ИССЛЕДОВАНИЕ МЕТОДОВ ТЕСТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ВЕБ-ПРИЛОЖЕНИЙ

Исак М. Email: Issak6101@scientifictext.ru

Исак Мухтар – бакалавр,
направление: информационные технологии,
Казахский национальный университет им. Аль-Фараби, г. Алматы, Республика Казахстан

Аннотация: *статья посвящена анализу дополнительных возможностей автоматизации функционального тестирования Web-приложений на основе технологии UniTesK. В ней рассматриваются существующие подходы к автоматизации функционального тестирования Web-приложений, обсуждаются их достоинства и недостатки. Кроме того, анализируются возможные варианты применения технологии UniTesK для тестирования данного класса приложений, и предлагается способ дополнительной инструментальной поддержки процесса разработки функциональных тестов.*

Ключевые слова: *целевой системой мы будем называть программную систему, которая выступает в качестве объекта тестирования. Далее в качестве синонима для термина «целевая система» мы будем также использовать словосочетание «тестируемая система».*

RESEARCH OF TESTING METHODS OF INFORMATION SYSTEMS AND WEB APPLICATIONS

Issak M.

Issak Mukhtar – Bachelor,
DIRECTION: INFORMATION TECHNOLOGY,
AL-FARABI KAZAKH NATIONAL UNIVERSITY, ALMATY, REPUBLIC OF KAZAKHSTAN

Abstract: *the article is devoted to the analysis of additional possibilities for automation of functional testing of Web-applications based on UniTesK technology. It examines existing approaches to automating functional testing of Web applications, discusses their advantages and disadvantages. In addition, possible options for using UniTesK technology for testing this class of applications are analyzed, and a method for additional instrumental support of the process of developing functional tests is proposed.*

Keywords: *we will call a target system a software system that acts in as an object of testing. Further, as a synonym for the term target system we will also use the phrase system under test.*

Введение

В настоящее время промышленное производство программного обеспечения (ПО) достигло таких масштабов и такой степени сложности, что необходимость в индустриально применимых технологиях систематического тестирования общепризнана. Особенно актуальным является создание таких технологий, которые обеспечивают одновременно качественное, систематическое тестирование целевого ПО и высокую степень автоматизации разработки тестов. Традиционные методы разработки тестов вручную уже не могут обеспечить качественное тестирование современных программных систем.

Обычно автоматизация тестирования сводится к автоматизации выполнения тестов и генерации отчетов по их результатам. Автоматизировать подготовку тестов и анализ полученных результатов труднее, поскольку при этом необходимо обращение к требованиям к ПО, соответствие которым, должно быть проверено во время тестирования. Требования же часто представлены в виде неформальных документов, а иногда - только как знания и опыт экспертов, аналитиков и проектировщиков ПО. Для того, чтобы вовлечь требования в автоматизированный процесс разработки тестов, необходимо перевести их в формальное представление, которое может быть обработано полностью автоматически. Для этой цели требования описывают в виде формальных спецификаций целевой системы, которые можно преобразовать в программы, выполняющие проверку соответствия работы целевого ПО зафиксированным в них требованиям.

Несмотря на активное развитие методов построения тестов на основе формальных спецификаций или формальных моделей в академическом сообществе, лишь немногие из них оказываются применимыми в индустрии производства ПО. Основная проблема здесь в том, что индустрии нужны не отдельные методы, а технологии, т.е. инструментально поддержанные системы методов для решения наборов связанных задач, относящихся к выделенному аспекту разработки ПО.

Данная статья представляет описание технологии UniTesK, которая была разработана в ИСП РАН на основе опыта нескольких проектов по верификации сложного промышленного ПО и нацелена на то, чтобы сделать возможным использование передовых методов тестирования в контексте индустриального производства ПО. UniTesK в первую очередь предназначена для разработки функциональных тестов на основе моделей требований к функциональности целевой системы. Проблемы построения тестов для проверки нефункцио-нальных требований выходят за рамки данной работы.

Структура статьи такова. Следующий за введением раздел содержит описание основных элементов технологии UniTesK, начиная с общего обзора ее базовых принципов и дальше раскрывая некоторые из них в деталях. В третьем разделе проводится сравнение UniTesK с другими подходами к разработке тестов на основе моделей. В четвертом разделе кратко описываются примеры приложений UniTesK и опыт использования этой технологии для тестирования промышленного ПО. В заключении рассматриваются направления дальнейшего развития этой технологии.

1. Существующие методы функционального тестирования веб-приложений.

Самым распространенным является подход, называемый Capture & Playback (другие названия – Record & Playback, Capture & Replay). Суть этого подхода заключается в том, что сценарии тестирования создаются на основе работы пользователя с тестируемым приложением. Инструмент перехватывает и записывает действия пользователя, результат каждого действия также запоминается и служит эталоном для последующих проверок. При этом в большинстве инструментов, реализующих этот подход, воздействия (например, нажатие кнопки мыши) связываются не с координатами текущего положения мыши, а с объектами HTML-интерфейса (кнопки, поля ввода и т.д.), на которые происходит воздействие, и их атрибутами. При тестировании инструмент автоматически воспроизводит ранее записанные действия и сравнивает их результаты с эталонными, точность сравнения может настраиваться. Можно также добавлять дополнительные проверки – задавать условия на свойства объектов (цвет, расположение, размер и т.д.) или на функциональность приложения (содержимое сообщения и т.д.). Все коммерческие инструменты тестирования, основанные на этом подходе, хранят записанные действия и ожидаемый результат в некотором внутреннем представлении, доступ к которому можно получить, используя или распространенный язык програм80 мирования (Java в Solex), или собственный язык инструмента (4Test в SilkTest от Segue, SQABasic в Rational Robot от IBM, TSL в WinRunner от Mercury). Кроме элементов интерфейса, инструменты могут оперировать HTTP-запросами, последовательность которых также может записываться при работе пользователя, а затем модифицироваться и воспроизводиться. Основное достоинство этого подхода – простота освоения.

Создавать тесты с помощью инструментов, реализующих данный подход, могут даже пользователи, не имеющие навыков программирования. Вместе с тем, у подхода имеется ряд существенных недостатков. Для разработки тестов не предоставляется никакой автоматизации; фактически, инструмент записывает процесс ручного тестирования. Если в процессе записи теста обнаружена ошибка, то в большинстве случаев создать тест для последующего использования невозможно, пока ошибка не будет исправлена (инструмент должен запомнить правильный результат для проверки). При изменении тестируемого приложения набор тестов трудно поддерживать в актуальном состоянии, так как тесты для изменившихся частей приложения приходится записывать заново. Этот подход лучше всего использовать для создания прототипа теста, который впоследствии может служить основой для ручной доработки. Одна из возможных доработок – параметризация теста для проверки тестируемого приложения на различных данных. Этот подход называется тестированием, управляемым данными Data Driven.

Основное ограничение – перебираемые данные не должны изменять поведение тестируемого приложения, поскольку проверки, записанные в тестовом сценарии, не подразумевают какой-либо анализ входных данных, т.е. для каждого варианта поведения нужно создавать свой сценарий тестирования со своим набором данных. Некоторые инструменты, реализующие Capture & Playback, предоставляют возможность по перебору данных; кроме того, над большинством распространенных инструментов существуют надстройки. Описанные подходы основываются на построении тестов с использованием тестируемого приложения. В подходе Keyword Driven. предпринимается попытка сделать процесс создания тестов независимым от реализации. Суть подхода заключается в том, что действия, выполняемые в ходе тестирования, описываются в виде последовательности ключевых слов из специального словаря («нажать», «ввести», «проверить» и т.д.). Специальный компонент тестовой системы переводит эти слова в воздействия на элементы интерфейса тестируемого приложения. Таким образом, никакого программирования для создания тестов не нужно. Единственное, что нужно менять при изменении интерфейса, – это компонент, который отвечает за перевод слов из «словаря» в последовательность воздействий на приложение. Комплект тестов может 81 разрабатываться пользователями, не владеющими навыками программирования, однако для поддержания комплекта в рабочем состоянии программирование все-таки необходимо. В качестве примера инструмента, поддерживающего такой подход к разработке тестов, можно привести Certify от WorkSoft, в котором поддерживается библиотека функций для работы с каждым компонентом интерфейса (окна, гиперссылки, поля ввода и т.д.) и предоставляется язык воздействий на эти элементы (InputText, VerifyValue и VerifyProperty). Основные преимущества этого подхода заключаются в том, что он позволяет создавать тесты, не дожидаясь окончания разработки приложения, руководствуясь требованиями и дизайном интерфейса. Созданные тесты можно использовать как для автоматического выполнения, так и для ручного тестирования. Основной недостаток этого подхода – отсутствие автоматизации процесса разработки тестов. В частности, все тестовые последовательности разрабатываются вручную, что приводит к проблемам, как на стадии разработки, так и на стадии сопровождения тестового набора. Эти проблемы особенно остро проявляются при тестировании Web-приложений со сложным интерфейсом.

1. Технология UniTesK

Большинство проблем, присущих рассмотренным подходам разработки тестов, решены в технологии UniTesK, разработанной в Институте системного программирования РАН. Технология хорошо себя зарекомендовала при функциональном тестировании разнообразных систем (ядро операционной системы, стеки протоколов,

компиляторы). Опыт применения технологии для тестирования Web-приложений показал, что UniTesK может служить хорошей базой для тестирования такого класса приложений. В этом разделе мы остановимся на основных моментах технологии UniTesK, в последующих разделах рассмотрим особенности применения технологии для тестирования Webприложений.

Технология UniTesK – это технология разработки функциональных тестов на основе моделей, которые используются для оценки корректности поведения целевой системы3 и автоматической генерации последовательностей воздействий, далее называемых тестовыми последовательностями. Результат воздействия (реакция системы) представляется выходными параметрами, значения которых могут зависеть от истории взаимодействий целевой системы с окружением. Информация об истории моделируется внутренним состоянием целевой системы. Внутреннее состояние влияет на выходные параметры интерфейсных функций

и может изменяться в результате их работы.

Следует заметить, что в рамках данной статьи для тестирования Webприложений рассматривается представление, в котором воздействия на целевую систему и получение ее реакции на это воздействие (выходные параметры интерфейсной функции) рассматриваются как атомарное действие. Под атомарностью действия понимается, что следующее воздействие можно произвести только после получения реакции на предыдущее. Технология UniTesK также позволяет представлять целевую систему и как систему с отложенными реакциями, т.е. как систему, разрешающую воздействие до получения всех реакций на предыдущее.

Корректность поведения целевой системы оценивается с точки зрения его соответствия поведению некоторой «эталонной» модели, называемой спецификацией. В технологии UniTesK эталонная модель описывается неявно в виде требований к поведению каждой интерфейсной функции. При задании эталонной модели можно описывать функции и их параметры в достаточно обобщенном виде, отвлекаясь от несущественных подробностей. Основными компонентами тестовой системы являются итератор тестовых воздействий, оракул и медиатор. Задачей итератора тестовых воздействий, работающего под управлением обходчика, является построение тестовой последовательности, обеспечивающей заранее определенный критерий тестового покрытия. Задачей оракула является оценка корректности поведения целевой системы. Задача медиатора – преобразовывать тестовое воздействие в последовательность реальных воздействий на целевую систему и на основании доступной информации построить новое модельное состояние целевой системы после вызова. В качестве языка описания компонентов тестовой системы используются спецификационные расширения обычных языков программирования, таких как C# и Java. В этих расширениях реализованы три вида специальных классов, предназначенных для описания компонентов тестовой системы. Из спецификационных классов генерируются оракулы, из медиаторных – медиаторы, а из сценарных – итераторы тестовых воздействий. В спецификационных классах описываются спецификационные методы, каждый из которых соответствует некоторой интерфейсной функции и содержит формальное описание требований к поведению целевой системы при взаимодействии с ней через данную интерфейсную функцию. Сценарные классы предназначены для описания тестовых сценариев, содержащих описание единичных воздействий и правил итерации их параметров. Медиаторы генерируются на основе медиаторных классов, которые связывают интерфейсные функции с воздействиями на целевую систему.

Основной шаг работы тестовой системы устроен следующим образом. Обходчик выбирает очередное сценарное воздействие. Сценарное воздействие содержит несколько обращений к целевой системе, представляющих собой вызов интерфейсной функции с определенным набором значений входных параметров. Вызов интерфейсной функции передается оракулу, который, в свою очередь, передает его медиатору. Медиатор преобразует вызов интерфейсной функции в последовательность действий над тестируемой системой, получает результат этих действий от тестируемой системы и преобразует его в значения выходных параметров интерфейсной функции. Медиатор также синхронизирует модель состояния тестируемой системы, используемую оракулом для оценки корректности поведения, с ее реальным состоянием.

В результате анализа функциональности необходимо определить интерфейс тестируемой системы. Для этого требуется выделить функции, предоставляемые системой, и для каждой такой функции определить, что выступает в качестве ее входных и выходных параметров. На этапе формализации требований для каждой интерфейсной функции, выявленной на предыдущем шаге, необходимо описать ограничения на значения выходных параметров в зависимости от значений входных параметров и истории предыдущих взаимодействий с тестируемой системой. Для этого в технологии UniTesK используется широко известный подход программных контрактов. В основе этого подхода лежат инварианты данных, а также предусловия и постусловия интерфейсных операций. При связывании требований с реализацией необходимо описать, как каждая интерфейсная функция отображается на реализацию тестируемой системы. В рамках этого отображения требуется установить правила преобразования вызовов интерфейсных функций в последовательность действий над тестируемой системой, а также правила построения модели состояния тестируемой системы. Для систем с прикладным программным интерфейсом, когда взаимодействие через интерфейсную функцию соответствует вызову функции тестируемой системы, установление такого отображения может быть автоматизировано при помощи интерактивных шаблонов, предоставляемых инструментами семейства UniTesK.

2. Применение UniTesK для тестирования Web-приложений

Технология UniTesK применялась для тестирования Web-приложений в нескольких проектах. В ходе разработки тестов выяснилось, что большая часть усилий тратится на создание медиаторов, которые переводят вызов интерфейсных функций в последовательность воздействий на Web-приложение. Анализ опыта показал, что большая часть этой работы может быть автоматизирована, опираясь на стандартизированную архитектуру

пользовательского интерфейса Web-приложений. В принципе, эту особенность Web-приложений можно было бы использовать для автоматизации других шагов технологии UniTesK. В этом разделе будут рассмотрены варианты моделирования поведения Web-приложения в контексте возможной автоматизации шагов технологии UniTesK. Моделирование определяется способом выделения интерфейсных функций и способом построения модели состояния Web-приложения. Первый вариант основывается на стандартном протоколе HTTP, который служит для взаимодействия между Web-браузером и Web-приложением. Поведение Webприложения рассматривается на уровне HTTP, и этот уровень считается единственно возможным для обращения к Web-приложению. Во втором варианте за основу берется формальное описание интерфейса в виде HTML, которое используется Web-браузером для организации взаимодействия с пользователем. В этом варианте взаимодействие с Web-приложением происходит только посредством Web-браузера. И, наконец, в третьем варианте поведение Web-приложения моделируется без привязки к конкретному способу обращения, основываясь лишь на тестируемой функциональности.

3.1. Моделирование поведения на уровне HTTP

В первом варианте модель Web-приложения представляется одной интерфейсной функцией, описывающей HTTP-запрос к Web-приложению. Параметры этой функции – это параметры запроса (например, тип запроса (GET или POST), адрес (URL), параметры заголовка и т.д.) и список данных, которые передаются Web-приложению. Выходные параметры функции формируются на основе HTTP-ответа, пришедшего от Web-приложения. Требования к функциональности формулируются в виде набора ограничений на выходные параметры в зависимости от значений входных параметров и модели состояния Web-приложения. Требования в большинстве случаев сильно различаются в зависимости от URL, поэтому спецификацию интерфейсной функции можно разделить на независимые спецификации нескольких интерфейсных функций, каждая из которых описывает поведение, характерное для конкретного семейства значений параметра, определяющего URL. Однако для больших Web-приложений такое описание требований получается громоздким и плохо структурируемым. Каждая интерфейсная функция соответствует определенному запросу с некоторыми параметрами; в процессе работы тестовой системы вызов интерфейсной функции преобразуется в посылку соответствующего HTTP-запроса серверу. HTTP-запрос строится на основе формальных правил преобразования параметров, поэтому шаг технологии UniTesK, на котором происходит связывание требований с реализацией, полностью автоматизируется. При создании тестовых сценариев для этого варианта наибольшую трудность представляет организация перебора параметров выделенных интерфейсных функций. Для каждого конкретного случая можно найти наиболее подходящий способ перебора параметров, однако это требует от тестировщика определенной квалификации и опыта. Следует отметить, что этот вариант позволяет тестировать Web-приложение на устойчивость к некорректным HTTP-запросам, так как можно имитировать ситуации, которые не должны появиться в процессе нормальной работы с Web-приложением посредством браузера.

3.2. Моделирование на уровне веб-браузера

Во втором варианте в качестве интерфейса системы рассматривается интерфейс, предоставляемый Web-браузером. В этом варианте интерфейсным функциям соответствуют воздействия на активные элементы интерфейса, в результате которых происходит обращение к Web-приложению. Такими элементами можно считать гиперссылки, кнопки форм и элементы интерфейса, для которых определена обработка на клиентской стороне, приводящая к обращению к серверу. Входные параметры этих функций – это данные, которые может вводить пользователь, например, при заполнении полей ввода или выборе значений из выпадающих списков и т.д. Таким образом, если 87 рассматривать HTML-форму, то нажатию на кнопку, по которой отправляются данные, будет соответствовать интерфейсная функция, параметрами которой являются значения полей этой формы. Стоит оговориться, что в этом варианте в качестве тестируемой системы рассматривается Web-приложение в целом, включая как серверную, так и клиентскую часть. Однако внимание акцентируется на тестировании серверной части, и не ставится задача покрытия функциональности клиентской части. В этом варианте считается, что выходные параметры интерфейсных функций отсутствуют, поскольку результат воздействия описывается как изменение состояния Web-приложения. Состояние Web-приложения в этом варианте разбивается на состояние интерфейса, отображаемого Web-браузером, и состояние сервера. К состоянию интерфейса можно отнести текущую отображаемую страницу и состояние элементов на ней. К состоянию сервера относится, например, состояние базы данных, с которой работает Web-приложение, или данные, описывающие сеанс работы пользователя. Интерфейсные функции доступны не во всех состояниях, так как не во всех состояниях пользовательского интерфейса присутствуют элементы интерфейса, воздействия на которые соответствуют этим интерфейсным функциям. Например, некоторые элементы интерфейса становятся доступны только после авторизации, HTML-формы с полями и кнопками располагаются не на всех страницах Web-приложения. Условия доступности описываются в предусловии и определяются состоянием Web-приложения. Требования к функциональности описываются в постусловии в виде ограничений на состояние, в которое переходит Web-приложение в результате воздействия, описываемого интерфейсной функцией. Часто одному URL соответствуют пользовательские интерфейсы, содержащие одни и те же наборы интерфейсных функций. В таких случаях эти наборы удобно объединять в группы функций и специфицировать их как методы одного спецификационного класса. В других случаях одни и те же функции могут присутствовать сразу на нескольких интерфейсах, соответствующих разным URL. В этом случае интерфейсные функции удобно объединять в группы в зависимости от функционального назначения и специфицировать отдельно. Это позволяет получить хорошо структурированные спецификации, в которых дублирование описания функциональности сведено к минимуму. По

сравнению с первым, этот вариант позволяет уделить большее внимание описанию именно функциональности Web-приложения, абстрагируясь от деталей обработки HTTP-запросов и ответов, что существенно упрощает моделирование работы пользовательских интерфейсов, обладающих сложным динамическим поведением.

3. Дополнительная инструментальная поддержка

Основной задачей, возлагаемой на инструментальную поддержку, является упрощение работы пользователя по созданию компонентов тестовой системы. Это достигается за счет дополнительной автоматизации шагов технологического процесса UniTesK с учетом специфики Web-приложений. Первый шаг технологического процесса UniTesK – анализ функциональности тестируемой системы – не предполагает инструментальной поддержки, однако для Web-приложений можно предложить способ выделения интерфейсных функций на основе автоматизированного анализа интерфейса Webприложения. На шаге формализации требований пользователь может описывать требования в виде условий на различные атрибуты элементов интерфейса; эти условия могут строиться с использованием поддержки инструмента. Информации, собранной при автоматизации первого и второго шагов, оказывается достаточно для автоматического связывания интерфейсных функций с Web-приложением. Для шага разработки тестовых сценариев предлагаются дополнительные возможности по описанию его компонентов в терминах интерфейса Web-приложения. Последний шаг не требует дополнительной автоматизации, так как все инструменты семейства UniTesK уже предоставляют развитые средства выполнения тестов и анализа их результатов.

При использовании дополнительной инструментальной поддержки процесс разработки тестов для функционального тестирования Web-приложений изменяется, и состоит из следующих шагов:

1) создание модели Web-приложения;

2) создание тестового сценария;

3) выполнение тестов и анализ результатов.

Первый шаг – создание модели Web-приложения – включает в себя определение интерфейсных функций, описание требований к ним и их связывание с Web-приложением, т.е. объединяет первые три шага технологии UniTesK. Основная задача этого шага – формализация требований к интерфейсным функциям – в отличие от второго шага технологии UniTesK может быть частично автоматизирована, а выделение интерфейсных функций и их связывание с Web-приложением происходит автоматически. Два последних шага соответствуют двум последним шагам технологии UniTesK и отличаются только уровнем автоматизации. На первом шаге должно быть получено описание модели, состоящее из набора интерфейсных функций и описания требований к ним. На втором шаге нужно получить описание тестов для Web-приложения. При создании тестов используется подход, предлагаемый технологией UniTesK. Согласно этому подходу тесты описываются в виде тестовых сценариев, в основе которых лежит алгоритм обхода графа переходов конечного автомата. По сравнению с базовым подходом UniTesK описанный подход обладает следующими преимуществами. Во-первых, уменьшается объем ручного труда за счет автоматизации действий, предписываемых технологией UniTesK. Во-вторых, снижаются требования к квалификации пользователей технологии, так как в этом подходе основным языком взаимодействия с пользователем является не язык программирования (или его расширение), а язык элементов интерфейса и воздействий на них.

Список литературы / References

1. *Raggett D., Le Hors A., Jacobs I.* HTML 4.0 Specification. [Электронный ресурс]. Режим доступа:: http://www.w3.org/TR/html40/ (дата обращения: 20.11.2020).
2. [Электронный ресурс]. Режим доступа: http://solex.sourceforge.net/ (дата обращения: 20.11.2020).
3. [Электронный ресурс]. Режим доступа: http://www.segue.com/ (дата обращения: 20.11.2020).
4. [Электронный ресурс]. Режим доступа: http://www-306.ibm.com/software/awdtools/tester/robot/ (дата обращения: 20.11.2020).
5. [Электронный ресурс]. Режим доступа: http://www.mercury.com/us/products/quality-center/functional-testing/winrunner/ (дата обращения: 20.11.2020).
6. *Shakil Ahmad.* Advance Data Driven Techniques. [Электронный ресурс]. Режим доступа: https://www.stickyminds.com/sites/default/files/article/file/2012/XDD2811filelistfilename1_0.pdf/ (дата обращения: 20.11.2020).
7. *Zambelich Keith.* Totally Data-Driven Automated Testing. [Электронный ресурс]. Режим доступа: http://www.sqa-test.com/w_paper1.html/ (дата обращения: 20.11.2020).