйозно обмежують її працездатність, називають нестабільною або, на жаргонній мові, «глючною», «глюкнутою», «забагованной», «бажною», «баг (а) нутою» (англ . unstable, buggy).
Багтрекінгова система	**Bug tracking system**	Система відстеження помилок, розроблена з метою допомогти розробникам програмного забезпечення (програмістам, тестувальникам та ін.) Враховувати та контролювати помилки (баги), знайдені в програмах, побажання користувачів, а також стежити за процесом усунення цих помилок та виконання або невиконання побажань.
Бамбук	**Bamboo**	див. Білд-сервер
Бандл АйДі	**BundleID**	Рядок виду com.g5e.XXXXXXXXXX, що ідентифікує додаток у системі. Необхідний для перевірки підпису додатку. Щоб перевірити Bundle ID, потрібно відкрити білд з грою за допомогою архіватора й знайти в папці * .app файл "info.plist", який можна відкрити за допомогою Блокнота. У відкритому

		файлі буде набір символів, серед яких потрібно знайти необхідний рядок. Актуальні Bundle ID можна подивитися в документі "bundle_apple_id.xlsx", який знаходиться на СВН: svn: //svn.g5e.com: 3790 / g5documents / Cross_Promotion
Термін UA:	**Термін EN:**	**Опис:**
Банер	**Banner**	(англ. banner – прапор, транспарант) – електронне графічне зображення рекламного характеру з можливістю переходу на сторінку рекламованого матеріалу.
Баундінг бокс	**Bounding box**	Активна зона предмету у вигляді мінімально можливого прямокутника, який повністю містить усередині цей предмет. Застосовується у випадку, якщо активна зона по контуру предмета занадто маленька, та якщо складно потрапити в цю зону.
Білд	**Build**	У галузі програмного забезпечення термін build (англ. «побудова») відноситься до процесу перетворення початкового коду програми, який може бути запущений на конкретному пристрої (пристроях) або перетворений в код, який виконується. Одним із кроків створення білда є процес компіляції вихідного коду, де файли структуруються та перетворюються в форму, доступну для тестування програми.
Білд-сервер	**Build server**	Білд-сервер – мережевий ресурс, призначений для зберігання та компіляції ресурсів різних проектів, що дозволяє вручну або автоматично (за часом) збирати актуальні ресурси по проекту в збірки ресурсів (білди) для тестування (розгляду або публікації), враховуючи останні внесені зміни.
Брифінг	**Briefing**	Ігрове вікно, що виводиться зазвичай перед початком рівня. Містить опис рівня, а також список завдань, які гравець повинен на рівні виконати.
Булет	**Bullet**	(от англ. Bullet - пуля) Виклад будь-якої інформації тезисно, ємко, найчастіше коротким списком.
Вейв	**Waive**	Рішення керівництва (зазвичай продюсера проекту) не виправляти будь-який баг.
Вейв кандидат	**Waive Candidate**	Баг, який представлений до вейв.
Версія	**Version**	Підназва програмного продукту, що використовується для руху по життєвому циклу додатку, тобто при виправленні в програмному продукті змінюється його версія, щоб кожного разу не міняти його (програмного продукту) назву.
Вікі	**Wikipedia**	(англ. Wikipedia – вільна) – веб-ресурс вільної енциклопедії, який використовується для збору та структурування інформації про компанію та внутрішні її процеси, доступні тільки для працівників компанії.
Віндоус	**Windows**	Microsoft Windows (вимовляється [майкрософт віндоус]) - сімейство операційних систем корпорації Майкрософт (Microsoft), орієнтованих на застосування графічного інтерфейсу при управлінні.
Віндоус РТ	**Windows RT**	Редакція операційної системи Windows 8 для планшетних та інших комп'ютерів на базі ARM-процесорів.

Термін UA:	Термін EN:	Опис:
Войсовер	**Voice-over**	Озвучка ігрових діалогів, тобто голос за кадром.
Воксель	**Voxel**	Воксель – елементарна частинка в растровій графіці. Аналог пікселя, але на відміну від нього, описується 3-ма координатами.
Випускне тестування	**Graduation testing**	Кінцева перевірка, яка проводиться в кінці кожного етапу проекту. Передбачає тестування всього базового функціоналу гри.
ГДД	**Game Design Document**	Максимально повний опис гри. GDD дозволяє розробникам гри скласти «план подальших дій» щодо втілення задуманого проекту в проект реальний. Також GDD – це назва етапу портирування, на якому створюється документація з гри.
Глосарій	**Glossary**	(лат. glossarium – «збірник глосс») – словник вузькоспеціалізованих термінів у якій-небудь галузі знань з тлумаченням, іноді перекладом іншою мовою, коментарями та прикладами.
Голд мастер	**Gold Master**	Стадія закінчення тестування, що припускає повну підготовку функціоналу та документації до сабміту проекту. Етап розробки Gold Master передбачає: - повну реалізацію всього задокументованого функціоналу гри; - повністю готові та перевірені локалізації гри; - повна відповідність вимогам платформи та магазину, на яких буде здійснюватися публікація продукту; - коректно функціонує додаток, що не має недоліків крупніше невеликих графічних дефектів, які не заважають ігровому процесу.
ГФЛ	**Global Feature List**	Документ, що описує базову функціональність фич, які повинні бути присутніми й, відповідно, працювати в кожному додатку, що випускається, при використанні на пристрої.
ГФЛ	**Global Feature checkList**	Документ, за яким необхідно перевіряти коректність роботи базових функціональних фич.
Дата	**Date**	Запис, що включає в себе число місяця, місяць та рік, іноді день тижня.
Дата	**Data**	Дані – різна інформація, представлена в форматі, що дозволяє зберігати, передавати та обробляти комп'ютером дану інформацію. Зазвичай всі дані зберігаються у файлах.
Дебрифинг	**Debriefing**	Ігрове вікно, що виводиться після завершення рівня. Містить перелік виконаних та невиконаних гравцем завдань, а також кількість набраних за рівень очок.
Девайс	**Device**	Пристрій (також жарг. девайс – від англ. device) – штучний об'єкт, що має внутрішню структуру, створений для виконання певних функцій, зазвичай в області техніки (технічний пристрій).
Дефект	**Defect**	Помилка, що припускає конкретну розбіжність реального програмного продукту з задуманою, заапрувленною версією. Синонім бага.

Термін UA:	Термін EN:	Опис:
Джі Файв	**G5 Entertainment AB**	G5 Entertainment – видавець та розробник крос-платформних онлайн-ігор формату «casual». Пріоритетним напрямком для G5 Entertainment є розробка та видання успішних оригінальних ігор, націлених на широку аудиторію. Основними платформами для компанії є iPhone, iPad, PC, Mac, інші переносні та домашні ігрові консолі. Офіси розробки компанії знаходяться у Харкові та Москві.
Джіра	**JIRA**	Система контролю проектів, яку QA використовує як багтрекінгову систему.
Діалогове вікно	**Dialog box**	Модальне вікно з текстом. Може з'являтися на екрані як у вигляді слів персонажа, так й у вигляді підказки гравцеві.
Дизайнер	**Designer**	див. Левел-Дизайнер
Драг	**Drag**	Дотик пальця до екрану та подальше його переміщення в довільному напрямку й з довільною швидкістю. Не плутати з Drug (наркотики).
Драг енд дроп	**Drag and Drop**	Механіка, що дозволяє комбінувати предмети у грі, перетягуючи один предмет на інший.
Завантаження	**Loading**	Процес актуалізації, докачки, додавання ресурсів у додатки для підключення нового функціоналу.
Замір пам'яті	**Measuring of memory**	Процес вимірювання споживаної пам'яті пристрою під час гри.
Замір FPS	**Measuring FPS**	Процес вимірювання кількості кадрів у секунду, що видаються на пристрої під час гри.
Зум	**Zoom**	Функціонал збільшення картинки на сенсорних екранах шляхом розведення / зведення двох пальців на екрані. (iPad / iPhone підтримують трехпальцевий зум)
Зум ін	**Zoom In**	За час повного розведення пальців (з положення щипка) – картинка збільшиться до свого максимального значення.
Зум аут	**Zoom Out**	За час повного зведення пальців (з положення, коли пальці (вказівний й великий) на екрані знаходяться на максимальному видаленні один від одного) – картинка повертається до свого повноцінного значення.
Інапи	**In App Purchase**	Сервіс дозволяє користувачеві здійснювати покупку всередині програми (наприклад, ігрової валюти, нових рівнів, ігрових предметів, розблокування повного контенту та ін.), застосовується з метою збільшення прибутку від гри.
Інвойс	**Invoice**	У міжнародній комерційній практиці документ, що надається продавцем покупцеві та містить перелік товарів, їх кількість та ціну, за якою вони будуть поставлені покупцеві, формальні особливості товару (колір, вага та ін.), умови поставки та відомості про відправника та одержувача. Виписка інвойсу свідчить про те, що (крім випадків, коли постачання здійснюється по передоплаті) у покупця з'являється обов'язок оплати товар відповідно до вказаних умов.
Інтест	**InTest**	Етап перевірки бага, куди баг потрапляє після виправлення розробником.
Ітерація	**Iteration**	Один крок циклу.

Термін UA:	Термін EN:	Опис:
Кат-сцена	**Cut-scene**	Епізод у грі, в якому гравець слабо або взагалі ніяк не може впливати на події, зазвичай з перериванням геймплея.
Клоуз-ап	**Close-up**	див. Контейнер.
Код	**Code**	Вихідний код, програмний код – текст комп'ютерної програми на будь-якій мові програмування, який може бути прочитаний людиною. В узагальненому сенсі – будь-які вхідні дані для транслятора. Вихідний код транслюється у виконуваний код цілком до запуску програми за допомогою компілятора, або може виконуватися відразу за допомогою інтерпретатора.
Кодер	**Coder**	див. Програміст.
Комміт	**Commit**	Процес підтвердження зміни ресурсів.
Комерційне програмне забезпечення	**Commercial software**	Програмне забезпечення, створене з метою отримання прибутку від його використання іншими, наприклад, шляхом продажу примірників.
Контейнер	**Container**	Елемент інтерфейсу гри, представлений у вигляді модального вікна, що показується поверх ігрової локації, та дозволяє більш детально оглянути яку-небудь область ігрового поля.
Косметик	**Cosmetic**	Незначна помилка в графічному оформленні гри, що не впливає на функціонал гри.
Кранч	**Crunch**	Схвалена керівництвом переробка у зв'язку з конкретними роботами по проекту.
Краш	**Crash**	Це раптове, непередбачене закриття гри, викликане помилкою (-ми) програмного коду, що виникає випадковим чином або внаслідок певних дій користувача. Часто характеризується втратою або псуванням даних. Синоніми: аварійний відмова, збій
Крашлог	**CrashLog**	Краш лог – файл, що містить інформацію про збій у роботі програми або системи, що призвів до краху додатку або системи відповідно.
Кредітси	**Credits**	Список осіб, що беруть участь у реалізації проекту, зазвичай показується по закінченню проходження гри.
Кроспромо	**CrossPromo**	Інтегрована в гру сторінка з усіма іншими іграми від G5 Entertainment, що служить для реклами ігор.
КроссПромо АйДі	**CrossPromoID**	Ідентифікатор гри виду "com.g5e.XXXXXXXXX", або подібний, необхідний для коректної роботи крос-промо, а також процесу відправки ігрових логів. Щоб перевірити крос промо айді, який вставлений в гру, необхідно: 1. Відкрити білд з грою за допомогою WinRAR 2. Знайти папку "xpromo", в якій знайти текстовий документ з назвою виду "config-com.XXXXXXXXXXXXXXX.txt" 3. Відкрити цей документ та знайти в ньому рядок "id = com.XXXXXXXXXXXXXXX" (цей рядок зазвичай перший) частина рядка "com.XXXXXXXXXXXXXXX" є крос промо ID

Термін UA:	Термін EN:	Опис:
КьюЕй	**Quality Assurance**	Контроль та оцінка будь-яких аспектів проекту, устаткування або виду послуг з метою збільшення (максимізації) ймовірності забезпечення встановлених (мінімальних) стандартів якості.
Лаг	**Lag**	(від англ. lag, [læg] – «запізнювання», «затримка») – затримка в роботі комп'ютерного додатку, коли він не реагує вчасно на введення користувачем даних. Похідне від нього "лага" широко використовуються користувачами Інтернету для позначення затримок у роботі різних інтернет-сервісів, онлайн-ігор. Також геймери використовують слово «лаг» відносно затримок самої програми. Часто зустрічається в онлайн іграх.
Лайан сендбоксінг	**Lion sandboxing**	Технологія в MAC OS X Lion, що забезпечує захист операційної системи від можливої шкоди, що можуть завдати сторонні додатки, шляхом обмеження функціональних можливостей цих додатків. Перевіряється шляхом контролю місця розташування файлів збережень гри. Файли з збереженнями ігри повинні лежати в строго визначеному місці: ~ / Library / Container / com.g5e.Project_name.mac, де Project_name - назва програми, що тестується (якщо у додатку немає яких-небудь унікальних особливостей).
Левел-Дизайнер	**Level-Designer**	Людина, що створює рівні для ігор – локації, місії, завдання та інше оточення. Зазвичай це робиться за допомогою редактора рівнів. Також дизайнер вносить необхідні зміни для адаптації портируємості ігор під необхідні платформи, наприклад, налаштування геймплея, зміна й адаптація інтерфейсів, ефектів та ін.
Лендінг Пейдж	**Landing Page**	Рекламна вставка в проектах компанії G5, що дозволяє користувачеві ознайомитися з іншими випущеними проектами компанії, підписуватися на розсилки. З'являється на початку гри перед завантаженням головного меню, а також (поки тільки для iOS-проектів) при поверненні в гру після будь-якої дії, при якому гра неактивна (наприклад, згорнута) не менше однієї хвилини.
Лід	**Lead**	Особа, що займає керівну посаду та підпорядкована безпосередньо главі відділу. Здійснює контроль термінів та якості виконання всіх проектів відділу.
Лоадінг скрін	**Loading screen**	Це зображення, яке показується гравцям під час завантаження будь-якої частини гри (рівня, карти, тощо). Як правило, на лоадінг-скрині є прогрес-бар.
Логін	**Login**	(від англ. log in – вписатися в журнал) – ім'я користувача в комп'ютерній мережі. Спільно з паролем служить для ідентифікації користувача.
Лоі	**List of Improvements**	Документ, що створюється продюсером й містить список усіх змін у порівнянні з грою-донором, а також нововведень, які повинні бути інтегровані в гру.
Локкіт	**Localization kit**	Документ, що містить переклади абсолютно всіх ігрових текстів одного або декількох мов, які підтримуються

		додатком.
Термін UA:	**Термін EN:**	**Опис:**
Логування	**Logging**	Процес відправки логів гри з пристрою на сервер. У логах відображаються всі основні дії гравця: запуск гри, відкриття крос-промо, покупка повної версії гри, отримання ачівки та ін.
Майлстоун	**Milestone**	Метафора, якою в software development позначають проміжний етап розробки проекту. Процес розробки розбивається майлстоуном відповідно до спеціального плану з метою мінімізувати ризики та встигнути в задані терміни. Кожен майлстоун включає список завдань, які повинні бути вирішені та втілені у проекті до дедлайну. Якщо вирішити всі завдання не встигають, відбувається зрив майлстоун. У цьому випадку або виділяють додатковий час на реалізацію та продляють терміни, або відмовляються від проблемної фичи зовсім. Майлстоун фіксує всі прийняті рішення, щоб у розробників не виникало спокуси переробляти все до нескінченності. Історично, майлстоун – це кам'яні плити із зазначенням відстаней до населених пунктів, які розставлялися уздовж доріг та служили орієнтиром мандрівникам.
Мак ОС	**Mac Operation System**	Пропріетарна операційна система від Apple. OS X входить у сімейство операційних систем OS X, до якого належить й Apple iOS. Також, OS X є спадкоємицею Mac OS 9 – останньої версії «класичної» Macintosh Operating System (Mac OS).
Мануал	**Manual**	(англ. user guide або user manual), керівництво по експлуатації, керівництво користувача – документ, призначення якого – надати людям допомогу у використанні деякої системи. Документ входить до складу технічної документації системи.
Метагейм	**Metagame**	(від грец. meta – за, після, через – й англ. game – гра) – «поза грою», у широкому розумінні – все, що відноситься до гравців, а не до їх персонажів. У вужчому розумінні метагейм – це ситуація, коли гравець діє, виходячи з власних міркувань, а не логіки персонажа, на шкоду виграшу, або використовує неігрову інформацію (тобто таку, яка відома йому, але не може бути відома його персонажу).
Мокап	**Mockup**	Мокап – це шаблон (макет), створений (зібраний) художником (дизайнером), за аналогією з яким у конкретну зону гри повинні бути коректно вставлені (розташовані) графічні елементи (можливо, несучі функціональне навантаження).
Мультитач	**Multi-touch**	Мультитач (англ. Multi-touch – множинний дотик) – функція сенсорних систем введення, що здійснює одночасне визначення координат двох та більше точок дотику, тобто, іншими словами, одночасне оброблення дотику до екрану двох та більше пальців. Мультитач може застосовуватися, наприклад, для зміни масштабу зображення (зума): при збільшенні відстані між точками дотику (розведення двох пальців в протилежні кути екрану) відбувається збільшення зображення. Крім того, мультитач екрану дозволяє працювати

		з пристроєм одночасно декільком користувачам.
Термін UA:	**Термін EN:**	**Опис:**
Покарання	**Punishment**	Застосування будь-яких неприємних або небажаних заходів відносно людини або тварини у відповідь на непокору або морально неправильну поведінку, професійну недбалість.
Неретина	**Not Retina**	Дозвіл екрану мобільних пристроїв нижче 800x480 пікселів.
Оверлей	**Overlay**	Метод програмування, що дозволяє створювати програми, що займають більше пам'яті, ніж встановлено в системі. Вбудовані комп'ютери часто використовують оверлеї, так як зазвичай Система на кристалі містить мало пам'яті й не підтримує віртуальну пам'ять.
Операційна система	**Operation System**	Комплекс програм, який дозволяє скористатися машинними можливостями пристрою.
ОС Ікс	**Operation System X**	див. Мак ОС
Помилка	**Error**	див. Баг
Пароль	**Password**	(фр. parole – слово) – це секретне слово або набір символів, призначений для підтвердження особи або повноважень. Паролі часто використовуються для захисту інформації від несанкціонованого доступу. У більшості обчислювальних систем комбінація «ім'я користувача – пароль» використовується для підтвердження користувача.
Партікл	**Particle**	У загальному випадку – дрібна частинка якої-небудь анімації. В іграх, як правило, поняття "партікл" має на увазі підсвічування (виділення) на ігровому полі об'єктів (зон), необхідних для просування по грі. Зазвичай виглядає як одна або декілька зірок, іскор, що з'являються на об'єкті.
Плашка	**Plate**	Будь-яке ігрове вікно порівняно невеликого розміру, що з'являється автоматично (наприклад, повідомлення про отримані досягнення) або за викликом гравця (вікно з описом якогось активного елементу при натисканні на нього).
Плейграунд	**Playground**	Жанр ігор, розрахованих на довгострокову перспективу, метою яких є підтримка інтересу гравців за рахунок постійно оновлюваного контенту (додавання ігрових завдань-квестів, нових рівнів, та ін.) та безперервне стимулювання до покупки різних ігрових бонусів, що дозволяють спростити процес гри, за рахунок порівняно невеликої вартості бонусів.
Плейн порт	**Plain Port**	Етап портирування гри. Основні завдання етапу: - перенесення функціоналу ігри-донора на платформу гри-реципієнта; - визначення витрат ресурсів на повне портирування гри. Більш докладно на Вікі: http://wiki.g5e.com/index.php/Porting_Project_Management_Process
Пауз-меню	**Pause-menu**	Меню, що викликається під час ігрового процесу та зупиняє гру. Як мінімум містить кнопки: виклику меню опцій, виходу

		в головне меню й продовження гри.
Термін UA:	**Термін EN:**	**Опис:**
Піксель	**Pixel**	(англ. pixel, pel – скорочення від pix element, в недо. іст. picture cell – букв. елемент зображень) або еліз (російський варіант терміна, який рідко використовується) – найменший логічний елемент двовимірного цифрового зображення в растровій графіці, а також елемент світлочутливої матриці (іноді званий сенсель – від sensor element) та елемент матриці дисплеїв, що формують зображення. Піксель є неподільним об'єктом прямокутної або круглої форми, що характеризується певним кольором.
Платформа	**Platform**	Апаратний та / або програмний комплекс, який використовується, як основа для операційної системи.
Поінт энд тап	**Point and tap**	Предмет має два стани: 1) Активний; 2) Неактивний. Щоб активізувати предмет, необхідно по ньому тапнути. Після того, як предмет став активним та гравець тапнув у вільному місці на ігровому полі, предмет повинен переходити в неактивний стан та повертатися на своє колишнє місце (скидатися). Якщо гравець після активації предмета, тапне по складеному об'єкту, частиною якого є активований предмет, тоді предмет вважається зібраним (застосовується). Якщо гравець тапне по складеному об'єкту, частиною якого не є, предмет повинен перейти в неактивний стан та повернутися на своє колишнє місце (скинутися). При спробі зуму предмет не повинен летіти на палець. Тобто, якщо була подія мультитача, то предмет на нього реагувати не повинен.
Користувач	**User**	Особа, яка використовує діючу систему для виконання конкретної функції.
Споживання пам'яті	**Memory Usage**	Процес використання запущеним додатком оперативної пам'яті пристрою. При надмірному споживанні пам'яті можливий краш програми. Для виявлення можливих перевищень обсягу споживаної пам'яті проводяться заміри.
Пріквел	**Prequel**	(англ. prequel, контамінація приставки pre- 'до-' і sequel, див. сиквел) – книга, кінофільм або комп'ютерна гра, час дії яких відбувається до подій раніше створеного твору та попередні їм по внутрішній хронології.
Додаток	**Application**	Програма, призначена для виконання певних завдань користувача, й розрахована на безпосередню взаємодію з користувачем. У більшості операційних систем прикладні програми не можуть звертатися до ресурсів комп'ютера безпосередньо, а взаємодіють з обладнанням та інш. за допомогою операційної системи. Також простою мовою – допоміжні програми.
Пріоритет	**Priority**	Це атрибут, який вказує на пріоритетність виконання завдання або усунення дефекту. Чим вище пріоритет, тим швидше потрібно виправити дефект.

Термін UA:	Термін EN:	Опис:
Провіжен	**Provision**	Файл-сертифікат, що містить список пристроїв для установки на ці пристрої додатків.
Програміст	**Programmer**	Фахівець, який займається програмуванням, тобто написанням додатків, а також виправляє функціональні баги.
Проект	**Project**	(від лат. projectus – кинутий вперед, виступаючий вперед) Процес, що складається із сукупності скоординованих та керованих видів діяльності з датами початку та закінчення, зроблений для досягнення мети, яка відповідає конкретним вимогам, що включає обмеження по термінах, вартості та ресурсах. Тимчасове підприємство, призначене для створення унікальних продуктів, послуг або результатів. Завершення наступає, коли досягнуті цілі проекту; або визнано, що цілі проекту не будуть або не можуть бути досягнуті; або зникла необхідністьу проекті.
Пропріетарне програмне забезпечення	**Proprietary software**	(від англ. proprietary – приватне, патентований, у складі власності та software – програмне забезпечення) – програмне забезпечення, яке є приватною власністю авторів або правовласників та не задовольняє критерії вільного ПЗ (наявності відкритого програмного коду недостатньо). Правовласник пропріетарного ПЗ зберігає за собою монополію на його використання, копіювання та модифікацію, повністю або в суттєвих моментах. Зазвичай пропріетарним називають будь-яке невільне, включаючи напіввільне.Розглядається поняття не пов'язане з поняттям комерційного програмного забезпечення.
Прошивка	**Firmware**	Системне програмне забезпечення, вбудоване («зашите») в апаратний пристрій.
Пуш нотіфікейшен	**Push Notification**	Система донесення інформації про зміни цін на ігри, оновлень та виходу новинок до користувача. http://wiki.g5e.com/index.php/Push_Notifications_1.0
Пуш	**Push**	(англ. Push, дослівно – «продавлювання» або «просування»), (також відомий як webcasting або netcasting) – один з варіантів розповсюдження контенту в Інтернеті, коли інформація надходить від сервера до клієнта на основі ряду параметрів, встановлених клієнтом. Звичайний користувач може підписатися на різні теми, інформацію від контент-провайдера, й кожен раз коли нове оновлення формується на сервері, це оновлення "просувається" на комп'ютер або смартфон користувача. Ця форма розповсюдження контенту відрізняється від спільного використання в Інтернеті, оскільки в цьому випадку інформація запитується користувачем на сервері.
Розширення екрану	**Screen resolutio**	Кількість точок (пікселів) по горизонталі та вертикалі екрану пристрою.
Рев'ю	**Review**	Вікно із запитом залишити свій відгук та оцінку на сторінці з грою в магазині. Виводиться в певні моменти ігрового процесу, коли користувач імовірно найбільш захоплений грою та готовий

		дати позитивний відгук. Мета цього вікна – підвищити рейтинг програми.
Термін UA:	**Термін EN:**	**Опис:**
Регресійне тестування	**Regression testing**	(від лат. regressio – рух назад) – збірна назва для всіх видів тестування програмного забезпечення, спрямованих на виявлення помилок у протестованих ділянках вихідного коду. Такі помилки, при яких після внесення змін до програми перестає працювати те, що повинно було продовжувати працювати, називають регресійними помилками (англ. Regression bugs).
Реджект	**Reject**	Повернення невиправленого бага або невиконаного завдання виконавцю для повторного виправлення / виконання.
Резолюція	**Resolution**	Рішення, прийняте посадовою особою, міжнародною організацією. У документообігу Резолюція документу – реквізит, що складається з напису на документі, зробленою посадовою особою та містить прийняте ним рішення.
Реліз	**Release**	Відрив пальця від екрану, після якого натискання вважається виконаним та відбувається запланована реакція гри.
Реліз	**Release**	Завантаження готової гри в магазин, для подальшого продажу.
Реліз кандидат	**Release Candidate**	Етап розробки, на якому основний функціонал доводиться до досконалості, вставляються локалізації, доводиться до досконалості покупка програми.
Рендер	**Render**	(англ. rendering – «візуалізація») – термін у комп'ютерній графіці, що позначає процес отримання зображення по моделі за допомогою комп'ютерної програми. Тут модель – це опис будь-яких об'єктів або явищ на строго визначеній мові або у вигляді структури даних. Такий опис може містити геометричні дані, положення точки спостерігача, інформацію про освітлення, міри наявності якоїсь речовини, напруженість фізичного поля та ін.
Ретіна	**Retina**	Дозвіл екрану мобільних пристроїв вище 800x480 пікселів.
Рефігс	**R-ussian E-nglish F-rench (Français) I-talian G-erman (Deutsch) S-panish (Español)**	Скорочена назва групи мов.
Сабміт	**Submit**	Процес фінального тестування гри, її випуску та завантаження в магазин.

Термін UA:	Термін EN:	Опис:
Саппорт	**Support**	Служба підтримки користувачів. Людина або група людей, що займаються вирішенням питань, як правило, технічного характеру при супроводі випущеного продукту.
Саспенд івент	**Suspend event**	Будь-яка подія, при якій гра переходить у фон (згортання гри, введення пристрою в sleep mode, та ін.). Після того, як відбувається повернення в гру, повинно з'являтися пауз-меню.
Свайп	**Swipe**	Безперервний рух пальця по екрану, що служить, наприклад, для гортання ігрової інформації.
СВН	**SubVersion**	TortoiseSVN – це безкоштовний клієнт для системи контролю версій Subversion, виконаний як розширення оболонки Windows та росповсюджується під ліцензією GPL. Будучи клієнтом Subversion, TortoiseSVN дозволяє управляти файлами та папками у часі. Файли зберігаються в центральному сховищі, в якому запам'ятовується кожна зміна, зроблена в збережених файлах та папках. Це дає можливість відновлювати старі версії файлів й вивчати історію їх зміни. Тому Subversion та інші системи контролю версій часто називають «машинами часу» для файлової системи.
Сеньйор	**Senior**	Старший фахівець, який може займати керівну посаду, бути здатним керувати групою, приймати відповідальні рішення та брати на себе відповідальність за діяльність всієї групи.
Сервер	**Server**	(англ. server від англ. to serve – служити) – в інформаційних технологіях програмний компонент обчислювальної системи, що виконує сервісні (обслуговуючі) функції по запиту клієнта, надаючи йому доступ до певних ресурсів або послуг.
Сиквел	**Sequel**	(англ. sequel [si:kwəl] – продовження) – гра, фільм або будь-який інший твір мистецтва, який за сюжетом є продовженням іншого твору, побудований на персонажах з нього та ін. Особливий розряд сиквелів складають «духовні сиквели», які не є прямими продовженнями, проте ж розглядають той же набір понять та ідей, що й твори, що передують по сюжету.
Сквозне тестування	**Through testing**	Вид тестування, при якому перевіряється основний функціонал при швидкому лінійному проходженні гри. Мета даного виду тестування полягає в перевірці працездатності всього функціоналу в цілому. Перевірка даним видом тестування проводиться на різних етапах розробки програми, коли є необхідність швидко упевнитися, що весь реалізований функціонал присутній у грі й функціонує коректно.
Скейл	**Scale**	Масштабування. Зміна розмірів об'єкта без зміни його положення або орієнтації.
Скіп	**Skip**	Функціонал, що дозволяє користувачеві пропустити яку-небудь частину гри (наприклад, відеоролик або навчання), яка не впливає на ігровий процес .
Скрол	**Scroll**	Переміщення інформації у вікні по вертикалі або горизонталі для перегляду даних, що не помістилися у вікні.
Сліп мод	**Sleep mode**	Сплячий (який чекає) режим пристрою, призначений для економії енергії, яка споживається пристроєм.

Термін UA:	Термін EN:	Опис:
Смарт тап	**Smart Tap**	Механіка обробки тапів, яка передбачає, що у випадку, якщо палець гравця при торканні екрану «накриває» більше однієї активної області, то оброблятися повинна та, чий центр ближче до місця дотику.
Смарт зум	**Smart Zoom**	У певні моменти рівень зуму повинен виставлятися автоматично. Перехід з початкового рівня зуму в необхідний рівень повинен здійснюватися плавно, неначе гравець змінює його сам. Час переходу з початкового рівня зуму в необхідний завжди має бути однаковий, але не перевищувати 1 сек.
Спамитм	**Spamming**	Проводити часті однакові дії з метою порушення функціоналу гри (пристрю).
Сплеш	**Splash**	Картинка з логотипом розробника, зазвичай з'являється відразу після запуску програми.
CEO	**Chief Executive Officer**	Головний виконавчий директор, (брит. англ. director general) - вища посадова особа компанії (генеральний директор, голова правління, президент, керівник). Визначає загальну стратегію підприємства, приймає рішення на вищому рівні, виконує представницькі обов'язки. Численні дискусії в спільноті, де проводиться переклад та які мають один результат: перекладати цей термін російською мовою треба як «генеральний директор», так як за своїми функціями CEO найбільш близький до цього російському поняттю.
COO	**Chief Operating Officer**	Головний операційний директор; один з керівників установи, що відповідає за повсякденні операції, за поточну діяльність. У російській мові та бізнесі цьому поняттю відповідає посаду «виконавчий директор».
Стейт	**State**	(від англ. Стан) Опис положення якого-небудь ігрового елемента.
Стрінг ID	**String ID**	ID (String від англ. Рядок, IDentifier (ID) від англ. ідентифікатор) – це унікальний ідентифікатор рядка в документах, що мають табличне походження, тобто мають рядки та стовпці.
СФО	**Chief Financial Officer**	Фінансовий директор – один з вищих управлінців компанії, відповідальний за управління фінансовими потоками бізнесу, за фінансове планування та звітність. Визначає фінансову політику організації, розробляє та здійснює заходи щодо забезпечення її фінансової стійкості. Керує роботою з управління фінансами виходячи зі стратегічних цілей та перспектив розвитку організації, за визначенням джерел фінансування з урахуванням ринкової кон'юнктури. У типовій схемі управління компанією займає пост віце-президента з фінансів та підзвітний президенту компанії або генеральному директору. Часто є членом ради директорів.
Сюжет	**Plot**	(від фр. sujet – предмет) – у літературі, драматургії, театрі, кіно та іграх – ряд подій (послідовність сцен, актів), що відбуваються в художньому творі (на сцені театру), та

		вбудованих для читача (глядача, гравця) за певними правилам демонстрації. Сюжет – основа форми твору.
Термін UA:	**Термін EN:**	**Опис:**
Тап	**Tap**	Короткий дотик пальця до певного місця на екрані, без його подальшого утримування на ньому.
Таск	**Task**	Завдання, поставлене конкретній особі, яке повинно бути виконано у встановлений термін.
Тач	**Touch**	Подія дотику до екрану
Тексел	**Texel**	(скорочення від англ. texture element) – мінімальна одиниця текстури тривимірного об'єкту. Піксел текстури.
Тестер	**Tester**	див. Тестувальник
Тестування апдейтов	**Testing of updates**	Вид тестування, який передбачає перевірку програми на коректність доданого в апдейт функціоналу, а також доопрацювання багів, які залишилися у роботі.
Тестування першої години гри	**First hour testing**	Під тестуванням «першої години» гри мається на увазі тестування відрізка гри до апселл скрина. Ця частина гри має бути максимально простою для проходження, навіть недосвідченим гравцем. У гравця не повинно виникати складнощів при проходженні, не повинно бути можливості «згорнути» її під час проходження гри до апселла. Гравець повинен дістатися до нього, не спіткнувшись на якому-небудь етапі. Тому особливу увагу при тестуванні першої години приділяється функціональному тестуванню.
Тестувальник	**Tester**	Спеціаліст, який проводить тестування програмного забезпечення з метою забезпечення та контролю якості програмного продукту або інформаційної системи.
Титри	**Credits**	див. Кредітси
Факап	**Fuck up**	Невдача, помилка, провал.
Ф.А.Кью	**Frequently Asked Question(s)**	FAQ, FAQ (акронім від англ. Frequently Asked Question (s) - питання, які часто задаються, вимовляється, «фак», «ФЕК», «факью», «еф-ей-кью») – збори поширених питань по якій-небудь темі та відповідей на них. Іноді зустрічається російський аналог цього скорочення – ЧАВО (що, як вважають, означає часті питання або ж актуальні питання та відповіді) або простий переклад англійської абревіатури ПЧЗ (питання, які часто задаються). Нерідко в рунеті зустрічаються й пряма транслітерація, ФАК («подивися в Факе»).
Фаллоф	**Falloff**	Похибка, після якої одна дія (наприклад, тап) вважається іншим (наприклад, свайпом).
Файд ефект	**Fade effect**	Ефект загасання, що використовується для більш плавного зникнення з екрану будь-яких об'єктів (зазвичай вікон, плашок).
Фідбек	**Feedback**	Відгук керівництва (наприклад, продюсера) після огляду гри, що представляє список вимог (багів, які необхідно виправити, або доповнень, які необхідно реалізувати). Вимоги з фідбек зазвичай є першочерговими для виконання.
Фікс	**Fix**	Процес виправлення помилок розробником.

Термін UA:	Термін EN:	Опис:
Фіксу	**Fiksu**	Система аналітики, що дозволяє відстежувати різні аспекти роботи мобільного додатку на пристроях користувача шляхом збору інформації про виконані дії користувачем та самих додатків. Процес перевірки інтеграції аналітики в додатках, описаних в платформенному чек-листі.
Фіча	**Software feature**	Фіча – (від англ. "feature" – особливість) функція, особливість, властивість. 1. Функція чого-небудь, наприклад, підтримка facebook нашими іграми; 2. Нетиповий результат дії програми, який може походити на збій або недоробки розробника, звідси відомий вислів «це не баг – це фіча ».
Флуд	**Flood**	Флуд (від невірно вимовного англ. "flood" – повінь, затоплення) – повідомлення, що займають великі об'єми та не несуть ніякої корисної інформації. Технічний флуд являє собою хакерську атаку з великою кількістю запитів, що приводить до відмови в обслуговуванні.
Флурі	**Flurry**	Аналог Fiksu.
Фонт	**Font**	см. Шрифт
Фріз	**Freeze**	У комп'ютерному сленгу – завмирання зображення на екрані, пов'язане з браком системних ресурсів комп'ютера (насамперед, оперативної пам'яті або відеопам'яті)
Фрі-ту-плей	**Free-to-play**	Спосіб поширення комп'ютерних ігор, що дозволяє користувачеві грати без внесення грошових коштів. Free-to-play позиціонується як безкоштовний, необмежений за часом доступ до всіх ігрових ресурсів, при цьому прибуток від гри досягається шляхом мікротранзакцій (поширення завантаження контенту або доступу до послуг, що надаються за невеликими цінами).
ФТП	**File Transfer Protocol**	FTP (File Transfer Protocol – протокол передачі файлів) – протокол, призначений для передачі файлів в комп'ютерних мережах.
ФТП сервер	**FTP server**	Сервер, що забезпечує обмін файлами по протоколу FTP.
Функціонал	**Functional**	Сукупність усіх функцій, доступних у грі.
Функціональне тестування	**Functional testing**	Тестування гри в цілях перевірки реалізованості функціональних вимог (описаних у специфікаціях), тобто здатності гри в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить гра, які завдання вона вирішує.
Хад	**Heads-Up Display**	Частина графічного інтерфейсу користувача, службовця для відображення важливої інформації безпосередньо під час ігрового процесу. Часто елементи HUD розташовуються по периферії екрану та / або напівпрозорі, щоб не заважати основному процесу гри. Елементами HUD можуть бути: кнопка виклику пауз-меню, кнопка підказки, інвентар, карта, окуляри здоров'я / енергії та ін.

Термін UA:	Термін EN:	Опис:
Хайден обджект гейм	**Hidden Object Game**	Жанр казуальних ігор, де основним завданням гравця є пошук різних предметів, різноманітні детективні розслідування, рішення логічних завдань, розгадка секретів, перевірка на спостережливість. У багатьох HO іграх також присутня квестова частина – це детективні ігри або ж ігри в детектива, тобто в них гравець стає детективом або сищиком, у завдання якого входять розкриття будь-якого злочину, пошуки зниклих людей або тварин, знаходження виходу із заплутаних ситуацій.
Хінт	**Hint**	Игрова підказка, яка активізується натиском відповідної кнопки і яка допомагає гравцю під час проходження. Підказка може вказувати на предмети, які необхідно знайти, або напрямок, куда треба йти, або дію, яку необхідно виконати.
Художник	**Artist**	Художники створюють графіку для однієї або декількох ігор. Художники відповідають за всі аспекти розробки ігор, які вимагають візуального мистецтва.
Цикл	**Cycle**	Різновид керуючої конструкції, призначена для організації багаторазового виконання набору інструкцій.
ЧАВО	**Frequently Asked Question(s)**	див. Ф.А.Кью.
Чаптер	**Chapter**	(з англ. Chapter – глава) В іграх важлива одиниця композиційного розділення, що є головною оповідною цензурою та позначає зазвичай тимчасову перерву протягом подій або при багатоплановому сюжеті – перехід від однієї сюжетної лінії до іншої.
Чекать	**Check**	від англ. Перевірити, тобто Чекать – це похідна від слова Чек означає Перевіряти.
Чекліст	**Checklist**	(англ. checklist – контрольний список, перелік, таблиця, карта) – список факторів, властивостей, параметрів, аспектів, компонентів, критеріїв або завдань, структурованих особливим чином з метою досягнення поставлених завдань.
Чіт	**Cheat**	(англ. cheat – «шахрайство», «обман»). Функціонал гри прихований від звичайних користувачів, що дає можливість впливати на ігровий процес.
Шоустоп	**Show stop**	Зависання гри або неможливість пройти далі.
Шрифт	**Font**	Шрифт (нім. Schrift ← schreiben – писати) – графічний малюнок накреслених літер, знаків, які складають єдину стилістичну та композиційну систему, набір символів визначеного розміру та малюнка.
Штраф	**Penalty**	Матеріальне або інше стягнення, яке стягується з особи або групи осіб за помилки в роботі.
Еір	**Airon App**	Мережевий ресурс з веб-інтерфейсом для зберігання останніх версій білдів по проектам.
ЕплАйДі	**AppleID**	Набір цифр, які вставляються в кінець посилання на сторінку з грою на AppStore, та є ідентифікатором гри. Актуальні Apple ID можна подивитися в документі "bundle_apple_id.xlsx", який знаходиться на СВН: svn: //svn.g5e.com: 3790 / g5documents / Cross_Promotion
ЕплАйДі	**AppID**	Набір символів виду 5PJ7R2GDWM.com.g5e.standofood. Потрібен

		для створення provision файлу. Наявність AppID візуально перевірити неможливо. Перевіркою наявності AppID є успішна установка гри на всі пристрої, які підтримуть її.
Термін UA:	**Термін EN:**	**Опис:**
Естімейт	**Estimate**	Оцінка кількості необхідних на певну задачу ресурсів (часу, людей, фінансів), надана експертом у певній галузі.
Етап	**Stage**	(від фр. étape – крок) – окремий момент, стадія якого-небудь процесу.
Юзер	**User**	див. Користувач