УДК 004.415.53

Н.А. Матвеева, В.В. Герасимов, Д.О. Игнатьева

# ИССЛЕДОВАНИЕ ТЕХНОЛОГИЙ МОДУЛЬНОГО ТЕСТИРОВАНИЯ НА ПЛАТФОРМЕ JAVA

*Аннотация. Проведено исследование основных технологий модульного тестирования, выявлены их функциональные особенности, достоинства и недостатки. Сделаны выводы касательно мест их применения.*

*Ключевые слова: модульное тестирование, тест, модуль, JUnit, TestNG, TestFX, Mockito, Hamcrest, FestAssertions.*

**Вступление.** Каждая сложная программная система состоит из отдельных частей — модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе системы в целом, необходимо вначале протестировать каждый модуль системы в отдельности. В случае возникновения проблем это позволит заранее выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них. Именно поэтому все большую популярность приобретает модульное тестирование.

Основная цель модульного тестирования — удостовериться в соответствии требованиям каждого отдельного модуля системы перед тем, как будет произведена его интеграция в состав системы. При этом в ходе модульного тестирования решаются четыре основные задачи: поиск и документирование несоответствий требованиям, поддержка разработки и рефакторинга низкоуровневой архитектуры системы и межмодульного взаимодействия, поддержка рефакторинга модулей и поддержка устранения дефектов и отладки [1].

**Целью работы** является рассмотрение наиболее популярных технологий модульного тестирования программного обеспечения на платформе Java, таких как JUnit, TestNG, TestFX, Mockito, FestAssertion, их сравнение по эффективности использования в различных сферах разработки программного обеспечения.

**Основная часть.** Первая технология, JUnit [2] — библиотека для модульного тестирования программного обеспечения. В основном

используется для модульного тестирования Java-проектов. JUnit былапортирована на другие языки, включая PHP (PHPUnit), C# (NUnit), Python (PyUnit), Fortran (fUnit), Delphi (DUnit), FreePascal (FPCUnit), Perl (Test:Unit), C++ (CPPUnit), Flex (FlexUnit), JavaScript (JSUnit), COS (COSUnit). Опыт, полученный при работе с JUnit, важен в разработке концепций тестирования программного обеспечения.

JUnit имеет следующие преимущества: являетсяполноценной-тестовой средой, не требует контроля со стороны пользователя во время исполнения, дает возможность запускать несколько тестов одновременно и выводить сообщения обо всех ошибках в ходе тестирования.

JUnit применяется для модульного тестирования, которое позволяет проверять на правильность отдельные модули исходного кода программы. Преимущество данного подхода заключается в изолировании отдельно взятого модуля от других. Тестовые классы JUnit 4 можно исполнять как с помощью интегрированной среды разработки, например, Eclipse, так и с помощью интерфейса командной строки. Благодаря использованию аннотаций в JUnit 4 используется гибкая фикстурная модель (fixturemodel). Аннотации позволяют исполнять одну и ту же фикстуру для каждого теста или один раз для всего класса, или не исполнять ее совсем.

К недостаткам JUnit можно отнести невозможность моделирования многопоточных ситуаций (запуск нескольких тестов одновременно не позволяет отследить момент их завершения) и резкое увеличение объема конфигурации при увеличении числа классов и возможных вариантов запуска.

Далее рассмотрим технологию TestNG [3]. TestNG — тестовый фреймворк для языка программирования Java наряду с фреймворками для модульного тестирования семейства XUnit [4], но он имеет новую мощную функциональность и более прост в использовании. TestNG — это фреймворк с открытым кодом, где NG в TestNG означает NextGeneration (Новое поколение). TestNG похож на JUnit, но намного функциональнее него.

В TestNG тесты описываются с помощью аннотаций, как и в JUnit 4, но здесь их гораздо больше и поддерживаются параметры аннотаций. Также здесь есть новые параметры, например,

dependsOnMethods, в котором записываются методы, от которых зависит тест, помеченный данной аннотацией — сначала будут выполнены они, а затем данный тестовый метод. Важным преимуществом TestNG является поддержка тестирования, управляемого данными с аннотацией для тестов @DataProvider [5], т.е. DataProvider ответственен за предоставление тестовых данных для методов, в которых данные используются. Преимущество заключается в том, что данные хранятся отдельно от тестов. В TestNG поддерживается многопоточность. Если необходимо проверить, как поведет себя приложение во многопоточном окружении, можно сделать так, чтобы тесты исполнялись одновременно из нескольких потоков. Также в TestNG расширение функционала может быть реализовано с помощью механизма разных типов слушателей.

TestNG поддерживается множеством инструментов и программных расширений, например, такими какEclipse, NetBeans, Maven и др. Одной из функций TestNG является мощная модель выполнения — больше нет TestSuite(наборов тестов), в то время как наборы тестов, группы и тесты, отобранные на выполнение, определяются и формируются файлами XML.

Главное преимущество TestNG — это возможность запускать все тесты одним xml-файлом. Автоматизация тестирования стала проще из-за следующих преимуществ TestNG:

- Встроенная система отчётов. То, что раньше приходилось разрабатывать отдельно, идёт готовым формализованным решением.
- Тестовые сценарии создаются в виде xml-файлов, а тестовые данные доставляются отдельно, что значительно упрощает создание тестовых данных.

Далее рассмотрим технологию TestFX [6]. TestFX— это интерфейс программирования приложений для тестирования пользовательских интерфейсов, написанных на JavaFX. Он автоматизирует тесты для приложений JavaFX путем имитации действий пользователя, таких как нажатие кнопок, ввод текста в поле ввода и многие другие операции, которые осуществляются в приложениях JavaFX.

TestFXоснован на известном фреймворке для модульного тестирования JUnit. Как и JUnit, TestFX очень простой в освоении и легкий в использовании. В TestFX тесты могут быть написаны аналогично тестамJUnit. Необходимо лишь добавить несколько дополне-

ний. Например, TestFX использует Hamcrest[7]вычислители поверх JUnit для утверждений тестов. Hamcrest вычислители по сравнению со стандартными утверждениями в JUnit имеют преимущество, которое заключается в том, что вы можете использовать натуральные утверждения и получить более полезные сообщения об ошибках, когда утверждение не выполняется. Это повышает качество кода тестов, а также снижает трудоемкость утверждений. Еще одним преимуществом TestFX является то, что он сохраняет в папку проекта точные скриншоты невыполненных тестов. Написание тестов с помощью TestFX не займет много времени.

Следующей рассмотрим технологию Mockito [8]. Mockito является одним из широко используемых интерфейсов тестирования для платформыJava. Mockito позволяет писать хорошие тесты с помощьючистого и простого API. Mockito-тесты очень легко читаются и выдают чистые ошибки проверки. Исследование 2013 года, проведенное над 10000 проектами GitHub, показало, что Mockito— 9-я по популярности библиотека Java.

Вообще, mock-объект (или заглушка) [9] — это объект, созданный, чтобы заменить объект, с которым ваш код будет сотрудничать. Ваш код может вызывать методы на mock-объекте, который будет добиваться результатов, заданных с помощью ваших тестов. Mock-объекты отлично подходят для тестирования части кода независимо от остального кода. Они являются пустыми оболочками, которые обеспечивают методы, позволяющие тестам управлять поведением всех бизнес-методов фальшивого класса. Также Mockito существенно упрощает разработку тестов для классов с внешними зависимостями.

Mockitoсовершенствует технологию создания тестовых сценариев. И речь не только omock-объектах, но и в целом это очень хороший выбор для разработчиков, способствующий созданию реальных устойчивых тестов, которые обеспечивают стабильность программного обеспечения [10].

Наиболее интересные возможности фреймворка следующие: возможно создание заглушки как для классов, так и для интерфейсов; можно проверять, вызывался ли метод и с какими параметрами; создана концепция "частичной заглушки" — в этом случае заглушка создается на класс и есть возможность задать поведение, требуемое для некоторых методов.

 ISSN 1562-9945

Mockito отслеживает все вызовы методов изначенияих параметров для mock-объекта. Для того, чтобы убедиться, что указанные условия соблюдены, т.е. что метод был вызван с определенными параметрами, используют метод verify() на mock-объект. Этот вид тестирования иногда называют тестированием поведения, потому что он проверяет не результат вызова метода, а то, что метод вызывается с нужными параметрами. Mockito отличается от других mock-фреймворков, позволяя разработчикам проверить поведение тестируемой системы без установки заранее ожидаемого результата. Одно из слабых мест mock-объектов заключается в том, что существует тесная связь тестового кода и тестируемой системы. Так как Mockito пытается ликвидировать шаблон «ожидаемое-запуск-проверка», удалив спецификацию ожиданий, связь сведена к минимуму. Результатом этой отличительной особенности является более простой тестовый код, который легче читать и модифицировать. Mockito также предоставляет некоторые аннотации, полезные для снижения шаблонности кода.

Объединение трех фреймворков, Mockito, Hamcrest и JUnit, может сделать с «закрытой» системой гораздо больше, чем доступ через открытые интерфейсы. Однако, эта особенность является также и недостатком для системы модульного тестирования (в случае TestDrivenDesign). Ведь основная цель модульного тестирования — удостовериться в соответствии требованиям каждого отдельного модуля системы. При изменении требований модули системы меняются, поэтому модульные тесты должны тоже менятся. Если в таком случае тесты не меняются, а меняются только матчеры (объекты, которые содержат логику принятия решения, знают, что ждали и что получили, о чем самостоятельно сообщают), то такие тесты неправильно тестируют измененные модули системы. В конечном итоге такие тесты способствуют созданиюплохихинтерфесов между компонентами вашей системы.

И, наконец,Mockito имеет определенные ограничения. Он не может проверить такие конструкции, как финальные классы, анонимные классы и примитивные типы.

Hamcrest— библиотека вычислителей (mathers) для построения тестовых выражений[7]. Hamcrest— это фреймворк, которыйиспользуютдля написания тестов для программного обеспечения на плат-

форме Java. Он поддерживает создание настраиваемых утверждений —матчеров, что позволяет соответствовать правилам, которые будут определены декларативно. Matcher-ы используются в модульном тестировании с помощьюJUnit. Типичные сценарии включают в себя фреймворки для тестирования, mock-библиотеки и правила проверки пользовательского интерфейса. Hamcrest была портирована на Java, C++, Objective-C, Python, PHP и Erlang. Hamcrest— библиотека не для тестирования: просто так получилось, что вычислители (matcher-ы) являются очень полезными для тестирования.

Далее рассмотрим технологию FestAssertion [11]. FestAssertion — это Java-библиотека, которая предоставляет свободный интерфейс для написания утверждений (assert-ов). Ее основная цель — улучшить читабельность кода тестов и сделать обслуживание тестов более легким. Для того, чтобы вам было быстрее и удобнее писать утверждения (assertions) она использует так называемый fluentinterface (способ реализации объектно-ориентированного API, нацеленный на повышение читабельности исходного кода программы). Festassertion требует Java SE 6.0 или более поздней версии и может использоваться в JUnit. Fest предоставляет утверждения для следующих типов данных: Object, String, Date, примитивные типы (boolean, int, char и т.д.), BigDecimal, Iterable, массивы Object, массивы примитивных типов, Map, Throwable, FileиInputStream, BufferedImage.

Начать использовать библиотеку очень легко — просто пишете assertThat() для тестируемого объекта, ставите точку, и любая интегрированная среда разработки Java покажет все доступные утверждения для типа объекта, который подлежит проверке. Нет больше путаницы с порядком «ожидаемых» и «фактических» значений, как и с читабельностью утверждений.

Среди удобств библиотеки можно выделить следующие: всего один import, автодополнение методов в IDE, более читаемый код, независимость от тестового фреймворка. Например, assertEquals() в JUnit и в TestNG имеет различный порядок аргументов. Если использовать Festassertions, то в случае необходимости вам будет проще перейти на другой тестовый фреймворк[12].

**Выводы.** Проанализировав все основные инструментарии, можно увидеть, что каждый продукт имеет свои достоинства, недостатки

 ISSN 1562-9945

и сферу использования. JUnit позволяет изолировать отдельные модули исходного кода программы и тестировать их независимо друг от друга, а также достаточно прост в использовании и имеет возможность расширения с помощью специальных правил и параметров запуска. TestNG похож на JUnit, но намного функциональнее него. TestNG позволяет запускать все тесты из одного xml-файла и в нем есть встроенная система отчетов. Гибкость TestNG особенно полезна при наличии больших наборов тестов. TestFXпредназначен для тестирования пользовательских интерфейсов, написанных на JavaFX. С помощью TestFX тесты будут написаны и выполнены быстро и при возникновении ошибок при тестировании последние буду сохранены в виде скриншотов. Библиотека Mockito, предназначенная для упрощения процесса создания mock-объектов, существенно упрощает разработку тестов, особенно для классов с внешними зависимостями. Библиотека FestAssertion не обязательна в использовании, но она позволяет легко писать и читать модульные тесты, написанные с помощью JUnitили TestNG.

## ЛИТЕРАТУРА

1. Академия Microsoft: Верификация программного обеспечения. Модульное тестирование — Режим доступу:
http://www.intuit.ru/studies/courses/1040/209/lecture/5394
2. Тестирование в Java. JUnit — Режим доступу:
http://habrahabr.ru/post/120101/
3. TestNG — Режим доступу: http://testng.org/doc/
4. XUnit — Режим доступу: https://ru.wikipedia.org/wiki/XUnit
5. Тестирование в Java. TestNG — Режим доступу:
https://habrahabr.ru/post/121234/
6. Bennet Schulz. Test JavaFX Apps with TestFX // Java magazine — 2015. — September/October 2015. — P. 20-25
7. Hamcrest — Режим доступу: https://en.wikipedia.org/wiki/Hamcrest
8. Unit tests with Mockito – Tutorial — Режим доступу: http://www.vogella.com/tutorials/Mockito/article.html
9. Vincent Massol, Ted Husted. JUnit in Action — USA: Manning — 2009. — 386 c.
10. JUnitMockito Example — Режим доступу:
http://examples.javacodegeeks.com/core-java/mockito/junit-mockito-example/
11. Fest Assertions 2.0 documentation — Режим доступу: https://github.com/alexruiz/fest-assert-2.x/wiki
12. Заметки о программировании. FestAssertion — Режим доступу: http://slava-semushin.blogspot.co.uk/2011/09/fest-assertions.html