УДК 004.415.532

О.А. РЕМІННИЙ
Вінницький національний технічний університет

ЕВОЛЮЦІЯ В ПІДХОДАХ ДО АВТОМАТИЗАЦІЇ КЛЮЧОВИМИ СЛОВАМИ

В даній роботі описано систему побудови тестових сценаріїв з набору функціональних методів, незалежну від утиліт автоматизованого тестування. Система дає змогу будувати сценарії людям, не знайомим з мовами програмування, та надає можливості перевірки скриптів на етапі побудови.

Ключові слова: Автоматизоване тестування, ключові слова, кейворди, користувацький інтерфейс

O.A. REMINNYI
Vinnytsianationaltechnicaluniversity

EVOLUTION OF KEYWORD DRIVEN AUTOMATION

Abstract –It's a widely known approach to build an automation solution based on keywords. Keywords are small reusable scripts. They are used mostly in order to give ability for manual testers to create automation tests without knowing of implementation languages and technologies.

But keyword automation has a set of significant disadvantages. Keyword implementation is usually bound to concrete automation tool. There is no implementation/build time validation for the scripts. The readability of the scripts is rather poor.

In this article a system described that helps to overcome these issues. It allows loading keyword libraries, implementation/build time validation, and fast scripting facilities though method chaining and method search filtering.

Keywords: Automatedtesting,keywords, userinterface

Вступ

Кейворди - це мініатюрні тестові скрипти. Наприклад, кейвордLogin може запускати сторінку входу в систему, перевіряти що вона відкрита, заповнювати поля логіну користувача і пароля потрібними значеннями, натискати кнопку «Вхід», перевіряти що користувач успішно ввійшов в систему. І все це - за допомогою одного окремого кроку. Для цього зазвичай створюється бібліотека ключових слів - кейвордів. Кейворд може оперувати декількома сторінками.

Приміром тест може виглядати так:

```
KwrdLib.getKeyword ("login"). Execute (newString [] {"user", "password"});
KwrdLib.getKeyword ("createNewUser"). Execute (newString [] {"user2", "password2"});
KwrdLib.getKeyword ("logout"). Execute (newString [] {});
KwrdLib.getKeyword ("login"). Execute (newString [] {"user2", "password2"});
KwrdLib.getKeyword ("logout"). Execute (newString [] {});
```

Через можливості сучасних мов програмування ці записи можна спростити ще більше і в результаті отримати щось на зразок наступного:

```
LOGIN.execute (newString [] {"user", "password"});
CREATE_NEW_USER.execute (newString [] {"user2", "password2"});
...
```

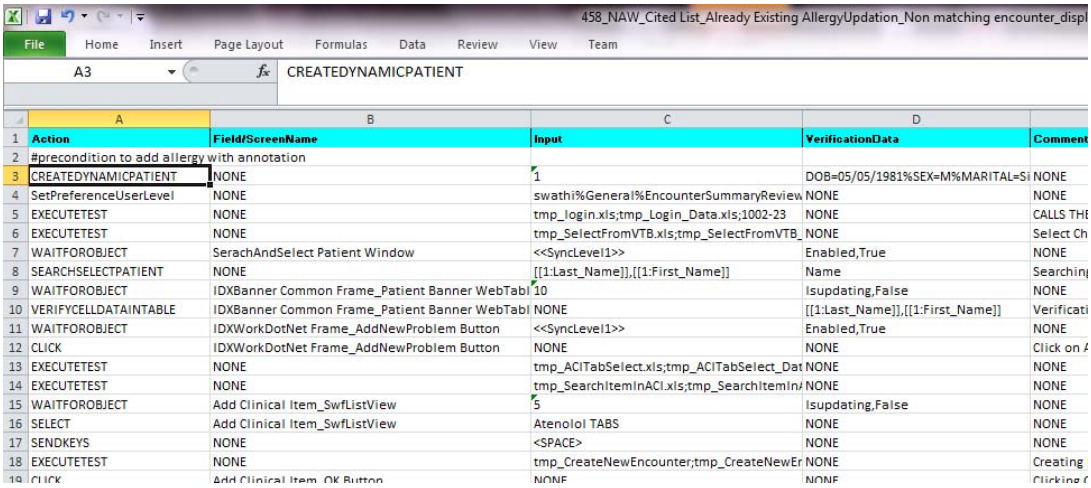
Відповідно, в класі KwrdLib буде статичний метод getKeyword який буде повертати екземпляр потрібного класу кейворда по імені.

```
publicstaticKeywordgetKeyword (StringkeywordName) {
    ...
    if (keywordName.equals ("login"))
returnnewLoginKeyword ();
        ...
}
```

А вже безпосередньо в класі кейворда буде описана маніпуляція з конкретними користувацькими формами.Питання визначення імплементації самих кейвордів ми залишимо за межами цієї статті.

В залежності від специфіки проекту автоматизованого тестування, системи взаємодії з користувацьким інтерфейсом автоматизованої системи, можуть бутиімплементовані різні підходи до використання ключових слів. Вже описаний механізм все-таки потребує певного компілятора скриптів. В деяких випадках опис може бути на рівні метапрограми – наприклад в певних текствих, xml чи Excel таблицях.

На рисунку 1 видно, як механізм організовується в Excel документі. Це реальний приклад працюючого скрипта, який виконується певним фреймворкомавтоматизованого тестування. Ключові слова в цьому випадку можуть бути як мінімальними скриптами автоматизації (CREATEDYNAMICPATIENT, SEARCHSELECTPATIENT), так і певними ключовими командами взаємодії з користувацьким інтерфейсом, як то CLICK для кнопок чи SENDKEYS для певних текстових полів.



458_NAW_Cited List_Already Existing AllergyUpdation_Non matching encounter_displ

File | Home | Insert | Page Layout | Formulas | Data | Review | View | Team

A3 | CREATEDYNAMICPATIENT

	A	B	C	D	
1	**Action**	**Field/ScreenName**	**Input**	**VerificationData**	**Comment**
2	#precondition to add allergy with annotation				
3	CREATEDYNAMICPATIENT	NONE	1	DOB=05/05/1981%SEX=M%MARITAL=Si	NONE
4	SetPreferenceUserLevel	NONE	swathi%General%EncounterSummaryReview	NONE	NONE
5	EXECUTETEST	NONE	tmp_login.xls;tmp_Login_Data.xls;1002-23	NONE	CALLS THE
6	EXECUTETEST	NONE	tmp_SelectFromVTB.xls;tmp_SelectFromVTB_	NONE	Select Ch
7	WAITFOROBJECT	SerachAndSelect Patient Window	<<SyncLevel1>>	Enabled,True	NONE
8	SEARCHSELECTPATIENT	NONE	[[1:Last_Name]],[[1:First_Name]]	Name	Searchin
9	WAITFOROBJECT	IDXBanner Common Frame_Patient Banner WebTabl	10	Isupdating,False	NONE
10	VERIFYCELLDATAINTABLE	IDXBanner Common Frame_Patient Banner WebTabl	NONE	[[1:Last_Name]],[[1:First_Name]]	Verificati
11	WAITFOROBJECT	IDXWorkDotNet Frame_AddNewProblem Button	<<SyncLevel1>>	Enabled,True	NONE
12	CLICK	IDXWorkDotNet Frame_AddNewProblem Button	NONE	NONE	Click on A
13	EXECUTETEST	NONE	tmp_ACITabSelect.xls;tmp_ACITabSelect_Dat	NONE	NONE
14	EXECUTETEST	NONE	tmp_SearchItemInACI.xls;tmp_SearchItemIn/	NONE	NONE
15	WAITFOROBJECT	Add Clinical Item_SwfListView	5	Isupdating,False	NONE
16	SELECT	Add Clinical Item_SwfListView	Atenolol TABS	NONE	NONE
17	SENDKEYS	NONE	<SPACE>	NONE	NONE
18	EXECUTETEST	NONE	tmp_CreateNewEncounter;tmp_CreateNewEr	NONE	Creating
19	CLICK	Add Clinical Item_OK Button	NONE	NONE	Clicking (

Рис. 1. Приклад кейвордскрипта

Загалом така імплементація ключовими словами є загальновідома, загальновизнана і популярна [1]. Стандартною є ситуація коли компанії імплементують власний фреймворк автоматизації для певних продуктів, і використовують саме такий підхід. Зазвичай трактується це тим, що використання ключових слів дасть змогу людям, які не розуміються на написанні програмного коду – ручним тестувальникам – простіше писати автоматизовані скрипти.

Підхід не повністю виправданий через значну кількість недоліків.

1. Написання кейвордів визначає пряму залежність їх імплементації від інструментів автоматизованого тестування (простіше кажучи програм, які будуть безпосередньо «нажимати кнопки» тестованої системи).

2. Верифікація скриптів значно ускладнюється:

а) Потрібен додатковий запуск всього скрипта, щоб перевірити чи правильно сформована послідовність кроків.

б) Опис тестового скрипта за допомогою стрічкових констант не дає можливості перевірити правильність написання ключових слів, помилка може бути спричинена просто опечаткою.

в) Параметри, які передаються ключовим словам, також не мають ніякого механізму верифікації на їх кількість та тип.

3. Читабельність таких скриптів катастрофічно низька.

Метою роботи є покращення якості автоматизованого тестування. Відповідно в рамках даної статті *задачею* є опис певних методик, механізмів та утиліт, які б могли покращити процес написання автоматизованих тестів, базовних на ключових словах.

Ключові слова як функціональні методи

Основною метою створення ключових слів як міні тестових скриптів є задача інкапсуляції певної логіки в рамках окремих перевикористовуваних модулів. Цей патерн в автоматизованому тестуванні можна ще назвати функціональним методом[2].

Однак пряме використання функціональних методів в високорівневих оточеннях розробки коду як MicrosoftVisualStudio, Eclipse, IntelliJIdea повертає нас до проблеми написання скриптів користувачами, які були б не знайомі з синтаксисом мов програмування. Тому першим кроком для покращення цієї ситуації було би використання існуючих бібліотек ключових слів, які б можна було підвантажувати в систему компоновки цих бізнес методів.

Система підвантаження бібліотек ключових слів

На рисунку 2 зображена можлива схема підвантаження бібліотек ключових словдля більш послідовного та логічного використання їх кінцевими користувачами.

Через специфічні для окремих технологій програмування (Java, .NET, Python…)парсери (TechnologyBridge)компоненти, бібліотеки ключових слів підвантажуються у систему і перетворюються в мета-данні, які вже не залежать від конкретної технології (TechnologyAgnosticMetadata) . Зберігається інформація про назви методів, опис, коментарі, параметри. Саме ця інформація і доступна користувачу через користувацький інтерфейс (UserInterface). Також користувацький інтерфейс дає змогу створювати скрипти, робити компоновку послідовності виклику методів, запускати окремі скрипти або їх набори на фреймворку запуску тестів (TestRunComponent).

В рамках дослідження така система була розроблена і відповідний TechnologyBridgeпобудований для технології програмування .NET. Як базова оболонка системи було обрано середовище MicrosoftVisualStudioShell, яке є безкоштовним для використання та перерозповсюдження (за виключенням мінімальних ліцензійних обмежень).

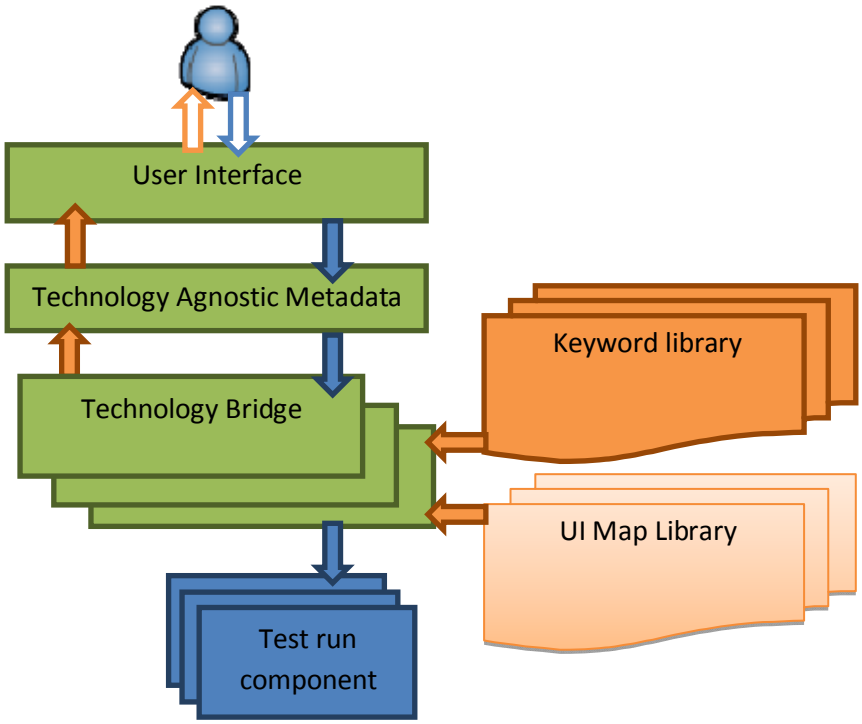


Рис. 2. Схематичне відображення системи побудови скриптів з бібліотек ключових слів

Користувацький інтерфейс цієї системи зображено на рисунку 3. Так

1 – список кроків в тесті. Відображає скомпоновані бізнес методи системи.

2 – область робочого тестового проекту. Відображає існуючі тести в проекті та дозволяє налаштовувати загальні для всіх тестів властивості.

3 – область властивостей активного вибраного компоненту. На рисунку активним вибраним компонентом є перший крок тестового скрипта, відповідно відображаються його властивості. Особливо важливими властивостями тут є вхідні параметри даного тестового кроку/бізнес методу/ключового слова. За рахунок технологічно специфічного парсера ми отримуємо можливість інформувати користувача про типи параметрів, які йому потрібно передати всередину кейворду.

5 – область результатів виконання тестів.

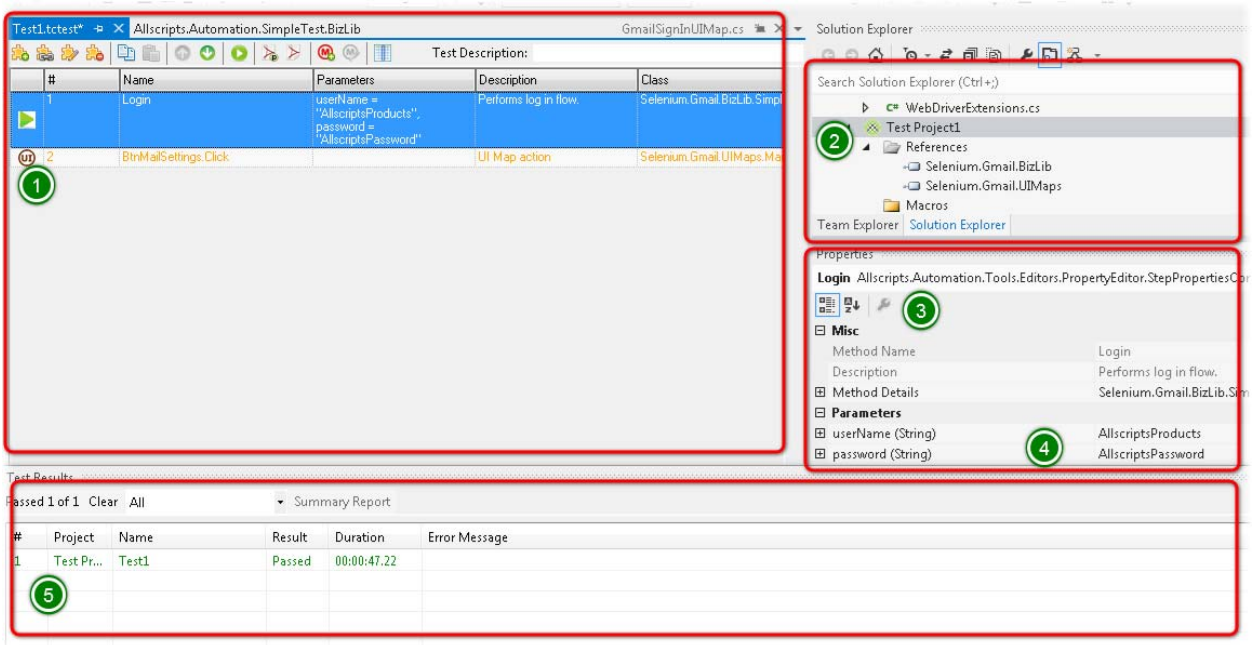


Рис. 3. Середовище компоновки тестових сценаріїв

Компонування функціональних методів в ланцюжки тестових сценаріїв

Однією з найцікавіших підзадач в рамках даного дослідження є можливість спрощення побудови самих скриптових тестових ланцюжків.

Отже, крок перший – як забезпечити *зручне додавання кроку в тестовий скрипт*? Маючи вже

підвантажений реєстр функціональних методів, нам перш за все потрібно спростити пошук потрібного методу. З власної практики різних тестових рішень, кількість методів в рішенні може досягати декількох тисяч. Тому коли користувач хоче додати наступний крок до тестового сценарію, він має наступні можливості фільтрації та відбору (рис. 4):

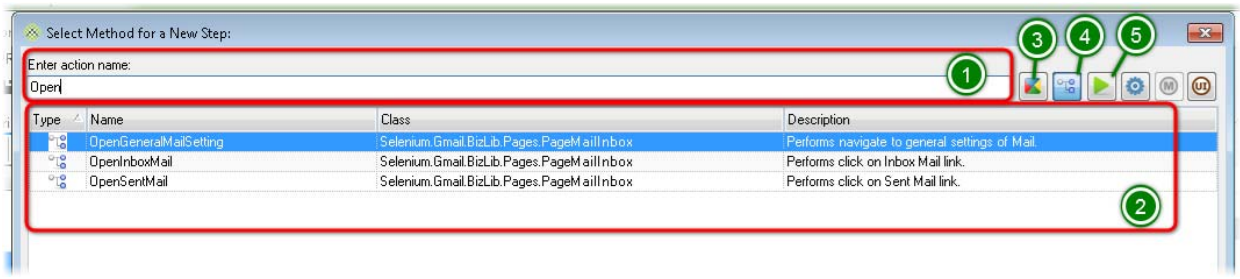


Рис. 4. Вибір нового кроку тестового скрипта

Перш за все, звернемось до поняття контексту в процесі виконання тестового скрипта. Тестовий сценарій по замовчуванню ніяк не зв'язаний з системою, над якою виконує операції. Однак різні методи поводять себе по різному. Наприклад метод Loginпевного автоматизованого рішення зазвичай починає роботу на сторінці входу в систему і закінчує своє виконання змінивши активну сторінку тестованої системи на домашню сторінку. Відповідно при виборі наступного методу нашої послідовності найбільш логічним буде не серед всієї множини можливих, а лише серед тих, які надає нам контекст домашньої сторінки тестованої системи.

Але як функціональному методу вказати який контекст наступний для конкретного методу? Найбільш логічно буде кожен функціональний метод описати специфічним чином (наприклад коментар, чи атрибут для компільованих мов програмування), в якому вказати, в який контекст переходить система після виклику даного методу. Також можливо використання ланцюжків методів (fluentinterface, methodchaining).

Саме так організована фільтрація в нашому випадку. Отже можливості діалогу вибору наступного кроку:

1 – фільтр пошуку функціональних методів по їх імені та по опису (коментарю).

2 – список отриманих в результаті фільтрації методів;

3 – фільтр відображення всіх методів, наявних в системі;

4 – відображення лише методів актуальних для теперішнього контексту;

5 – відображення вхідних точок тестових сценаріїв.

Крізний доступ від системи побудови тестових сценаріїв до елементів користувацького інтерфейсу

Імплементація ключових слів дає змогу перевикористовувати елементи коду у великих об'ємах. Однак є певне обмеження з підтримкою самих ключових слів – з часом кількість тестових сценаріїв може збільшуватись, а у проекту може не бути додаткових ресурсів для розробки нових кейвордів/функціональних методів. Як було описано у вступі, у таких випадках може бути використаний підхід розробки загальних універсальних ключових слів, який буде взаємодіяти з абстрактним користувацьким інтерфейсом (CLICK, SENDKEYS). Однак використання цих методів при побудові тестових сценаріїв може затягуватись через багато причин, як то знову таки відсутність верифікації/компіляції.

Відповідно схема, зображена на рис. X також відображає можливість підвантаження в тестовий сценарій так званих бібліотек мап користувацьких інтерфейсів(UIMapLibrary).

Мапа користувацьких інтерфейсів(МКІ) – це унітарна складова нижнього рівня імплементації тестового рішення в багатошаровому рішенні автоматизованого тестування [4].Зазвичай більшість утиліт автоматизованого тестування (CodedUI, TelerikAutomation, Ranorex…)підтримують функціонал автоматичної генерації МКІ. Відповідно після автоматичної генерації, маючи відповідний механізм завантаження МКІ в систему компонування тестових методів може дозволити їх пряме використання в тестових скриптах.

В рамках дослідження було розроблено універсальний механізм завантаження будь-яких МКІ, згенерованих утилітами автоматизованого тестування з підтримкою .NET API.

На рис. 5 зображено можливість завантаження елементів МКІ. Відповідно фільтр 2 показує ті елементи, які завантажені саме з цього рівня (результат – секція 3 на рис. 5). Також для того щоб мати певний набір утилітарних методів, незалежних від програми автоматизації, було розроблено набір системних методів 1 (їх список видно у секції 4).

Таким чином, комбіноване використання ключових слів та елементів МКІ (Рис.3, секція 1 – перший крок – функціональний метод, другий крок – виклик методу МКІ), покриває практично повний спектр розробки рішень автоматизованого тестування, при цьому залишаючи основні переваги використання метасистеми завантаження ключових слів та елементів МКІ.

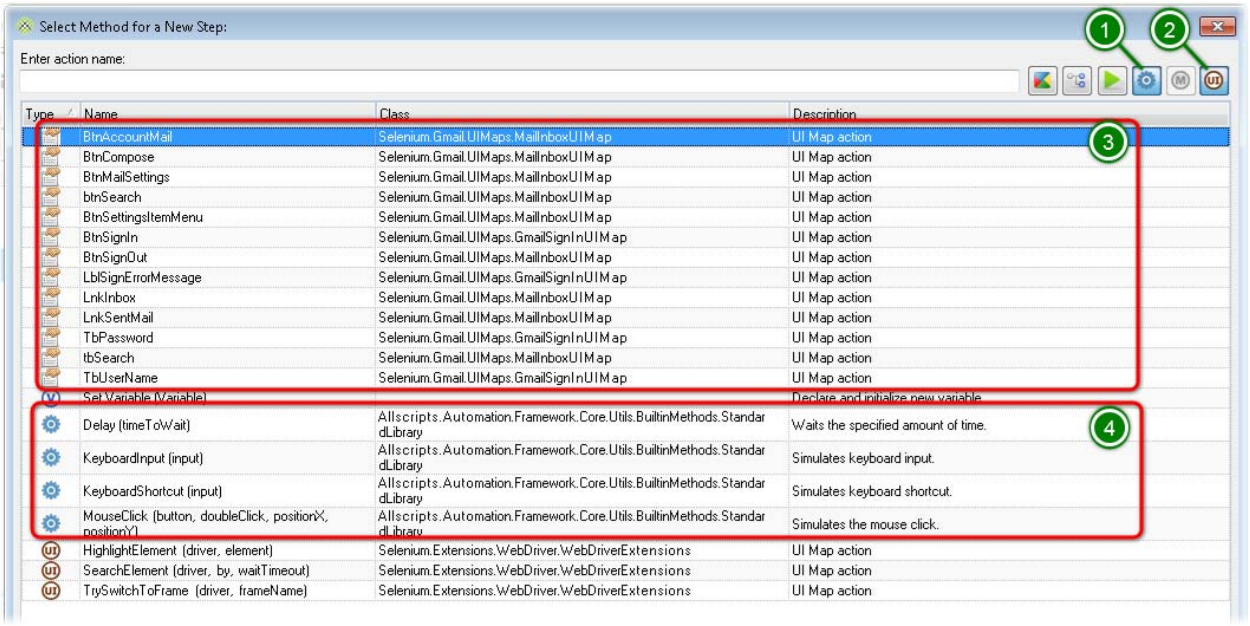


Рис. 5. Вибір нового кроку тестового скриптапри прямому доступі до МКІ

Висновки

В статті вперше запропонована система побудови скриптових сценаріїв на базі підвантажених ключових слів та елементів МКІ, які не залежали б від мови імплементації МКІ та утиліти автоматизованого тестування.

Описана система надає ряд незаперечних переваг:

- Кейворди абстраговані від інструментів автоматизованого тестування.
- Початкове завантаження ключових слів та МКІ у вигляді бібліотек не дає змоги помилитись в написанні методів та елементів МКІ, адже інтерфейс спроектований лише на базі фільтрів, і відповідо неіснуючий метод вибрати неможливо.
- Присутня перевірка на обов'язкові параметри та їх обов'язкова типізація.
- Єдина методика побудови скрипта дозволяє дещо покращити його читабельність.

Література

1. В. П. Котляров, Т. В. Коликова, Основы тестирования программного обеспечения– 2006 – 248с.

2. Elfriede Dustin, Thom Garrett and Bernie Gauf, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality/ Addison-Wesley Professional – 2009. – 368p.

3. Пишем систему автоматизированных тестов с нуля [Електронний ресурс] // http://www.protesting.ru/automation/practice/ automation_from_scratch.html

4. OleksandrReminnyi, Functional GUI Testing Automation Patterns [Електронний ресурс] // Режим доступу до файлу: http://www.infoq.com/articles/gui-automation-patterns

References

1. V. P. Kotlyarov, T. V. Kolykova, Osnovy testyrovanyya prohrammnoho obespechenyya – 2006 – 248s.

2. Elfried Dustin, ThomGarrett and Bernie Gauf , Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality / Addison-Wesley Professional – 2009. – 368p.

3. Pyshem systemu avtomatyzyrovannykh testov s nulya // http://www.protesting.ru/automation/practice/ automation_from_scratch.html (Aug 25 2013).

4. Oleksandr Reminnyi, Functional GUI Testing Automation Patterns // http://www.infoq.com/articles/gui-automation-patterns (Aug 25 2013).

Рецензія/Peer review : 4.9.2013 р. Надрукована/Printed :17.10.2013 р.
Рецензент: д.т.н., проф., зав. каф. програмного забезпечення автоматизованих систем ВНТУ А. М. Пєтух