УДК 004.415.532.3

О.Н. Пчельникова-Гротова
Московский авиационный институт
(Национальный исследовательский университет)

АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ СЦЕНАРИЕВ ИНТЕРФЕЙСА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В статье рассматриваются результаты разработки технологии автоматизированного тестирования сценариев интерфейса с целью повышения качества программного обеспечения. Приведены архитектура и инфраструктура предложенной технологии тестирования, а также инструменты и методы для ее реализации. Рассмотрен состав и содержание блоков фреймворка, а также сценарий запуска автотестов по разработанной технологии. Оценка результатов тестирования интерфейса по разработанной технологии показала ее высокую эффективность.

Автоматизация тестирования, программное обеспечение, тестирование интерфейса, гибкие методики разработки, регрессионное тестирование.

Автоматизация тестирования программного обеспечения (ПО) является важной задачей для разработчиков, поскольку позволяет, с одной стороны, значительно ускорить процесс тестирования и сократить время и затраты на разработку ПО в целом, а с другой – выявлять ошибки и проблемы, которые могут быть пропущены при ручном тестировании, что обеспечивает высокое качество программного продукта.

Одной из ключевых задач автоматизации тестирования, сложность которой связана, с одной стороны, с динамичностью его элементов, а с другой – с процессами Continuous Integration/Continuous Delivery (CI/CD), предусматривающими быструю и автоматическую сборку и развертывание ПО, является тестирование интерфейса [1]. В то же время для таких видов тестирования, как регрессионное тестирование и тестирование критических сценариев, автоматизация является обязательным процессом обеспечения качества ПО.

Для обеспечения качества ПО в условиях гибкой разработки технология автоматизации тестирования интерфейса должна обладать рядом свойств:

- гибкость и адаптивность,
- надежность,
- простота использования,
- поддержка распределенной командной разработки,
- возможность интеграции с другими инструментами разработки.

Одним из ключевых методов для обеспечения качества программного обеспечения является регрессионное тестирование [2], которое позволяет выявить потенциальные ошибки, возникающие в результате внесенных изменений в исходный код и исправления выявленных ранее ошибок (рис. 1). Поэтому необходимым условием для эффективного обеспечения качества программного продукта является автоматизация такого тестирования, которая дает возможность значительно ускорить процесс тестирования и повысить эффективность выявления ошибок.

Особенности тестирования критических сценариев интерфейса связаны с тем, что интерфейс является главным каналом взаимодействия системы с пользователем. Поэтому важно не только тестировать функциональность, но и учитывать различные типы пользователей и специфику их взаимодействия с системой.

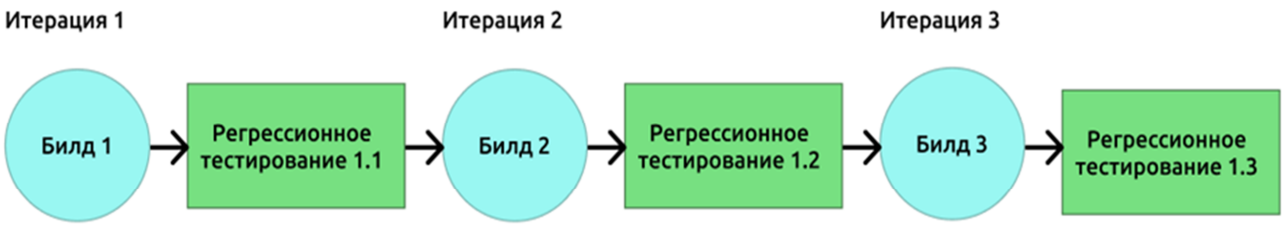

Рис. 1. Итеративность регрессионного тестирования

Рис. 2. Архитектура системы автоматизации тестирования

При автоматизации критических сценариев команды разработки и тестирования должны иметь возможность постоянного мониторинга состояния автотестов, а в случае срабатывания с ошибкой – возможность определять конкретную причину сбоя и оперативно устранять ее [3].

Архитектура системы автоматизации тестирования представляет собой структуру программного обеспечения, которая используется для разработки, создания и выполнения тестовых скриптов. Она включает в себя компоненты и модули, которые позволяют эффективно управлять и автоматизировать процесс тестирования, а также обеспечивать надежную и точную проверку функций и возможностей приложения.

В существующих архитектурах систем автоматизации для того, чтобы автоматизировать тест, в тестовом коде прописываются шаги взаимодействия с приложением, а затем итоговые данные извлекаются и сравниваются с эталоном [4]. Такая архитектура малоэффективна, поскольку с ее помощью невозможно проверить отдельный функционал интерфейса: на результат прохождения автотеста влияют любые расхождения с эталоном, а не только ошибки в работе проверяемых компонент. Кроме того, поскольку автотест обращается к приложению и содержит в себе шаги тестового сценария, любые изменения, внесенные в код приложения или функционал интерфейса, влекут ошибки срабатывания автотеста, что делает невозможным автоматизацию регрессионных тестов, которые необходимо запускать сразу после выхода новой сборки приложения, и тестирование критических сценариев, поскольку оно предполагает мониторинг их прохождения.

Для решения перечисленных проблем предложено изменить технологию автоматизации тестирования таким образом, чтобы код не взаимодействовал с тестируемым приложением напрямую, для чего тестовый сценарий (шаги теста, связанные с выполнением определенных действий в приложении) извлекается из автотеста и выполняется автономно (рис. 2).

Извлечение тестового сценария из автотеста требует написания нового фреймворка, представляющего собой набор библиотек и средств для автоматизации процесса тестирования [5].

В этом случае автотест осуществляет проверку только извлеченных из приложения данных исключительно по конкретному функционалу. Для идентификации причины ошибок срабатывания автотеста и определения некорректности работы проверяемого функционала необходимо проверять как входные, так и выходные условия.

Чтобы обеспечивать возможность добавления нового функционала без лишних затрат времени и усилий, позволять разрабатывать сложные проверки путем повторного использования небольших блоков тестового кода, обеспечить управление набором тестов, легкую настройку их по параметрам, а также группировку тестов по категориям и подкатегориям, фреймворк должен обладать гибкостью и расширяемостью, поддерживать стандартные форматы обмена данными, такие как XML и JSON, а также обеспечивать возможность запуска тестов параллельно и локально [6].

Был разработан фреймворк, отвечающий перечисленным требованиям, в состав которого входят несколько функциональных блоков:

1. Блок управления тестовым окружением предназначен для контроля и настройки компонентов, необходимых для проведения тестирования программного продукта (создание, настройка, конфигурация, мониторинг и управление тестовыми ресурсами и инфраструктурой).

2. Блок управления и обработки данных о входных и выходных условиях, извлеченных из тестируемого приложения в процессе выполнения тестового сценария (в качестве формата для хранения извлеченных данных выбран JSON – понятный, легко читаемый и расширяемый формат).

3. Блок управления и выполнения автотестов для локального запуска автотестов и проведения их отладки.

4. Блок формирования отчетов о выполнении тестов для отображения результатов запуска автотестов, их статуса и времени выполнения тестов.

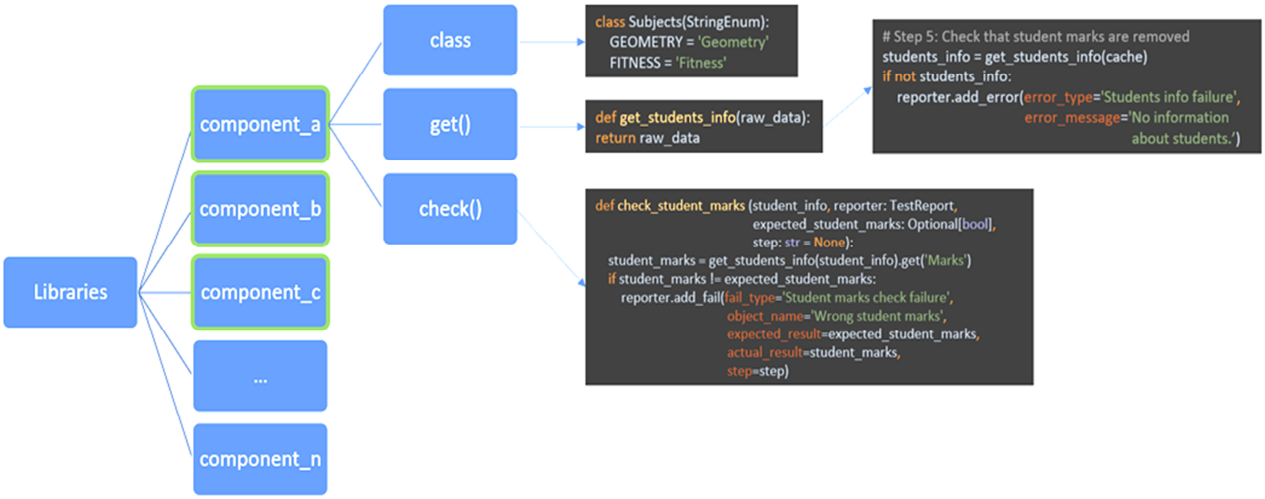

Рис. 3. Структура функционального блока управления и обработки тестовых данных фреймворка

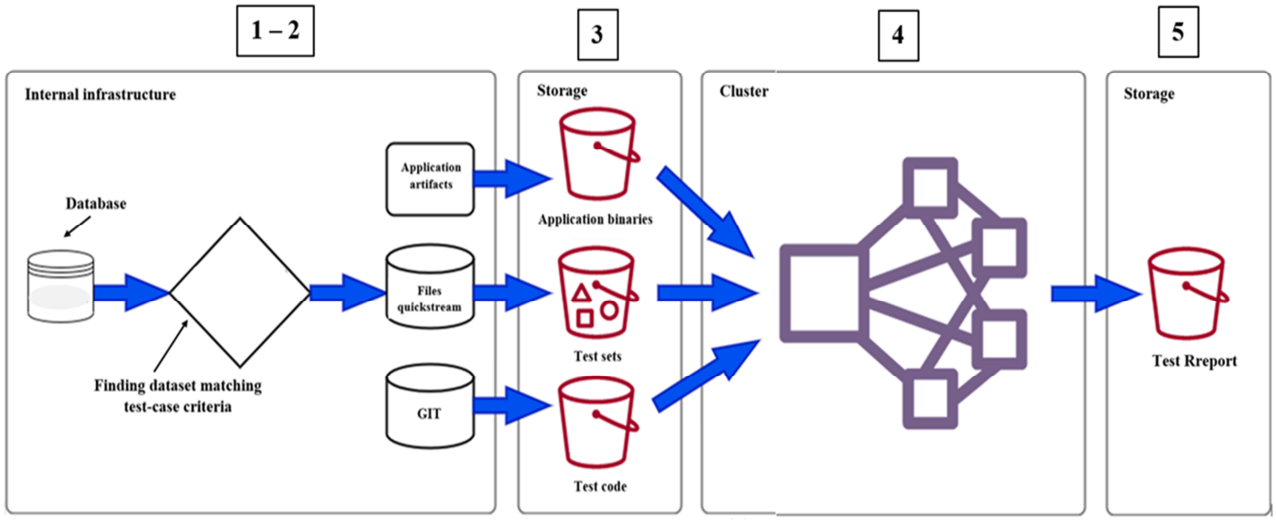

Рис. 4. Инфраструктура технологии автоматизации тестирования

Ключевым в созданном фреймворке является функциональный блок для управления и обработки тестовых данных (рис. 3). Поскольку тестируемое приложение является многокомпонентным, и за каждую компоненту отвечает отдельная команда разработки и отдельная команда тестирования, важной является группировка классов и методов по компонентам. Для каждой компоненты создаются наборы: классов class – шаблоны для создания объектов, описывающие свойства и методы объекта; функций для получения данных get() – функции обработки полученных данных для проверки с помощью программного кода; функций для проверки данных check() – функции подтверждения соблюдения требований или обнаружения ошибок.

Инфраструктура автоматизации тестирования представляет собой систему технических и программных инструментов, которые используются для оптимизации процесса тестирования, повышения производительности и минимизации рисков, связанных с введением новых изменений в программное обеспечение. В разработанной технологии тестирования использованы перечисленные ниже элементы инфраструктуры (рис. 4):

1. Внутренняя база данных, из которой извлекается датасет, соответствующий необходимым критериям.

2. Система контроля версий для совместной разработки автотестов.

3. Внешнее хранилище, в которое поставляется набор тестовых данных, данные об актуальной сборке приложения, тестовый код.

4. Кластер для параллельного запуска автотестов.

5. Внешнее хранилище для отчетов о выполненных автотестах.

Для организации внешней инфраструктуры была выбрана облачная платформа Amazon Web Services, которая предлагает широкий спектр услуг, включая хостинг, вычислительные мощности, хранилище данных, базы данных и другие ресурсы, которые могут быть использованы для разработки и доставки различных приложений и сервисов. Кроме того, использование AWS для параллельного запуска автотестов на кластерах предоставляет возможность значительной экономии на инфраструктурных решениях и

улучшает качество программного обеспечения за счет большего тестового покрытия [7].

Для включения автоматизации тестирования в CI/CD использовались инструменты Jira Software, Bitbucket, Bamboo и Confluence, поскольку они являются продуктами одного разработчика и уже частично интегрированы между собой. Интеграция между Jira и Bitbucket позволяет связать трекер задач и систему контроля версий, обеспечивает автоматическое создание и обновление баг-репортов в Jira, если происходят изменения в коде программы в Bitbucket, а также гарантирует прозрачность процесса разработки, что позволяет командам иметь более точное представление о текущем состоянии проекта, быстро выявлять проблемы и искать решения.

Интеграция Jira и Bamboo является необходимым процессом для улучшения производительности команды разработчиков и обеспечения высокого качества выпускаемого продукта. Она позволяет автоматизировать процессы сборки, тестирования и развертывания приложений, что уменьшает количество времени и трудозатрат на выполнение задач, а также своевременно выявлять ошибки и проблемы в процессе разработки.

Процесс интеграции инструментов происходит в следующем порядке:

1. Формирование в Jira задачи на автоматизацию мануального теста, в котором каждому шагу должна соответствовать проверка в автотесте, что помогает идентифицировать причину ошибок срабатывания автотеста.

2. Создание на Bitbucket запроса на добавление автотеста в основную ветку, и добавление автотеста после подтверждения от ответственных за компоненту лиц.

3. Компиляция на Bamboo актуальной версии программы, одновременно идет тестирование и проверяется отсутствие конфликтов в версиях.

4. Выбор из внутреннего хранилища нужной версии приложения, запускающей автотест.

5. Обработка автотеста в кластере на AWS с возможностью параллельного запуска.

6. Формирование отчета для хранения на внешней инфраструктуре. Отчеты по регрессионным тестам проверяются и перед очередным запуском найденные ошибки исправляются, что позволяет поддерживать качество приложения одновременно с его обновлением.

Рассмотренные инструменты автоматизированного тестирования поддерживают интеграцию с инструментами CI/CD – возможно передавать данные для тестов прямо в план действий (пайплайн) и запускать тесты поэтапно, получая результаты после каждого шага, а также переводить сборку на другой этап в зависимости от результата тестов на предыдущем этапе.

В соответствии с разработанной технологией автоматизированного тестирования для запуска автотеста подготовлены несколько файлов: файлы датасета, использующиеся при выполнении шагов теста в приложении; PYTHON-файл, содержащий код автотеста для выполнения проверки извлеченных из приложения данных; XML-файл со сценарием теста; BAT-файл, содержащий команду для запуска автотеста.

При запуске автотеста команды выполняются в следующей последовательности (рис. 5). Сначала команда из XML-файла сценария, по которой из датасета извлекаются исходные данные в формате JSON. Затем с помощью PYTHON-файла выполняется проверка исходных данных (в случае ошибки появляется сообщение о некорректных исходных данных и выполнение автотеста прерывается). Потом по сценарию (XML-файл) выполняются тестовые шаги, включающие взаимодействие с приложением и из датасета извлекаются итоговые данные в формате JSON. В заключение выполняется проверка итоговых данных (PYTHON-файл), и в случае ошибки появляется сообщение о некорректной работе проверяемого функционала.

Таким образом, в процессе тестирования исключается возможность появления «ложных» ошибок срабатывания автотеста, не связанных с ошибками в работе проверяемого функционала. Кроме того, разработанный подход к выполнению проверок позволяет выяснить конкретную причину ошибки срабатывания автотеста, что значительно сокращает время на ее исправление и повышает качество разрабатываемого ПО.

Апробация разработанной технологии проводилась на девяти автотестах (по 3 на компоненту приложения), в ходе которой оценивалось время ручного выполнения шагов теста, время автоматизированного выполнения автотеста, подтверждение стабильности выполнения автотеста. Результаты запуска тестов по разработанной технологии приведены в таблице.

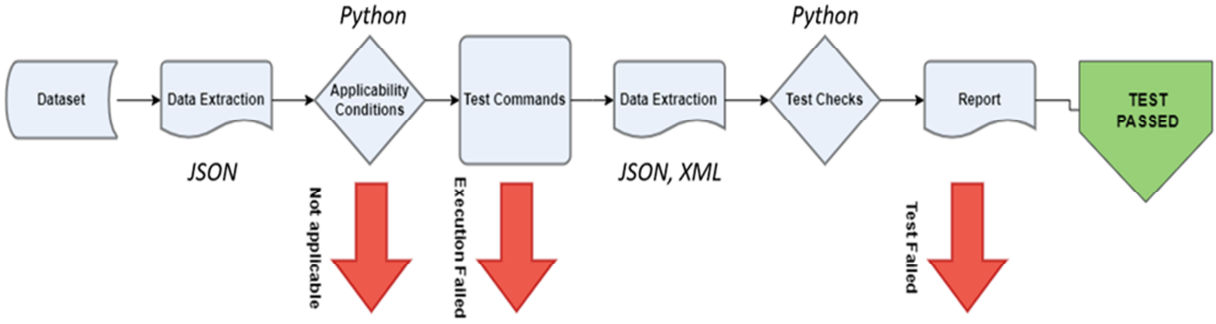

Рис. 5. Сценарий запуска автотеста

Таблица

Оценка результатов выполнения автотестов

Компонента	*№ теста*	*Выполнение*		*Стабильность*
		Ручное	*Автоматизированное*	
А	1	4 мин 29 сек	1 мин 27 сек	+
	2	7 мин 38 сек	2 мин 4 сек	+
	3	5 мин 36 сек	1 мин 22 сек	+
Б	4	9 мин 21 се	1 мин 59 сек	+
	5	11 мин 34 сек	1 мин 56 сек	+
	6	4 мин 23 сек	1 мин 23 сек	+
В	7	8 мин 11 сек	2 мин 13 сек	+
	8	3 мин 54 сек	0 мин 58 сек	+
	9	6 мин 18 сек	2 мин 1 сек	+
Итого:	**9**	**61 мин 24 сек**	**15 мин 23 сек**	**+**

По результатам запуска автотестов можно сделать вывод о сокращении времени тестирования в 4 раза при 100-процентной стабильности выполнения тестов и достижении высоких показателей обеспечения качества ПО.

Таким образом, предложена новая архитектура системы автоматизации тестирования, для реализации которой был разработан расширяемый фреймворк, поддерживающий форматы обмена данными XML и JSON. Выбранные инструменты и методы организации инфраструктуры автоматизации тестирования обеспечили совместимость разработанной технологии с CI/CD-процессом. Результаты запуска автотестов показали, что приведенная выше технология позволяет создавать стабильные и быстрые автотесты, соответствующие выявленным особенностям тестирования интерфейса.

Литература

1. Henry van Merode Continuous Integration (CI) and Continuous Delivery (CD) Apress Berkeley, CA Number of Pages: XIII, 422. – https://doi.org/10.1007/978-1-4842-9228-0.

2. Minhas N., Petersen K. Regression testing for large-scale embedded software development – Exploring the state of practice // Volume. – 2020. – № 120.

3. Хасанов, А. С., Шишкин, В. В. Автоматизация интеграционного тестирования программного обеспечения с повышенными требованиями к критичности // Вестник УлГЛУ. – 2021. – № 2. – С. 61–66.

4. Бойко, В. А. Архитектура интеллектуальной системы тестирования // Integral. – 2021. – № 2. – С. 347–352.

5. Jin Kim, Jez Humble The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations // Mann. – 2018. – С. 507.

6. Нутрудинов, Э. Э. Непрерывное развертывание ПО // Вестник науки. – 2022. – №5. – С. 103–107.

7. Брогзунова, Н. П. Обзор актуальных средств и методик разработки мобильных приложений // Наука и образование. – 2021. – № 4.

O.N. Pchelnikova-Grotova
Moscow Aviation Institute
(National Research University)

AUTOMATION OF SOFTWARE INTERFACE SCENARIOS TESTING

The article discusses the results of developing technology for automated testing of interface scripts in order to improve the quality of software. The architecture and infrastructure of the proposed testing technology, as well as tools and methods for its implementation, are presented. The composition and content of the sides of the framework, as well as the scenario for launching autotests using the developed technology, are considered. Evaluation of the results of interface testing using the developed technology showed its high efficiency.

Test automation, software, interface testing, flexible development methodologies, regression testing.