УДК 004.415.533

О.А. РЕМІННИЙ
Вінницький національний технічний університет

СТВОРЕННЯ ФРЕЙМВОРКУ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ГРАФІЧНИХ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

В даній роботі описано процес розробки базового фреймворку автоматизованого тестування користувацьких інтерфейсів. Визначено основні компоненти фреймворку, кращі практики.

В результаті дослідження було створено набір базових характеристик, які прискорюють процес розробки рішень автоматизованого тестування.

Ключові слова:графічний користувацький інтерфейс, автоматизоване тестування, фреймворк

O.A. REMINNYI
Vinnitsa national technical university

CREATING GUI AUTOMATION TESTING FRAMEWORK

Abstract – The goal of this article is to share information about experience related to implementing corporate level automation framework that simplified automation solution coding and code maintaining processes by using such principles as layered architecture, automation patterns, fluent interface, and aspects. Also given approaches have shortened the project bootstrapping time by providing basic mechanisms for writing reports, reading configuration, providing validation etc.

Keywords: GUI, automated testing, framework

Вступ

Термін автоматизації широко використовується в наш час. Його часто пов'язують з тестуванням програмного забезпечення, модульним тестуванням, автоматизацією процесів. Однак досить часто уявлення про його значення зовсім різне у різних індивідів. Давайте почнемо із з'ясування різниці між модульним тестуванням та функціональним тестуванням.

Технологія Unit-тестування є однією з провідних та загальновизнаних технологій у сфері автоматизації тестування програмного забезпечення [1, 2]. Ідея unit-тестів полягає в тому, що для кожного класу або для невеликої їх сукупності, у випадку, коли один клас не є функціонально автономним, пишеться тестова програма, яка перевіряє функціональність модуля (класу або групи класів) на працездатність. Тобто тестована програма розглядається не як якийсь цілісний програмний продукт, а як набір самостійних модулів зі своєю функціональністю, яка тестується не в контексті всієї програми, а як самостійний додаток [1].

Функціональне тестування [2] - це тестування програмного забезпечення (ПЗ) в цілях перевірки реалізованості функціональних вимог, тобто здатності ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить ПЗ, які завдання воно вирішує. Зазвичай, функціональне тестування більш сильно абстраговане від тестованої системи. Так само як і тестери нічого не знають про вихідний код, модулі та архітектуру програми, функціональне тестування зосереджується на функціональності зовнішніх інтерфейсів системи.

У процесі функціонального тестування інструмент автоматизованого тесту дозволяє емулювати дії тестера (клацання кнопки миші, натискання клавіш і т.д.) Інструмент автоматизованого тестування може бути частиною фреймворку автоматизації, але він не обов'язково має визначеним в рамках фреймворку. Фреймворк абстрагується від будь-яких засобів автоматизації за замовчуванням.

Фреймворк для автоматизованого тестування

Розглянемо одну з компаній, для якої у мене був шанс розробляти фреймворк автоматизованого тестування. Десять тисяч працівників, більше 50 найменувань програмних продуктів у продажу, розподілена мережа розробницьких офісів в усьому світі. Основна проблема, пов'язана з автоматизацією продуктів - відсутність однакового систематичного підходу, придатного для повторного використання на більшості проектів. Таким чином нашою задачею було створити такий підхід і базові принципи. У той же час ми повинні були врахувати той факт, що різні команди використовують різні інструменти автоматизованого тестування.

Технологічний стек компанії був Microsoft орієнтований, тому фреймворк розроблявся для мови програмування тестів C# .NET технології. Однак принципи, описані в наступному розділі є абсолютно незалежними від мови програмування та конкретних технологій.

Отже, фреймворк автоматизованого тестування складається з набору припущень, концепцій та інструментів, які забезпечують підтримку автоматизованого тестування програмного забезпечення. Він має наступні функції:

- Визначення методології написання сценаріїв для автоматизації ПЗ;
- Забезпечення механізму для підключення до тестованої системи (System Under Test - SUT);
- Виконання тестів і представлення результатів;
- Зниження часових затрат на початкових стадіях автоматизації продукту;

- Створення єдиного стандарту.

Принципи фреймворку

Основною ціллю розробки фреймворку автоматизованого тестування (AF), а потім його використання для написання автоматизованих тестів є збільшення повторного використання коду та зменшення часу на підтримку цього коду. Давайте припустимо, що у нас є складні програмні додатки з багатим користувача інтерфейсом користувача і безліччю елементів управління. Все це розташовано на двох формах. Той факт, що програмний додаток є складним, може означати, що він може мати десятки тест кейсів для покриття свого функціоналу. Скажімо, є 50 тестів. Всі ці тести використовують ті ж два екрани. Так як ми можемо спростити підтримку таких автоматизованих тестів?

Багаторівнева архітектурна модель

Ідея розділення коду системи програмного забезпечення архітектурно на окремі шари є досить розповсюджена. Перший рівень інкапсулює логіку користувацького інтерфейсу, другий рівень забезпечує бізнес-логіку, а третій відповідає за зберігання даних. Використання цієї парадигми дозволяє знизити вартість обслуговування коду додатків, так як компоненти всередині кожного рівня можуть зазнавати змін без впливу на код інших рівнів. Той самий підхід може бути застосований до системи автоматизованого тестування.

Тестовий код може бути розділений на три шари: шар доступу до користувацького інтерфейсу через інструмент автоматизованого тестування, шар функціональної логіки і шар тестового скрипта. Кожен шар має певну відповідальність із загальною метою зменшення витрат на підтримку коду тесту і полегшення створення нових тестів [3, 4].

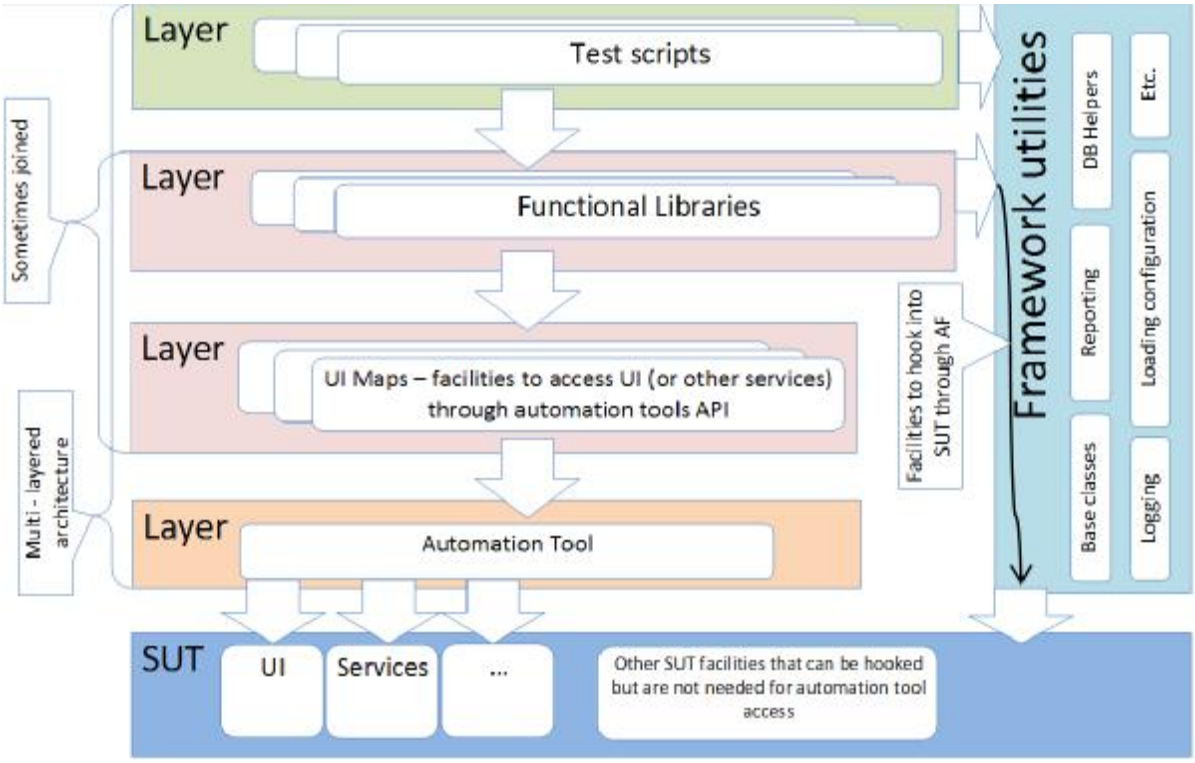

Рис. 1. Архітектурні архетип - багатошарова архітектура тестової системи

Патерн Об'єкт сторінки (PageObject)

Ідея створення окремих логічних шарів для тестових сценаріїв, функціональної логіки та шар доступу до інтерфейсу для кожного окремого тесту дає нам змогу змінювати окремі шари в майбутньому, у разі зміни функціональної логіки в кожному конкретному тесті.

Давайте припустимо, що наш додаток є веб-пошта Gmail і однією з двох форм додатку є екран входу в систему. Функціонал логіну використовується у всіх 50-ти тестах (для того, щоб дістатися до другої форми програмного додатку, в першу чергу ми повинні увійти (авторизуватись) в додаток).

Припустимо, що щось змінилося в інтерфейсі цієї логін форми. Наприклад для нашого конкретного випадку, Gmail веб-пошта тепер вимагає код підтвердження CAPTCHA при кожному логіні.

Це означає, що кожен тест з 50-ти тепер має бути оновлений відповідно до нового робочого процесу входу в систему. Але загалом було б логічно, що оновлення потребує лише одна перевикористовувана частину коду. Саме тут ми можемо оцінити користь патернів «об'єкт сторінки» та «функціональний метод» [5]. Якби ми мали окремий PageObject для екрану авторизації з одним методом Login(), який приймає параметри ім’я користувача і пароль, нам потрібно було б оновити лише код цього функціонального методу, щоб охопити всі випадки, пов'язані з цією зміною.

Рис. 2. Форма логіну до пошти Google

Рис. 3. Форма логіну до пошти Google з CAPTCHA підтвердженням

Текучий інтерфейс (Fluent Interface)

Ідея текучого інтерфейсу полягає в тому [6], що всі функціональні методи певного об'єкта сторінки повертають інший інстанційований об'єкт сторінки (відповідно до нового контексту, в якому знаходиться SUT після виконання попереднього функціонального методу). Наприклад, метод входу в наш додаток повертає інстанційований об'єкт домашньої сторінки Gmal сервісу (поштова скринька). Цей підхід дає можливість описувати кроки функціонального тесту один за іншим, використовуючи ланцюжки викликів (method chaining) [7]. У результаті такого підходу функціональні методи мають змогу підказувати розробнику, які наступні можливі функціональні методи можна викликати через IntelliSense засоби середовищ розробки коду (рис. 4, 5).

```
[BusinessMethod("Allows to login into application")]
public HomePageObject Login(string username, string password)
{
    if (!string.IsNullOrEmpty(username))
    {
        loginPageUiMap.TbUserName.SendKeys(username);
    }

    if (!string.IsNullOrEmpty(password))
    {
        loginPageUiMap.TbPassword.SendKeys(password);
    }

    loginPageUiMap.BtnSignIn.Click();
    return HomePageObject.Instance;
}
```

Рис 4. Додавання можливостей Fluent Interface для бізнес методу, описаного на мові програмування C#

```
LoginPageObject.Instance.Login("login", "password");
NavigationPageObject.Instance.Compose();
NewMailPageObject.Instance.FillInEmail(
    new NewMailModel
    {
        To = "to@gmail.com",
        ToCC = "tocc@gmail.com",
        ToBCC = "tobcc@gmail.com",
        Subject = "Test Subject 1",
        Body = "Test Email text"
    });
NewMailPageObject.Instance.InsertPhotos();
AddPhotosPageObject.Instance
    .AddWebLinkPhoto("http://imagelink.jpg");

NewMailPageObject.Instance.AddWebLink();

AddWebLinkPageObject.Instance
    .AddWebLinkUrl("http://sitelink.com");
```

а

```
LoginPageObject.Instance
    .Login("login", "password")
    .NavigationProvider
    .Compose()
    .FillInEmail(
    new NewMailModel
    {
        To = "to@gmail.com",
        ToCC = "tocc@gmail.com",
        ToBCC = "tobcc@gmail.com",
        Subject = "Test Subject 1",
        Body = "Test Email text"
    })
    .InsertPhotos()
    .AddWebLinkPhoto("http://imagelink.jpg")
    .AddWebLink()
    .
```


б

Рис. 5. Використання засобів IntelliSense та Fluent Interface в середовищі Visual Studio 2010 для скеровування дій розробника: а - без використання fluent interface, б - з використанням fluent interface

Аспектно-орієнтоване програмування

Аспекти [8, 9] дають нам додаткові функціональні можливості при роботі над такими задачами як логування коду, створення звітів і т.п. Наприклад, нам потрібно огорнути конкретний метод блоками перехвату помилок (try/catch), або внести запис в систему логування/звітування про вхід / вихід з конкретного методу. В рамках розробки рішення автоматизованих тестів, використання аспектів дозволяє поліпшити якість коду, а також робить його більш читабельним і зрозумілим. Для наших потреб ми використовували PostSharp фреймворк [9]. Цей продукт має безкоштовну Starter Edition ліцензію, яка дозволяє комерційне використання продукту.

```
[BusinessMethod("Allows to login into application")]
public HomePageObject Login(string username, string password)
{
    DateTime methodStart = DateTime.Now;
    AutoEngine.Instance.Report.DebugFormat(
        "Started Login method exec. username: {0}, pwd: {1}",
        username,
        password);

    if (!string.IsNullOrEmpty(username))
    {
        loginPageUiMap.TbUserName.SendKeys(username);
    }

    if (!string.IsNullOrEmpty(password))
    {
        loginPageUiMap.TbPassword.SendKeys(password);
    }

    loginPageUiMap.BtnSignIn.Click();

    double ellapsedMsec =
        (DateTime.Now - methodStart).TotalMilliseconds;

    AutoEngine.Instance.Report.DebugFormat(
        "Finished Login method exec. Ellapsed: {0} msec",
        ellapsedMsec);

    return HomePageObject.Instance;
}
```

а

```
public HomePageObject Login(string username, string password)
{
    if (!string.IsNullOrEmpty(username))
    {
        loginPageUiMap.TbUserName.SendKeys(username);
    }

    if (!string.IsNullOrEmpty(password))
    {
        loginPageUiMap.TbPassword.SendKeys(password);
    }

    return HomePageObject.Instance;
}
```

б

Рис. 6. Винесення логування часу виконання бізнес методу у PostSharp аспект:
а - логування в бізнес методі, б - винесення логіки логування в аспект

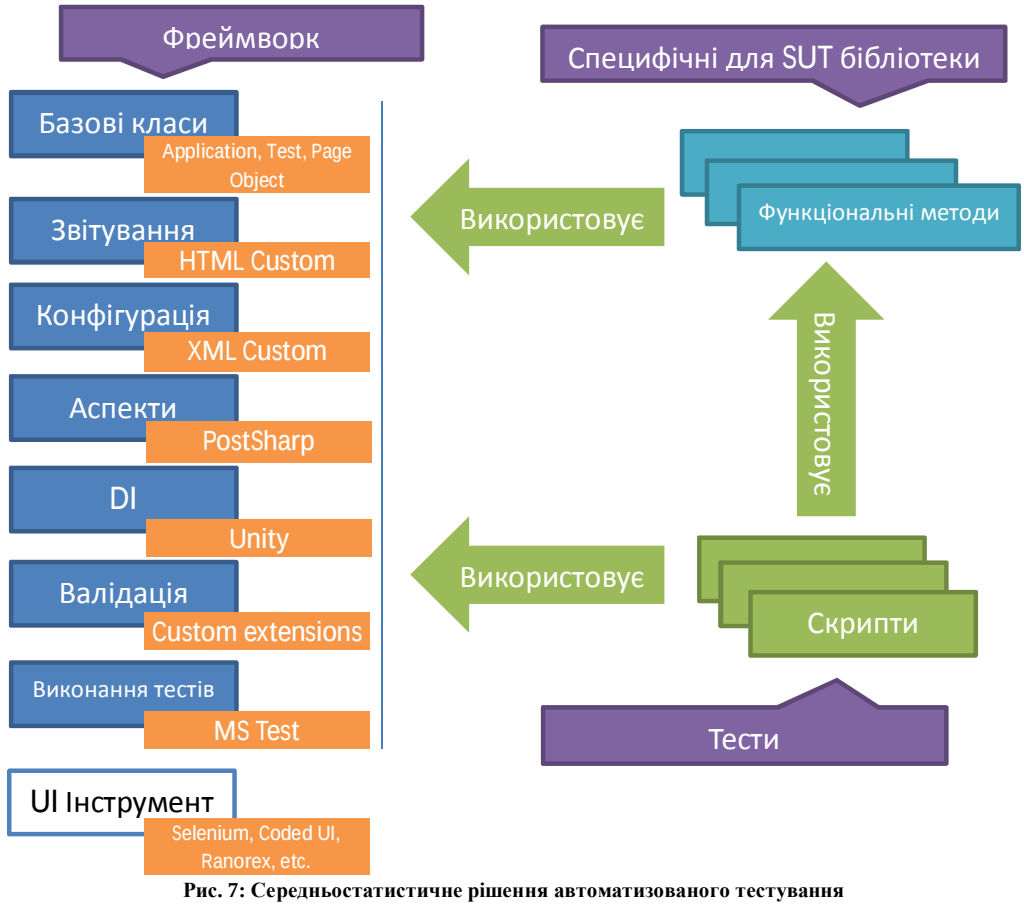

Рис. 7: Середньостатистичне рішення автоматизованого тестування

Також призначення фреймворку автоматизованого тестування - визначення корпоративного стандарту розробки коду автоматизованих тестів. Тому в рамках фреймворку було додатково розроблено

ряд аспектів, які використовуються для перевірки чи слідував розробник автоматизованого тесту рекомендаціям.

Ось що перевіряли ці аспекти:

- Найменування тесових сценаріїв (відповідно до RegEx шаблону);
- Перевірка наявності атрибуту BusinessMethod у публічних методах PageObject - ів;
- Перевірка типізації параметрів бізнес методів для забезпечити відсутність конкретного стану тестах (stateless tests).

Фреймворк зсередини

Фреймворк зсередини складається з набору базового функціоналу, який повторно використовують різні тестові сценарії і бізнес-бібліотеки. Схема наведена на рисунку 7.

Модулі всередині фреймворку, є незалежні і тому можуть бути пере використані для різних проектів автоматизації.

Базові класи

Фреймворк автоматизованого тестування надає набір базових класів. Їх призначення – спростити реалізацію бібліотеки PageObject-ів та тестових сценаріїв.

Абстрактні класи WorkflowItem, PageObject

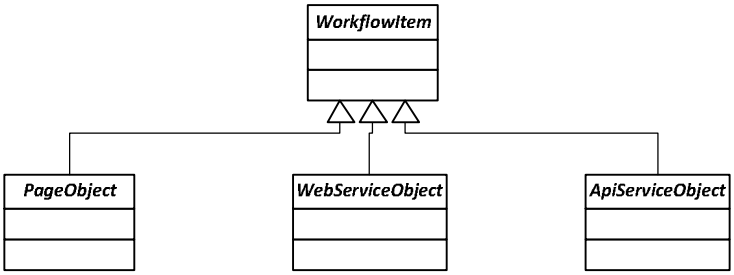

Рис. 8. Ієрархія PageObject елементів

IWorkflowItem інтерфейс

Цей інтерфейс є основним інтерфейсом для всіх WorkflowItem класів, які обговорюватимуться пізніше в цій статті. Він представляє собою базовий інтерфейс для аспекту WorkflowItemValidationAspect (який перевіряє всі типи, похідні від цього інтерфейсу).

ApplicationUnderTestBase & SecuredApplicationUnderTestBase

Ці базові класи служать в якості стартової точки для будь-якого тесту. Вони мають свою власний секцію конфігурації, яка визначає, як тестована система має бути запущена. (URL для браузера або абсолютний шлях до .exe файлу програми на локальному комп'ютері).

Конфігураційна інформація зчитується всередині Run методу класу ApplicationUnderTestBase. Крім того, метод Run має перевантажений метод, який приймає параметри запуску для додатків у випадку якщо тетс не має конфігураційної секції. Дані класи відповідають патерну «об’єкт запуску та закриття системи» (SUT Runner)[5].

FunctionalTestBase

Цей клас має використовуватись в якості базового класу для всіх функціональних тестів. Він надає наступні можливості:

- Завантажує конфігурацію для тесту перед початком його виконання;
- відкриває-закриває файл звіту для поточного тесту;
- Записує інформацію про результат тесту у звіт;
- Дає змогу перевірити слідування рекомендаціям імплементації тестових класів відповідно до аспекту FunctionalTestAspect.

AutoEngine

AutoEngine це статичний клас, який дозволяє розробнику автоматизації доступ до усіх основних функцій фреймворку, як наприклад:

- Завантаження додаткових конфігураційних файлів за допомогою ConfigurationLoader;
- Запис додаткової інформації в звіт або лог тесту.

Крім того, він використовується для ініціалізації функціональних тестів.

Аспекти

Як було описано, фреймворк надає набір аспектів для перевірки коду та для ін’єкції коду.

Валідаційні аспекти, які спрацьовують під час компіляції тестів:

- FunctionalTestAspect – перевірка слідування кращим практикам під час написання класів функціональних тестів;
- WorkflowItemValidationAspect - перевірка слідування кращим практикам під час написання бібліотек функціональних методів;

Аспекти ін’єкції коду:

- ParametersValidationAspect – перевіряє чи параметри, які передаються всередину бізнес-методів не мають порожніх (null) значень. У випадку потреби передати пусте значення в бізнес метод, параметр потрібно відмітити атрибутом [OptionalParam];
- TestExceptionAspect - записує інформацію про причину помилки у виконанні тесту у звіт, також робить скріншот екрану;
- TraceAspect - записує інформацію про вхід/вихід з кожного функціонального методу в звіт тесту.

Конфігурація

Створення конфігурації тестів є важливим кроком на шляху до створення серйозного рішення автоматизованого тестування. Конфігурація дозволяє змінювати налаштування тестованої системи, тест конкретні параметри бізнес методів без потреби перекомпіляції тестів. У нашому випадку вона є досить простою та гнучкою.

Конфігурація функціонального тесту

Важливо для тесту не використовувати жорстко закодовані значення конфігурації, але взяти їх з деякого зовнішнього реєстру. Фреймворк надає власний механізм для завантаження конфігурації з зовнішнього джерел.

Конфігураційний файл представляє собою XML-файл, який має таке ж ім'я, що й функціональний клас тесту. Так, тестовий клас з ім'ям Test12345 має мати відповідний конфігураційний файл Test12345.xml. Цей файл буде шукати і читати функціонал базового FunctionalTestBase класу.

Ось зразок файлу конфігурації, який придатний для читання фреймворком автоматизованого тестування (структури XML-файла з урахуванням регістру):

```
<WorkflowConfiguration>
  <section name="Login">
    <add key="login" value="nameLogin" />
    <add key="password" value="********" />
  </section>
  <section name="bootstraperInfo" reference="CommonSections/bootstraperInfo.xml"/>
  <section name="QuestionWithNote">
    <add key="Question">
      <MedQuestion>
        <Name>QuestionName ${RND:[9][99]}</Name>
        <IsRequired>true</IsRequired>
        <AnswerType>Other</AnswerType>
        <IncludeQuestionNote>true</IncludeQuestionNote>
      </MedQuestion >
    </add>
   </section>
</WorkflowConfiguration>
```

Рис. 8: Приклад конфігурації файлу тесту

Фреймворк має парсер для такого конфігураційного файлу, який дає наступні функції:

- Можливість отримати пару ключ-значення в коді тесту, звернувшись до властивості базового класу Configuration.
- Можливість виносити спільні секції в окремі перевикористовувані файли (приклад: секція bootstrapperInfo);
- Можливість завантажити складний об'єктів безпосередньо з XML-специфікації, використовуючи стандартні методи XML серіалізації:

```
MedQuestion question = Configuration.SafeGetModel<MedQuestion>("QuestionWithNote", "Question");
```

- Можливість завантажити унікальні значення з конфігурації при кожному запуску тесту за допомогою технік попередньої обробки значень в XML файлі (${RND: [9] [99]} після завантаження буде замінено на випадкове значення в діапазоні від 9 до 99.

Всередині тестового класу, успадкованого від FunctionalTestBase, значення пароля входу з логін секції можна отримати через успадковану властивість Configuration:

```
[TestMethod]
public void RunTest_000000()
{
    string password = Configuration["Login"]["password"];
}
```

Валідація

Валідаційні класи

Фреймворк автоматизованого тестування має класи для перевірки коректності стану тестованої системи. Відповідно є два статичних класи - Validate і CollectionValidate. Вони замінюють стандартні класи Microsoft Test фреймворку Assert і CollectionAssert.

Переваги власних валідаційних методів:

- Інформація про результат перевірки автоматично записується у звіт;

- результатом перевірки є логічне значення true/false, тому результати перевірки можуть бути використані в бізнес-логіці тесту;
- Класи валідації не обов'язково переривають виконання тесту (як то робить стандартний механізм валідації Assert) у випадку неуспішної перевірки, тому розробники можуть дозволити продовження виконання тесту після некритичних валідаційних помилок.

Валідаційні методи розширення (extension methods)

Extension Methods – можливість розширювати базового фунуціоналу класів мови програмування без зміни самих класів. В технології Microsoft .NET така можливість існує. Мігрувала вона в цю технологію з функціональних мов програмування.

Фреймворк автоматизованого тестування має набір валідаційних методів розширення для більшості базових типів .NET технології. Вони значно спрощують програмування перевірок в тестовому скрипті:

```
int a = 10;
// використання статистичних класів
Validate.AreEqual(a, 10);
Validate.IsTrue(a > 5 && a < 15);
// використання методів розширення
a.ShouldBe(10);
a.ShouldBeInRange(5, 15);
```

Висновок

У цій статті представлено процес розробки фреймворку автоматизованого тестування. Його використання на реальних проектах дозволило отримати значне зниження часу навчання нових членів команд автоматизованого тестування шляхом встановлення загальних стандартів і валідації рекомендацій. Також було досягнуто спрощення процесу кодування та підтримки коду тестів з використанням таких принципів, як багаторівнева архітектура, патерни автоматизованого тестування, текучий інтерфейс, аспектно-орієнтоване програмування. Скорочено час початкового старту проекту шляхом надання основних механізмів для написання звітів, читання конфігурації, валідаційних класів. Підходи були успішно впроваджені вже на трьох незалежних проектах автоматизованого тестування.

Література

1. Unit Testing [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/ wiki/Unit_testing
2. Functional Testing [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/ wiki/Functional_testing
3. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software/ Addison-Wesley Professional – 1994. - 416p.
4. Ryan Gerard, Amit Mathur, Meta-Framework: A New Pattern for Test Automation [Електронний ресурс]/ Режим доступу до файлу: http://www.associationforsoftwaretesting.org/?dl_name=Meta-Framework.pdf
5. Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code / Gerard Meszaros – 2007. – 833p.
6. Текучий інтерфейс [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/wiki/Fluent_interface
7. Ланцюжок методів [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/wiki/Method_chaining
8. Аспектно-орієнтоване програмування [Електронний ресурс] // Режим доступу до файлу: https://en.wikipedia.org/wiki/Aspect-oriented_programming
9. PostSharp – Фреймворк аспектно-орієнтованого програмування для технології .NET [Електронний ресурс] // Режим доступу до файлу: http://www.postsharp.net/

References

1. Unit Testing [Elektronnij resurs] // Rezhim dostupu do fajlu: http://en.wikipedia.org/ wiki/Unit_testing
2. Functional Testing [Elektronnij resurs] // Rezhim dostupu do fajlu: http://en.wikipedia.org/ wiki/Functional_testing
3. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software/ Addison-Wesley Professional – 1994. - 416p.
4. Ryan Gerard, Amit Mathur, Meta-Framework: A New Pattern for Test Automation [Elektronnij resurs]/ Rezhim dostupu do fajlu: http://www.associationforsoftwaretesting.org/?dl_name=Meta-Framework.pdf
5. Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code / Gerard Meszaros – 2007. – 833p.
6. 6. Tekuchij interfejs [Elektronnij resurs] // Rezhim dostupu do fajlu: http://en.wikipedia.org/wiki/Fluent_interface
7. 7. Lancjuzhok metodiv [Elektronnij resurs] // Rezhim dostupu do fajlu: http://en.wikipedia.org/wiki/Method_chaining
8. 8. Aspektno-orієntovane programuvannja [Elektronnij resurs] // Rezhim dostupu do fajlu: https://en.wikipedia.org/wiki/Aspect-oriented_programming
9. PostSharp – Frejmvork aspektno-orієntovanogo programuvannja dlja tehnologiї .NET [Elektronnij resurs] // Rezhim dostupu do fajlu: http://www.postsharp.net/

Рецензія/Peer review : 19.5.2013 р. Надрукована/Printed :29.9.2013 р.
Рецензент: д.т.н., проф. кафедри АІВТ ВНТУ Бісікало О.В.