# Автоматизация тестирования web-приложений, основанных на скриптовых языках

*Д. В. Силаков*
*silakov@ispras.ru*

**Аннотация.** Статья рассказывает о подходе к тестированию Web-приложений, основанных на скриптовых языках, позволяющем за короткое время создать достаточно качественный набор тестов. Описываемый подход основан на анализе исходного кода (т.е. относится к методам тестирования на основе «белого ящика») и использует некоторые особенности кода приложений, предназначенных для работы в Web. Рассматриваются возможные процедуры вынесения вердикта об успешности прохождения теста, не требующие вмешательства человека. Также описано применение предложенной методики к тестированию LSB Навигатора — Web-инструмента для просмотра и анализа содержимого спецификационной базы данных стандарта Linux Standard Base, разрабатываемого в ИСП РАН в рамках совместного проекта с Linux Foundation.

## 1. Введение

В настоящее время все большее распространение получают приложения, взаимодействующие с пользователем посредством Web-интерфейса. Такие приложения не занимаются непосредственно отображением пользовательского интерфейса, а перекладывают эту задачу на посредника — как правило, в этой роли выступает Web-браузер, получающий от приложения документы в формате HTML [1], отображающий их в своем окне, и передающий приложению информацию о различных действиях, совершаемых пользователем. В зависимости от этих действий приложение создает и передает Web-браузеру новую страницу HTML, и работа продолжается дальше.

Страницы HTML, как правило, генерируются приложением в процессе работы; при этом могут использоваться различные шаблоны, задающие стиль и структуру документа, в то время как содержательная часть создается динамически. При наполнении страницы может использоваться некоторое хранилище данных (в роли которого, как правило, выступает база данных). Такой подход широко распространен в системах управления информацией (Content Management Systems, CMS), различных интернет-форумах, и т.п.



Отделение данных от остальной инфраструктуры приложения обеспечивает возможность удобно и быстро изменять содержимое интернет-порталов и сайтов; в то же время изменение оформления или структуры страниц не требует какой-либо работы с данными.

Для написания программ, генерирующих страницы HTML с использованием данных из внешнего хранилища, широко используются скриптовые языки программирования [2], например, Perl и PHP. Такие языки являются интерпретируемыми, что позволяет программистам не беспокоиться о том, на какой программно-аппаратной платформе будет работать приложение (естественно, при условии, что для данной платформы существует интерпретатор соответствующего языка).

Как и для любого программного обеспечения, для Web-приложений важным является вопрос обеспечения их качества — программы должны выдавать те страницы, которые ожидает пользователь. Таким образом, необходимо решать задачу функционального тестирования таких программ (других важных аспектов проверки качества Web-приложений, например, нагрузочного тестирования, мы в этой статье касаться не будем). Кроме того, большинство Web-приложений постоянно развиваются и модифицируются, поэтому важным является наличие регрессионных тестов, позволяющих удостовериться, что в результате внесения изменений функциональность приложения не нарушается.

Непосредственное тестирование Web-приложения человеком (заключающееся, фактически, в переходе по различным ссылкам внутри приложения и анализа отображаемых страниц) отнимает много времени и в случае больших приложений малоэффективно. Под «большими» здесь стоит понимать программы, не только содержащие много строк кода, но и обладающие большим количеством входных параметров, работающие с базами данных сложной структуры и с большим количеством записей. При разработке таких программных продуктов вопрос об автоматизации процесса их тестирования стоит особенно остро.

## 2. Существующие подходы к тестированию Web-приложений

Большинство из существующих подходов к тестированию Web-приложений являются тестированием по принципу "черного ящика". Все, чем обладает тестировщик — это само приложение, с которым можно взаимодействовать, например, через Web-браузер, и список требований, которым приложение должно удовлетворять. Тестирование заключается в переходе по различным ссылкам внутри приложения и анализе получаемых страниц. Существуют инструменты для автоматизации этих процессов, однако ведущая роль в разработке тестов все-таки отводится человеку.

Существует много решений, позволяющих записывать сценарии поведения пользователя (т.е. цепочку ссылок, по которым осуществлялся переход) — IBM Rational Robot [3], HP WinRunner [4], Empirix e-TEST [5] и другие. Записанный единожды сценарий может далее воспроизводиться автоматически. Однако создание сценариев — трудоемкое занятие, причем отдельной задачей является анализ требований к приложению с целью определить, какие именно сценарии должны быть созданы для обеспечения хорошего качества тестирования. Некоторые инструменты (например, компонент PureAgent в системе PureTest [6]) позволяют создавать сценарии на основе действий реальных пользователей, работающих с приложением. Однако и при таком подходе при достаточно большом количестве пользователей встает вопрос о выборе из множества возможных сценариев относительно небольшого набора, который, тем не менее, обеспечит хорошее качество тестирования.

Существуют инструменты, позволяющие автоматически генерировать ссылки для обращения к приложению по протоколу HTTP, получать соответствующие страницы и производить их анализ. Однако при генерации ссылок также обычно применяется подход «черного ящика» — исходный код приложений не анализируется, а ссылки, которые необходимо генерировать, должен описать тестировщик с использованием специфических для каждого инструмента средств. Например, eValid [7] позволяет перебирать значения параметров для скриптов, которые будут подставляться в создаваемые ссылки, но список параметров с возможными значениями для каждого скрипта должен составлять тестировщик. При достаточно большом количестве скриптов и параметров создание такого списка потребует много времени; кроме того, список должен постоянно поддерживаться в актуальном состоянии — тестировщик должен следить за изменением состава скриптов и их параметров. Инструмент Puffin [8] позволяет генерировать для параметров произвольные значения, однако такой подход во многих случаях сильно снижает качество тестирования по сравнению с ручным заданием значений. В случае Puffin, опять же, список имен параметров должен составлять и поддерживать тестировщик. Кроме того, существующие инструменты не предоставляют удобных средств для перебора различных комбинаций параметров, а во многих случаях интерес представляет именно перебор комбинаций, поскольку разные сочетания параметров могут задействовать различные части приложения.

Многие инструменты анализируют получаемые в процессе тестирования страницы, извлекая из них ссылки на другие части приложения и имитируя переход по ним (опять же, с дальнейшим анализом получаемых страниц; например, для таких целей предназначен компонент 'Web Crawler', входящий в PureTest). Таким образом, осуществляется переход по всем ссылкам, которые могут быть достигнуты, начиная с определенной страницы. Поскольку все части приложения, как правило, взаимосвязаны, то в идеале, начав с некоторой стартовой страницы и посетив все достижимые из нее ссылки,



можно протестировать всю функциональность приложения, доступную пользователю. Однако число ссылок может быть чрезвычайно велико, и лавинообразно расти с увеличением числа посещенных страниц. К сожалению, современные средства перехода по ссылкам достаточно примитивны и просто осуществляют переходы по всем встреченным ссылкам (ввиду чего они часто используются при нагрузочном тестировании [9]). Настройка либо доработка инструментов для более «интеллектуального» выбора ссылок, по которым надо осуществлять переходы, требует тщательного анализа самого приложения.

Проверка правильности может производиться путем сравнения получаемых страниц с эталонными. Такие проверки полезны при регрессионном тестировании, однако если что-то изменилось в структуре страниц с момента создания эталонов, то эталоны должны быть созданы заново, а проверка корректности страницы — произведена вручную.

Кроме того, во многих случаях стоит цель тестирования не только части приложения, занимающейся формированием страниц HTML, но всего программного комплекса, неотъемлемой частью которого является постоянно изменяющаяся база данных. Сравнение страниц целиком в таких случаях не оправдано. Например, в случае приложения, формирующего страницу с ежечасно обновляемыми новостями, документы, выдаваемые по одному и тому же запросу в различные моменты времени, с большой вероятностью будут отличаться. Можно поддерживать еще и заранее заданный набор данных, на котором проводится тестирование, однако это не избавляет от проблем в случае изменения структуры или оформления страниц.

В качестве альтернативы предлагаются менее строгие проверки; так, уже упоминавшийся eValid позволяет производить более 20 сравнений, среди которых можно отметить следующие:

- 'URL' — проверка ссылки, на которой оказался пользователь после совершения определенных действий, записанных в сценарии;
- 'Title' — проверка названия страницы;
- 'Elements' — проверка числа элементов в DOM-модели страницы;
- 'Byte Size' — проверка размера страницы;
- 'Last Modified Date' — проверка даты последнего изменения страницы;
- 'Checksum' — проверка контрольной суммы для текста страницы;
- 'Text' — проверка выделенных участков текста страницы;
- 'Screen Rectangle' — сравнение изображения определенного участка страницы с тем, что наблюдал тестировщик во время записи сценария.

Остальные проверки являются вариациями проверки части страницы как изображения либо являются комбинациями указанных проверок.

Можно отметить, что несмотря на достаточно большое количество доступных проверок, ни одна из них не пригодна в случае, когда изменяется

содержательная часть страницы — часть проверок (такие, как 'Title' и 'URL') могут вообще не зависеть от этой составляющей документа, а другие ('Byte Size', 'Checksum', 'Screen Rectangle') с большой вероятностью сообщат об ошибке (т.е. об отличии полученного на новых данных результата от эталонного), но такие сообщения скорее всего не будут свидетельствовать о реальном нарушении функциональности приложения.

Существуют и более сложные подходы к разработке тестов для Web-приложений, позволяющие абстрагироваться от различных аспектов, связанных непосредственно с языком разметки HTML, и генерировать тесты на основе формальной модели данных, передаваемых приложению — см., например, [10]. Однако такие подходы трудоемки и требуют достаточно высокой квалификации тестировщиков.

В то же время исходный код Web-приложений, основанных на скриптовых языках, обладает рядом особенностей, позволяющих на основе его анализа автоматически генерировать ссылки для тестирования, что и будет продемонстрировано в данной статье. При этом соответствие кода некоторым условиям существенно повышает качество создаваемых тестов, что может быть учтено при разработке приложения. Также будут рассмотрены методы и инструменты, которые можно применять в процессе анализа генерируемых страниц для вынесения вердикта об успешности тестирования. Основная цель предлагаемого подхода — быстрое создание тестового набора, покрывающего достаточно большую часть функциональности приложения, не требующего больших затрат по поддержанию тестов в актуальном состоянии, но в то же время способного выявлять достаточно большой спектр ошибок.

## *3. Использование исходного кода для генерации тестов*

### 3.1. Извлечение имен параметров и их значений

Как уже было отмечено ранее, при работе с Web-приложением пользователь непосредственно взаимодействует с посредником в виде Web-браузера. Браузер, в свою очередь, взаимодействует с Web-сервером, на котором работает приложение, по протоколу HTTP (конечно, возможно использование других посредников и других протоколов, однако они применяются достаточно редко, и здесь мы их рассматривать не будем). Для передачи данных Web-браузера серверу протоколом HTTP [11] предусмотрено несколько методов передачи параметров, из которых в большинстве Web-приложений используются два — GET и POST. Параметры GET — это параметры, передаваемые непосредственно в адресной строке Web-браузера. Параметры POST передаются вместе с пакетами данных (и используются, как правило, либо для передачи больших объемов данных, поскольку не имеют ограничений на размер, либо чтобы не загромождать адресную строку браузера).



Чтобы протестировать приложение, необходимо знать, какие имена параметров оно ожидает увидеть в запросе и какие значения должны принимать эти параметры, чтобы выполнилась та или иная часть программы. Для выполнения этих задач необходим анализ потока данных в приложении. В общем случае эта задача нетривиальна и требует использования методов статического анализа кода; однако для каждого конкретного приложения с большой вероятностью такой анализ может быть достаточно простым. Тем более что при создании тестов не обязательно задаваться целью проанализировать все возможные варианты поведения приложения в зависимости от входных данных — следует исходить из желаемого соотношения качества тестов ко времени и ресурсам, необходимым для их разработки. Сложность создания инструмента для анализа потока данных зависит от структуры исходного кода приложения. Как будет показано ниже, во многих случаях за короткое время можно создать инструмент, производящий достаточно подробный разбор потока данных.

Каждый скриптовый язык, рассчитанный на применение в Web-приложениях, предоставляет программистам удобные и унифицированные способы доступа к параметрам запроса. Так, в языке PHP доступ к параметрам, переданным скрипту методами POST и GET, обычно осуществляется через ассоциативные массивы $_POST и $_GET соответственно (или с помощью их устаревших аналогов, $HTTP_POST_VARS и $HTTP_GET_VARS), либо с использованием массива $_REQUEST, который в дополнение к таким переменным содержит пользовательские данные, передаваемые Web-браузером — так называемые cookie. Ключами в этих массивах являются имена параметров; в ответ на обращение по ключу возвращается значение соответствующего параметра. Аналогичный подход с использованием ассоциативных массивов применяется в Perl.

Таким образом, в случае PHP для определения имен параметров, воспринимаемых скриптом, достаточно выделить из текста скрипта использования массивов $_REQUEST, $_POST и $_GET, и проанализировать, по каким ключам производится выборка из этих массивов. В простейшем и самом распространенном случае ключ, по которому производится выбор элемента — это и есть имя параметра. Процесс извлечения таких имен из исходного кода легко автоматизируется. Не стоит при этом забывать, что скрипт может подключать другие файлы; анализ кода подключаемых файлов также может быть востребован.

Естественно, обращение к массивам может производиться по ключу, формирование которого производится в процессе работы программы. Сложность определения такого формирования ключей зависит от приложения; стоит или нет пытаться выявить возможные значения ключа в этом случае и насколько глубоким должен быть анализ — решать тестировщикам, исходя из предполагаемой трудоемкости работы и потенциальных выгод, которые она может принести.

Однако знание одних лишь имен параметров может и не сильно облегчить задачу создания качественных тестов; к тому же во многих случаях, даже для больших приложений, список имен можно составить и вручную. Гораздо больший интерес представляют возможные значения параметров, ведь при разных значениях одного и того же параметра приложение может вести себя по-разному. Здесь также во многих случаях может помочь исходный код приложения. Одним из способов узнать возможные значения параметров, которые могут влиять на процесс работы, — это анализ условий в операторах ветвления типа 'if' и 'switch' — если при вычислении условия для оператора ветвления присутствует сравнение параметра запроса с некоторой константой, то эта константа и есть интересующее нас значение.

Для решения этой задачи необходимо выявить, где в вычислениях условий операторов ветвления используются параметры запроса, а также какие значения являются операндами в операциях сравнения, где один из операндов — параметр запроса. Сложность решения такой задачи также зависит от структуры исходного кода приложения. Для языка PHP можно сказать, что, как и в случае поиска имен параметров, поиск значений проще всего автоматизировать, если при вычислении условий операторов ветвления производится непосредственное обращение к массивам $_REQUEST, $_POST или $_GET, а сравнение производится с константными выражениями.

Подводя итоги вышесказанного, можно сказать, что автоматизация процесса поиска имен параметров и их значений в исходном коде приложения, написанном на PHP, является достаточно простой задачей, если выполнены следующие условия:

- обращение к массивам $_REQUEST, $_POST и $_GET производится непосредственно по именам параметров;
- если в вычислении условия для оператора ветвления используется значение, полученное в запросе, то обращение к этому значению производится непосредственно через обращение к массивам $_REQUEST, $_POST и $_GET;
- если в операции сравнения один из операндов — параметр запроса, то второй — константа.

Стоит отметить, что для приложений, работающих с базами данных, параметры запроса могут соответствовать некоторым полям из таблиц базы. При этом приложение может вести себя по-разному в зависимости от того, есть ли в базе данных запись, где соответствующее поле имеет значение, равное переданному значению параметра, или нет. Выявить такие ситуации сложно, поэтому можно порекомендовать всегда использовать в тестах как минимум два значения для каждого из параметров — одно из которых есть в базе данных, а другое отсутствует. Поскольку определить, какое значение есть в базе, на основе одного лишь кода программы зачастую невозможно, то здесь необходимо вмешательство человека. Однако и эту задачу можно



автоматизировать, если в процессе создания тестов есть возможность доступа к базе данных, с которой работает приложение. В таком случае достаточно указать, какие поля каких таблиц базы соответствуют именам параметров, и возложить задачу непосредственного выбора значений на генератор тестов.

## 3.2. Зависимости параметров

Помимо имен параметров и их возможных значений, полезно знать взаимосвязи между параметрами; это позволяет существенно сократить количество генерируемых тестов без потери качества. Например, если приложение использует параметр *param1*, только если в запросе присутствует *param2*, то нет смысла создавать запросы, в которых будет присутствовать *param1* и отсутствовать *param2*.

Можно выделить два основных вида зависимостей между параметрами:

- значения параметров {a1, a2, ... an} используются, только если в запросе присутствуют параметры {b1, b2, ... bm};
- значения параметров {a1, a2, ... an} используются, только если в запросе присутствуют параметры {b1, b2, ... bm} и их значения лежат в интервалах {v1, v2, ... vm} соответственно;

Задача определения всех зависимостей в общем случае также достаточно трудна, однако и здесь для каждого конкретного приложения могут существовать простые способы определить существенную часть зависимостей (достаточно большую, чтобы существенно сократить тестовый набор). В большинстве случаев анализ ветвлений (условных операторов и операторов выбора), производимых с учетом значений параметров запроса, с последующим анализом того, какие параметры используются в каждой ветке, может дать неплохой результат. Здесь снова встает вопрос о том, как определить, что некоторый параметр участвует в вычислении условия для ветвления, а также то, что некоторый параметр используется внутри ветки. Т.е. задача схожа с задачей выявления имен параметров и их возможных значений, рассмотренной ранее. Для нее справедливы все приведенные выше утверждения; в частности, можно сказать, что она достаточно хорошо автоматизируется, если исходный код приложения удовлетворяет условиям, перечисленным в конце предыдущего раздела.

Зная имена параметров, принимаемых скриптами, их возможные значения и взаимосвязи, можно формировать тестовые запросы, содержащие различные параметры с различными значениями. Однако перебор всех возможных сочетаний значений даже для относительно небольших наборов параметров может привести к слишком большому количеству тестов. Учет взаимосвязей помогает сократить количество тестов, однако этого сокращения может оказаться недостаточно. В таких случаях полезно сделать генератор тестов параметризуемым, чтобы он в зависимости от некоторых настроек выбирал только часть из допустимых тестовых воздействий. Настройки генератора

выбираются из соображений качества получаемого тестового набора, а также, зачастую, исходя из времени, требуемого для выполнения всех тестов.

## 4. Вынесение вердикта о правильности работы приложения

Создание тестовых данных является важным, но не единственным аспектом проверки качества ПО. Любому тесту необходимо уметь выносить вердикт о правильности работы тестируемого приложения. В случае Web-приложений это означает, что для каждой страницы, полученной от приложения по сгенерированной ссылке, необходимо установить, были ли допущены ошибки при ее формировании. Осуществить такую проверку в полном объеме под силу лишь человеку; однако существуют некоторые виды проверок, которые можно выполнять автоматически. Примеры таких проверок, производимые инструментом eValid, приведены во введении; для их более детального описания можно обратиться к документации инструмента [7]. Похожие виды проверок предоставляют многие инструменты, однако в силу своей универсальности они достаточно примитивны и способны выявлять очень узкий класс ошибок.

Рассмотрим ряд проверок, которые не столь универсальны (в том плане, что детали их конкретных реализаций могут сильно зависеть от тестируемого приложения), но позволяют производить более глубокий анализ результатов, получаемых в процессе выполнения тестов.

### 4.1. Ошибки взаимодействия с внешними системами

Как отмечалось выше, многие Web-приложения активно используют хранилища данных; кроме того, они могут взаимодействовать с другими внешними системами. Если успешность взаимодействий является критичной для функциональности приложения, то в процессе тестирования важно выявлять ошибки, происходящие во время обращения к внешним системам либо хранилищам данных. Одним из методов, позволяющих автоматически распознавать такие ошибки во время тестирования, является использование для взаимодействия функций, которые в случае возникновения ошибок выводят сигнализирующее об этом сообщение в специальный журнал либо непосредственно в текст генерируемого документа.

Во многих случаях для обращения к внешним объектам используются функции, предоставляемые внешними системами, которые в случае неудачного выполнения возвращают код ошибки либо выставляют переменные среды в определенные значения. Например, для выполнения запросов к базам данных, как правило, используются стандартные функции, принимающие на вход запрос в виде текстовой строки. В этом случае для них могут быть написаны функции-обертки, вызывающие внутри себя реальные функции, анализирующие результат их работы и в случае ошибок



добавляющие в журнал либо в генерируемый документ некоторое сообщение. При добавлении сообщения в создаваемый документ необходимо убедиться, что текст сообщения не может появиться в документе в случае нормального функционирования приложения. Тогда появление такого сообщения будет сигнализировать об ошибке.

Процесс поиска заданного текста в документе достаточно прост, и многие инструменты тестирования предоставляют такую возможность. Некоторые программы (например, Jmeter [12]) позволяют искать текст, соответствующий определенному регулярному выражению.

### 4.2. Соответствие спецификации XHTML

Web-приложение создает страницы в формате HTML либо XHTML [1]. Стандарт XHTML является более строгим; соответствие страниц этому стандарту облегчает работу Web-браузеров по их отображению и уменьшает время их загрузки. Соответствие спецификации XHTML позволяет избежать ошибок, связанных со структурной разметкой страницы (отсутствие необходимых тегов, использование атрибутов, которые могут поддерживаться не всеми браузерами и т.п.). Стандарт предъявляет строгие требования только к структуре документа, не затрагивая его содержимое; однако, как показывает практика, несоответствие сгенерированной страницы спецификации XHTML часто является следствием некорректной обработки данных, а не ошибкой форматирования страницы (при условии, что шаблоны разметки, используемые для формирования страницы, соответствуют стандарту XHTML).

Например, рассмотрим приложение, которое выводит текстовые данные, внутри которых могут содержаться угловые скобки — '<' и '>' (например, декларации шаблонов C++). Если угловые скобки печатать в документ HTML «как есть», то браузер будет воспринимать текст между двумя парными скобками как тег, и отображать его не будет. Однако поскольку такого «тега», скорее всего, не существует (не предусмотрено спецификацией XHTML), то при проверке на соответствие документа XHTML будет выявлена ошибка.

Для проверки соответствия документа спецификации XHTML существуют свободно доступные инструменты, например, Offline HTMLHelp.com Validator [13].

### 4.3. Анализ лог-файлов Web-сервера

Важной особенностью работы Web-приложений является то, что сообщения об ошибках, генерируемые интерпретатором скриптов могут (при соответствующих настройках) автоматически записываются в лог-файл HTTP-сервера. Среди ошибок могут быть и такие, которые не мешают интерпретатору продолжить выполнение скрипта, но могут привести к некорректности получаемых страниц. Одной из самых распространенных ошибок такого рода в случае PHP и Perl является выход за границы массива в

случае обычных массивов либо отсутствие значения для ключа в ассоциативном массиве. Например, в следующем коде

```
$select = 'SELECT Aid, Aname FROM Application';
$res = Query($select);
while( $row = mysql_fetch_array($res) ) {
       print      'Application      '.$row['Aname'].'
'.$row['Aversion'];
}
```

осуществляется попытка вывести на страницу значения полей Aname и Aversion из таблицы Application, но поле Aversion в запросе отсутствует. В результате будет выведено только поле Aname, а это не совсем то, что хотел программист. При этом никаких ошибок работы с базой данных не будет, также не будет нарушено соответствие страницы стандарту XHTML. Помимо непосредственного просмотра страницы обнаружить эту ошибку можно, просмотрев логи Web-сервера, в которых появится запись вида

```
Notice:  Undefined index: Aversion
```

При этом будет указан файл и номер строки, где произошло обращение по некорректному ключу.

Таким образом, анализ лог-файлов Web-сервера после выполнения тестов также может служить важной составляющей процесса тестирования.

## *5. Тестирование LSB Навигатора*

В рамках совместного проекта ИСП РАН и Linux Foundation по развитию инфраструктуры стандарта LSB в ИСП РАН разрабатывается LSB Навигатор (LSB Navigator) [14] — Web-приложение, позволяющее пользователям в удобной и доступной форме получать и анализировать данные, находящиеся в спецификационной базе данных LSB [15]. LSB Навигатор полностью написан на PHP и в настоящее время представляет собой 120 скриптов, содержащих около 17000 строк кода. В процессе работы может создаваться порядка 250 различных типов страниц. Количество используемых параметров запроса варьируется от 5 до 50 для разных скриптов. База данных LSB содержит 62 таблицы, а также 167 так называемых кэш-таблиц, создаваемых автоматически и используемых для ускорения работы Навигатора. По состоянию на декабрь 2007 года (дата выхода LSB 3.2), обычные таблицы содержат около 18 миллионов записей, кэш-таблицы — около 6 миллионов. Общее количество различных страниц, которые можно получить при работе с приложением, составляет около ста миллионов.

LSB Навигатор активно развивается, и ввиду достаточно сложной внутренней структуры программы актуальной является проблема регрессионного тестирования — изменения в некоторых участках кода могут повлиять на десятки страниц, и определение и проверка таких страниц вручную —



трудоемкий процесс. Кроме того, для постоянно появляющихся новых возможностей необходимы новые тесты. Применяемый подход позволил за короткое время получить достаточно качественный набор регрессионных тестов. Более того, тесты для новых возможностей генерируются без участия человека; таким образом, регрессионный набор автоматически поддерживается в актуальном состоянии.

### 5.1. Анализ кода

Код LSB Навигатора имеет ряд особенностей, благодаря которым для анализа скриптов в целях генерации тестов не требуется полноценного парсера PHP. Так, каждый скрипт, к которому можно обратиться из адресной строки браузера, имеет следующую структуру:

```
<инициализационная часть; возможно, некоторые проверки
параметров запроса>
switch( $_REQUEST['cmd'] ) {
      case "cmd1":
            function1(              $_REQUEST["param1"],
$_REQUEST["param2"], ... );
            break;
      case "cmd2":
            function2( ... );
            break;
      ...
      case "":
      default:
            default_function( ... );
}
```

Для простоты здесь не приведены различные действия с параметрами запроса перед вызовом функций — например, проверка корректности их значений. Кроме того, внутри каждой ветви оператора выбора могут присутствовать вложенные конструкции ветвления и какие-то дополнительные действия, не относящиеся к обработке параметров запроса.

Помимо массива $_REQUEST, используются также обращения к массивам $_GET и $_POST. Инициализационная часть также может содержать обращения к этим массивам, однако она, как правило, не содержит ветвлений и проста для анализа.

Основная же функциональность скрипта заключается в проверке значения параметра 'cmd' и вызова соответствующих функций в зависимости от этого значения. Аргументы для функций формируются из других параметров запроса, причем для различных значений 'cmd' (т.е. для разных функций) могут использоваться различные параметры.

Нетрудно видеть, что такой исходный код хорошо поддается автоматизированному анализу с помощью приемов, описанных ранее. В

соответствии с изложенным выше подходом для генерации тестов для LSB Навигатора был создан инструмент, разбирающий исходный код скриптов, анализирующий используемые параметры запроса, их возможные значения и взаимосвязи. Процесс работы анализатора кода построен следующим образом:

- определение параметров запроса, используемых в инициализационной части;
- определение списка значений параметра 'cmd', встречающихся в инструкциях 'case';
- определение имен и возможных значений параметров запроса, используемых в исходном коде внутри каждого блока 'case'; при этом производится рекурсивный анализ вложенных конструкции 'if' и 'switch' внутри каждой ветви.

Заметим, что анализ вложенных конструкций 'switch' может быть применен ко всему скрипту целиком, начиная с предложения 'switch( $_REQUEST['cmd'] )', и явного выделения такого предложения не требуется. Однако в тестах для LSB Навигатора использование $_REQUEST['cmd'] рассматривается отдельно — в данном случае стоит цель проверить реакцию приложения на все возможные значения параметра 'cmd', в то время как для остальных параметров часть значений может быть либо опущена, либо использована только с определенными значениями других параметров.

Вызываемые функции function1, function2, ..., default_function определены в отдельных файлах, к которым нельзя получить доступ через Web-браузер, и которые не содержат обращений непосредственно к массивам $_REQUEST, $_POST или $_GET. В настоящее время тела вызываемых функций не анализируются, хотя потенциально функции могу вести себя по-разному в зависимости от значений своих аргументов, и такой анализ мог бы способствовать повышению качества тестов.

## 5.2. Генерация тестов

На основе сведений, собранных в процессе анализа скрипта, производится генерация тестов для этого скрипта. Тест представляет собой ссылку вида

```
http://test.host.name/script_name?param1=value1&pa
ram2=value2...
```

В настоящее время получение страниц, соответствующих сгенерированным ссылкам, возлагается на внешнюю программу (утилиту wget). При этом все параметры помещаются непосредственно в ссылку, т.е. фактически являются параметрами типа GET. Поскольку реально приложение может ожидать получения некоторых параметров только методом POST, то на время тестирования версии приложения обращения к массиву $_POST заменяются обращениями к массиву $_REQUEST. При этом если LSB Навигатор ожидает получить некоторый параметр методом POST, то он не использует параметр с



таким же именем, но получаемый методом GET. Это позволяет гарантировать, что при описанном изменении метода передачи параметров коллизий не возникает и функциональность приложения не нарушается.

В процессе генерации перебираются имена параметров, извлеченные из текста скрипта. Для каждого параметра перебираются значения из множества, формируемого следующим образом:

- если значение параметра сравнивалось в тексте скрипта с некоторой константой, то такая константа заносится во множество возможных значений параметра. Так, для параметра 'cmd' перебираются все значения, найденные в соответствующих предложениях 'case';
- если имя параметра совпадает с именем поля в одной из таблиц спецификационной базы данных LSB, то во множество потенциальных значений заносятся следующие значения:
  - значение, которое содержится в соответствующем поле хотя бы одной записи базы данных;
  - значение-строка, которое гарантированно не встречается ни в одной записи базы данных;
  - значение-число, которое гарантированно не встречается ни в одной записи базы; такое значение заносится в список только для полей БД числового типа. Для определения такого значения из существующих значений выбирается максимальное и к нему прибавляется единица. При этом производится проверка, что полученное число умещается в диапазон значений поля; если это не так, будет выдано предупреждение (однако автоматический выбор нового значения в настоящее время не реализован, поскольку в реальной базе данных таких ситуаций пока не возникает). Стоит отметить, что правила именования полей в базе данных LSB позволяют отличить поля, имеющие числовой тип, от полей текстового типа, на основе самого имени;
- есть несколько параметров, для которых все возможные значения задаются вручную.

Поскольку количество скриптов и их параметров велико, то перебор всех возможных сочетаний значений параметров нецелесообразен. В целях уменьшения количества тестов применяются различные стратегии перебора комбинаций параметров — на данный момент генератор может быть настроен на использование каждого второго из всех возможных сочетаний, каждого третьего, и т.д. Недостатком на данный момент является то, что одна и та же стратегия перебора применяется ко всем комбинациям параметров; если для каких-то параметров необходимо применить другую стратегию, то

необходима ручная правка генератора тестов. В будущем планируется сделать генератор более удобным в настройке.

## 5.3. Вынесение вердикта

Вся содержательная часть страниц, генерируемых LSB Навигатором, формируется на основе спецификационной базы данных LSB. Поэтому одним из основных критериев успешности выполнения скриптов, образующих Навигатор, является успешность взаимодействия с базой данных.

В качестве СУБД в Linux Foundation используется MySQL, для работы с ней из Навигатора используются соответствующие функции PHP. В соответствии с описанным ранее подходом, непосредственно из скриптов вызываются функции-обертки, производящие обращение к реальным функциям и первичную обработку возвращаемых результатов. Так, при получении ошибки от MySQL функции-обертки выводят на генерируемую страницу фразу 'MySQL ERROR', за которой следует непосредственно текст ошибки. Эти функция — единственный потенциальный источник фразы 'MySQL ERROR' на странице; при нормальном функционировании приложения она не может возникнуть, поэтому ее появление в тексте свидетельствует об ошибке.

Каждая получаемая в процессе тестирования страница проверяется на соответствие стандарту XHTML 1.0. Для проверки соответствия страницы стандарту XHTML используется Offline HTMLHelp.com Validator [13].

Также после выполнения производится анализ логов HTTP-сервера (в случае LSB Навигатора — Apache), из которого выбираются предупреждения и ошибки, выдаваемые интерпретатором PHP.

В будущем планируется расширить анализ текста страниц HTML, генерируемых приложением в процессе тестирования. Например, на данный момент встречаются ошибки, связанные с некорректным формированием ссылок из страницы на другие части документа. В случае LSB Навигатора можно утверждать, что если ссылка содержит обращение к скрипту, передавая ему некоторые параметры, но при этом часть параметров имеет пустое значение, то такая ссылка некорректна — перейдя по ней, пользователь получит сообщение о некорректном значении параметров. Поиск таких ссылок в документе является достаточно простой задачей.

Кроме того, интересной представляется возможность более глубокого анализа содержимого страниц на основе запроса, в ответ на который они были получены. В общем случае для такой проверки понадобится создание программы, фактически дублирующей функциональность приложения. Однако вполне вероятно, что можно выявить зависимости между параметрами запроса и некоторыми элементами генерируемой страницы, которые позволят не только автоматически создавать тестовые запросы, но и вычислять, какие элементы должны обязательно присутствовать на полученной в результате запроса странице.



## 5.4. Результаты

Автоматические тесты для LSB Навигатора позволили к настоящему времени выявить около 30 ошибок, связанных с функциональностью. При этом общее число ошибок, занесенных в базу ошибок Навигатора за время его разработки — 250. Кроме того, автоматические тесты играют роль регрессионных и позволяют следить за тем, что генерируемые приложением страницы соответствуют стандарту XHTML; выполнение этой работы вручную нереалистично из-за сложности приложения и его базы данных. Разработка же анализатора скриптов и генератора тестов заняла 2 человеко-дня.

Характеристики генерируемых тестовых наборов по состоянию на декабрь 2007 года приведены в Таблице 1. Тесты делятся на две категории, в соответствии с режимом работы LSB Навигатора, для которого они предназначены. LSB Навигатор предоставляет два режима работы — режим пользователя (“Browse mode”) и режим администратора (“Administration mode”). При работе в режиме пользователя не допускается никаких изменений в базе данных (т.е. осуществляются только выборки данных при помощи оператора SELECT), в то время как в режиме администратора предоставляется практически полный контроль над базой данных. Скрипты, реализующие режим администратора, могут осуществлять гораздо больше различных запросов к базе данных и принимают на вход значительно большее число параметров. Ввиду этого тестирование работы в режиме администратора занимает больше времени, чем в случае режима пользователя.

| Режим работы | Количество тестов | Время генерации | Время тестирования |
|---|---|---|---|
| Пользователь | 2912 | 15 с | 1ч. 30 мин. |
| Администратор | 28740 | 10 мин. | 6 ч. 15 мин. |

*Таблица 1. Характеристики тестовых наборов для режимов пользователя и администратора LSB Навигатора.*

Число тестов для режима администратора превосходит число тестов для режима пользователя в примерно 10 раз, а время их генерации — в 40. Это обусловлено тем, что при генерации тестов для режима администратора анализатор кода использует существенно больше системных ресурсов, что, в частности, приводит к активному использованию swap-раздела. В то же время затраты на выполнение тестов для этого режима отличаются не столь сильно; этот факт объясняется тем, что любой запрос, выполняемый в режиме администратора, затрагивает только одну таблицу базы данных (более того, существенная часть запросов обращается только к одной записи таблицы). В режиме же пользователя выполняется много сложных агрегатных запросов, выбирающих данные сразу из нескольких таблиц (например, при сборе различных статистических данных).

Большую часть времени (порядка 80%) занимает проверка получаемых страниц на соответствие спецификации XHTML. Соответственно, отключение этой проверки позволяет за то же время сгенерировать и выполнить большее количество тестов (например, перебирая больше различных комбинаций параметров). Более того, время анализа результатов тестирования сильно зависит от числа выявляемых при такой проверке ошибок (что обусловлено особенностями используемого для проверки инструмента) — так, наличие ошибок, проявляющихся в 250 тестах для режима пользователя (8% от общего количества) привело к увеличению времени их выполнения до 8 часов.

Что касается непосредственно ошибок, выявляемых тестами, то их можно разделить на следующие категории.

- Вывод имен классов, типов и шаблонов без преобразования угловых скобок, знака '&' и других специальных символов (на страницах, содержащих декларации функций и классов C++), которые могут быть интерпретируемых Web-браузером как часть разметки страницы, а не как данные. Такие ошибки выявляются в процессе проверки страницы на соответствие спецификации XHTML.

- Ошибки, связанные с экранированием символов в строках, используемых в запросах к базам данных. Отсутствие экранирования приводит к некорректности запроса SQL и получению ошибки при работе с СУБД. Обнаружение таких ошибок чрезвычайно важно, т.к. их наличие в приложении делает потенциально возможным использование SQL-инъекций — способа взлома сайтов и программ, работающих с базами данных, основанного на внедрении в запрос произвольного SQL-кода [16].

- Ошибки, связанные с некорректной обработкой переданных параметров. При определенных значениях параметров на их основе может быть создан некорректный SQL-запрос, что приведет к ошибке взаимодействия с СУБД.

- Обращение к ассоциативному массиву по некорректному ключу. Такие ошибки выявляются на основе анализа лог-файлов HTTP-сервера. Как правило, они являются следствием некорректного задания имен полей, выбираемых в запросе к базе данных, либо банальных опечаток.

Измерение покрытия кода показало, что в процессе тестирования покрывается 60% строк кода скриптов (как уже упоминалось ранее, всего скрипты содержат около 17000 строк кода). Для сравнения — если просто обратиться к каждому скрипту без передачи ему каких-либо параметров, то покрытие составит 15%. Для измерения покрытия используется свободный инструмент PHPCoverage компании Spike Source [17]. Стоит отметить, что анализ



покрытия кода также помогает выявлять недостижимый код, удаление которого приводит к увеличению покрытия.

Несмотря на достаточно высокое значение тестового покрытия, в процессе ручного тестирования встречаются ошибки, которые потенциально могли быть выявлены и автоматическими тестами (как правило, это ошибки работы с базой данных, либо ошибки формирования страницы, приводящие к нарушению спецификации XHTML). Такие ошибки проявляются на специфических значениях параметров при определенном их сочетании; они не выявляются автоматически, поскольку в процессе генерации тестов перебираются не все сочетания значений параметров. Как уже отмечалось ранее, генератор тестов обладает некоторыми средствами настройки стратегий перебора параметров, эти средства осуществляют достаточно грубую настройку. В результате при задании достаточно жестких ограничений на комбинации параметров тесты пропускают некоторые ошибки, а даже незначительное ослабление этих ограничений приводит к слишком большому количеству тестов, выполнение которых может длиться больше недели. Поэтому возможность более тонкой настройки генератора является одним из приоритетных направлений развития в ближайшем будущем; также возможно, что уменьшению числа тестов без потери качества может помочь более глубокий анализ зависимостей между параметрами.

Тем не менее, в автоматических тестах предусмотрена возможность использования запросов, созданных человеком, наряду со сгенерированными тестовыми запросами. Все запросы, которые привели к ошибкам, но при этом не попали в список сгенерированных, добавляются в «ручной» список. Таким образом, можно гарантировать, что если по каким-то причинам ошибка появится снова, то она будет выявлена автоматически.

## *6. Заключение*

Роль человека в тестировании приложений с графическим пользовательским интерфейсом, к которым относятся и Web-приложения, переоценить трудно; проверка многих аспектов, касающихся интерфейса, с трудом поддается автоматизации. Однако в случае Web-приложений ситуация несколько упрощается, поскольку для многих из них большую часть того, что видит пользователь на экране, можно получить автоматически и в текстовой форме — в виде документа HTML. Многие современные инструменты используют этот факт, предоставляя тестировщикам возможность проверки свойств различных элементов страницы, отображаемой Web-браузером, сводя такое тестирование к проверке соответствующих элементов HTML-кода страницы.

В то же время большинство современных средств тестирования Web-приложений используют в своей работе сценарии, создаваемые людьми; полноценное тестирование больших приложений с применением такого подхода требует больших людских ресурсов. Также ресурсоемким оказывается создание и поддержание в актуальном состоянии набора

регрессионных тестов. Причем для приложений, у которых содержательная часть одной и той же страницы постоянно меняется, возможность создания сложных тестовых сценариев сомнительна - ведь страницы, генерируемые при создании сценария, не могут быть использованы в качестве эталонов при вынесении вердикта об успешности прохождения теста. В таких случаях приходится применять менее строгие проверки, чем сравнение страниц с эталоном целиком. Однако создание тестов и выполнение ряда несложных проверок может быть автоматизировано, что позволяет существенно экономить ручной труд. В то же время даже достаточно простые проверки могут помочь выявить серьезные ошибки.

Одним из методов автоматизации тестирования Web-приложений является генерация тестов на основе анализа исходного кода приложения, возможные подходы к которой изложены в данной статье. Для выявления ошибок в генерируемых приложением страницах HTML может быть использован ряд автоматических проверок, которые либо вовсе не зависят от кода приложения, либо требуют добавления в код достаточно несложных конструкций, не влияющих на функциональность.

Предлагаемый подход к генерации тестов на основе исходного кода приложения гораздо больше зависит от структуры кода. Впрочем, эта структура не налагает ограничений на саму возможность применения метода; однако именно от нее зависит сложность реализации метода для конкретного приложения. Соответствие кода некоторым условиям может существенно упростить создание тестов и в тоже время сделать их более качественными. Конкретные формулировки таких условий зависят от используемого языка программирования (более точно — от способа доступа к параметрам, передаваемым приложению, который предоставляет язык); в статье приводятся формулировки для приложений, написанных на PHP.

Такие условия могут быть учтены при создании приложения. При их соблюдении для анализа кода не требуется полноценного парсера используемого языка программирования; достаточным оказывается создание существенно более простого инструмента. Примером приложения, для которого предлагаемый подход позволяет получать достаточно качественные тестовые наборы за короткое время, является LSB Навигатор (распространяющийся по лицензии GPL; исходный код может быть получен в соответствующем разделе Linux Foundation Bazaar [18]). Естественно, реализация метода зависит от языка программирования, на котором написано приложение. Однако для многих скриптовых языков, используемых при создании Web-приложений, сложность реализации этого подхода примерно одинакова; ключевым фактором все-таки является структура исходного кода.

**Литература**


[1] XHTML 1.0. The Extensible HyperText Markup Language (Second Edition). http://www.w3.org/TR/xhtml1/.





[2] Википедия — свободная энциклопедия. Скриптовый язык. http://ru.wikipedia.org/wiki/Скрипт.

[3] IBM Rational Robot. http://www-306.ibm.com/software/awdtools/tester/robot/.

[4] HP WinRunner Software. https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24%5E1074_4000_100__.

[5] Empirix e-TEST suite. http://empirix.com/products-services/w-testing.asp.

[6] PureTest — Users Guide. Minq Software, 2006. http://minq.se/products/pureload/doc/html/puretest/usersguide/index.html

[7] eValid Web Testing & Analysis Suite. http://www.soft.com/eValid/.

[8] Web Application Testing with Puffin. http://www.ibm.com/developerworks/opensource/library/os-puffin.html.

[9] С. Рогов, Д. Намиот. Тестирование производительности Web-серверов. Открытые системы, 12:55–64, 2002.

[10] А. А. Сортов, А. В. Хорошилов. Функциональное тестирование Web-приложений на основе технологии UniTESK. Труды ИСП РАН, 8(1):77–97.

[11] RFC 2616. Hypertext Transfer Protocol — HTTP/1.1.

[12] Jmeter User's Manual. Jakarta Project, 2007. http://jakarta.apache.org/jmeter/usermanual/index.html.

[13] Offline HTMLHelp.com Validator. http://www.htmlhelp.com/tools/validator/offline/index.html.en.

[14] LSB Navigator. http://linux-foundation.org/navigator.

[15] LSB Specification Database. http://ispras.linux-foundation.org/index.php/LSB_Database_Home.

[16] Википедия — свободная энциклопедия. Инъекция SQL. http://ru.wikipedia.org/wiki/Инъекция_SQL.

[17] Spike PHPCoverage. http://developer.spikesource.com/wiki/index.php/Projects:phpcoverage.

[18] Linux Foundation Bazaar — DB Navigator module. http://bzr.linux-foundation.org/lsb/devel/dbadmin.