Автоматизация в тестировании web-приложений

Мычко А. И.
Кафедра ЭВМ, ФКСиС
Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
e-mail: hor2zont@gmail.com

***Аннотация*—Производится рассмотрение сути и принципов тестирования web-приложений. Рассматривается как производится процесс автоматизация тестирования. Показываются реальные процессы, которые при этом происходят. Также выдвигается идея, которая позволит упростить и ускорить процесс автоматизации тестирования.**

Ключевые слова: web-приложения; html; тестирование; test driven development; continuous integration; selenium; автоматизация тестирования, внедрение тестов в web-приложение.

I. Введение

Тестирование – один из основополагающих принципов создания качественного и надежного программного обеспечения (ПО).

Если тестирование не проводить – то нельзя говорить о том, что приложение будет достаточно качественным и пригодным для работы. Большинство проблем с ПО находится как раз на стадии тестирования, а не разработки. Однако минусом является большая трудоемкость и необходимость привлечения новых людей для работы над нужным ПО.

Чтобы уменьшить ручной труд можно воспользоваться различной автоматизацией процесса тестирования. Но автоматизация накладывает новые проблемы, связанные со сложностью этого процесса и необходимостью программистов писать 2 кода: приложения и тестов для этого приложения.

II. Виды тестирования

Тестирование можно разделить на множество видов [1]:

1. По знанию внутренностей системы (черный ящик, серый ящик, белый ящик).
2. По объекту тестирования (функциональное тестирование, тестирование интерфейса пользователя, тестирование локализации, тестирование скорости и надежности, тестирование безопасности, тестирование опыта пользователя, тестирование совместимости).
3. По времени проведения тестирования (до передачи пользователю — альфа-тестирование, тест приемки – *smoke test,* тестирование новых функциональностей, регрессивное тестирование, тест сдач, после передачи пользователю — бета-тестирование).
4. По критерию "позитивности" сценариев (позитивное тестирование, негативное тестирование).
5. По степени изолированности тестируемых компонентов (компонентное тестирование, интеграционное тестирование, системное тестирование.
6. По степени автоматизированности тестирования (ручное тестирование, автоматизированное тестирование, смешанное тестирование).
7. По степени подготовки к тестированию (тестирование по документации, интуитивное тестирование).

Для оценки работоспособности web-приложений обычно речь идет о функциональном тестировании – проверка какой-либо функциональности программы на корректность ее работы. Проверяются как отдельные модули (классы) приложения – unit-тестирование, так и функционирование целой системы, или ее отдельных частей – интеграционное тестирование. Unit-тесты показывают правильность работы каждого модуля в приложении. Интеграционные тесты – как правильно модули работают при взаимодействии между собой.

Интеграционные тесты могут различаться по своему назначению: smoke-тесты (применяются для поверхностной проверки всех модулей приложения на предмет работоспособности и наличие быстро находимых критических и блокирующих дефектов), регрессионные (для проверки изменений, сделанных в приложении для подтверждения того факта, что существующая ранее функциональность работает как и прежде).

Именно эти виды тестирования составляют большую часть тестов для приложения и в первую очередь нуждаются в автоматизации.

III. Организация автоматизации тестирования

Процесс создания web-приложения может происходить по-разному. Чтобы обеспечить наибольшее качество проекта необходимо этот процесс объединить с тестированием приложения. Здесь лучше всего подходит принцип Test driven development (TDD). TDD - это методика разработки ПО, которая основывается на коротких циклах работы, где сначала создаётся тест, а потом функционал [4].

При этом подходе в первую очередь создаются именно unit-тесты для будущих классов, а затем пишется код этих классов. Код тестов обычно пишется самими программистами. Здесь выгода в том, что в дальнейшем при любом изменении в коде приложения можно запустить тесты и посмотреть работоспособность приложения. Т.е. процессом автоматизации модульного тестирования занимаются сами программисты. Чтобы протестировать приложение целиком одних unit-тестов не достаточно. Для оценки работы всего приложения еще необходимо

Библиотека БГУИР

протестировать его со стороны пользователя. Здесь уже появляются интеграционные тесты. Эти тесты выполняют тестировщики. Чтобы автоматизировать этот процесс можно воспользоваться средством selenium.

Selenium – это *инструмент для программного управления браузерами [2]. Он может реализовать любое действие пользователя. Существует несколько различных продуктов, позволяющих управлять браузером. Используя один из них можно описать те действия, которые может совершать пользователь. Такие тесты обычно пишут не те программисты, которые разрабатывают само приложение.*

Чтобы автоматизированное тестирование можно было производить необходимо вести учет тех функциональностей, которые необходимо проверить. Также необходимы алгоритмы тестов, которые позволят с наибольшей вероятностью сказать, правильно ли работает приложение. Список функциональностей называется check-list, а алгоритм для проверки одной функциональности – test-case. При написании тестов и их поддержке создается новое приложение, которое включает в себя все вышеперечисленные алгоритмы. При этом тесты должны быть как можно более универсальными, т.к. тестируемое приложение постоянно изменяется и тесты должны под него подстраиваться.

IV. Подход «continuous integration»

Однако просто наличие тестов не подразумевает то, что ошибки будут находиться быстро. Т.к. запускаться они будут не всегда. Чтобы сделать процесс разработки web-приложения еще более эффективным можно следовать принципу continues integration.

Continuous integration (CI) – принцип разработки ПО, подразумевающий постоянную интеграцию свежего кода в проект и проверки приложения после каждого изменения [3]. Также могут выполнять еще различные другие действия с проектом, но для обеспечения качества приложения, вышеперечисленных действий достаточно.

Сборки проекта должны быть быстрыми. Такими же должны быть и тесты. Однако если unit-тесты проверяют каждый блок по отдельности и выполняются быстро, то интеграционные тесты требуют запуск браузера и повторяют всё поведение пользователя, здесь каждый тест может занимать по несколько минут. Чтобы процесс CI проходил правильно, нельзя, чтобы программист ждал большое количество времени, чтобы понять, как его изменения повлияли на работоспособность всего приложения.

В этом случае процесс тестирования приложения нужно разнести на 2 части:

1. Unit-тесты запускаются после каждой сборки проекта (т.е. после малейшего изменения в проекте). Считается, что работоспособность каждого модуля по одиночке дает определенную гарантию на то, что приложение осталось работоспособным.
2. Интеграционные тесты запускаются только в определенное время, обычно ночью, и проходят один раз в день. Здесь идет уже другая проверка приложения, как оконечного продукта. Такой подход позволяет контролировать стабильность приложения и все ошибки находятся сразу после их возникновения.

V. «Внедренных тестов» в самом web-приложении

Автоматизированные интеграционные тесты являются самостоятельной большой программой, которая позволяет проверять качество другого приложения. Но написание тестов отнимает много времени и делать это можно только после создания тестируемого приложения. Т.е. как только web-приложение создано, для него еще нет тестов, и чтобы сказать, что оно работоспособно приходится нанимать тестировщиков и заниматься отдельно создание автоматизированных тестов.

Чтобы избежать этой проблемы, связанной с затратами на написание интеграционных тестов, можно пойти следующим путем – создавать шаблон для тестов, в момент написания web-приложения. Для этого можно предусмотреть внедрение дополнительных тегов и атрибутов в верстку html. Они укажут на те элементы в верстке, которые будут участвовать в тестировании. Если задать этим атрибутам специальные параметры, говорящие об очередности выполнения действий и о том, какой должен получиться результат, то полученные «встроенные» тесты приобретут необходимую гибкость и логику. Они смогут выполнять полноценное тестирование. Если для тестирования будут необходимы какие-то данные, то их нужно будет вынести в специальный файл. Обычно для Smoke-тестов вышеперечисленных возможностей вполне достаточно. Т.е. можно создать Smoke-тесты без больших усилий.

Для запуска таких тестов нужна будет только программа, которая будет анализировать html страницы, искать в них специальные атрибуты, запускать selenium и начинать работать с теми тегами, которые помечены.

Для сложных тестов, которые могут содержать очень много действий, полностью всю логику можно поместить в отдельный файл. Здесь потребуются дополнительные усилия на написание теста, но описываться тест будет в простом ключе, это будет уже не сложная программа, а просто шаги из test-case.

Еще при разработке приложения, затратив немного больше времени для создания верстки страницы, можно параллельно указать то, как нужно тестировать приложение. В результате этого, как только приложение станет работоспособным, для него сразу будут существовать интеграционные тесты.

[1] Роман Савин, “Тестирование dot com” – М.: Дело, 2007 – 312 с.

[2] Проекты Selenium [Электронный ресурс]. – Электронные данные. Режим доступа: http://seleniumhq.org/projects/

[3] Continuous Integration [Электронный ресурс]. – Электронные данные. Режим доступа: http://lib.custis.ru/Continuous_Integration

[4] Introduction to Test Driven Development [Электронный ресурс]. – Электронные данные. Режим доступа: http://www.agiledata.org/essays/tdd.html

Библиотека БГУИР