МІНІСТЕРСТВО ОСВІТИ І НАУКИ

Національний авіаційний університет

ПРАКТИКА З ТЕСТУВАННЯ ПЗ

Методичні рекомендації до виконання практики
для студентів спеціальності 123
«Комп'ютерна інженерія»
спеціалізації «Системне програмування»

Київ 2017

УДК 004.415.538(076.5)

ББК 3973.20-018.2р

П691

Укладачі: *С.М. Станко – завдання 1.1.-1.3, 2.1, 2.5*

І.-Ф.Ф. Кашкевич – завдання 1.4, 2.2-2.4

Рецензент: *Б.Я. Корнієнко* – д-р техн. наук, доцент

Затверджено методично-редакційною радою Національного авіаційного університету (протокол № ___ від _____ р.).

Практика з тестування ПЗ: методичні рекомендації до виконання
П691 практики / уклад.: С.М. Станко, І.-Ф.Ф. Кашкевич – К.: НАУ, 2017. – 48 с.

Містить загальні методичні рекомендації до виконання практики з тестування програмного забезпечення. Наведено стислий теоретичний матеріал за темами, порядок виконання роботи, завдання для самостійної роботи, контрольні запитання.

Для студентів спеціальності 123 «Комп'ютерна інженерія» спеціалізації «Системне програмування».

ЗАГАЛЬНІ МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Метою практики є оволодіння студентами сучасними методами тестування програмного забезпечення та формами організації праці в галузі їх майбутньої професії, формування та поглиблення на базі одержаних у вищому навчальному закладі знань, професійних умінь і навичок, здатності прийняття самостійних рішень під час конкретної роботи в реальних виробничих умовах.

Розглядаються основні положення тестування програмного забезпечення, тестова документація, методи та види тестування.

У результаті вивчення навчальної дисципліни студенти повинні знати основні етапи розроблення, форми та методи організації та тестування програмного забезпечення; вміти розроблювати тестові набори, складати план тестування, документувати всі етапи тестування, обирати найбільш оптимальні шляхи перевірки якості програмного забезпечення й аналізувати результати тестів.

Кожне завдання студент має виконувати індивідуально. Перед виконанням завдання студент повинен продемонструвати викладачеві досягнутий рівень самостійної підготовки: дати відповіді на додаткові запитання зі списку, наведеного в кінці кожного завдання. Після виконання завдання студент повинен показати результати роботи, відповісти на додаткові запитання, скласти письмовий звіт на основі результатів виконаної роботи.

Звіт має містити титульний аркуш, завдання, результати виконання, екранні форми та висновки.

МОДУЛЬ 1. ОСОБЛИВОСТІ ПРОЦЕСУ ТА ТЕХНОЛОГІЇ ТЕСТУВАННЯ

Завдання 1.1. Дослідження основних критеріїв вибору тестів

Мета роботи: ознайомитися з видами тестів, оволодіти навичками вибору тестів.

Основні теоретичні відомості

1. Основні визначення

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як «коректність», «повнота», «безпечність», але може містити більше технічних вимог, які описані в стандарті ISO/IEC 25010:2011. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою виявлення помилок.

Якість не є абсолютною – це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури. Тестування пронизує весь життєвий цикл програмного забезпечення, починаючи від проектування і закінчуючи невизначено тривалим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою

тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування.

Існує безліч підходів до вирішення завдання тестування та верифікації програмного забезпечення, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких.

З точки зору ISO/IEC 25010:2011 якість (програмних засобів) можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових: надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність. Склад і зміст документації, супутньої процесу тестування, визначається стандартом IEEE 829-2008 Standard for Software Test Documentation.

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування. Зазвичай виділяють наступні:

1) За об'єктом тестування:
- функціональне тестування (functional testing);
- тестування продуктивності (performance testing):
- навантажувальне тестування (load testing);
- стрес-тестування (stress testing);
- тестування стабільності (stability / endurance / soak testing);
- тестування зручності використання (usability testing);
- тестування інтерфейсу користувача (ui testing);
- тестування безпеки (security testing);
- тестування локалізації (localization testing);
- тестування сумісності (compatibility testing).

2) За знанням системи:
- тестування чорного ящика (black box);
- тестування білого ящика (white box);
- тестування сірого ящика (gray box).

3) За ступенем автоматизації:
- ручне тестування (manual testing);
- автоматизоване тестування (automated testing);

- напівавтоматизоване тестування (semiautomated testing).

4) За ступенем ізольованості компонентів:
- компонентне (модульне) тестування (component / unit testing);
- інтеграційне тестування (integration testing);
- системне тестування (system / end-to-end testing).

5) За часом проведення тестування:
- Альфа-тестування (alpha testing):
- тестування при прийманні (smoke testing);
- тестування нової функціональності (new feature testing);
- регресивне тестування (regression testing);
- тестування при здачі (acceptance testing);
- Бета-тестування (beta testing).

6) За ознакою позитивності сценаріїв:
- позитивне тестування (positive testing);
- негативне тестування (negative testing).

7) За ступенем підготовленості до тестування:
- тестування за документацією (formal testing);
- інтуїтивне тестування (ad hoc testing).

2. Критерії вибору тестів

Вимоги до ідеального критерію тестування наступні:

1) Критерій повинен бути достатнім, тобто показувати, коли деякої кінцевої кількості тестів достатньо для тестування даної програми.

2) Критерій повинен бути повним, тобто в разі помилки повинен існувати тест з безлічі тестів, що задовольняють критерію, який розкриває помилку.

3) Критерій повинен бути надійним, тобто будь-які дві множини тестів, які відповідають йому, одночасно повинні розкривати або не розкривати помилки програми.

4) Критерій повинен легко перевірятися, наприклад на обчислювальних тестах.

3. Класи критеріїв

Функціональні критерії формулюються в описі вимог до програмного продукту (критерії так званого «чорного ящика»), оскільки в цьому випадку вхідний код програми недоступний. Критерієм

повноти тестування вважається перебір всіх можливих значень вхідних даних. Основні функціональні критерії:

- Тестування класів вхідних даних - набір тестів, метою якого є перевірка зразкового екземпляру з кожного класу вхідних даних не менше одного разу.

- Тестування правил - набір тестів для перевірки кожного правила, якщо вхідні й вихідні значення описуються набором правил деякої граматики.

- Тестування класів вихідних даних - набір тестів для перевірки зразка з кожного вихідного класу.

- Тестування пунктів специфікації.

- Тестування функцій - набір тестів, який повинен забезпечити перевірку кожної дії, реалізованої модулем, що тестується, не менш одного разу.

- Комбіновані критерії для програм і специфікацій - набір тестів для перевірки всіх комбінацій несуперечливих умов програм і специфікацій не менш одного разу.

Структурні критерії (критерії так званого «білого ящика») використовують інформацію про структуру програми у вигляді потокового графа керування. Повним тестуванням у цьому випадку буде таке, що приведе до перебору всіх можливих шляхів на графі передач керування програми. Основні структурні критерії:

- Критерій тестування команд (критерій С0) - набір тестів у сукупності повинен забезпечити проходження кожної команди не менш одного разу.

- Критерій тестування рішень (критерій С1) - набір тестів у сукупності повинен забезпечити проходження кожної гілки не менш одного разу.

- Критерій тестування шляхів (критерій С2) - набір тестів у сукупності повинен забезпечити проходження кожного шляху не менш 1 разу.

Критерії стохастичного тестування формулюються в термінах перевірки наявності заданих властивостей у тестованої програми, засобами перевірки деякої статистичної гіпотези. Відповідно до даного методу створюється необхідна кількість незалежних тестів, у яких вхідні дані генеруються випадковим чином.

Мутаційний критерій полягає у штучному внесенні помилок у програму (мутації) - змінюються значення змінних, модифікуються

індекси й границі циклів, вносяться мутації в умови. Програми, що відрізняються від вихідних програм, штучно внесеними помилками називають мутантами. Якщо сформована множина тестових наборів виявляє всі мутації у всіх мутантах, то воно відповідає мутаційному критерію. Якщо тестування вихідної програми та мутанту на заданій множині тестових наборів не виявило помилок, то програма оголошується еквівалентною мутанту. У випадку мутаційного тестування важливо створити таке число мутантів, яке б охоплювало всі можливі ділянки прояву помилок.

Хід виконання роботи

1. Розглянемо використання функціональних критеріїв на прикладі наступної програми: програма приймає від користувача три цілих числа *A, B, C* – значення довжин сторін трикутника, визначає чи є даний трикутник рівностороннім, рівнобедреним, або ж простим. Набір вхідних даних представлений в табл. 1.

Таблиця 1

Таблиця набору вхідних даних для функціонального тестування

№ тесту	Мета тестування	Набір вхідних даних
1	2	3
1.	Трикутник є рівностороннім	47, 47, 47
2.	Трикутник є рівнобедреним	8, 8, 12 17, 23, 17 15, 89, 89
3.	Трикутник є простим	56, 78, 64
4.	Трикутник не існує	6, 5, 11 6, 5, 13 7, 9, 1 7, 9, 2 23, 8, 15 37, 8, 15
5.	Довжина сторони дорівнює нулю	0, 25, 37 48, 0, 37 28, 25, 0 0, 0, 0
6.	Довжина сторони від’ємна	-7, 5, 4 9, -3, 18 12, 10, -2

Продовження таблиці 1

1	2	3
7.	Довжина не є цілим числом	2.5, 8, 7 3, 7.2, 5 9, 8, 6.4
8.	Довжина не є числом	*, 2, 1 8, т, 12 5, 6, %
9.	Невірна кількість вхідних аргументів	1 2, 3 7, 19, 21, 16
10.	Реакція на числа, що знаходяться біля межі діапазону цілих чисел.	32000, 32000, 32000

2. Розглянемо приклад застосування мутаційного критерію. Для основної програми (лістинг 1), що знаходить кількість одиниць кожного розряду п'ятизначного числа, створюємо дві програми-мутанти. В першій (лістинг 2) змінюємо значення змінної n з 5 на 4. В другій програмі-мутанті (лістинг 3) змінюємо початкове значення лічильника циклу з n на n-1.

Лістинг 1. Основна програма знаходження кількості одиниць кожного розряду п'ятизначного числа:

```
int main()
{
      int n=5;
      int num, i;
      int M[n]={0};
      printf("Enter number in 5 digits: ");
      scanf("%d", &num);
      for (i=n; i >=0; i--)
      {
            M[i-1] = num % 10;
            if(num > 10)
                  num = (num - M[i-1])/10;
      }

      for (i = 0; i <= n; i++)
            printf("%d\n", M[i]);
      return 0;
```

```
}
```

Лістинг 2. Перша програма-мутант

```
int main()
{
      int n=4;
/*змінене значення змінної з 5 на 4 */
      int num, i;
      int M[n]={0};
      printf("Enter number in 5 digits: ");
      scanf("%d", &num);
      for (i=n; i >=0; i--)
      {
            M[i-1] = num % 10;
            if(num > 10)
                  num = (num - M[i-1])/10;
      }

      for (i = 0; i <= n; i++)
            printf("%d\n", M[i]);
      return 0;
}
```

Лістинг 3. Друга програма-мутант

```
int main()
{
      int n=5;
      int num, i;
      int M[n]={0};
      printf("Enter number in 5 digits: ");
      scanf("%d", &num);
      for (i=n-1; i >=0; i--)
/*змінене початкове значення лічильника з n на n-1 */
      {
            M[i-1] = num % 10;
            if(num > 10)
                  num = (num - M[i-1])/10;
      }
      for (i = 0; i <= n; i++)
```

```
            printf("%d\n", M[i]);
      return 0;
}
```

Запуск тестів з вхідними значеннями 12345, 10345, 27893 дозволив виявити помилки в основній програмі, а також в програмах-мутантах.

Завдання на виконання

1. Написати власну версію програми, описаної в ході виконання роботи. При складанні програми врахувати всі наведені вище тести. Здати готовий текст програми викладачу.
2. Отримати у викладача програму, написану іншим студентом. Провести її тестування.
3. Результати тестування занести в звіт.
4. Отримати у викладача програмний додаток для тестування.
5. Дослідіть основні критерії вибору тестів для отриманої програми.
6. Опишіть результати роботи в звіті.

Контрольні запитання

1. Що таке тестування?
2. Чим визначається якість програмного забезпечення?
3. Опишіть вимоги до критеріїв тестів.
4. Які класи критеріїв тестів ви знаєте?
5. В чому полягає сутність критерії «білого ящика»?
6. В чому відмінність критеріїв «чорного ящика» від критеріїв «білого ящика»?
7. Опишіть принципи критеріїв стохастичного тестування.

Завдання 1.2. Тестування технічного завдання до програми

Мета роботи: оволодіти навичками тестування технічного завдання до програмного забезпечення.

Основні теоретичні відомості

При перевірці технічного завдання до програмного забезпечення враховуються наступні критерії:

1) Повнота. Кожна вимога має повністю описати функціональні можливості, які будуть отримані. Документ повинен містити всю інформацію, необхідну розробнику для розробки і реалізації.

2) Правильність. Кожна вимога має точно описати функціональність, яка буде побудована. Тільки представники користувачів можуть визначити правильність призначених для користувача вимог, тому користувачі або їхні представники повинні переглянути вимоги.

3) Здійсненність. Повинна бути забезпечена можливість реалізувати кожну вимогу в межах відомих можливостей і обмежень системи і її операційного середовища. Розробник може забезпечити перевірку реальності можливості або неможливості виконання, а також виконання лише за надлишкової вартості.

4) Необхідність. Кожна вимога має документувати можливості того, що клієнти дійсно потребують відповідно до вимог до зовнішньої системи або стандарту. Кожна вимога повинна виходити з положення, яке має повноваження визначати ці вимоги.

5) Пріоритетність. Призначають пріоритет реалізації для кожної функціональної вимоги, функції.

6) Недвозначність. Всі читачі вимоги повинні прийти до єдиної, послідовної інтерпретації прочитаного.

Хід виконання роботи

Проаналізуємо для прикладу деякі вимоги, зазначені в технічному завданні.

Вимога 1. Програмний додаток повинен підтримуватись усіма версіями всіх операційних систем.

Зауваження:

1. Які саме операційні системи підтримують продукт?

2. Які версії даних операційних систем можуть використовуватись?

3. Які вимоги до апаратного забезпечення?

Вимога 2. Будь-яка інформація про сім'ю може бути розміщена в одному файлі, який буде збережено з пустими полями.

Зауваження:

1. Про яку саме сім'ю заноситься інформація?

2. В одному файлі зберігається інформація про всі сім'ї чи для кожної з них створюється окремий файл?

3. Навіщо зберігати пусті поля?

Вимога 3. Декілька додатків можуть бути підключені одночасно.

Зауваження:

1. Що це за додатки?

2. Як саме користувач буде одночасно запускати кілька додатків?

Вимога 4. Коли запускається додаток, вікно повинно містити меню з трьома стандартними кнопками «Нова база даних», «Відкрити Базу даних», «Зберегти базу даних».

Зауваження:

1. Потрібно акцентувати увагу на вкладки, а лише потім описувати кнопки.

2. При описі кнопок немає їх зображень і користувачу доведеться здогадуватись про їхні назви.

Завдання на виконання

1. Отримати у викладача технічне завдання для програмного додатку.
2. Провести детальний аналіз тексту отриманого технічного завдання.
3. Результати аналізу занести до звіту.

Контрольні запитання

1. Яке призначення у технічного завдання?
2. Які критерії використовуються при перевірці технічного завдання?
3. Хто має змогу перевірити здійсненність вимог?
4. Хто має повноваження визначати правильність вимог технічного завдання?

Завдання 1.3. Створення тестових наборів програми

Мета роботи: оволодіти навичками створення тестових наборів програми.

Основні теоретичні відомості

Тестовий набір – це комплект перевірочної вхідної інформації, умов виконання та очікуваних результатів, розроблених для конкретної мети, наприклад, відстеження виконання програми або для перевірки відповідності до конкретного технічного завдання. Перерахуємо основні групи тестів:

- зручність користування;
- функціонування;
- помилкові введення;
- порівняння з аналогами;
- перевірка супровідної документації;
- сумісність з іншими продуктами або системами;
- зовнішній вигляд (графічний інтерфейс);
- тестування на екстремальні навантажувальні умови;
- вимоги – чи є вимоги і перевірити відповідність до них.

Оформлюються тестові набори у вигляді таблиці, що складається з наступних полів (табл. 2):

– перевірочна дія;
– очікуваний результат;
– фактичний результат.

Таблиця 2

Структура таблиці тестових наборів

Перевірочна дія	**Очікуваний результат**	**Фактичний результат**
Передумови		
виконати дію А1	перевірити стан В1	виконано
виконати дію А2	перевірити стан В2	помилка
Опис тестового випадку		
виконати дію А3	перевірити стан В3	заблоковано
Післяумови		
виконати дію А4	перевірити стан В4	

Тестовий набір включає в себе 3 частини:

- Передумови – список кроків, які приводять систему, що перевіряється до стану, придатного для тестування, або список перевірок умов того, що система вже знаходитися в необхідному стані.

- Опис тестового випадку– список дій, за допомогою яких здійснюється основна перевірка функціоналу.

- Післяумови - список дій, які повертають систему в початковий стан.

Тестування програмного продукту завжди починається з так званого «димового тестування». Цей вид тестування являє собою поверхневу перевірку всіх модулів програми на предмет працездатності і наявність критичних і блокуючих дефектів. За результатами димового тестування робиться висновок про те, чи приймається встановлена версія програмного забезпечення в тестування або експлуатацію.

Хід виконання роботи

Для прикладу візьмемо програму Notepad. Опишемо для обраної програми тестові набори.

1. Опишемо сценарій димового тестування:

1.1. Запустити програму.

1.2. Відкрити новий файл.

1.3. Ввести текст.

1.4. Вивести текст на друк.

1.5. Зберегти файл.

1.6. Перевірити. чи внесені у файл зміни дійсно збереглись. Для цього слід закрити даний файл і знову його відкрити.

1.7. Перевірити можливість редагування тексту.

2. Опишемо послідовність формування переліку критичних тестів:

2.1. Описати всі способи запуску програми.

2.2. Розбити тестування програми на окремі частини відповідно до функціоналу програми: кожен пункт меню – це окрема частина.

2.3. Описати перевірку кожної частини, деталізуючи до перевірки кожного окремого елемента, який входить в зазначену частину тестування.

Завдання на виконання

1. Отримати у викладача програмний додаток для тестування та технічне завдання до нього.

2. Створити тестовий набір для перевірки отриманого програмного додатку.

3. Занести таблицю тестових наборів до звіту з практики.

Контрольні запитання

1. Що таке тестовий набір?
2. Для чого використовується димове тестування?
3. З чого складається тестовий набір?
4. Які основні групи тестів зазвичай входять до тестового набору?
5. Як оформлюються тестові набори?

Завдання 1.4. Створення плану тестування програми

Мета роботи: оволодіти навичками створення плану тестування програми

Основні теоретичні відомості

План тестування – це документ, що описує обсяг, підхід, ресурси і графік намічених контрольних заходів. Він визначає тестові завдання, функції для перевірки, завдання тестування для виконання кожної задачі, і будь-які ризики, що вимагають планування на випадок непередбачених обставин [1].

Типова структура плану тестування згідно зі стандартом IEEE Std 829-2008 виглядає наступним чином:

1. Вступ
1.1. Ідентифікатор документу
1.2. Сфера дії
1.3. Посилання
1.4. Рівень в загальній послідовності
1,5. Класи тестувань і загальні умови проведення випробувань
2. Деталі тестового плану
2.1 Тестові пункти та їх ідентифікатори
2.2 Матриця простежуваності тесту
2.3 Властивості, які підлягають тестуванню
2.4 Властивості, які не підлягають тестуванню
2.5 Підхід
2.6 Критерії успішності або провалу тесту
2.7 Критерії призупинення і вимоги відновлення тестування
2.8 Очікувані результати тестів
3. Управління тестуванням
3.1 Плановані заходи і завдання; проходження тесту
3.2 Навколишнє середовище / інфраструктура
3.3 Обов'язки та повноваження
3.4 Інтерфейси між сторонами, які беруть участь в розробці
3.5 Ресурси та їхній розподіл
3.6 Навчання
3.7 Розклади, кошториси та витрати
3.8 Ризики і форсмажорні обставини

4. Загальна частина

4.1 Процедури контролю якості

4.2 Метрики

4.3 Покриття тестами

4.4 Словник

4.5 Процедури зміни документу та історія

Хід виконання роботи

Розглянемо наповнення плану тестування згідно зі стандартом IEEE Std 829-2008.

1. Вступ

Цей розділ ідентифікує документ, а також життєвий цикл проекту і затрати ресурсів на його тестування. У цьому розділі також визначаються види і умови випробувань для певного рівня тестування.

1.1. Ідентифікатор документу

Унікально ідентифікує версію документу (інформація про дату випуску, організацію, авторів, затвердження і статус або версію).

1.2. Сфера дії

Узагальнює програмний продукт, або системний елемент та функції, які повинні бути протестовані на даному рівні тестування.

1.3. Посилання

Список всіх використаних довідкових документів. Існують «зовнішні» (зовнішні по відношенню до проекту) та «внутрішні» (в межах проекту) посилання.

1.4. Рівень в загальній послідовності

Відображає місце даного рівня в загальній ієрархії тестування.

1,5. Класи тестувань і загальні умови проведення випробувань

Узагальнений унікальний характер даного рівня тесту, а також описує умови проведення випробувань.

2. Деталі тестового плану

У цьому розділі описані конкретні елементи, які будуть підлягати тестуванню на даному рівні, а також матриця зв'язків елементів.

2.1 Тестові пункти та їх ідентифікатори

Ідентифікація елементів тестування, які є об’єктами випробування, такі, як специфічні атрибути, інструкції з встановлення, інструкції користувача, взаємодія апаратного забезпечення, програм-

не забезпечення для перетворення даних, включаючи їх версію. А також ідентифікує будь-які процедури, які передаються до тестованого обладнання з іншого.

2.2 Матриця простежуваності тесту

Надає список вимог(програмного забезпечення та/або системи; таблиці чи бази даних), які будуть здійснюватись на даному рівні тесту і відповідні текст кейси чи процедури.

2.3 Властивості, які підлягають тестуванню

Визначає всі можливості програмного продукту чи програмного забезпечення і комбінації функцій програмного забезпечення чи системи, які підлягають тестуванню.

2.4 Властивості, які не підлягають тестуванню

Визначає всі особливості і відомі суттєві комбінації функцій, які не будуть протестовані та обґрунтування щодо їх виключення з тесту.

2.5 Підхід

Опис загального підходу тестування на даному рівні. А також підхід для кожної основної функції, або групи функцій, який буде гарантувати, що вони перевірені належним чином.

2.6 Критерії успішності або провалу тесту

Визначає критерії, які будуть використовуватись для визначення успішності проходження тесту.

2.7 Критерії призупинення і вимоги відновлення тестування

Визначає критерії, що використовуються для припинення всіх, або частини тестів, а також заходи, які повинні буди проведені при поновленні тестування.

2.8 Очікувані результати тестів

Визначає всю інформацію, яка повинна бути отримана після проходження тестів (документи, дані, та ін.).

3. Управління тестуванням

У цьому розділі описуються тестові роботи та завдання для даного рівня, а також рівень і їх прогрес. Тут визначається інфраструктура, обов’язки, повноваження, ресурси, навчання, графіки і ризики, якщо вони не були визначені на більш високому рівні документу.

3.1 Плановані заходи і завдання; проходження тесту

Визначає набір завдань, необхідних для підготовки та проведення тестування, та обмеження.

3.2 Навколишнє середовище / інфраструктура

Опис необхідних та бажаних властивостей середовища що тестується і відповідних даних.

3.3 Обов'язки та повноваження

Визначення особи, або групи осіб, відповідальних за управління, проектування, підготовку, виконання і перевірку результатів даного рівня тестування, а також вирішення знайдених невідповідностей.

3.4 Інтерфейси між сторонами, які беруть участь в розробці

Опис засобів і змісту зв’язків між окремими особами і групами, що зазначені в плані тестування.

3.5 Ресурси та їхній розподіл

Опис будь-яких додаткових необхідних ресурсів, які ще не задокументовані в інших частинах плану.

3.6 Навчання

Визначення варіантів навчання необхідним навичкам для проведення тестування .

3.7 Розклади, кошториси та витрати

Включає всі тестові етапи, визначені в програмному забезпеченні чи графіку системного проекту і супровідні заходи та витрати.

3.8 Ризики і форсмажорні обставини

Виявлення потенційних ризиків, які можуть негативно вплинути на успішне завершення випробувань на даному рівні тестування.

4. Загальна частина

У цьому розділі описуються процедури контролю якості та метрики, глосарій і інформація про частоту та спосіб зміни документу.

4.1 Процедури контролю якості

Визначає засоби забезпечення якості процесів тестування процесів і продуктів та відстежуються аномалії

4.2 Метрики

Визначаються конкретні показники, які будуть зібрані, проаналізовані та повідомлені. Ці показники використовуються тільки на конкретному рівні тестування.

4.3 Покриття тестами

Визначення рівня покриття тест-кейсами тестованого елементу.

4.4 Словник

Алфавітний список термінів та їх визначення, які використовувались в плані тесту.

4.5 Процедури зміни документу та історія

Вказує засоби для ідентифікації, затвердження, реалізації та засобу змін документу.

Завдання на виконання

1. Отримати у викладача програмний додаток для тестування та технічне завдання до нього.
2. Отримати у викладача зразок плану тестування. Проаналізувати даний документ.
3. Створити власний план тестування отриманого програмного додатку.
4. Створений документ плану тестування роздрукувати і долучити до звіту.

Контрольні запитання

1. Що таке план тестування?
2. Які документи регламентують оформлення плану тестування?
3. Що може входити до плану тестування?

МОДУЛЬ №2 «ДОСЛІДЖЕННЯ РІЗНОВИДІВ ТЕСТУВАННЯ ПЗ»

Завдання 2.1. Модульне тестування програми

Мета роботи: оволодіти навичками модульного тестування.

Основні теоретичні відомості

Модульне тестування – ізольоване тестування кожного окремого елемента програмного продукту в штучно створеному середовищі. Перевірка окремих елементів виконується з використанням драйверів і заглушок.

Елемент – це найменший компонент програмного забезпечення, який можна протестувати.

Драйвер – певний модуль тесту, який виконує елемент, що тестується.

Заглушка – частина програми, яка симулює обмін даними з компонентом, що тестується, і виконує імітацію робочої системи.

Заглушки потрібні для:

– імітації відсутніх компонентів для роботи даного елемента.

– подачі або повернення модулю певного значення.

– надання можливості тестувальнику самому ввести потрібне значення.

– відтворення певних ситуацій (виключення або інші нестандартні умови роботи елемента).

Завдання модульного тестування – це довести, що кожен окремий модуль є працездатним.

Хід виконання роботи

Проведемо модульне тестування з використанням NUnit - інструмента для модульного тестування додатків .Net, який дозволяє писати тести безпосередньо в середовищі розробки.

1. Створюємо новий проект в Visual Studio. Обрати тип проекту – «Консольний додаток C#».

2. Створимо функцію, що обчислює степінь цілого додатного числа з наступним кодом:

```
[code=c#]
static UInt32 Degree(UInt32 x, UInt32 y) {
      UInt32 i;
      if (x < 0 || y < 0)
            exit 1;
      for (i = 2; i <= y; i++)
            x *= x;
      return x;
} [/code=c#]
```

3. Додати в створений проект посилання на бібліотеку NUnit. Для цього необхідно обрати меню Проект->Додати посилання, у вікні, що з'явилось, на вкладці .NET обрати пункт nunit.framework (рис. 1).

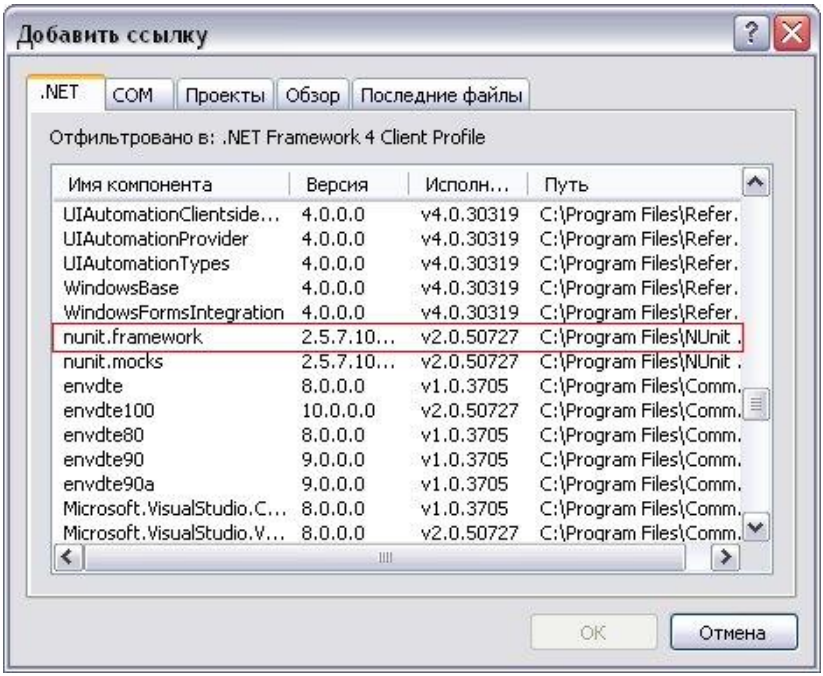


Рис. 1 – Створення посилання на бібліотеку NUnit

4. Підключити відповідний простір імен:

```
[code=c#]using NUnit.Framework;[/code=c#]
```

5. До головного класу Program додати атрибут [TestFixture] наступним чином:

```
[code=c#]
[TestFixture]
class Program
[/code=c#]
```

6. Для нашої функції тестовими випадками будуть наступні твердження:

1) результатом функції повинно бути ціле число;

2) результат роботи функції повинен бути невід'ємним;

3) функція повинна повертати степінь числа.

7. Для тестування створимо зовнішню функцію MyTestFunc() і встановлюємо їй атрибут [Test]. Її текст буде виглядати наступним чином:

```
[code=c#]
[Test]
public static void MyTestFunc() {
UInt32 x = 3, y = 5;
/// тест 1. результатом функції повинно бути ціле число
Assert.IsInstanceOfType(typeof(UInt32),
                          Degree(x,y));
/// тест 2. результат роботи функції повинен бути невід'ємним
Assert.IsTrue(Degree(x, y)>=0);
/// тест 3. функція повинна повертати степінь числа
Assert.AreEqual((UInt32)Math.Pow(x,y),
                 Degree(x,y));
} [/code=c#]
```

8. Скомпілювати та запустити програму на виконання.

9. Запустити NUnit. Відкрити в ньому скомпільований файл проекту.

10. Запустити на виконання тестування, натиснувши кнопку Run. Перевірка пройшла успішно. Можна зробити висновок, про те, що наша функція працює коректно.

11. Внесемо умисні помилки до нашої функції, наприклад замінимо початкове значення лічильника i з 2 на 0:

```
[code=c#]
static UInt32 Degree(UInt32 x, UInt32 y)
{
      UInt32 i;
      if (x < 0 || y < 0)
            exit 1;
      for (i = 0; i <= y; i++)
            x *= x;
      return x;
}
[/code=c#]
```

12. Запустити тести в вікні NUnit. Змінена функція тестування не пройшла.

Завдання на виконання

1) Отримати у викладача програмний додаток для тестування.
2) Провести модульне тестування отриманого додатку за допомогою NUnit.
3) Результати тестування занести до звіту з практики.

Контрольні запитання

1) В чому полягає модульне тестування?
2) Що таке елемент модульного тестування?
3) Які функції виконує драйвер в модульному тестуванні?
4) Яке призначення у заглушки в модульному тестуванні?
5) Опишіть особливості використання NUnit для модульного тестування.

Завдання 2.2. Інсталяційне тестування програми

Мета роботи: оволодіти навичками інсталяційного тестування програми

Основні теоретичні відомості

Інсталяційне тестування – один з видів нефункціонального тестування, несе відповідальність за якість установки програмного забезпечення, його оновлення та деінсталяцію.

Тестування установки необхідно проводити при створенні програмного забезпечення, після появи нової версії, а також при зміні конфігурації. Тип тестування буде залежати від декількох факторів:

- які платформи та операційні системи підтримуються;
- яким чином буде поширюватися програмне забезпечення;
- хто буде виконувати установку програмного забезпечення, тощо.

Інсталяційне тестування є перевіркою не тільки самого процесу установки програмного забезпечення, а й плану установки, що забезпечує ефективність і надійність інсталяції програмного забезпечення.

Хід виконання роботи

Розглянемо перевірки, які необхідно проводити при проведенні інсталяційного тестування:

1) Встановлення повинно початися при натисканні на кнопку, яка підтверджує дану дію.
2) Встановлення у всіх підтримуваних середовищах і на всіх підтримуваних платформах.
3) Встановлення в середовищах, що не підтримуються, а також в середовищах з некоректними налаштуваннями.
4) Права, які вимагає інсталяція, найчастіше повинні бути адміністраторські. Перевірити встановити додаток як гість.
5) Встановлення при відсутності будь-яких можливих пов'язаних файлів і попередніх версій.
6) Чи підраховується при встановленні кількість вільного місця на диску і чи видається попередження, якщо місця недостатньо.

7) Встановлення завантаженого попередньо додатку, а також пряме встановлення з використанням мережі.

8) Чи відновлюється процес встановлення при раптовому його переривання (відключення пристрою, відмова мережі, відключення бездротового з'єднання тощо).

9) Чи розпізнається наявність в системі програмних додатків, необхідних для коректної роботи програми, що встановлюється.

10) Повторний запуск установки додатка при вже поточному встановленому повинен видавати коректне повідомлення про те, що подвійне встановлення неможливе.

11) Процес встановлення може налаштовуватись або прийматись за умовчанням. Переконатися що обидва способи коректно працюють.

12) Наявність кнопки, яка запропонує зберегти додаток в певну папку і так само вказує розташування за умовчанням.

13) Чи правильно встановлені та збережені в коректних папках файли програми.

14) Наявність створених ярликів, чи коректно вони розташовані.

15) Після встановлення в системній вкладці "Програми та засоби" повинні бути доступні: назва програми, іконка, ім'я розробника, розмір програми, дата встановлення та номер версії

16) Налаштування змінних середовища PATH

17) Переконатися, що ліцензійний ключ зберігається в Windows Registry library.

18) Чи підтримує додаток функції 'UnInstall', 'Modify', 'ReInstall' і чи коректно вони працюють.

19) Робота програми з вже існуючими DLL-файлами, з DLL-файлами додатків, які необхідні для коректної роботи програми, що встановлюється.

20) Наявність інформації або повідомлення про те, коли закінчується термін дії встановленої пробної версії додатка.

21) Чи підтримує додаток функцію оновлення й автооновлення.

22) Чи зберігаються призначені для користувача налаштування при спробі завантажити нову версію або оновлення старої версії

23) При спробі оновити версію повинні бути доступні функції видалити додаток і відновити додаток

24) При оновленні переконатися, що номер версії програми змінився на новий.

25) Запустити програму і переконатися що вона працює коректно.

26) Спробувати встановити стару версію на новішу.

27) Наявність коректного повідомлення при спробі відкату.

28) Чи не залишається в системі якихось папок, файлів, ярликів або ключів реєстру після повного видалення програми.

29) Чи коректно працює система після установки і подальшого видалення програми.

Завдання на виконання

1) Отримайте у викладача програмний додаток для тестування.

2) Проведіть інсталяційне тестування отриманого програмного додатку, виконавши дії, описані в ході виконання даної роботи.

3) Результати перевірки занесіть у звіт з практики.

Контрольні запитання

1) Що таке інсталяційне тестування?

2) Що перевіряється під час інсталяційного тестування?

3) У яких випадках застосовується інсталяційне тестуванні?

4) Що потрібно перевіряти після встановлення нового програмного забезпечення?

5) Що потрібно перевіряти після деінсталяцій програмного забезпечення?

6) Що потрібно перевіряти після оновлення програмного забезпечення?

Завдання 2.3. Функціональне тестування користувацького інтерфейсу

Мета роботи: оволодіти навичками роботи з багатовимірними масивами

Основні теоретичні відомості

GUI тестування являє собою процес тестування користувальницького інтерфейсу програми та переконатися в тому , що вона підтверджує вимоги до конструкції.

Зручність використання призначеного для користувача інтерфейсу – показник його якості, який визначає кількість зусиль, необхідних для вивчення принципів роботи з програмною системою за допомогою даного інтерфейсу, її використання, підготовки вхідних даних та інтерпретації вихідних. Тобто, зручність використання визначає ступінь простоти доступу користувача до функцій системи, що надаються безпосередньо через людино-машинний інтерфейс.

На зручність використання призначеного для користувача інтерфейсу впливають такі чинники:

- легкість навчання – чи швидко людина вчиться використовувати систему;
- ефективність навчання - наскільки швидко людина працює після навчання;
- запам’ятовуваність навчання – чи легко запам’ятовується все, чого людина навчилась;
- помилки – наскільки часто людина припускається помилок в роботі;
- загальна задоволеність – наскільки позитивним є загальне враження від роботи з системою.

Перевірка інтерфейсу додатку на зручність полягає в:

1) Оцінці відповідності дизайну додатку до його функціональності, заданої замовником.

2) Аналізі використовуваних графічних елементів, колірного оформлення з точки зору сприйняття.

3) Оцінці зручності навігації і структури посилань.

4) Аналізі текстового наповнення сайту.

5) Оцінці зручності використання функцій програми

6) Аналізі шрифтового оформлення тексту.

Як правило, при тестуванні зручності використання призначеного для користувача інтерфейсу використовуються деякі евристичні критерії і характеристики, які замінюють точні оцінки в класичному тестуванні програмних систем.

Хід виконання роботи

Виділимо основні критерії для функціонального аналізу користувацького інтерфейсу.

1. Текстове поле.

1.1. Перемістіть курсор миші над усіма текстовими полями, що призначені для вводу даних. Курсор повинен змінитися від стрілки на панель для вставки.

1.2. Якщо цього не станеться, то текст в полі повинен бути сірим або залишатись незмінним.

1.3. Спробуйте переповнювати поле вводу, набравши будь-яку кількість символів.

1.4. Введіть неприпустимі символи – букви в числові поля, спробуйте такі символи, як +, - * тощо у всіх областях.

1.5. Комбінація клавіш «SHIFT» + «Стрілка» повинна виділяти символи. Виділення повинне бути також можливим з допомогою миші. Подвійне натискання на кнопку миші повинне виділяти весь текст в полі.

2. Радіо-перемикач

2.1. Ліві і праві стрілки повинні рухатися по відбору. Також мають працювати стрілки вгору і вниз.

2.2. Виберіть з допомогою миші, натиснувши кнопку.

3. Прапорці

3.1. Натискання мишею на полі або на тексті повинне призводити до включення або виключення даного елемента.

3.2. Клавіша «Пробіл» повинна викликати ті ж самі дії.

4. Командні кнопки

4.1. Якщо кнопка викликає призводить до виклику іншого екрану, і якщо користувач може ввести або змінити деталі на цьому іншому екрані, то текст на кнопці повинен бути доповнений трьома крапками.

4.2. Всі кнопки для підтвердження та скасування дій повинні бути підписані та мати доступ до них. Про це свідчать підкреслені букви в тексті кнопки. Кнопка повинна бути активована натисненням комбінації клавіш «ALT» + виділена буква.

4.3. Натисніть на кожну кнопку один раз за допомогою миші - це має її активувати.

4.4. Натисніть клавішу «Пробіл» - це має активувати кнопку.

4.5. Натисніть клавішу «RETURN» - ця дія повинна активізувати дану кнопку.

5. Естетичні умови

5.1. Чи правильний колір було обрано для загального фону екрану?

5.2. Чи правильний колір було обрано для поля підказки?

5.3. Чи правильний колір було обрано для фону поля?

5.4. В режимі тільки для читання чи правильний колір застосовано до підказки поля?

5.5 В режимі тільки для читання чи правильний колір застосовано до фону поля?

5.6. Чи всі екранні підказки позначені правильним екранним шрифтом?

5.7. Чи у всіх областях екрану для тексту обрано правильний шрифт?

5.8. Наскільки ідеально вирівняні підказки всіх полів на екрані?

5.9. Наскільки ідеально вирівняні всі поля редагування на екрані?

5.10. Чи можливо змінювати розмір вікна?

5.11. Чи можливо мінімізувати розмір вікна?

5.12. Чи вводяться дані з врахуванням верхнього та нижнього регістрів послідовно?

Завдання на виконання

1) Отримайте у викладача програмний додаток для тестування.
2) Проведіть функціональне тестування отриманого програмного додатку, виконавши дії, описані в ході виконання даної роботи.
3) Результати перевірки занесіть у звіт з практики.

Контрольні запитання

1) В чому полягають особливості тестування графічного інтерфейсу користувача?
2) Які тести потрібно виконати над самим вікном програми?
3) Яким чином перевіряються поля вводу?
4) Які особливості у тестуванні радіоперемикача?
5) Як тестується перемикач «прапорець»?
6) Яким чином тестуються кнопки?

Завдання 2.4. Регресивне тестування програми

Мета роботи: оволодіти навичками регресивного тестування

Основні теоретичні відомості

Розробка і підтримка програмного забезпечення – це процес, який включає виправлення помилок, оптимізацію, розширення і іноді усунення існуючого функціоналу. Всі ці модифікації можуть стати причиною некоректної роботи системи.

Мета регресивного тестування – переконатися, що нові зміни в коді не вплинули негативно на існуючий функціонал. Проведення даного виду тестування гарантує, що старий код все ще працює в той час як зроблені нові зміни в коді в тому числі ті, які спрямовані на усунення раніше виявлених помилок.

Регресивне тестування необхідно проводити в наступних випадках

1) зміни у вимогах і зміна в коді згідно нових вимог;
2) додавання нового функціоналу;
3) виправлення функціонального дефекту;
4) виправлення нефункціонального дефекту;

Згодом набори регресивних тестів стають досить великими. Повне їх виконання вимагає великих витрат часу і коштів. Стає складно зменшувати тестові набори з метою досягнення максимального покриття коду.

Регресивне тестування може проводиться з використанням наступних технік:

1) повторний прогін всіх існуючих тестових наборів. Це один з методів регресивного тестування, де всі існуючі тестові набори відтворюються повторно. Це дуже дорогий метод, так як це вимагає величезної кількості часу і коштів;

2) відбір тестових наборів для регресивного тестування. Щоб повторно не відтворювати цілі набори тестів, проводять їх відбір базуючись на певних умовах, а саме:

- тестові набори, які при попередніх тестуваннях виявили найбільшу кількість дефектів;
- функціонал, найбільш видимий або часто використовуваний користувачем;
- тестові набори, які перевіряють основний функціонал продукту;
- тестові набори, де функціонал піддавався змінам найбільше і функціонал з недавніми змінами;
- всі інтеграційні тестові набори;
- всі складні тестові набори;
- тестові набори, де проводилося тестування граничних значень;
- приклади успішно пройдених тестових наборів;
- приклади тестових наборів, які не пройшли перевірку.

3) приорітезація тестових наборів у залежності від бізнес-логіки, критичності і частоти використання функціоналу. Відбір тестових наборів, базуючись на пріоритетності, значно зменшить кількість регресивних тест кейсів.

Якщо продукт піддається частим змінам, витрати на регресивне тестування збільшуються. Ручне виконання тест кейсів вимагає великих витрат часу і ресурсів. Тому в більшості випадків вдаються до автоматизації регресивного тестування. Для автоматизації тестування використовують різні інструменти. Одним з найбільш популярних є Selenium. Також використовується: Quick Test Professional, Rational Functional Tester, Microsoft Visual Studio for Testers.

У процесі розробки програмного забезпечення код постійно модифікується, частота релізів або ітерацій безпосередньо впливає на кількість тестування. Існують «гнучкі» методології розробки програмного забезпечення, такі як Agile: Scrum, XP, Kanban, Lean, пропагують короткі ітерації, відповідно істотно збільшується кількість необхідного регресивного тестування.

Щоб гарантувати ефективне регресивне тестування потрібно дотримуватися певних правил:

– код, який піддається регресивному тестуванню, повинен бути під контролем так званого інструменту управління конфігураціями. Це потрібно для того, щоб відстежувати зміни в коді;

– неприпустимо вносити зміни в код в процесі регресивного тестування;

– база даних, яка використовується для регресивного тестування, повинна бути ізольована. Неприпустимою є зміна бази даних.

Є кілька видів регресивних тестів:

1) Верифікаційні тести, які можна розділити на:

- Санітарне тестування – проводиться для того, щоб переконатися в виправленні помилок, виявлених в попередній збірці проекту, а також що внесений дефект не пошкодив суміжний функціонал.

- Димове тестування, яке представляє собою набір тестів, мета яких – переконатися, що в новій збірці проекту основна функціональність програми не порушена. Тобто, димове тестування – це коротке тестування всіх основних функцій виготовленого ПЗ.

2) Безпосередньо саме регресивне тестування – повторне виконання тестів, які були написані і проведені в попередніх збірках і не виявили помилок.

3) Тести які проводилися в попередніх збірках і виявили помилки. Надалі помилки були виправлені і закриті. Але якщо в ході розробки ділянки коду, де був раніше виявлений дефект змінився, то швидше за все помилка проявиться знову. Щоб цього не допустити необхідно час від часу проводити тести, які раніше виявили помилки.

Починати регресивне тестування потрібно з димового. Якщо хоча б один тест призводить до дефекту, то більш детальне тестування вважається недоречним. Збірка повертається на доопрацювання.

Якщо димове тестування пройшло успішно, то проводиться санітарне тестування.

З регресивних тестів, які раніше не виявили помилок, відбираються ті, які тим чи іншим способом «стикаються» зі змінами в збірці.

Аналогічним чином відбираються тести для регресивного тестування раніше виявлених помилок.

Регресивні тести, які успішно виконувалися і не приводили до помилки протягом декількох збірок, вважаються закритими.

Для тестів регресії, які передбачається проводити більше 3-5 разів рекомендується писати скрипти для автоматизації процесу. Це відноситься до всіх груп тестів регресії.

Відбір тестів для фінального регресивного тестування здійснюється за такими принципами:

1) В першу чергу відбирають тести забраковані два і більше разів. У тому числі й ті, які виявляли помилки, що вимагають доопрацювання.

У другу чергу відбираються тести забраковані один раз, і успішно пройдені повторно.

Далі відбираються всі тести, які були пройдені успішно, але проводилися тільки один раз.

Потім проводяться всі інші тести, в залежності від поставленого завдання.

Хід виконання роботи

Розглянемо приклад регресивного тестування. Отримавши звіт про помилку, програміст аналізує вихідний код, знаходить помилку, виправляє її і модульно або інтеграційно тестує результат. У свою чергу тестувальник, перевіряючи внесені програмістом зміни, повинен:

1. Для перевірки і затвердження виправлення помилки необхідно виконати зазначений в звіті тест, за допомогою якого була знайдена помилка.

2. Спробувати відтворити помилку яким-небудь іншим способом.

3. Протестувати наслідки виправлень. Можливо, що внесені виправлення привнесли помилку (наведену помилку) в код, який до цього справно працював. Наприклад, при тестуванні класу TCommandQueue запускаємо тести

```
// Тест перевіряє, чи створюється об'єкт типу
// TCommand і додається він в кінець черги.
private void TCommandQueueTest1 ()
```

```
// Тест перевіряє додавання команд в чергу на
// зазначену позицію. Також перевіряється
// правильність видалення команд з черги.
private void TCommandQueueTest2 ()
```

4. При цьому перший тест виконується успішно, а другий ні, тобто команда додається в кінець черги команд успішно, а на зазначену позицію - ні. Розробник аналізує код, який реалізує функціональність, що тестується:

```
if ((Position <-1) &&
     (Position <= this .Items.Count))
   {
      this .Items.Insert (Position, Command);
   }
else
{
   if (Position == - 1)
   {
      this .Items.Add (Command);
   }
} ...
```

5. Аналіз показує, що помилка полягає в використанні невірного знака порівняння в першому рядку фрагмента. Далі програміст виправляє помилку, наприклад таким чином:

```
if ((Position> = - 1) &&
     (Position <= this .Items.Count))
{
   this.Items.Insert (Position, Command);
} else
{
   if (Position == - 1)
   {
      this .Items.Add (Command);
   }
} ...
```

6. Для перевірки скоригованого коду можна пропустити тільки тест TCommandQueueTest2. Можна переконатися, що тест

TCommandQueueTest2 буде виконуватися успішно. Однак однієї цієї перевірки недостатньо. Якщо ми повторимо пропуск двох тестів, то при запуску першого тесту, TCommandQueueTest1, буде виявлений новий дефект. Повторний аналіз коду показує, що гілка else не виконується. Таким чином, виправлення в одному місці призвело до помилки в іншому, що демонструє необхідність проведення повного перетестування. Однак повторне перетестування вимагає значних зусиль і часу. Виникає запитання - відібрати скорочений набір тестів з вихідного набору (може бути, поповнивши його поруч додаткових - знову розроблених - тестів), якого, проте, буде досить для вичерпної перевірки функціональності відповідно до обраного критерію. Організація повторного тестування в умовах скорочення ресурсів, необхідних для забезпечення заданого рівня якості продукту, забезпечується регресивний тестуванням

Завдання на виконання

1) Отримайте у викладача завдання і створіть за ним програмний додаток для тестування.
2) Здайте програмний додаток викладачу.
3) Отримайте у викладача програмний додаток, розроблений іншим студентом.
4) Проведіть тестування отриманого програмного додатку. Звіт з тестування здайте викладачу.
5) Отримайте у викладача звіт з тестування додатку, попередньо розробленого вами. Виправте знайдені іншим студентом помилки. Здайте програмний додаток викладачу.
6) Проводьте регресивне тестування до тих пір, доки в програмі будуть віднаходитись помилки.
7) Опишіть проведений процес регресивного тестування в звіті.

Контрольні запитання

1) В чому полягають особливості регресивного тестування?
2) Які тести потрібно виконати над самим вікном програми?
3) Яким чином перевіряються поля вводу?
4) Які особливості у тестуванні радіоперемикача?
5) Як тестується перемикач «прапорець»?
6) Яким чином тестуються кнопки?

Завдання 2.5. Автоматизоване тестування програми

Мета роботи: оволодіти навичками автоматизованого тестування програмного забезпечення

Основні теоретичні відомості

Автоматизоване тестування передбачає використання спеціального програмного забезпечення (крім тестованого) для контролю виконання тестів і порівняння очікуваного і фактичного результату роботи програми. Цей тип тестування допомагає автоматизувати часто повторювані, але необхідні для максимізації тестового покриття завдання.

Деякі завдання тестування, такі як низькорівневе регресивне тестування, можуть бути трудомісткими і вимагають багато часу, якщо виконувати їх вручну. Крім того, мануальне тестування може недостатньо ефективно знаходити деякі класи помилок. У таких випадках автоматизація може допомогти заощадити час і зусилля проектної команди.

Кращими з точки зору автоматизації є:

- тестові набор, виконання яких проводиться регулярно, наприклад, попереднє або регресивне тестування);
- тестування допустимих конфігурацій апаратного забезпечення і операційних систем;
- певна вибірка наборів для тестування користувацького інтерфейсу;
- навантажувальне і стресове тестування;
- тестові набори, які використовують введення великих масивів даних;
- тестові набори, в яких автоматизована верифікація отриманих даних є кращою по відношенню до ручної.

Після створення автоматизованих тестів, їх можна в будь-який момент запустити знову, причому запускаються і виконуються вони швидко і точно. Таким чином, якщо є необхідність частого повторного прогону тестів, значення автоматизації для спрощення супроводу проекту і зниження його вартості важко переоцінити. Адже навіть мінімальні зміни коду можуть стати причиною появи нових помилок.

Існує кілька основних видів автоматизованого тестування:

1) автоматизація тестування коду– тестування на рівні програмних модулів, класів і бібліотек;

2) автоматизація тестування графічного інтерфейсу користувача – спеціальна програма, яка дозволяє генерувати дії користувача (натискання клавіш, кліки мишкою) і відслідковувати реакцію програми на ці дії – чи відповідає вона специфікації;

3) автоматизація тестування API – тестування програмного інтерфейсу програми. Тестуються інтерфейси, призначені для взаємодії, наприклад, з іншими програмами або з користувачем.

Основними стадіями процесу автоматизації тестування є:

- прийняття рішення про автоматизоване тестування: оцінюються потенційні можливості і економічний ефект;

- вибір засобу автоматизованого тестування: попередньо визначається основний набір вимог до даного засобу;

- попереднє планування: визначаються і при необхідності модифікуються цілі, стратегії, а також види тестів, придатні для автоматизації, перевіряється сумісність засобів автоматизації та тестової програми, а також тестового середовища;

- планування: визначаються стандарти розробки тестових скриптів, посібників, вимоги до апаратного та програмного забезпечення, мережевого оточення, набори тестових даних. Складається попередній графік тестування, визначаються методи контролю тестових конфігурацій і оточення, а також система моніторингу дефектів системи;

- розробка тестових скриптів: попередньо визначаються методи тестування, умови тестування і проводиться оцінка кількості необхідних тестів;

- виконання тестових скриптів і документування помилок;

- звіти за результатами тестування.

Серед засобів автоматизованого тестування слід виділити такі продукти як Rational Robot, Mercury Quick Test Pro, WinRunner, Computerware TestPartner, NeoLoad і ін.

Коли, що і як автоматизувати і чи автоматизувати взагалі – дуже важливі питання, відповіді на які повинна дати команда розробки. Вибір правильних елементів програми для автоматизації в великій мірі буде визначати успіх автоматизації тестування в принципі.

Потрібно уникати автоматизації тестування ділянок коду, які можуть часто змінюватися.

Як ручне, так і автоматизоване тестування можуть використовуватися на різних рівнях тестування, а також бути частиною інших типів і видів тестування.

Автоматизація зберігає час, сили і гроші. Одного разу складений автоматизований тест можна запускати знову і знову, докладаючи мінімум зусиль. Однак автоматизоване тестування не має сенсу для маленьких і короткострокових проектів, оскільки початкові витрати занадто високі.

Вручну можна протестувати практично будь-який додаток, в той час як автоматизувати варто тільки стабільні системи. Автоматизоване тестування використовується головним чином для регресії. Крім того, деякі види тестування, наприклад, дослідницьке тестування, можуть бути виконані тільки вручну.

Ручне тестування набагато більш гнучке, ніж автоматичне. Використовуючи автоматизовані засоби, важко змінити значення в програмі, після початку тестування. При ручному тестуванні, ви можете швидко перевірити результати, і це дозволить побачити, який функціонал працює найкраще.

Таким чином, на реальних проектах часто використовується комбінацію ручного і автоматизованого тестування, причому рівень автоматизації буде залежати як від типу проекту, так і від особливостей постановки виробничих процесів в компанії.

Хід виконання роботи

Розглянемо автоматизоване тестування на прикладі тестування веб-сайту за допомогою засобів Apache Jmeter. Для перевірки було обрано сайт http://dynamo.kiev.ua/.

JMeter є дуже потужним інструментом навантажувального тестування з можливістю створення великої кількості запитів одночасно завдяки паралельній роботі на декількох комп'ютерах.

Запустивши програму, ліворуч бачимо 2 пункти - Test Plan і WorkBench. Додамо Thread group в Test plan, для цього потрібно натиснути правою клавішею на Test plan - Add - Threads (Users) - Thread group.

Далі вже всередині Thread group створимо елементи налаштувань тесту, натискаємо правою клавішею на Thread group - Add - Config Element і вибираємо по черзі 4 пункти:

HTTP Authorization Manager. Його використовуємо, коли потрібно проводити тест з авторизацією користувача. Натискаємо Add і заповнюємо Base URL ресурсу, у User і Password вказуємо логін і пароль облікового запису.

HTTP Cookie Manager автоматично буде зберігати cookie.

HTTP Header Manager до кожного записаному пакету додасть інформацію про заголовку, для перегляду якої потрібно буде «розкрити» рядок (дерево на скріншоті буде створено нами трохи пізніше). Тема можна редагувати.

У налаштуваннях HTTP Proxy Server вказуємо будь-яку унікальну адресу порту (наприклад 8089) і знімаємо позначку Capture HTTP Headers.

В будь-якому браузері налаштовуємо проксі-сервер (в даному випадку обрано браузер Mozilla Firefox). У налаштуваннях мережі вказуємо використання проксі і вписуємо HTTP Proxy - localhost, порт - використовуваний нами 8089. Видаляємо виключення для localhost.

В Jmeter в HTTP Proxy Server натискаємо кнопку Start. Тепер в браузері завантажуємо всі інтерфейси, які потребують проведення навантажувального тестування. В даному випадку http://dynamo.kiev.ua/.

Після цього в Jmeter натискаємо Stop в HTTP Proxy Server.

Далі очистимо проект від непотрібної інформації. Деякі пакети посилаються до сторонніх сервісів, на які нам не потрібно створювати навантаження. Їх потрібно видалити. Для цього потрібно перебрати їх всі і подивитися, яке зазначене має Server Name or IP. Якщо ім'я сервера не має відношення до тестованого ресурсу - ми не створимо на нього додаткового навантаження. Такі url видаляємо.

Додамо в Thread Group по черзі елементи перегляду. Натискаємо правою клавішею на Thread Group - Add - Listener - Graph Results, Aggregate Report, View Result in Table і View Result Tree.

Остання настройка перед запуском - вказати кількість користувачів (Number of Threads) і кількість ітерацій (Loop Count), натиснувши на Thread Group і встановивши відповідні параметри.

Запускаємо тест, натисканням зеленого трикутника на верхній панелі і відкриваємо будь-який з елементів перегляду - перед нами статистика онлайн. Оскільки ми встановили Loop Count - Forever, натиснемо Stop після тестування. Graph Results відображає результат у вигляді графіків. Значення надані в мілісекундах.

Data - час відгуку кожної окремої одиниці даних тобто кожного перевіреного url.

Average - усереднений час відгуку, об'єктивний графік зміни навантаження.

Median - значення медіани (використовується в статистиці).

Deviation - похибка, стандартне відхилення.

Throughput - пропускна здатність виконуваних запитів.

З отриманих графіків можна зробити висновок, що час відгуку значно збільшується, чим вище пропускна здатність запитів (менше мілісекунд), тим більше часу потрібно серверу для обробки.

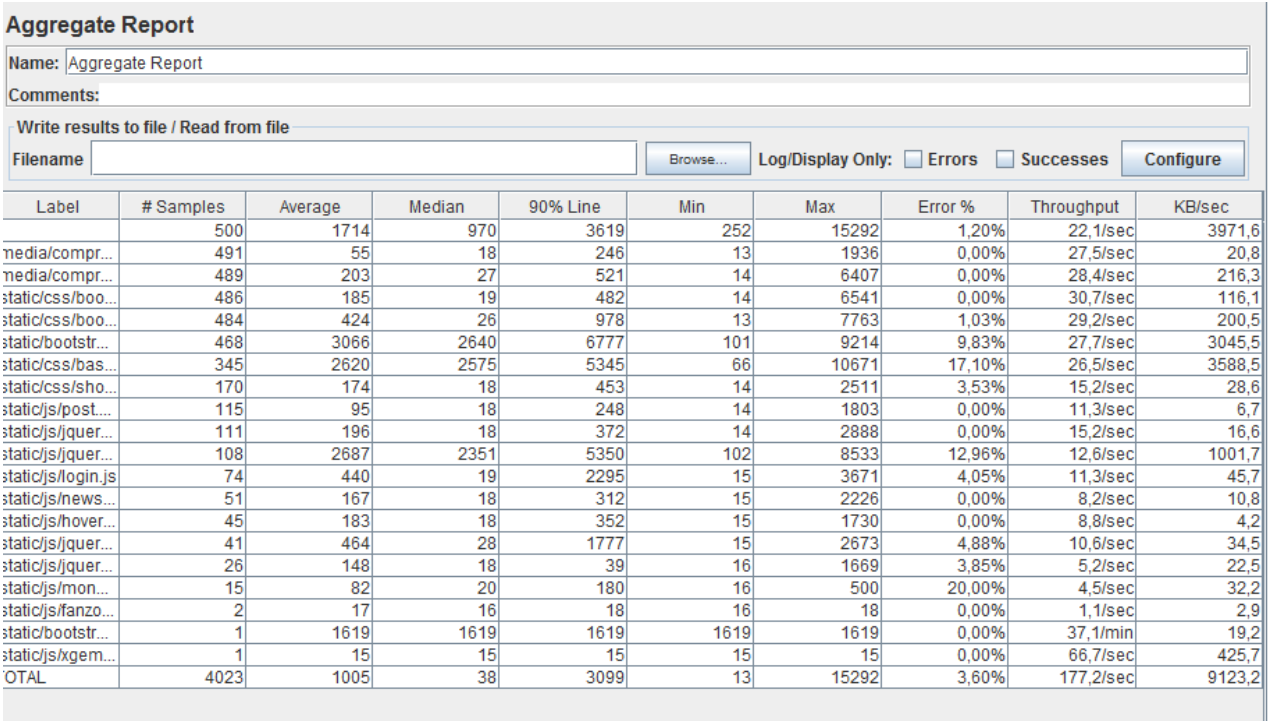
Агрегатний звіт відображає статистику по кожному індивідуальному url окремо (рис. 2).

У стовпці Average бачимо середній час відгуку, логічно припустити, що чим воно більше - тим більше навантаження на даний url.

Результат у вигляді таблиці відображений на рис. 3. Було знайдено помилки (червоний трикутник із знаком оклику). Чому саме вони виникли, можна перевірити в звіті у вигляді дерева.

Завдання на виконання

1) Отримайте у викладача програмний додаток для тестування.
2) Проведіть автоматизоване тестування отриманого програмного додатку, виконавши дії, описані в ході виконання даної роботи.
3) Результати перевірки занесіть у звіт з практики.



Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
	500	1714	970	3619	252	15292	1,20%	22,1/sec	3971,6
media/compr...	491	55	18	246	13	1936	0,00%	27,5/sec	20,8
media/compr...	489	203	27	521	14	6407	0,00%	28,4/sec	216,3
static/css/boo...	486	185	19	482	14	6541	0,00%	30,7/sec	116,1
static/css/boo...	484	424	26	978	13	7763	1,03%	29,2/sec	200,5
static/bootstr...	468	3066	2640	6777	101	9214	9,83%	27,7/sec	3045,5
static/css/bas...	345	2620	2575	5345	66	10671	17,10%	26,5/sec	3588,5
static/css/sho...	170	174	18	453	14	2511	3,53%	15,2/sec	28,6
static/js/post....	115	95	18	248	14	1803	0,00%	11,3/sec	6,7
static/js/jquer...	111	196	18	372	14	2888	0,00%	15,2/sec	16,6
static/js/jquer...	108	2687	2351	5350	102	8533	12,96%	12,6/sec	1001,7
static/js/login.js	74	440	19	2295	15	3671	4,05%	11,3/sec	45,7
static/js/news...	51	167	18	312	15	2226	0,00%	8,2/sec	10,8
static/js/hover...	45	183	18	352	15	1730	0,00%	8,8/sec	4,2
static/js/jquer...	41	464	28	1777	15	2673	4,88%	10,6/sec	34,5
static/js/jquer...	26	148	18	39	16	1669	3,85%	5,2/sec	22,5
static/js/mon...	15	82	20	180	16	500	20,00%	4,5/sec	32,2
static/js/fanzo...	2	17	16	18	16	18	0,00%	1,1/sec	2,9
static/bootstr...	1	1619	1619	1619	1619	1619	0,00%	37,1/min	19,2
static/js/xgem...	1	15	15	15	15	15	0,00%	66,7/sec	425,7
OTAL	4023	1005	38	3099	13	15292	3,60%	177,2/sec	9123,2

Рис. 2 – Агрегатний звіт при навантаженні у 500 віртуальних користувачів

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency
3622	14:04:18.825	Thread Group 1-2...	/static/css/base.css	6430		166851	285
3623	14:04:22.473	Thread Group 1-2...	/static/css/base.css	2716		166851	591
3624	14:04:25.168	Thread Group 1-85	/static/js/jquery.co...	17		4560	17
3625	14:04:25.171	Thread Group 1-26	/static/js/hover.js	15		489	15
3626	14:04:22.408	Thread Group 1-4...	/static/css/base.css	2777		166851	33
3627	14:04:24.986	Thread Group 1-23	/static/js/jquery-1....	347		1345	32
3628	14:04:19.630	Thread Group 1-1...	/static/js/jquery-1....	5694		1345	2164
3629	14:04:22.371	Thread Group 1-4...	/static/css/base.css	2965		1345	2599
3630	14:04:22.369	Thread Group 1-4...	/static/css/base.css	2972		1345	19
3631	14:04:25.256	Thread Group 1-3...	/static/css/shop-in...	17		1926	17
3632	14:04:22.591	Thread Group 1-2...	/static/css/base.css	2755		1345	52
3633	14:04:23.073	Thread Group 1-2...	/static/js/jquery-1....	2214		93108	17
3634	14:04:22.323	Thread Group 1-4...	/static/bootstrap/c...	3028		1345	909
3635	14:04:18.367	Thread Group 1-66	/static/bootstrap/c...	6977		124508	1473
3636	14:04:16.635	Thread Group 1-1...	/static/bootstrap/c...	8566		124508	33
3637	14:04:18.952	Thread Group 1-2...	/static/bootstrap/c...	6441		1345	81
3638	14:04:18.571	Thread Group 1-2...	/static/bootstrap/c...	6817		1345	21
3639	14:04:22.397	Thread Group 1-2...	/static/css/base.css	2979		1345	35
3640	14:04:22.512	Thread Group 1-3...	/static/css/base.css	2693		166851	17
3641	14:04:19.491	Thread Group 1-3...	/static/css/bootstr...	5877		1345	2305
3642	14:04:22.326	Thread Group 1-67	/static/css/base.css	2980		166851	2567
3643	14:04:25.287	Thread Group 1-13	/static/css/shop-in...	74		2264	0
3644	14:04:19.069	Thread Group 1-35	/static/css/base.css	6296		1345	20
3645	14:04:19.230	Thread Group 1-2...	/static/css/base.css	6204		1345	42
3646	14:04:23.114	Thread Group 1-1...	/static/js/jquery.sli...	2183		3417	1793
3647	14:04:18.917	Thread Group 1-2...	/static/bootstrap/c...	6506		124508	18

Scroll automatically? Child samples? No of Samples 4023 Latest Sample 17 Average 1005 Deviation 1775

Рис. 3 – Перегляд результату у вигляді таблиці при навантаженні у 500 віртуальних користувачів

Контрольні запитання

1) В чому полягає автоматизоване тестування?
2) Які існують інструменти для автоматизованого тестування?
3) В чому переваги і недоліки автоматизованого тестування?
4) В чому переваги і недоліки ручного тестування?
5) Які стадії включає в себе автоматизоване тестування?

СПИСОК ЛІТЕРАТУРИ

1. IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation.
2. ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.
3. ДСТУ 2844-94. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення. — Введ. 1.08.1995. — К.: Держстандарт України, 1995. — 57 с.
4. ДСТУ 2850-94. Програмні засоби ЕОМ. Показники і методи оцінювання якості. — К.: Держстандарт України, 1994. – 20 с.
5. Майерс Г., Баджет Т., Сандлер К. Искусство тестирования программ. – М.: Диалектика, 2016. - 272 с.
6. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. - С.-Пб.: Питер, 2004. - 318 с.
7. Криспин Л., Грегори Дж. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. – М.: Диалектика-Вильямс, 2010. - 464 с.
8. Блэк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. – М.: Лори, 2006. - 576 с.
9. Бек К. Экстремальное программирование: разработка через тестирование. – С.-Пб.: Питер, 2003. - 224 с.
10. Макгрегор Д., Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. – М.: Диасофт, 2002. – 417 с.
11. Плаксин М.А. Тестирование и отладка программ – для профессионалов будущих и настоящих. - М.: Бином, 2004. - 167 с.
12. Тамре Л. Введение в тестирование программного обеспечения. – М.: Вильямс, 2003. – 368 с.
13. Котляров В.П., Коликова Т.В. Основы тестирования программного обеспечения. – М.: Бином, 2006. – 288 с.
14. Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. – М.: Лори, 2003. – 592 с.

ЗМІСТ

Навчальне видання

Практика з тестування ПЗ
Методичні рекомендації до виконання практики
для студентів спеціальності 123
«Комп'ютерна інженерія»
спеціалізації «Системне програмування»

Укладачі: СТАНКО Світлана Михайлівна
КАШКЕВИЧ Іван-Фарход Фуркатович

Редактор
Технічний редактор
Комп'ютерна верстка

Підп. до друку _____. Формат 60x84/16. Папір офс.
Офс. друк. Ум. друк. арк.42. Обл.-вид. арк. 2.
Тираж 100 пр. Замовлення № _____.

Видавець і виготовлювач
Національний авіаційний університет
03680. Київ – 58, проспект Космонавта Комарова, 1

Свідоцтво про внесення до Державного реєстру ДК № 977 від 05.07.2002