УДК 519.682:004.423
DOI: 10.15827/0236-235X.128.601-606
Дата подачи статьи: 11.04.19
2019. Т. 32. № 4. С. 601–606

Особенности применения предметно-ориентированных языков для тестирования веб-приложений

В.Г. Федоренков [1], *студент, vlad.fedorenkov@gmail.com*
П.В. Балакшин [1], *к.т.н., доцент, pvbalakshin@gmail.com*

[1] *Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО), г. Санкт-Петербург, 197101, Россия*

Перед разработчиками как крупных, так и небольших проектов, стремящихся выпустить по-настоящему качественный, хороший продукт с минимальным числом ошибок, часто встает немало вопросов относительно реализации процесса его тестирования. Данная работа посвящена поиску ответов на эти вопросы.

В статье сравниваются основные методы, а также существующие программные средства создания и поддержки доменно-ориентированных языков (англ. DSL, domain specific language), нацеленных на использование в контексте работы с тестовыми сценариями для тестирования интерфейсов веб-приложений. Определены наиболее подходящие технические приемы и средства для решения подобных задач на основе опыта авторов и результатов схожих исследований других специалистов. Проведен обзор существующих подходов к работе с инструментом Selenium, активно использующимся (как в данной работе, так и в большинстве подобных проектов) при автоматизации процесса тестирования для имитации действий пользователя в веб-браузере. Описаны преимущества использования DSL в тестировании, определены функциональные и нефункциональные требования к созданию предметно-ориентированных языков для их дальнейшего эффективного использования, рассмотрены различные методы создания DSL с точки зрения структуры языка.

В качестве одного из основных критериев для работы было выбрано вовлечение нетехнических специалистов на каждом этапе тестирования (решение так называемой проблемы перевода), что особенно важно для реализации всестороннего тестирования программного продукта.

Одной из ключевых особенностей статьи является демонстрация реализации прототипа DSL на основе Selenium с последующим тестированием и оценкой применимости реализованного прототипа. В упрощенном виде продемонстрирована структура внутреннего устройства языка по Java-пакетам. Сформулированы рекомендации по написанию DSL на основе ранее определенных требований и произведенной оценки прототипа. Показан способ разработки дополнительного инструмента метапрограммирования для дальнейшего упрощения создания, поддержки, модификации тестовых сценариев и их миграции на новые платформы.

Ключевые слова*: DSL, программное обеспечение, тестирование, веб-приложение, разработка, интерфейс, функциональность, Selenium.*

Считается, что ПО для тестирования имеет решающее значение для обеспечения качества финального продукта. Часто стратегии тестирования в средних и крупных проектах (из-за сравнительно большого числа участников данного процесса) сильно варьируются в плане используемых инструментов, уровней тестирования, методологий, уровня автоматизации [1, 2]. При этом неизменно самым желанным результатом является выпуск качественного продукта с минимальным числом ошибок. В общем случае ключ к качественному тестированию можно найти, ответив на три фундаментальных вопроса:

– каким образом можно обеспечить раннее начало процесса тестирования и наименьший интервал между запусками тестов с целью снижения рисков;

– каким образом можно обеспечить достаточную продолжительность поддержки тестов, их улучшение и повторное использование;

– как можно вовлечь в процесс тестирования все заинтересованные стороны с целью максимального соответствия поставленным требованиям.

В настоящее время веб-приложения являются одной из наиболее распространенных областей разработки и применения современного ПО. Следовательно, необходимо ответить на три поставленных вопроса именно с точки зрения веб-приложений.

Данная работа посвящена поиску ответов на эти вопросы, главным инструментом которого является концепция использования DSL – доменно-ориентированных языков, разработанных для решения узкоспециализированных задач (в рассматриваемом случае – задач тестирования пользовательских интерфейсов веб-приложений).

Средства создания и поддержки DSL

Принято разделять два основных типа доменно-ориентированных языков на внешние и внутренние. Внешние DSL имеют собственный синтаксис, отделенный от основного языка приложения. Внутренние DSL используют в своей основе язык программирования общего назначения, но отличаются тем, что используют конкретное подмножество возможностей этого языка в определенном стиле [3, 4]. Для разработки DSL существует достаточное число всевозможных языков, платформ и сред разработки [5, 6]. Выбор средства необходимо определять исходя из требований тестируемой системы (англ. SUT, system under test), необходимых навыков у команды разработчиков и тестировщиков. Опыт авторов и результаты других исследований [5, 7, 8] показывают, что для дальнейшей качественной поддержки и применения в тестировании программных продуктов целесообразно использовать средства, основанные на грамматиках языков программирования общего назначения.

Использование DSL в тестировании

DSL для SUT дает возможность команде стандартизировать определенный набор терминов для описания домена, что минимизирует ошибки перевода. Кроме того, DSL позволяют нетехническим заинтересованным сторонам взаимодействовать с тестировщиками более конструктивно. Правильно спроектированный DSL, как минимум, сделает тесты легко читаемыми для всех заинтересованных сторон. При правильном развитии такого DSL любые нетехнические пользователи, в том числе эксперты по предметным вопросам (англ. SME, subject matter expert), могут быть вовлечены непосредственно в процесс написания тестов.

В рамках тестирования интерфейсов веб-приложений главным инструментом, вокруг которого и будет сфокусирована разработка DSL, является популярная библиотека Selenium.

Такой выбор действительно прост и однозначен, ведь только Selenium предоставляет достаточный функционал для взаимодействия с приложением посредством большинства современных браузеров [9, 10]. Взаимодействие возможно благодаря набору драйверов, выпускаемых непосредственно компаниями-разработчиками браузеров. Именно они дают возможность осуществлять имитацию действий пользователя в веб-приложении, алгоритм которых может быть изложен на одном из множества поддерживаемых прикладных языков программирования, например, на Ruby, Python, Scala, C#, F#, Haskel и других [5, 11]. Для разработки прототипа языка в данной работе был использован язык Java.

В качестве домена, то есть веб-приложения, для тестирования которого и разработан прототип, было использовано приложение, представляющее собой пользовательский интерфейс для работы с системой комплексной диагностики сети, применяемое в немецком концерне Deutsche Telekom AG. Интерфейс этого приложения отличается нетривиальной структурой и содержит практически все возможные веб-компоненты, что позволяет отлично продемонстрировать работу DSL. Были определены требования к языку, поскольку именно формирование четких требований позволяет избежать всех проблем и сложностей, включая создание языковой инфраструктуры, появляющихся у специалистов по тестированию [5]. Функциональные требования:

– полная совместимость с актуальной версией библиотеки Selenium;

– максимальное соответствие используемой бизнес-модели;

– имплементация работы со всеми наиболее часто используемыми элементами веб-страниц (кнопки, текстовые поля, селекторы, выпадающие списки и т.п.);

– возможность параллельного выполнения операций;

– наличие понятного для конечного пользователя языка функционала для логирования.

К нефункциональным требованиям относится читаемость (в частности, для нетехнических пользователей).

Для выбора оптимальной структуры языка были рассмотрены три основных метода создания DSL: на основе вложенных функций, на основе цепочек методов, на основе лямбда-выражений.

Из них был выбран метод создания DSL на основе цепочек методов. Такой выбор обуслов-

лен наиболее простой структурой с точки зрения конечного пользователя языка, что критически важно для вовлечения в процесс тестирования нетехническеских пользователей.

- DSL на основе вложенных функций:

```
Graph(
   edge(from("a"), to("b"), weight(12.3)),
   edge(from("b"), to("c"), weight(10.5))
);
```

- DSL на основе лямбда-выражений:

```
Graph(g -> {
   g.edge(e -> {
      e.from("a");
      e.to("b");
      e.weight(12.3);
   });
   g.edge(e -> {
      e.from("b");
      e.to("c");
      e.weight(10.5);
   });
});
```

- DSL на основе цепочек методов:

```
Graph()
   .edge()
      .from("a")
      .to("b")
      .weight(12.3)
   .edge()
      .from("b")
      .to("c")
      .weight(10.5);
```

Разработанный DSL

В упрощенном виде структура внутреннего устройства языка включает (по Java-пакетам):

– Authorize – функции для basic-авторизации, реализованные для различных браузеров (так, например, для Selenium 3 способы авторизации при помощи Google Chrome и Internet Explorer 11 принципиально различаются);

– Config – всесторонняя конфигурация тестов от аспектов бизнес-логики до настроек записи видео;

– Constants, Enums, Exception – определяемые бизнес-логикой константы, перечисления, исключения;

– Ctx – контекст выполнения, управляющий потоками и содержащий всю необходимую в ходе теста информацию;

– Driver – функционал для работы с различными браузерами;

– Helpdesk – пакет, включающий классы и методы для работы со всеми используемыми в тестируемом веб-приложении элементами (кнопками, radio-кнопками, полями для ввода, иконками и т.п.);

– Logic – методы, реализующие логические операторы в DSL;

– Recording – функционал для сохранения скриншотов и видео выполнения тестов (с использованием ffmpeg).

На основе разработанных требований к языку были сформулированы следующие рекомендации по его написанию.

- Для обеспечения наилучшей читаемости разрабатываемый язык должен оперировать только в терминах бизнес-логики. Любые технические операторы лучше оставить за кадром.
- Должны существовать различные способы конфигурирования тестов для различных сценариев их использования. Для нетехнических пользователей будет намного понятнее конфигурирование теста непосредственно в его коде при помощи специальных конструкций в DSL.
- При использовании структуры языка, где каждый метод представляет собой обратный вызов, цепочка из которых начинает свое исполнение лишь при достижении замыкающего метода такой цепочки, целесообразно использование Java Concurrency API. Это заметно усложняет внутреннее устройство языка (но не саму структуру DSL), но позволяет удобно организовывать параллельную обработку выполняемых в тесте проверок.
- Уже на начальном этапе разработки языка стоит спланировать процесс логирования. На практике это позволяет значительно быстрее находить источник ошибки, причем подобный функционал работоспособен даже при запуске браузера в headless-режиме, что особенно полезно для автоматических тестов.

Очевидно, что основными критериями для определения оценки применимости прототипа на базе DSL являются его удобочитаемость и наглядность. Так, например, выглядит код теста без использования DSL:

```
   logger.debug("Verifying icons in Di-
agnosestart panel");
   logger.debug("1) Icon 'VoIP fehler-
haft' expected state - Error");
   assertTrue(Tools.isDiagnoseIconError
("VoIP fehlerhaft", wait));
   logger.debug("2) Icon 'PFS-Fehler'
expected state - Error");
   assertTrue(Tools.isDiagnoseIconError
("PFS-Fehler", wait));
   logger.debug("3) Icon 'Ereignis-Er-
mittlung' expected state - OK");
   assertTrue(Tools.isDiagnoseIconOk
("Ereignis-Ermittlung", wait));
   logger.debug("4) Icon 'Konfiguration
fehlerfrei' expected state - OK");
```

```
   assertTrue(Tools.isDiagnoseIconOk
("Konfiguration fehlerfrei", wait));
```

А так аналогичный код с использованием DSL:

```
   .diagnoseProzesse()
      .togglePanel("Diagnosestart")
         .isProcessIndicatorError("VoIP
fehlerhaft")
         .isProcessIndicatorError("PFS-
Fehler")
         .isProcessIndica-
torOk("Ereignis-Ermittlung")
         .isProcessIndicatorOk("Konfigu-
ration fehlerfrei")
      .ende()
   .ende()
```

Применение метапрограммирования с разработанным прототипом DSL

Для ряда программных продуктов, подразумевающих продолжительный цикл разработки, поддержки и сопровождения, помимо создания ранее рассмотренного DSL с целью оптимизации процесса тестирования, может быть целесообразно внедрение дополнительных инструментов метапрограммирования [4, 12].

Само по себе использование DSL при тестировании веб-приложений значительно сокращает время написания тестов благодаря оперированию тестировщиками понятными для них терминами. Но даже это можно практически автоматизировать, доверив генерирование кода теста программе и поручив пользователю выполнять привычные для него действия, сохранив при этом все те преимущества, которые дает DSL [13].

Иными словами, с готовым DSL-языком достаточно просто можно разработать дополнительный инструмент, который позволил бы переводить действия тестировщика (если речь идет о тестировании веб-приложений, то это, очевидно, действия пользователя на веб-странице: навигация, клики, ввод текста и т.п.) в готовый к исполнению код на этом DSL. Это позволило бы свести затраты времени и усилий на каждый новый тест к минимуму, обеспечив ту же степень читаемости кода и простоты его сопровождения, как если бы он был написан непосредственно человеком [6].

В рамках данной работы для той же предметной области, что и ранее рассмотренный язык, было разработано расширение для браузера Chromium на языке Javascript. Выбор формата обусловлен простотой использования конечным пользователем, так как в таком случае для написания теста от последнего потребуется лишь наличие совместимого браузера. Расширение позволяет интегрировать в просматриваемую веб-страницу специальный Javascript-код, отображающий некий оверлей для всех ключевых элементов, таких как кнопки, таблицы, иконки, текстовые элементы и т.п.

Благодаря этому пользователь может буквально в несколько нажатий мыши осуществить определенную проверку (например, соответствие иконки или текста ячейки таблицы ожидаемым значениям), которая далее будет отражена в сгенерированном коде.

Таким образом, создание нового теста приложения сводится к однократному выполнению тестировщиком всех необходимых действий, а результатом является код на DSL, который легко читать и сопровождать в дальнейшем.

В результате внедрения такого DSL в работу проекта можно сделать, в том числе на основе отзывов конечных пользователей языка, а также статистики тестов на сервере непрерывной интеграции, следующие выводы.

- Читаемость кода действительно становится намного выше. Даже грамотно оформленный и прокомментированный код теста в обычном формате будет в лучшем случае более нагруженным, например, для заказчика, который имеет слабое представление о классических языках программирования.
- Тесты на основе DSL проще создавать и сопровождать. В среднем для нетехнического пользователя создание теста на основе DSL занимает примерно на 20–30 % меньше времени по сравнению с традиционным подходом.
- При изменениях, например, в html-разметке тестируемого приложения, DSL позволяет восстановить работоспособность тестов при минимальном числе правок, как правило, оставляя код самих тестов нетронутым, что дает возможность значительно эффективнее организовать работу автоматических тестов (например, при использовании непрерывной интеграции).
- Для ряда программных продуктов может быть целесообразна разработка дополнительного инструмента генерирования DSL-кода. При его использовании создание нового теста приложения сводится к однократному выполнению тестировщиком всех тех действий, которые он хочет реализовать, а результатом является код на DSL, легко читаемый и сопровождаемый в дальнейшем.

Заключение

Использование различных средств автоматизации практически всегда приводит к сокращению расходов и улучшению тестирования автоматизируемого процесса. Для сферы информационных технологий данный тезис является полностью верным. Для широко распространенных в настоящее время веб-приложений создание и поддержку автоматизации тестирования целесообразно делать, применяя доменно-ориентированные языки с использованием элементов метапрограммирования. Это обеспечит раннее начало процесса тестирования, улучшит его сопровождение, миграцию на новые платформы, браузеры и т.п., а также за счет небольшого порога вхождения позволит SME и другим заинтересованным сторонам участвовать в тестировании на любой его стадии.

Литература

1. Куликов С.С. Тестирование программного обеспечения. Базовый курс. Минск, 2017. 312 с.
2. Ермыкин А.А. Разработка метода построения комплекса нагрузочного тестирования распределенной информационной системы. СПб: Изд-во СПбГУ ИТМО, 2005. 147 с.
3. Fowler M. Domain specific languages. Addison-Wesley Professional, 2010, 640 p.
4. Kirgizov G.V., Kirilenko I.A. Heterogeneous architectures programming library. Proc. ISP RAS, 2018, vol. 30, iss. 4, pp. 45–62. DOI: 10.15514/ISPRAS-2018-30(4)-3.
5. Ботов Д.С. Обзор современных средств создания и поддержки предметно-ориентированных языков программирования // Вестн. ЮУрГУ. 2013. Т. 13. № 1. С. 10–15.
6. JetBrains. Meta Programming System. Create your own domain-specific language. URL: https://www.jetbrains.com/mps/ (дата обращения: 17.01.2019).
7. Ratiu D., Voelter M., Pavletic D. Automated testing of DSL implementations – experiences from building mbeddr. Software Quality J., 2018, vol. 26, iss. 4, pp. 1483–1518. DOI: 10.1145/2896921.2896922.
8. Сухов А.О. Разработка предметно-ориентированных языков на основе онтологий // Современные проблемы математики и ее прикладные аспекты: сб. тез. конф. Пермь, 2013. С. 45.
9. Gochenour P. Getting Started with Selenium for Automated Website Testing. URL: https://wiki.saucelabs.com/display/DOCS/Getting+Started+with+Selenium+for+Automated+Website+Testing (дата обращения: 29.12.2018).
10. Leotta M., Clerissi D., Ricca F., Spadaro C. Improving test suites maintainability with the page object pattern: an industrial case study. Proc. ICSTW, Luxembourg, 2013, pp. 108–113. DOI: 10.1109/ICSTW.2013.19.
11. Gray J., Yue S. SPOT: A DSL for Extending Fortran Programs with Metaprogramming. Advances in Software Engineering, 2014, vol. 2014, art. ID 917327, 23 p. DOI: 10.1155/2014/917327.
12. Handy non-default settings for MPS. URL: http://dslfoundry.com/category/meta-programming-system/ (дата обращения: 11.03.2019).
13. Radojičić M. Selenium + JavaScript Best Practices. URL: https://blog.testproject.io/2018/03/08/selenium-javascript-best-practices/ (дата обращения: 02.02.2019).

Software & Systems
DOI: 10.15827/0236-235X.128.601-606
Received 11.04.19
2019, vol. 32, no. 4, pp. 601–606

Domain-specific languages for testing web applications

V.G. Fedorenkov [1], *Student, vlad.fedorenkov@gmail.com*
P.V. Balakshin [1], *Ph.D. (Engineering), Associate Professor, pvbalakshin@gmail.com*

[1] *The National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, 197101, Russian Federation*

Abstract. The desire to release a high quality product with minimal errors often raises many problems regarding product testing for developers of both large and smaller projects. This work is devoted to searching for solutions for these problems.

The paper compares the main methods as well as the existing software tools for creating and supporting domain specific languages aimed at working with test scripts to testing interfaces of web applications. It also

considers existing tools for working with Selenium, reviews the methodology of writing DSL (with further selection of the most appropriate), shows how to implement a prototype of DSL based on Selenium and to test and assess the applicability of a prototype. It describes the advantages of using DSL in testing, its functional and non-functional requirements, shows the developed DSL in a simplified form, the language structure (Java-packages).

One of the main criteria for working with all of the abovementioned is the involvement of non-technical specialists at each testing stage (solving the so-called translation problem), which is important for implementing comprehensive testing of a software product.

One of the key features of the article is the demonstration of implementing a DSL prototype based on Selenium, followed by testing and evaluating the applicability of the implemented prototype. The paper shows a method of creating an additional metaprogramming tool for further simplification of creation, support, and modification of the developed test scripts.

Keywords: DSL, software, testing, web application, development, user interface, functionality, Selenium.

References

1. Kulikov S.S. *Software Testing. Base Course*. 2nd ed. Minsk, 2017, 312 p.
2. Yermykin A.A. *Development of a Method for Constructing a Load-Testing Complex of a Distributed Information System*. PhD Thesis. St. Petersburg, SPb ITMO Publ., 2005, 147 p.
3. Fowler M. *Domain Specific Languages*. Addison-Wesley Prof. Publ., USA, 2010, 640 p.
4. Kirgizov G.V., Kirilenko I.A. Heterogeneous Architectures Programming Library. *Proc. ISP RAS*. 2018, vol. 30, iss. 4, pp. 45–62. DOI: 10.15514/ISPRAS-2018-30(4)-3.
5. Botov D.S. Review of modern development and support tools for domain-specific programming languages. *Bulletin of South Ural State Univ. Series Computer Technologies, Control, Radioelectronics*. 2013, vol. 13, no. 1, pp. 10–15 (in Russ.).
6. *JetBrains. Meta Programming System. Create Your Own Domain-Specific Language*. Available at: https://www.jetbrains.com/mps/ (accessed January 17, 2019).
7. Ratiu D., Voelter M., Pavletic D. Automated testing of DSL implementations – experiences from building mbeddr. *Software Quality J*. 2018, vol. 26, iss. 4, pp. 1483–1518. DOI: 10.1145/2896921.2896922.
8. Sukhov A.O. Ontology based creation of domain specific languages. *Conf. Proc. Actual problems of mathematics and its application aspects*. Perm, 2013, p. 45 (in Russ.).
9. Gochenour P. *Getting Started with Selenium for Automated Website Testing*. Available at: https://wiki.saucelabs.com/display/DOCS/Getting+Started+with+Selenium+for+Automated+Website+Testing (accessed December 29, 2018).
10. Leotta M., Clerissi D., Ricca F., Spadaro C. Improving test suites maintainability with the page object pattern: An industrial case study. *Proc. 2013 IEEE 6th Intern. Conf. on Software Testing, Verification and Validation Workshops*. Luxembourg, 2013, pp. 108–113. DOI: 10.1109/ICSTW.2013.19.
11. Gray J., Yue S. SPOT: A DSL for extending fortran programs with metaprogramming. *Advances in Software Engineering*. 2014, vol. 2014, art. ID 917327, 23 p. DOI: 10.1155/2014/917327.
12. *Handy Non-Default Settings for MPS*. Available at: http://dslfoundry.com/category/meta-programming-system/ (accessed March 11, 2019).
13. Radojičić M. *Selenium + JavaScript Best Practices*. Available at: https://blog.testproject.io/2018/03/08/selenium-javascript-best-practices/ (accessed February 2, 2019).

Для цитирования

Федоренков В.Г., Балакшин П.В. Особенности применения предметно-ориентированных языков для тестирования веб-приложений // Программные продукты и системы. 2019. Т. 32. № 4. С. 601–606. DOI: 10.15827/0236-235X.128.601-606.

For citation

Fedorenkov V.G., Balakshin P.V. Domain-specific languages for testing web applications. *Software & Systems*. 2019, vol. 32, no. 4, pp. 601–606 (in Russ.). DOI: 10.15827/0236-235X.128.601-606.