УДК 004.054
DOI 10.25205/1818-7900-2019-17-3-93-110

Методика автоматического тестирования развивающегося веб-приложения

А. В. Ткачев, **Д. В. Иртегов**

Новосибирский государственный университет
Новосибирск, *Россия*

Аннотация

Статья посвящена методике автоматизированного тестирования системы автоматической оценки заданий по программированию NSUts. При разработке методики главным приоритетом было параллельное тестирование старой и новой версий приложения так, чтобы одни и те же или минимально модифицированные тесты проходили на двух версиях системы с различными архитектурами. Мы надеемся, что наш опыт будет полезен при выстраивании процесса разработки других приложений с длительным жизненным циклом.

Чтобы тестировать не только серверную, но и клиентскую часть веб-приложения, мы предлагаем использовать инструменты типа Selenium WebDriver для симуляции действий пользователей, посылая команды настоящим браузерам. В методике применяется известный шаблон проектирования Page Object и рассматривается ряд приемов, позволяющих снизить хрупкость разрабатываемых тестов и упростить их адаптацию для работы с новой версией системы.

В статье также описано применение данной методики для организации тестирования системы NSUts и проведен анализ ее эффективности. Анализ показал, что оценочное покрытие кода данными тестами достаточно высоко, и потому методику можно считать эффективной и применять на схожих веб-приложениях.

Ключевые слова

тестирование веб-приложений, Selenium WebDriver, функциональное тестирование

Для цитирования

Ткачев А. В., *Иртегов Д. В.* Методика автоматического тестирования развивающегося веб-приложения // Вестник НГУ. Серия: Информационные технологии. 2019. Т. 17, № 3. С. 93–110. DOI 10.25205/1818-7900-2019-17-3-93-110

Developing Web-Application's Automated Testing Technique

A. V. Tkachev, **D. V. Irtegov**

Novosibirsk State University
Novosibirsk, *Russian Federation*

Abstract

The article is devoted to the technique of automated testing of NSUts – automatic assessment system for programming tasks developed at NSU. The main priority for this technique is to test both the old and the new versions of the application, so that the same or minimally modified tests could be executed on two versions of the system with different architectures. This could be useful while organizing the development process for other applications with a long life cycle.

To test not only the server but also the client side of the web application, we suggest using tools like Selenium WebDriver to simulate user actions by sending commands to real browsers. We use the well-known Page Object design pattern to handle differences in HTML layout and functionality, and describe a number of ways to make developed tests less fragile and easily adapt those to work with the new version of the system.

The article also describes the use of this technique to organize automated testing of the NSUts system and analyzes its effectiveness. The analysis shows that the estimated code coverage by these tests is quite high, and therefore the technique can be considered effective and applied to other similar web applications.

Keywords
web-application testing, Selenium WebDriver, functional testing
For citation
Tkachev A. V., Irtegov D. V. Developing Web-Application's Automated Testing Technique. *Vestnik NSU. Series: Information Technologies*, 2019, vol. 17, no. 3, p. 93–110. (in Russ.) DOI 10.25205/1818-7900-2019-17-3-93-110

Введение

При разработке системы автоматической оценки заданий по программированию NSUts авторы столкнулись с необходимостью улучшить процедуры контроля качества. Необходимость была связана, главным образом, с планом перевода системы на новую архитектуру и стек технологий. При этом у заказчиков и пользователей возникал резонный вопрос: можно ли доверять новому коду в той же степени, что и старой системе, у которой был накоплен большой (и преимущественно позитивный) опыт эксплуатации? Чтобы снять этот вопрос, было принято решение организовать автоматизированное тестирование старой и новой систем. Методике организации этого тестирования и посвящена данная работа. Мы надеемся, что сама методика и соображения, исходя из которых мы ее создавали, могут быть полезны при решении аналогичных задач, в первую очередь при построении систем поддержки образовательного процесса.

Разработку программного обеспечения можно описать как формализацию требований. В начальный момент требования заказчика доступны лишь в виде неформального описания, даже не всегда в виде единого документа на естественном языке. На основе этих описаний разрабатывается описание системы на машинно-читаемом языке или языках. Машинно-читаемость сама по себе налагает достаточно высокие требования на уровень формализации. В свете этого рассуждения контроль качества программного обеспечения можно описать как повторную формализацию тех же требований, как правило, на другом машинно-читаемом языке (например, на входном языке разрабатываемой системы) в надежде на то, что вероятность дважды совершить одну и ту ошибку существенно ниже.

В зависимости от характера задач формализацию целесообразно выполнять на разных этапах и в разных видах. В некоторых приложениях требования просты и стабильны, а стоимость возможной ошибки очень велика, поэтому целесообразно вложиться в формализацию самих требований, после чего можно применять инструменты для автоматической генерации кода и тестов к нему, а иногда и автоматической верификации.

В других ситуациях требования сложны или просто велики по объему и меняются часто, так что формализация на раннем этапе оказывается экономически нецелесообразной. В этом случае часто прибегают к ручному тестированию, когда формализацию требований в итоге выполняет тестировщик. В нашем случае занимать студентов ручным тестированием было политически невозможно, поэтому автоматическое тестирование было сочтено наиболее приемлемым вариантом.

Для системы NSUts уже были разработаны комплекты автоматизированных тестов на основе Selenium, но попытки применить их для тестирования новой системы выявили их хрупкость, в первую очередь чувствительность к изменениям верстки HTML. Поскольку новый стек технологий (генерация HTML на стороне клиента при помощи фреймворка React) предполагал полное изменение верстки, это делало старые тесты практически непригодными. Поэтому было решено разработать методику, позволяющую составить автоматические тесты таким образом, чтобы они работали как с новой, так и со старой версией системы, и требовали при этом минимальных изменений для запуска после внесения модификаций.

Обзор предметной области

В НГУ давно существует, активно развивается и используется система для автоматической оценки заданий по программированию NSUts. Она используется в учебном процессе,

для проведения школьных и вузовских олимпиад по программированию и тренировок сборной НГУ. Система достаточно сложна по сравнению с типичными студенческими проектами (1 406 файлов, 148 115 строк кода) и содержит около двадцати крупных функциональных модулей: собственно оценка заданий, построение рейтинга по различным правилам, возможность задавать вопросы и т. д. В 2010 г. для системы было разработано техническое задание в соответствии с требованиями ГОСТ 19.201-78, которое заняло 49 страниц. Поскольку система развивается, появляются новые требования, и этот документ потерял актуальность. Ему на замену в 2018 г. было составлено новое описание функциональности, которое содержит 37 пунктов, соответствующих отдельным страницам системы. Для каждой из этих страниц описано в среднем по 5 функций. Сам документ занимает более 50 страниц понятного человеку текста (рис. 1).

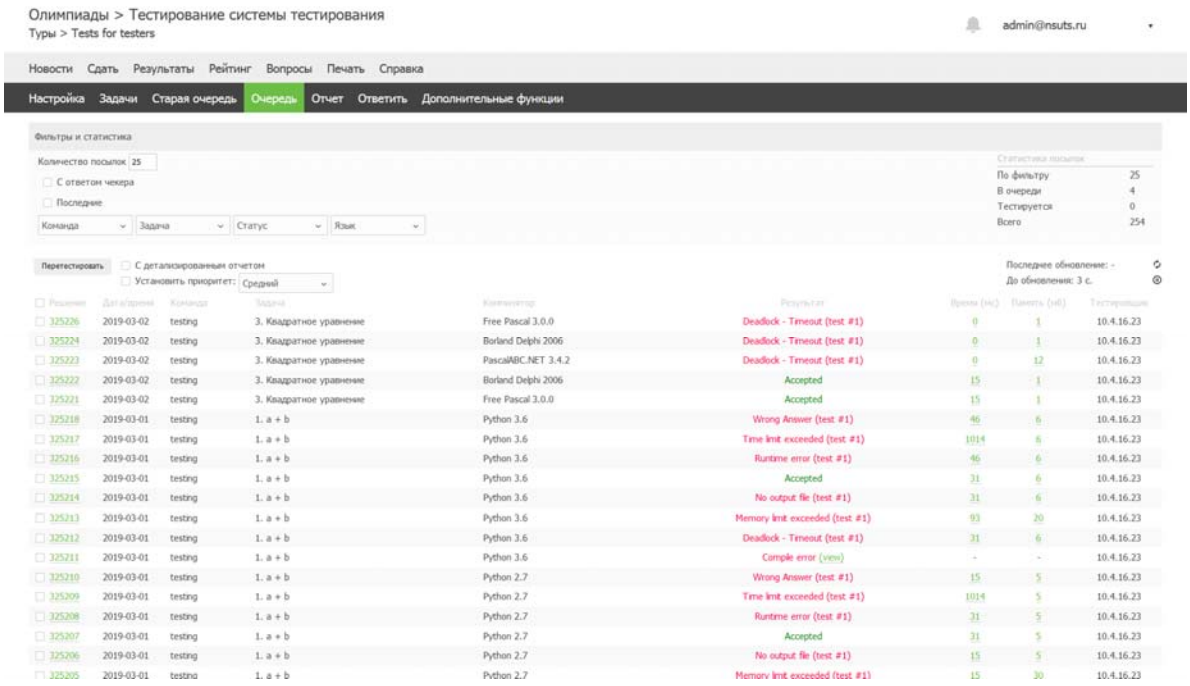

Рис. 1. Внешний вид одного из модулей системы

Fig. 1. Interface of One of the System's Modules

Ежегодно в системе проводится несколько различных олимпиад. Кроме того, на протяжении всего учебного года она используется преподавателями и студентами для сдачи заданий по курсам «Программирование на языке высокого уровня» и для тренировки олимпиадных команд НГУ. Каждый год на Открытой Всесибирской олимпиаде им. И. В. Поттосина представляется тур со специальной задачей с уникальными правилами.

Правила проведения некоторых олимпиад – например, регионального этапа Всероссийской олимпиады по информатике (РОИ) – меняются за несколько месяцев до их проведения, что требует оперативного внесения изменений в код системы. Иногда это требует существенной доработки, поскольку изменения затрагивают множество модулей сразу. Например, обновление правил РОИ в 2019 г. привело к изменению 972 строк в 14 файлах.

Обзор существующих решений

Изначально требования заказчика выражены неформально – даже если они подробно описаны в документах типа технического задания, они все еще описаны на естественном языке,

по которому невозможно сгенерировать конечный продукт. Потому при контроле качества, как и при разработке, эти требования рано или поздно формализуются в программный код или в действия, которые выполняет тестировщик. В зависимости от того, насколько сложны требования и как часто они изменяются, экономически выгодно формализовать их так или иначе (рис. 2).

Рассмотрим основные применяемые на практике подходы к контролю качества:

- ручное тестирование – формализация в ходе выполнения описанных неформально сценариев;
- формальная верификация – требования описываются на формальном языке, что позволяет затем автоматически генерировать тесты или верифицировать саму программу на соответствие этим требованиям;
- автоматическое тестирование – формализация требований в виде программного кода (тестовых сценариев).

Рис. 2. Схема разных подходов к формализации требований.
Сверху вниз: ручное тестирование, формальная верификация, автоматическое тестирование

Fig. 2. Different Approaches to Requirements Formalization.
From top to bottom: manual testing, formal verification, automated testing

Ручное тестирование

Если требования достаточно сложны или меняются часто, может оказаться очень эффективным простой подход ручного тестирования: человек, который достаточно хорошо знаком с неформально выраженными требованиями (иногда не в исходной их форме, а в форме плана тестирования), по сути формализует их в процессе выполнения различных сценариев в приложении либо может записать их в виде последовательности действий, которую должен выполнить другой специально обученный человек. Данная технология широко используется при разработке приложений поддержки бизнеса, учебного процесса, веб-приложений и др.

В процессе отладки нового кода, разработчики в той или иной форме выполняют отдельные операции по ручному тестированию. Но при этом ошибка в коде и пропуск ее при тести-

ровании могут быть причинно связанными и не могут считаться независимыми в теоретико-вероятностном смысле. Поэтому главный принцип, изложенный во введении, – что при самом программировании и при контроле качества мы выполняем формализацию требований, по возможности независимо, – не выполняется. Поэтому полагаться на тестирование самими разработчиками для контроля качества невозможно. В промышленности тестированием обычно занимаются другие люди. При этом ручное тестирование составляет значительную долю общих трудозатрат проекта, и во многих случаях эту долю воспринимают как проблему и так или иначе стараются снизить [1; 2].

Одно из усовершенствований идеи ручного тестирования привело к созданию подхода «Capture and Replay», при котором компьютер записывает действия пользователя и затем может их воспроизводить. Этот подход особенно популярен при тестировании приложений с графическим и веб-интерфейсом. Построенные таким образом тесты довольно хрупки, т. е. перестают работать при незначительных изменениях в приложении. Во многих случаях даже небольшие изменения функциональности или визуального дизайна требуют полной перестройки всего набора тестов. Встречаются также попытки генерации тестов по журналам доступа реальных пользователей [3; 4]. При обсуждении данного подхода обычно не рассматривают вопрос, насколько правомерна генерация тестов не на основе требований.

Ключевым недостатком Capture and Replay является сам принцип работы записывающих действия пользователей продуктов. Человек выполняет некоторые действия осмысленно, а программа механически записывает его действия. При взаимодействии с различными элементами на странице программа запоминает какие-то отличительные свойства этих элементов, чтобы в дальнейшем найти эти же самые элементы снова. Такими свойствами могут быть, например, текстовые надписи, значение атрибута id или XPath-выражение. id – это уникальная строковая метка, привязанная разработчиками веб-приложения, а язык XPath позволяет указать на конкретный элемент в иерархии элементов, описанных на языке разметки HTML. В листинге 1 представлен пример (рис. 3).

```
<div id="header" class="panel">
    <div class="title"><a href="select_olympiad.cgi">Выбор олимпиады</a></div>
    <div class="right">
        <div class="dropdown">
            <span><span>admin@nsuts.ru</span></span>
            <div class="links">
                <a href="edit_profile.cgi">Профиль</a>
                <a href="login.cgi?logout=1">Выйти</a>
            </div>
        </div>
    </div>
</div>
```

Рис. 3. Фрагмент HTML-кода типовой страницы NSUts

Fig. 3. Fragment of Typical NSUts Page HTML-Code

Для этого примера автоматические инструменты могли бы записать id элемента “header”, при нажатии на ссылки – их текст, например, «Выйти». Поскольку далеко не каждый элемент может иметь id, а текст может повторяться по несколько раз, многие инструменты просто записывают соответствующее XPath-выражение. Ссылке «Профиль» соответствует следующее выражение: /div/div[2]/div/div/a[1].

Однако к приложениям со сложным внутренним состоянием, в которых выполнение одной и той же последовательности действий приводит к разным результатам, данный подход оказывается неприменим. Например, при создании в NSUts олимпиады в базе появляется новая запись, и пользователь в списке видит новый пункт. При записи сценария человек нажимает на такой пункт, и инструмент запомнит какое-то его свойство. Если создать новую

олимпиаду, эти свойства уже будут другими, а записанный сценарий будет обращаться к старым. С другой стороны, если у олимпиады будет такое же название, то неизвестно, старую или новую выберет инструмент. Получается, что для приложений подобного рода невозможно составить адекватный сценарий, не описывая некоторую логику приложения, чего запись действий пользователя не предполагает.

Формальная верификация

На другой стороне спектра находится формальная верификация. Формализация требований происходит в процессе их описания на специальном языке, по которому в дальнейшем можно сгенерировать код или верифицировать, что написанный код соответствует описанным требованиям. Этот подход больше соответствует предметным областям, в которых требования к функциональности могут быть относительно простыми, но требование к надежности и безопасности стоит во главе угла. Примерами таких областей могут быть авионика [5; 6], медицина [7], автомобильная промышленность [8], разработка распределенных вычислительных программ [9] и аппаратного оборудования в соответствии с индустриальными стандартами [10].

Для решения таких задач ведется разработка специальных технологий и инструментов, но эти инструменты приспособлены именно к языкам, операционным системам, протоколам и др., которые применяются именно в соответствующих областях (например, к языку Ada). В нашей же задаче требования сложны и нестабильны, и потому данный подход является неприменимым. Поэтому даже перевод проекта на технологии и языки, для которых есть средства формальной верификации, не мог бы решить стоящую перед нами проблему.

Автоматическое тестирование при помощи тестовых сценариев

С учетом изложенных ранее соображений для нашей задачи наиболее разумным подходом представляется автоматическое тестирование, при котором формализация происходит при написании программного кода тестовых сценариев. По оценкам [1; 2], начальное написание тестирующих скриптов занимает больше времени по сравнению с Capture and Replay, но дальнейшая их доработка после внесения изменений в тестируемую систему оказывается дешевле, и уже после 2–3 обновлений общие трудозатраты на поддержку Capture and Replay становятся выше.

Часто встречаются различные попытки генерации тестов: по различным графам, соответствующим приложению [11], или с помощью автоматических инструментов по самому приложению [12; 13]. Идея тестирования в принципе заключается в независимой проверке тех же требований, а генерация тестов по коду приложения сводит эту идею на нет. Кроме того, поддерживать автоматически сгенерированный код, как правило, гораздо сложнее, нежели написанный людьми. Поэтому можно предположить, что по мере развития системы также окажется, что поддержка написанных вручную скриптов – наиболее выгодный по трудозатратам вариант.

Также в рассмотренной литературе встречается описание генерации тестов по модели в виде конечного автомата [14]. В этом случае страницы описываются группами состояний, а функциональность – переходами между этими состояниями. Система автоматически обходит все возможные пути в данном автомате, проверяя, что все требования выполняются после каждого перехода.

По сути, это иной способ записать требования формально: не в виде программного кода, а в виде диаграммы состояний конечного автомата. Дискуссия о сравнительных достоинствах разработки программ в виде текстов и в виде диаграмм состояний продолжается уже много десятилетий. На взгляд авторов, ограниченный успех CASE-инструментов, ориентированных на диаграммы, свидетельствует о том, что все-таки писать и поддерживать программы в форме текста людям почему-то удобнее. Кроме того, авторы работ по данной технологии тестирования предлагают инструменты только для языка Java, но не для языков, используемых в нашем проекте.

Проектирование методики

Как было показано ранее, наиболее подходящим подходом для нашего проекта является формализация требований в виде тестирующих скриптов. Для того чтобы упростить эту формализацию, можно собрать и систематизировать имеющиеся требования в виде некоторого документа. Таким документом может быть, например, техническое задание, но в целом достаточно будет и простого, но подробного описания функциональности.

Инструменты для тестирования веб-приложений можно разбить на две категории:

- те, что «притворяются» браузерами и отправляют только HTTP-запросы;
- настоящие браузеры или их эмуляторы, исполняющие также клиентский Javascript-код.

Первые применимы для тестирования серверной части веб-приложения, но если на клиентской стороне также присутствует нетривиальная логика, которую мы хотим проверить, то нужно использовать вторые. В нашем случае даже старая версия системы реализовала ряд нетривиальных функций (например, вопросы и ответы, новости) при помощи Javascript, а новая версия системы полностью основана на генерации страниц средствами Javascript, что делает тестирование на уровне запросов HTTP полностью неприменимым.

Ко второй категории можно отнести headless-браузеры (которые не отрисовывают страницу при работе) и другие инструменты, предоставляющие API для работы с каким-то конкретным браузерным движком. Такие версии браузеров, а тем более сторонние инструменты на их основе, имеют ряд отличий (иногда недокументированных) от обычных браузеров, и их использование может привести к неожиданным результатам при тестировании.

Кроме того, headless-браузеры предоставляют API, несовместимые между собой. В NSUts заявлена поддержка Firefox и Chrome / Chromium, поэтому разработка тестов при помощи headless-браузера потребовала бы работы с двумя разными API.

Наиболее популярный инструмент, решающий эти проблемы – это Selenium [1]: он позволяет обращаться к реальным браузерам и имеет при этом общий для них всех API, что упрощает написание тестирующих скриптов.

В прошлом в нашем проекте уже предпринимались попытки разработать тестовые сценарии с использованием инструмента Selenium WebDriver, который будет подробнее рассматриваться далее. Сценарии были разработаны на языке Perl, для унификации с основным кодом системы. Как отмечено в работе [2], поддержка тестов является трудоемкой и дорогостоящей задачей, поскольку должна выполняться вручную [15]. Из-за нехватки ресурсов и некоторых специфических проблем, которые будут обсуждаться далее, поддержка этих скриптов проводилась в недостаточном объеме, поэтому они отстали от реальной системы и оказались неприменимы.

Кроме того, продолжали развиваться браузеры и Selenium, и получилось так, что библиотека, использованная для реализации скриптов в нашем проекте, была снята с поддержки, а доступные для новой версии Selenium библиотеки для Perl неофициальны и несовместимы со старыми [2]. Поскольку в новой архитектуре системы мы также отказываемся от Perl, было решено было решено писать новые тесты на Python. Предполагается, что применение методики позволит сделать процесс более системным, а тесты – более поддерживаемыми.

Особенности применения методики

Обновление архитектуры в NSUts

Проведенное командой разработчиков NSUts исследование [16] показало, что прототипы системы, использующие технологии AJAX / AJAJ, создают гораздо меньшую нагрузку на сервер, что критично при большой нагрузке. Например, во время интернет-тура Открытой Всесибирской олимпиады по программированию количество одновременно активных поль-

[1] URL: https://www.seleniumhq.org/

[2] См.: https://metacpan.org/release/Test-WWW-Selenium и https://metacpan.org/release/Selenium-Remote-Driver

зователей превосходило тысячу. При этом активность многих этих пользователей заключалась в частом (иногда раз в несколько секунд) обновлении страницы рейтинга, что приближало систему к исчерпанию мощности процессоров и ряда других параметров. Поэтому в настоящее время система находится в процессе поэтапного перехода на новую архитектуру. Системой постоянно пользуются студенты и преподаватели, и потому необходимо избежать негативных последствий при внесении изменений: нужно осуществить этот переход так, чтобы с точки зрения пользователей система продолжила работать как прежде.

Таким образом, перед нами стоит следующая задача: написать тесты так, чтобы они не только поддерживались в будущем, но и позволяли убедиться, что модули, разработанные на замену оригинальным, имеют ту же функциональность. Для этого хотелось бы составить тесты так, чтобы их можно было запускать на обеих версиях системы, т. е. тестировать именно функциональность системы, а не ее реализацию. Понятно, что тесту в любом случае придется учитывать различия из-за особенностей реализации, поэтому в ходе применения методики нужно было придумать способ, с помощью которого адаптация написанного для исходной системы теста к работе с новой версией будет требовать минимального написания нового кода.

Использование Selenium

Selenium WebDriver позволяет взаимодействовать со страницей – передавать нажатия мыши и клавиш клавиатуры элементам страницы. Для этого нужно иметь возможность указать, с каким элементом (например, ссылкой или кнопкой) надо взаимодействовать. В Selenium есть 8 способов найти элемент:

- по HTML-атрибуту id;
- по HTML-атрибуту name, который обычно имеют только различные поля ввода;
- по XPath-выражению, соответствующему элементу;
- по тексту ссылки;
- по частичному тексту ссылки;
- по имени HTML-тега;
- по классу элемента;
- по CSS-селектору.

XPath – это место элемента в структуре HTML-документа, поэтому он, по определению, существует у любого элемента страницы. Но при малейших изменениях в верстке структура документа меняется, и XPath может перестать соответствовать тому элементу, которому он соответствовал раньше. Остальные способы применимы не для всех элементов (например, не все элементы являются ссылками) или являются недостаточно точными (например, на странице может быть много элементов с одинаковым именем HTML-тега).

Поиск элементов по id – метке, которую привязывают сами разработчики, – оказывается наиболее точным и наименее хрупким из доступных способов указать конкретный элемент. Атрибут id можно оставить прежним даже после полного изменения верстки страницы. Это позволяет устранить целый класс различий между двумя реализациями системы: можно «семантически связывать» элементы, которые выполняют одну и ту же функцию в разных реализациях, просто указав для них одинаковый id.

Главный недостаток этого способа в нашей ситуации заключается в том, что при разработке старого кода не было действующих соглашений по стилю кодирования, требовавших расставлять эти метки. Поэтому большинство элементов, с которыми нужно взаимодействовать тестирующим скриптам, этих меток не имели. Внесение изменений в старый код, особенно при отсутствии тестов, является нежелательным, поскольку может привести к созданию ошибок. Однако использование id имеет значительные преимущества и вдобавок является достаточно безопасным изменением, поскольку добавление атрибута id не влияет на функциональность.

Первоначально ставилось требование, чтобы тесты проходили на старом коде системы без каких-либо изменений этого кода. Преимущества, которые давала расстановка id, были сочтены настолько существенными, что это требование было ослаблено.

Другая проблема состоит в том, что в системах типа NSUts содержимое страниц в основном состоит из списков различных сущностей: олимпиады, задачи, решения. Из-за этого на странице присутствует множество однотипных элементов, и потому в id элементов добавляется переменное значение (чаще всего, первичный ключ сущности, рис. 4):

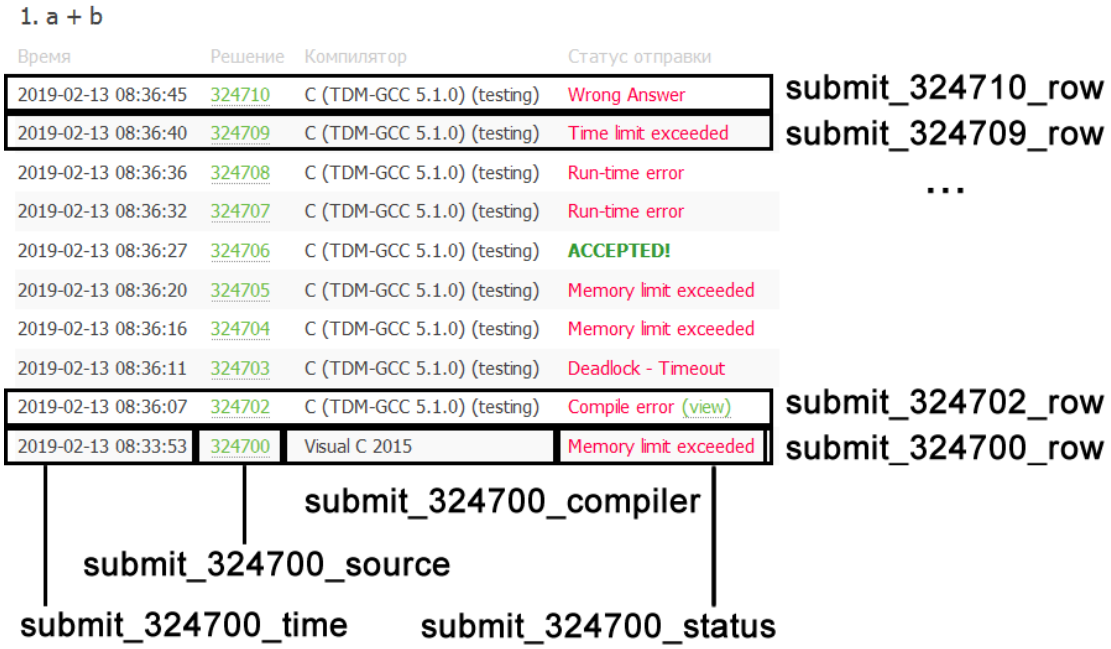

Рис. 4. id однотипных элементов
Fig. 4. Similar Elements' ids

Тестирующему скрипту нужно взаимодействовать с конкретным элементом, а для этого необходимо выяснить, какое переменное значение ему соответствует. Предлагается при создании сущностей генерировать уникальные значения – например, при отправке текста программы на проверку в системе NSUts можно добавить комментарий с текущим временем и случайно сгенерированной строкой. Затем придется перебрать все доступные сущности в поисках этого значения и таким образом вычислить искомый id.

Шаблон проектирования Page Object

Чтобы минимизировать изменения, необходимые для адаптации тестов к новым версиям старых модулей, был разработан промежуточный программный интерфейс для взаимодействия с приложением. Для разработки этого интерфейса использован шаблон проектирования, известный как Page Object. Он состоит в том, что каждой странице приложения в соответствие ставится класс (в нашем случае – класс языка Python), а доступная на странице функциональность описывается методами этого класса. Это позволяет описывать в тестовых сценариях логику взаимодействия в терминах высокоуровневого функционального описания системы, а не дублировать множество команд, например, нажатия мыши по конкретным элементам страницы. Код, использующий библиотеки для управления браузером, скрыт внутри классов – наследников описания страницы. Исследования [17] показывают, что ввиду указанных причин применение этого шаблона упрощает поддержку тестовых скриптов.

Поскольку функциональность старой и новой версий модулей должна совпадать, можно сделать классы страниц абстрактными и предоставлять для них по две реализации. Применение полиморфизма позволит передавать экземпляры классов для разных реализаций в код тестовых сценариев, написанный для работы с интерфейсом их базового класса. Код самих тестовых сценариев при этом может вообще остаться без изменений.

```
...

PRINT_PAGE_SUBMIT_TOO_LARGE = u"Превышено ограничение на размер файла!"
TEST_PRINT_LARGE_SUBMIT = "T" * (262144 + 1)

class TestPrintPage(NsutsTest):
    def setUp(self):
        NsutsTest.setUp(self)
        self.tour_id = PageAdminTour.make_test_tour(self, OLYMPIAD_ID, do_logout=False)

    def test_1_submit_on_print(self):
        ...

    def test_2_submit_large(self):
        self.page.open()
        submit_successful, message = self.page.submit_text(TEST_PRINT_LARGE_SUBMIT)
        self.assertFalse(submit_successful)
        self.assertEqual(message, PRINT_PAGE_SUBMIT_TOO_LARGE)

    def test_3_admin_mark_printed(self):
        ...

    def tearDown(self):
        PageAdminTour.remove_test_tour(self, OLYMPIAD_ID, self.tour_id)
        NsutsTest.tearDown(self)
```

Рис. 5. Фрагмент тестового сценария, использующего методы базового класса страницы
Fig. 5. Fragment of Test Script, which Uses Methods of Page's Base Class

```
...

class PrintPageFormPerl(PrintPageForm):
    def __init__(self, engine):
        PrintPageForm.__init__(self, engine)

    def input_text(self, code):
        ...

    def submit(self):
        self.engine.remember_page()
        self.engine.driver.find_element_by_xpath("//*[@type='submit']").click()
        self.engine.wait_for_page_to_change()

        # waiting for redirect to happen
        self.engine.remember_page()
        self.engine.wait_for_page_to_change()

        return self.engine.driver.find_element_by_id("print_form_message").get_attribute('innerHTML')

```

Рис. 6. Фрагмент реализации класса страницы, использующий Selenium
Fig. 6. Fragment of Page Class Implementation with Selenium Usage

```
...

class PrintPageFormReact(PrintPageForm):
    def __init__(self, engine):
        PrintPageForm.__init__(self, engine)

    def input_text(self, code):
        ...

    def submit(self):
        self.engine.driver.find_element_by_id("print_form_submit_button").click()

        # wait for ajax request
        time.sleep(2)

        return self.engine.driver.find_element_by_id("print_form_message").get_attribute('innerHTML')

```

Рис. 7. Фрагмент реализации класса страницы, использующий Selenium
Fig. 7. Fragment of Page Class Implementation with Selenium Usage

С тех пор, как систему начали переводить на новую архитектуру, прошло уже несколько лет, и за это время появились новые требования. Было решено реализовывать новую функциональность только в новых версиях модулей, чтобы способствовать переходу на них. Влияние новой функциональности нужно учитывать наравне с различиями в реализации модулей. Это делается путем написания разной реализации одних и тех же методов в соответствующих Page Object. Совпадающую функциональность можно описывать в базовом классе. Как было сказано, во многих простых случаях достаточно использовать одинаковые id элементов, чтобы устранить различия для тестирующих скриптов.

Параметризация тестов

В системе может быть множество параметров, которые влияют на функциональность отдельных ее элементов. Например, в NSUts бывают разные правила олимпиад, настройки тура, различный набор привилегий у пользователей. При этом необязательно писать разные тесты: помимо разных реализаций для страниц тест может и принимать другие параметры, и учитывать их. Это, однако, может привести к комбинаторному взрыву, при котором из-за различных комбинаций значений всех параметров получается слишком большое число возможных вариантов, и потому сильно возрастает время прохождения тестов. Были обнаружены исследования [18], в которых описаны методики сокращения числа тестов, обеспечивающие при этом приемлемый уровень кодового покрытия.

В настоящий момент наши тестовые скрипты работают с небольшим числом параметров, но в дальнейшем рассматривается возможность добавления новых и применения указанной методики для эффективного управления временем прохождения тестов.

Организация тестового стенда

Для запуска тестов и проверки написанных разработчиками новых версий страниц был подготовлен тестовый стенд. Разработчики могут запускать тесты и на собственных машинах, однако можно сэкономить их время, проделав настройку один раз. В стенде используются Selenium Server, последние версии Google Chrome и Mozilla Firefox, соответствующие им Selenium-драйвера и библиотека для работы с Selenium. Все эти компоненты периодически обновляются, и чтобы тесты работали, необходимо использовать согласованный набор компонент. На машинах разработчиков может не быть всех браузеров или установлены старые версии компонент, из-за чего тесты, которые запускаются в одном месте, перестают работать в другом. Чтобы этого избежать, можно использовать стенд, который поддерживается в актуальном состоянии одним из разработчиков.

Компоненты тестового стенда

В тестовый стенд должны входить:

- экземпляр тестируемой системы;
- одна или несколько машин, на которых запускаются браузеры и Selenium Server;
- машина, на которой запускаются тестирующие скрипты, управляющие браузерами через Selenium Server.

Все эти компоненты могут располагаться на разных компьютерах.

В нашем случае тестируемая система – NSUts – состоит из двух частей: веб-сервера, работающего на Debian, и набора тестирующих клиентов, работающих на Windows. На последних установлены компиляторы разных версий, и запущен процесс, который общается с веб-сервером NSUts, получает решения участников, компилирует и запускает их проверку на наборах входных данных, после чего сообщает веб-серверу, успешно ли решение прошло проверку. Схема работы изображена на рис. 8.

Рис. 8. Схема работы системы NSUts

Fig. 8. Scheme Illustrating NSUts Workflow

В целях верификации работы веб-сервера NSUts не обязательно разворачивать полноценные тестирующие клиенты – достаточно, чтобы сервер получил запрос на выдачу решения, а затем получил отчет о его проверке. Поэтому в наших тестирующих скриптах реализованы «поддельные» тестирующие клиенты, которые скачивают решение и выдают заранее согласованный ответ. Также это сокращает время, необходимое для тестирования веб-сервера, поскольку «поддельные» клиенты могут сообщить о результатах проверки мгновенно, а при использовании настоящих клиентов приходилось бы ожидать, пока этот процесс действительно произойдет.

Запуск браузеров на разных машинах позволяет сократить время выполнения тестов, запуская их параллельно. Пока что нам это не потребовалось, и потому для тестового стенда достаточно одной машины, на которой установлены все необходимые браузеры. Поскольку большинство пользователей NSUts пользуются ОС Microsoft Windows, необходимо запускать браузеры именно в этой ОС. Веб-сервер NSUts работает на Debian, и его можно развернуть

либо на отдельной машине, либо внутри виртуальной машины. Нами был выбран второй вариант: виртуальная машина работает на той же машине, на которой запускаются браузеры. Таким образом, тестовый стенд представлен всего одним компьютером.

Запуск тестирующих скриптов происходит на машине разработчика. Для этого ему требуется установить дистрибутив языка, на котором реализованы скрипты (в нашем случае Python) и библиотеку для работы с Selenium на этом языке. Данный подход сопряжен с серьезной проблемой: в Selenium Server не реализована аутентификация, поэтому размещение его на публичном или даже локальном в рамках сети НГУ IP-адресе привело бы к тому, что скрипты (не обязательно разработанные нами) мог бы запускать кто угодно. Для решения этой проблемы, удаленный доступ к Selenium Server осуществляется через ssh-туннель.

Рабочий цикл тестового стенда

Тестовый стенд подразумевает однократную первоначальную настройку, т. е. развертывание тестируемой системы и установку браузеров, Selenium и драйверов, а затем многократный запуск тестов, перед которым выполняется обновление тестируемой системы. По ряду причин (часть из которых описана ниже) очистка от результатов предыдущих тестов осуществляется как отдельная операция и должна активироваться вручную.

Для удобства развертывания системы как для тестирования, так и для разработки, был создан набор скриптов, упрощающий данный процесс. Пользователю нужно загрузить эти скрипты из специального репозитория, и по мере их работы указывать желаемые настройки. Скрипты сами устанавливают необходимые пакеты с помощью менеджера пакетов, создают необходимые директории и файлы (например, конфигурацию для веб-сервера apache), создают и заполняют таблицы в базе данных, загружают код веб-приложения из отдельного репозитория. Поскольку развертывание системы проводится один раз при создании стенда, необходимость пользовательского ввода считается некритическим недостатком.

«Боевая» система также не переразвертывается каждый раз как минимум в том смысле, что данные (учетные записи пользователей, олимпиады, задачи, сданные решения и т. д.) должны сохраняться, даже когда вносятся изменения в схему базы данных. Поэтому в репозиторий добавляются специальные скрипты, которые инкрементально обновляют развернутую систему, например SQL-запросы для изменения схемы БД. Тестируемая система обновляется точно так же: перед очередным запуском тестов из репозитория загружаются изменения и запускаются все новые скрипты. В отличие от набора скриптов для развертывания системы данные скрипты не требуют пользовательского ввода, поэтому процедура обновления происходит полностью автоматически.

Чтобы запустить тестирующие скрипты, разработчики получают их из отдельного репозитория и указывают в конфигурационном файле, по какому адресу доступны Selenium Server и веб-сервер NSUts. Наличие конфигурационного файла позволяет легко использовать машину с Selenium Server и браузерами для тестирования реальной системы. Схема работы стенда показана на рис. 9.

Применение методики для тестирования системы NSUts

В настоящее время по указанной методике разработаны тесты для четырех модулей системы NSUts, и два из этих модулей были успешно переведены на новую архитектуру. После минимальных изменений в коде классов страниц новые модули были протестированы. В одном случае изменения не потребовались вовсе, несмотря на значительные различия во внешнем виде пользовательского интерфейса и в структуре HTML (табл. 1). Сейчас указанные модули уже используются вместо их старых аналогов, и готовятся новые версии модулей, для которых уже написаны тесты.

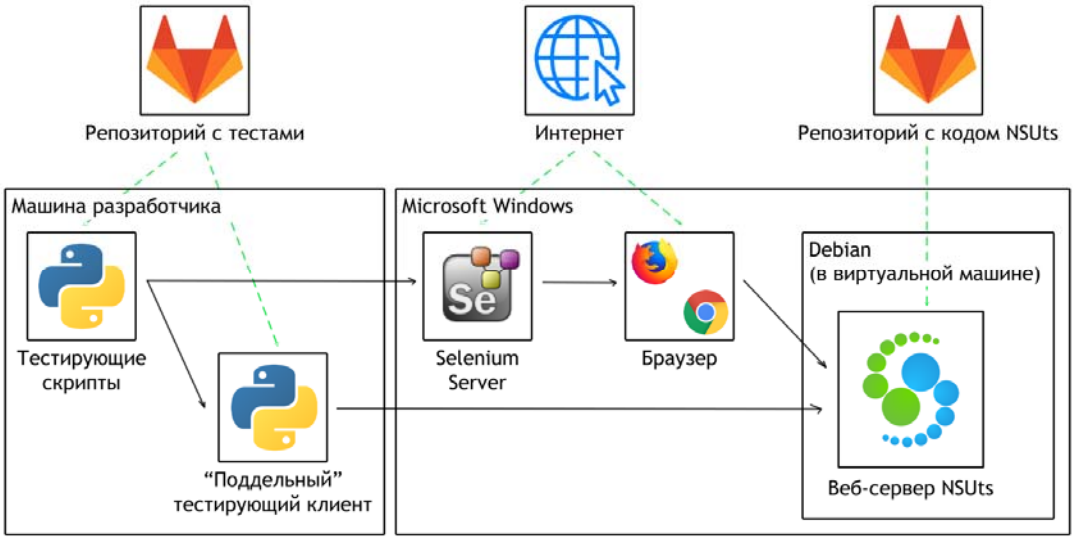

Рис. 9. Схема работы тестового стенда
Fig. 9. Test Stand Workflow

Таблица 1

Объем кода тестов для страниц системы
и изменений для адаптации тестов к новой версии системы

Table 1

Amount of Tests Code for System Pages
and Changes to Adapt Tests to Work with the New Version of the System

Страница	Число строк	
	общего кода	добавленных
Новости	399	–
Отправить	315	–
Печать	241	95
Результаты	392	0

Изменения не потребовались на странице со статической информацией (результаты отправленных на тестирование решений), т. е. новым представлением тех же данных, и нескольким доступным действиям, не ведущим к их изменениям (например, просмотр текста отправленного решения или ошибки компиляции). Здесь очень сильно пригодилось использование совпадающих значений атрибута id.

Однако на странице с возможностью отправки информации (печать текста на принтере) потребовалось сделать отдельные, хотя и очень схожие, реализации класса страницы для разных модулей. Дело в том, что страницы работают по-разному, хотя для пользователя поведение системы выглядит практически неотличимым. Так, в одном случае при отправке формы происходит несколько перенаправлений, а в другом – асинхронный запрос, и это различие необходимо отразить в коде соответствующих странице классов.

Анализ эффективности методики

Одной из самых популярных метрик, используемых для оценки качества тестов, является процент покрытия ими исходного кода. К сожалению, данную метрику сложно измерить без

специализированных инструментов. В нашем случае используется Perl в старой архитектуре и PHP + Javascript в новой, и не было обнаружено инструмента, который поддерживал бы все три языка.

Поэтому для оценки покрытия был использован метод Монте-Карло, заключающийся в следующем: в файле выбирается случайная строка, она изменяется с целью изменить логику программы, и запускаются тесты. Изменения в строку вносятся вручную, чтобы новая строка была корректна синтаксически, но некорректна семантически (например, замена условия на противоположное). Если тесты проходят успешно, то строка считается непокрытой, и наоборот. Не все строки в файле влияют на логику программы (например, комментарии, пустые строки или строки с различными скобками), и потому такие строки не учитываются. Номера строк генерируются псевдослучайным образом. Если генератор выдает номер строки, которая уже была протестирована, она все равно учитывается в выборке. По полученной выборке можно построить график, показывающий, как изменяется процент покрытия с увеличением размера выборки. То, что процент покрытия прекращает существенно изменяться, говорит о том, что этот метод действительно позволяет оценить реальный процент покрытия.

Подсчет был совершен для двух модулей, имеющих реализацию на обеих архитектурах. Далее представлены графики сходимости покрытия (рис. 10) и приведены получившиеся результаты (табл. 2).

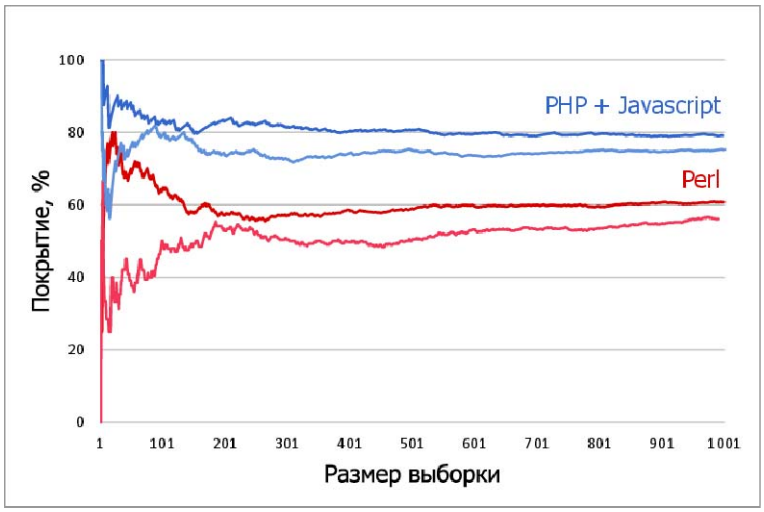

Рис. 10. Сходимость покрытия страниц «Печать» и «Результаты» (на Perl и PHP + Javascript)
*Fig. 10. C*overage Convergence for “Print” and “Report” Pages (in Perl and in PHP + Javascript)

Таблица 2

Оценка покрытия кода модулей тестами, %

Table 2

Evaluated Code Coverage, %

Страница	Perl	PHP + Javascript
Печать	60,88	79,1
Результаты	56,05	75,2

Из табл. 2 видно, что тест покрывает код на новой архитектуре даже лучше, чем на старой. Это позволяет предположить, что в старой системе существует значительная доля «мертвого» кода. Покрытие неполное, поскольку на страницах присутствуют элементы, ко-

торые не влияют на функциональность, например заголовки и другой текст, предназначенный для пользователя. Некоторые элементы просто не обладают функциональностью (например, дата отправки в таблицах на указанных страницах), и потому тесты на них написаны не были. Соответствующие им строки действительно не покрыты тестами, ведь их изменение или отсутствие не отражается на прохождении тестов.

Строки, изменение которых привело к провалу тестов, реально соответствуют тестируемой функциональности, поэтому методику можно считать применимой и, учитывая достаточно высокий процент покрытия, эффективной. При необходимости после получения оценки покрытия тесты можно доработать, и протестировать изменения в менее важных или недостающих, если таковые имеются, элементах и функциональности.

Заключение

В работе представлена методика автоматического тестирования развивающихся веб-приложений. Поскольку мы разрабатывали методику для приложения со сложными и часто меняющимися требованиями, она основывается на написании тестирующих сценариев. А так как мы разрабатывали ее для веб-приложения, она предлагает использование Selenium WebDriver. Важной особенностью методики является применение полиморфизма при разработке скриптов, позволяющее написать тест так, чтобы он мог проверять несколько разных реализаций одной и той же функциональности. Адаптация скрипта к новой реализации может вообще не потребовать написания нового кода, если приложение соответствующим образом адаптировано к тестированию. В нашем случае главным методом адаптации является расстановка HTML-тегов id у управляющих элементов интерфейса приложения.

Методика используется при организации тестирования системы NSUts, и спроектированные по ней тесты существенно упрощают задачу разработчиков, позволяя быстро и надежно убедиться, что переводимые ими на новую архитектуру модули имеют ту же функциональность, что и их старые аналоги. Показано, что оценочное покрытие кода данными тестами достаточно высоко.

Список литературы / References

1. **Leotta M.**, **Clerissi D.**, **Ricca F.**, **Tonella P.** Approaches and Tools for Automated End-to-End Web Testing. *Advances in Computers*, 2016, vol. 101, p. 193–237. DOI 10.1016/bs.adcom.2015.11.007
2. **Leotta M.**, **Clerissi D.**, **Ricca F.**, **Tonella P.** Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In: Proceedings – Working Conference on Reverse Engineering, 2013, p. 272–281. DOI 10.1109/wcre.2013.6671302
3. **Шмейлин Б. З.** Современные технологии тестирования Web приложений // Системы и средства информ. 2009. Доп. выпуск. С. 138–147.
   Shmeilin B. Z. Sovremennye tekhnologii testirovaniya Web prilozhenii [Modern technologies of web-applications testing]. *Sistemy i sredstva inform*., 2009, add. issue, p. 138–147. (in Russ.)
4. **Chen C.**, **Chen Y.**, **Miao H.**, **Wang H.** Usage-pattern based statistical web testing and reliability measurement. *Procedia Computer Science*, 2013, p. 140. DOI 10.1016/j.procs.2013.09.020
5. **Muñoz C.**, **Narkawicz A.**, **Dutle A.** From formal requirements to highly assured software for unmanned aircraft systems. *Lecture Notes in Computer Science*, 2018, vol. 10951, p. 647–652. DOI 10.1007/978-3-319-95582-7_38

6. **Denman W.**, **Zaki M. H.**, **Tahar S.**, **Rodrigues L.** Towards flight control verification using automated theorem proving. *Lecture Notes in Computer Science*, 2011, vol. 6617, p. 89–100. DOI 10.1007/978-3-642-20398-5_8
7. **Harrison M. D.**, **Freitas L.**, **Drinnan M.**, **Campos J. C.**, **Masci P.**, **di Maria C.**, **Whitaker M.** Formal techniques in the safety analysis of software components of a new dialysis machine. *Science of Computer Programming*, 2019, vol. 175, p. 17–34. DOI 10.1016/j.scico.2019.02.003
8. **Nellen J.**, **Rambow T.**, **Waez M. T. B.**, **Abraham E.**, **Katoen J. P.** Formal verification of automotive Simulink controller models: empirical technical challenges, evaluation and recommendations. *Lecture Notes in Computer Science*, 2018, vol. 10951, p. 382–398. DOI 10.1007/978-3-319-95582-7_23
9. **Khanna D.**, **Sharma S.**, **Rodríguez C.**, **Purandare R.** Dynamic symbolic verification of MPI programs systems. *Lecture Notes in Computer Science*, 2018, vol. 10951, p. 466–684. DOI 10.1007/978-3-319-95582-7_28
10. **Steiner W.**, **Dutertre B.** Automated formal verification of the TTEthernet synchronization quality. *Lecture Notes in Computer Science*, 2011, vol. 6617, p. 375–390. DOI 10.1007/978-3-642-20398-5_27
11. **Liu C.-H.**, **Kung D. C.**, **Hsia P.**, **Hsu C.-T.** An object-based data flow testing approach for web applications. *International Journal of Software Engineering and Knowledge Engineering*, 2001, vol. 11, no. 2, p. 157–179. DOI 10.1109/apaq.2000.883773
12. **Choudhary S.**, **Dincturk**, **M.E.**, **Mirtaheri**, **S.M.** Building rich internet applications models: Example of a better strategy. In: Proc. of the International Conference on Web Engineering, ICWE. Springer, 2013. DOI 10.1007/978-3-642-39200-9_25
13. **Benedikt M.**, **Freire J.**, **Godefroid P.** VeriWeb: Automatically testing dynamic web sites. In: Proc. 11th Intern. WWW Conf. May, 2002.
14. **Chen J.**, **Chovanec S.** Towards Specification-Based Web Testing. Springer, 2002, vol. 2376, p. 165–171. DOI 10.1007/3-540-45745-3_15
15. **Mirzaaghaei M.** Automatic test suite evolution. In: Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering – SIGSOFT/FSE'11, 2011, p. 396–399. DOI 10.1145/2025113.2025172
16. **Боженкова Е. Н.**, **Иртегов Д. В.**, **Колбин Я. С.** Оптимизация производительности веб-интерфейса приложения NSUts средствами динамического HTML // Вестник НГУ. Серия: Информационные технологии. 2015. Т. 13, № 2. С. 13–21.
    Bozhenkova E. N., **Irtegov D. V.**, **Kolbin Ya. S.** Optimizing performance of NSUts application web-interface via dynamic HTML. *Vestnik NSU. Series: Information Technologies*, 2015, vol. 13, no. 2, p. 13–21. (in Russ.)
17. **Leotta M.**, **Clerissi D.**, **Ricca F.**, **Spadaro C.** Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study. In: IEEE 6th Int. Conf. on Software Testing, Verification and Validation Workshops, 2013, p. 108–113. DOI 10.1109/icstw.2013.19
18. **Masuda S.**, **Matsuodani T.**, **Tsuda**, **K.** A method of creating testing pattern for pair-wise method by using knowledge of parameter values. In: Procedia Computer Science, 2013, p. 521. DOI 10.1016/j.procs.2013.09.131

Материал поступил в редколлегию
Received
04.03.2019

Сведения об авторах / Information about the Authors

Ткачев Александр Витальевич, магистрант факультета информационных технологий Новосибирского государственного университета (ул. Пирогова, 1, Новосибирск, 630090, Россия)

Alexander V. Tkachev, Master's Student, Faculty of Information Technologies, Novosibirsk State University (1 Pirogov Str., Novosibirsk, 630090, Russian Federation)

alexander@tkachov.ru
D-4763-2019
ORCID 0000-0001-7721-9527

Иртегов Дмитрий Валентинович, доцент, заведующий лабораторией, факультет информационных технологий Новосибирского государственного университета (ул. Пирогова, 1, Новосибирск, 630090, Россия)

Dmitry V. Irtegov, Associate Professor, Head of Laboratory, Faculty of Information Technologies, Novosibirsk State University (1 Pirogov Str., Novosibirsk, 630090, Russian Federation)

fat@nsu.ru
ORCID 0000-0003-1007-134X