УДК 004.9, 004.415.53

ПАТТЕРНЫ ТЕСТИРОВАНИЯ КАК ИНСТРУМЕНТ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

© 2022 Л. А. Сазанова

Уральский государственный экономический университет (Екатеринбург, Россия)

Статья посвящена автоматизации тестирования программного обеспечения. Актуальность темы связана с большим разнообразием существующих паттернов автотестирования, отличающихся особенностями использования и степенью востребованности. При проведении исследования использованы теоретические и эмпирические общенаучные методы: сбор данных, работа с электронными источниками, систематизация, обобщение и анализ. Представлена классификация подходов к тестированию производительности, выявлен ряд проблемных моментов в этой области. Также дана сравнительная характеристика наиболее популярных фреймворков, используемых в области автоматизации тестирования, их достоинств и недостатков. Отмечено, что автотестирование не являясь универсальным средством обеспечения качества программного продукта, представляет собой сложный процесс, нуждающийся в совершенствовании с учетом требований разработчиков, заказчиков и ИТ-сферы в целом.

Ключевые слова: *автоматизация тестирования, программное обеспечение, паттерн, шаблон, фреймворк, оптимизация, автотесты*.

Введение. Процесс тестирования является необходимой составляющей разработки любого современного программного обеспечения (далее – ПО). Под тестированием в программной инженерии понимается деятельность, выполняемая с целью оценки и улучшения качества ПО [1]. Оно базируется на обнаружении дефектов и проблем в программных системах и обеспечивает нахождение ошибок как в коде, так и в постановках задач на разработку. Существует ряд направлений тестирования в зависимости от его назначения [2-4]; в частности, по степени автоматизации выделяют ручное и автоматизированное тестирование [5]. При этом выбор конкретного метода и его применение с учетом требований к разработке ПО представляет собой нетривиальную задачу. Данное исследование посвящено анализу особенностей автотестов и проблем, связанных с их использованием.

Разновидности автотестирования, их достоинства и недостатки. В ходе автоматизированного тестирования ПО основные этапы тестов реализуются посредством применения программ, осуществляющих запуск, создание, отладку, выполнение и анализ результатов прогона т. н. тест-скриптов. Автотестирование весьма эффективно при наличии большого числа рутинных операций, например, когда используются формы с множеством полей для заполнения, сохранения и проверки, или, когда рассматриваемый процесс требует проведения точных математических расчетов. Однако, выбор и реализация оптимального подхода к автоматизации процесса тестирования представляет существенную проблему в силу разнообразия требований к создаваемому ПО, а также из-за обилия самих подходов [6]. В частности, одним из наиболее актуальных является т. н. *тестирование производительности*, выполняемое с целью выяснения, насколько корректно тестируемая система работает с точки зрения скорости и стабильности при определенной рабочей нагрузке. Основные направления в области тестирования производительности представлены в таблице 1.

Сазанова Лариса Анатольевна – Уральский государственный экономический университет, канд. физ.-мат. наук, e-mail: sazanovalarisa@rambler.ru.

Таблица 1

Направления тестирования производительности

Тестирование производительности	нагрузочное
	стресс-тестирование
	тестирование стабильности
	конфигурационное

Наиболее простым из представленных выше является нагрузочное тестирование. Оно производится с целью изучения поведения системы в условиях ожидаемой нагрузки. Для выявления надежности, стабильности и отказоустойчивости особо важных частей ПО при большой нагрузке используется обычно стресс-тестирование. В ходе тестирования стабильности отслеживается использование памяти, что немаловажно при обнаружении снижения производительности. Наконец, конфигурационное тестирование служит для оценки эффекта влияния на производительность изменений в конфигурации системы. Примером такого тестирования является эксперимент по балансировке нагрузки. Этот тип тестирования может сочетаться с остальными тремя типами.

Обеспечивая возможность проверки приложений после внесения изменений, автоматизация тестирования способствует повышению эффективности и охвата тестированием разрабатываемого ПО благодаря уменьшению времени на получение отчета о результатах проверки и экономии человеческих ресурсов. Качественное автотестирование добавляет ценность всем заинтересованным сторонам разработчикам, заказчикам, конечным пользователям. Улучшая имидж бренда и обеспечивая более высокий доход потребителю ПО, оно косвенно способствует росту инвестиций в исследования продуктов и инновационные процессы. С другой стороны, автотесты не лишены ряда недостатков, которые следует учитывать и по возможности минимизировать. Во-первых, это «хрупкость» автотестов: они могут не давать требуемого результата из-за ряда факторов, таких, как незначительное изменение UI-интерфейса, «падение» сервиса, проблемы с сетью, загруженность тестовой машины. Во-вторых, написание автотестов и поддержание их в актуальном состоянии требует временных затрат. Заметим, что автоматизация не является сама по себе тестированием. Автотесты – это лишь запрограммированные шаги для выполнения заданного сценария. Многие, кто впервые сталкивается с автоматизацией, стремятся автоматизировать все тест-кейсы, чтобы полностью избавиться от тестирования вручную, что невозможно, т. к. существуют проблемы, с которыми автотесты не справляются.

Паттерны проектирования автотестов и условия их применения. Паттерны или шаблоны автоматического тестирования способны воспроизводить определенные, предварительно записанные действия, сравнивать результаты с ожидаемым поведением и сообщать об успехе или неудаче инженеру. Они позволяют получить решения для повторяющихся проблем, адаптируемые к потребностям разработчиков. При этом одной из основных проблем их использования по-прежнему остается необходимость получения точных спецификаций. Только тогда можно оценить, насколько шаблон соответствует тестируемой системе. Паттерн проектирования – это часто встречающееся решение определённой проблемы при проектировании архитектуры программ [7]. В отличие от готовых функций или библиотек, паттерн нельзя взять и скопировать в программу, поскольку это не конкретный код, а общая концепция решения проблемы; его еще нужно «подстроить» под нужды программы. Описание паттерна обычно включает: указание решаемой им задачи, обоснование выбранного способа решения, описание структур классов, составляющих решение, пример на одном из языков программирования, перечень особенностей реализации и связи с другими паттернами.

Вторая проблема создания и использования паттернов связана со сложностью используемых в них структур. Ряд шаблонов используют динамические функции (связывание, типизация), что делает процесс тестирования с их применением «непрозрачным» и подверженным ошибкам. С ростом скорости разработки сокращение времени, отведенного на тестирование, при одновременном достижении максимально возможного

покрытия неисправностей, становится все более трудно решаемой задачей. Для одновременного достижения указанных целей – качественного тестирования с покрытием максимального числа неисправностей, наиболее эффективной является автоматическая генерация тестовых наборов. При этом, в свою очередь, необходимо обеспечить координацию между алгоритмами (например, между упорядочиванием тестов и управлением потоком тестовых данных), что позволит избежать избыточности и обеспечить компромиссы между различными шаблонами. Существуют алгоритмы, фокусирующиеся на эффективном сокращении количества испытаний даже при использовании моделей с множественными неисправностями, и они не очень просты в реализации [8].

Наряду с проблемами, нельзя не отметить ряд важных преимуществ, получаемых при использовании паттернов тестирования. Основные аспекты процесса тестирования, в направлении которых наблюдаются значительные эффекты улучшения при использовании автотестов, отражены на рисунке.

Рисунок. Эффекты от автоматизации процесса тестирования

Существует достаточно много паттернов, которые могут решать комплексные проблемы, связанные не только с кодом программного обеспечения, но и с поведением системы в целом.

Стратегии создания и особенности использования паттернов тестирования. Выделяют две наиболее популярных стратегии создания автотестов [9]. Первая представляет собой моделирование поведения приложения, реализуемое путем создания простых объектов страниц, отображающих части тестируемого ПО (например, создание объекта главной страницы либо страницы для авторизации). Такой паттерн носит название Page object. Он отражает известный в проектировании принцип единой ответственности, обеспечивающий автоматическое и быстрое внесение изменений в тестовый код и получение понятного, читаемого интерфейса прикладного программирования (API) для тестирования. Этот шаблон также придерживается популярной практики разработки ПО – принципа DRY [5], согласно которому каждый фрагмент логики кодируется только один раз. Итоговым результатом применения данного паттерна является повышение устойчивости автоматических функциональных тестов.

Вторая стратегия состоит в применении принципов проектирования SOLID [10] к автоматизированному приемочному тестированию. Паттерн Screenplay берет объекты страницы и разбивает их на мелкие части, обеспечивая удобство и надежность. Еще одно существенное преимущество данного шаблона заключается в том, что он делает тестовые сценарии более читабельными. Сценарий использует идею пользователей, задач и целей, чтобы выразить тесты с точки зрения бизнеса, а не с точки зрения взаимодействия с системой.

Паттерны тестирования являются основой для написания кода автотестов. Исходя из их логики создано множество фреймворков, облегчающих автотестирование. В таблице 2 представлены результаты сравнительного анализа наиболее популярных фреймворков тестирования на языке Python, признанного в 2021 г. лучшим языком программирования [11].

Таблица 2

Лучшие фреймворки Python для автоматизации тестирования

Название фреймворка	Особенности	Преимущества	Недостатки
1	2	3	4
Robot Framework	Применим для разработки, основанной на приемочных тестах; использует подход на основе ключевых слов и тестовые библиотеки.	Поддерживает все ОС; упрощает создание примеров. Расширяемый, имеет много API; запускает параллельные тесты с помощью Selenium Grid.	Проблемы при создании отчетов в HTML (возможны короткие отчеты в формате xUnit). Нестабилен при параллельном тестировании. Не очень прост в освоении.
Pytest.	Применяется для всех видов тестирования ПО; имеет открытый исходный код, прост в освоении. Для его запуска требуется Python версии 3.5 или выше.	Позволяет писать тесты компактно, сохраняя значения внутри них и сообщая, какое выполнено, какое – нет. Расширяемый; имеет процедуры для минимизации ошибок; подходит для параллельного тестирования.	Использует специальные подпрограммы на Python, поэтому создает проблемы с совместимостью.
Behave.	Реализует методологию Agile, очень популярен; похож на SpecFlow и Cucumber. Поведение определяется набором методов.	Поведение системы выражено полуформальным языком, понятным разработчикам и аналитикам, что облегчает координацию команд.	Наиболее успешно работает только при тестировании по методу «черного ящика». Отсутствует возможность параллельного запуска тестов.
TestProject	Реализует облачные HTML-отчеты. Поддерживает Python версии 3.6 и выше, фреймворки Pytest и Unittest. Имеется кроссплатформенная поддержка для Mac, Windows, Linux и Docker.	Бесплатен. Легко автоматизирует тестирование мобильных, веб- и иных приложений с помощью открытого SDK. Актуален за счет стабильных версий драйверов Selenium/Appium.	Агент выполняет по одному тесту за раз, поэтому для параллельного тестирования потребуются агенты Docker. При работе в автономном режиме функции коллаборации из гибридного облака ограничены.
PyUnit (Unittest)	Является частью стандартной библиотеки Python. Удобен для начального обучения, а также при знакомстве с xUnit-фреймворками.	Простое и гибкое выполнение тест-кейсов; быстрая генерация отчетов о тестах в XML и unittest-sml-reporting.	Большое число шаблонного кода. Для именования используется camelCase, а не snake_case, характерный для Python.

Продолжение таблицы 2

1	2	3	4
Nose2	Преемник Nose, основанный на PyUnit, но с плагинами. Добавлены поддержка выполнения тестов и другие возможности.	Легок в освоении для начинающих. Имеет встроенные плагины; поддерживает параллельное тестирование; автоматически собирает тесты.	Не очень активная поддержка в сравнении с другими фреймворками и отсутствие развернутой документации.
Lettuce	BDD-фреймворк, основанный на Cucumber. Требует версии Python 2.7.14 и выше; подходит для black-box тестирования и небольших проектов.	Позволяет создавать тесты на естественном языке, совместим с Gherkin. Способен тестировать различные модели взаимодействия серверов и баз данных.	Не очень широкий функционал и недостаточно активная поддержка. Требует особо тесной коммуникации между всеми участниками проекта.
Testify	Создан как замена Unittest и Nose, но по сравнению с ними обладает расширенным функционалом. Прост для тех, кто знаком с Unittest.	Множество плагинов, простой синтаксис фикстур. Хорош для модульного, интеграционного и системного тестирования.	Сложная реализация параллельного тестирования и отсутствие развернутой документации.

Как видно из таблицы, особенности и цели тестирования существенно определяют используемый инструмент. Скажем, если требуется внедрить фреймворк, дающий возможность ручным тестировщикам и бизнес-аналитикам создавать автоматизированные тесты, подходящим решением будет Robot Framework. Если же предполагается использование небольших и модульных тестов, поддерживающих сложные сценарии, необходим Pytest и т. д. Также выбор фреймворка тестирования может зависеть от того, модульное или функциональное тестирование выполняется; имеется ли у команды технический бэкграунд и опыт в программировании и от многих других нюансов, определяющих требования к продукту и процессу его тестирования.

Заключение. Тестирование ПО – важный процесс, позволяющий отследить выполнение всех бизнес-сценариев и требований конечных пользователей, а также выявить и устранить возможные проблемы и дефекты ИТ-продуктов. Паттерны тестирования существенно ускоряют этот процесс, повышая эффективность и обеспечивая надежное качество ПО, соответствующее требованиями клиентов. Особенно это становится очевидно на больших проектах. Существует еще один эффект от развития методов тестирования ПО: оно, отвечая потребности в постоянном разрешении проблем, способствует усовершенствованию требований к качеству разработок и совершенствованию отраслевых стандартов в целом.

В то же время, автотестирование, не являясь единственным и универсальным инструментом в борьбе за качество конечного продукта, по-прежнему порождает ряд проблемных моментов. К ним относятся трудоемкость создания автотестов и их поддержка в работоспособном состоянии, требующая постоянных обновлений и, как следствие, привлечения ИТ-специалистов высокой квалификации. Также, сложность представляет собой и сама процедура отбора тестов, поскольку далеко не все ситуации легко поддаются автоматизации. Остаются трудности и в плане формализации процедур тестирования. Наибольшая польза от автоматизации тестирования ожидается при проверке функциональности ПО, тогда как визуальное тестирование эффективнее проводить вручную. Если проект в чем-то уникален, то тестирование будет представлять собой сложную неформализованную задачу. Все вышеперечисленные

моменты неизменно отражаются на стоимости конечного продукта. Программисты и тестировщики продолжают искать оптимальные подходы к решению задач в данной области, в том числе – при реализации методов гибкой разработки ПО [12]. Наконец, остаются и организационные проблемы: одни разработчики совершенствуют процессы тестирования в рамках своих проектов самостоятельно, другие осуществляют аудит и контроль качества тестирования с помощью экспертов-консультантов, стремясь обеспечить высокое качество результата и уложиться в сроки.

СПИСОК ИСТОЧНИКОВ

1. Классификация видов тестирования. – URL: https://qa-academy.by/qaacademy/news/klassifikaciya-vidov-testirovaniya/ (дата обращения: 03.07.2022).

2. Паттерны проектирования в автоматизации тестирования. – URL: https://habr.com/ru/company/jugru/blog/338836/ (дата обращения: 03.07.2022).

3. Артюхова А. С. Проблемы автоматизации тестирования и подходы к их решению. /А. С. Артюхова // Научное периодическое издание «CETERIS PARIBUS». – 2016. – № 7. – С. 5-9.

4. Основные аспекты создания скриптов для нагрузочного тестирования. – URL: http://www.protesting.ru/automation/practice/load_scripts_writing.html (дата обращения: 03.07.2022).

5. Паттерны проектирования в автоматизации тестирования. – URL: https://www.pvsm.ru/testirovanie/264567 (дата обращения: 03.07.2022).

6. Пышкин Е. В. Проблемы автоматизации приемочного тестирования программного обеспечения при использовании подхода «Разрабртка, управляемая описанием поведения» / Е. В. Пышкин // Научно-технические ведомости СПбГПУ. – 2012. – №6. – С.90-97.

7. Четыре лучших паттерна проектирования автоматизированного тестирования. – URL: https://habr.com/ru/company/otus/blog/493796/ (дата обращения: 03.07.2022).

8. Модульное тестирование ПО встроенных систем в среде Unity. – URL: https://club.shelek.ru/viewart.php?id=348 (дата обращения: 03.07.2022).

9. Восемь лучших фреймворков для тестирования с помощью Python в 2021 году. – URL: https://habr.com/ru/company/otus/blog/576760/ (дата обращения: 03.07.2022).

10. Лучшие паттерны для автоматизации тестирования. – URL: https://testmatick.com/ru/luchshie-patterny-dlya-avtomatizatsii-testirovaniya/ (дата обращения: 03.07.2022).

11. Самые популярные языки программирования 2021 года. – URL: https://blog.skillfactory.ru/samye-populyarnye-yazyki-programmirovaniya-2021-goda/ (дата обращения: 03.07.2022).

12. Проблемы с автоматизацией тестирования и современным QA. – URL: https://invest-map-nnov.com/problems-with-test-automation# (дата обращения: 03.07.2022).

TEST PATTERNS AS A SOFTWARE DEVELOPMENT TOOL

© 2022 L. A. Sazanova

Ural State University of Economics (Yekaterinburg, Russian Federation)

The article is devoted to software testing automation. The relevance of the topic is associated with a wide variety of existing autotesting patterns, which differ in their use and degree of demand. During the study, theoretical and empirical general scientific methods were used: data collection, work with electronic sources, systematization, generalization and analysis. A classification of approaches to performance testing is presented, a number of problematic issues in this area are identified. A comparative description of the most popular frameworks used in the field of test automation, their advantages and disadvantages is also given. It is noted that self-testing, not being a universal means of ensuring the quality of a software product, is a complex process that needs to be improved taking into account the requirements of developers, customers and the IT sector as a whole.

Keywords: test automation, software, pattern, template, framework, optimization, autotests.