An approach to automating software product quality assurance using templates for all levels of the testing pyramid

Dmitriy Cherepovskiy
Institute of Computer Science and Technology
Peter the Great St.Petersburg Polytechnic University
Saint – Petersburg, Russian Federation
gdk1743@gmail.com

Pavel Drobintsev
Institute of Computer Science and Technology
Peter the Great St. Petersburg Polytechnic University
Saint – Petersburg, Russian Federation
drobintsev_pd@spbstu.ru

***Abstract* — software product quality assessment is an important activity in the software development lifecycle. Testing involves running a program on a specific set of test data and comparing the visible results with the expected results. Nowadays, IT companies are increasingly switching from manual testing to automated testing. Automation brings many benefits that speed up test execution time, increase the accuracy of the testing process, and minimize software support costs. However, the support and development of automation tools is not so simple and requires a certain qualification in the field of development and testing. In order to save money and time, SDET (Software Development Engineer in Test) specialists are increasingly developing tools that simplify the writing and support of automated test cases. This article presents a framework that solves the problem of different understanding of test design among the project team and provides detailed reporting on all types of automated tests.**

Keywords — framework, automated tests, templates, reporting.

I. Test Pyramid

A key concept in the automation of product quality assessment is the testing pyramid described by Mike Cohn. [1,2] The testing pyramid shown in Fig. 1 consists of three layers, which are arranged depending on the number of test cases and the goals of the testing.

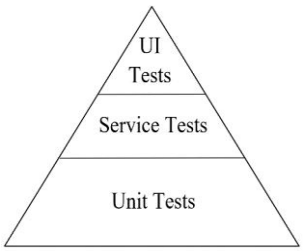

Fig. 1 Testing pyramid

- *User Interface Tests* – this type of test is responsible for the correct response of the system to the actions of the end user.
- *Service Tests* – this type of tests checks the availability of the web services that the application accesses. Most often, working with them is organized using the API requests.
- *Unit Tests* – this type of tests checks the behavior of a function or method in the source code.

When writing auto-tests, experts try to adhere to the basic rules that come from the testing pyramid:

Test execution speed as well as the tested objects isolation both decrease from the bottom up (from Unit to UI).

This is due to the simplification of support for automated cases, as well as optimization of the speed of execution of a certain test set.

When using the concept of the testing pyramid, most often for each type of test, its own reports are compiled, which is created by the test coverage analysis or by comparison of the number of written tests with the test base. This work is quite time-consuming and with a large number of test scenarios, some of them may be skipped or incorrectly compared with automated tests during manual analysis. The article describes a tool that generates testing reports at all levels of the pyramid and provides it to the project team in a single format.

II. Test-Driven Development

Within TDD [3], both testers and developers participate in writing auto-tests at all levels of the testing pyramid [4], since most tests are written at the initial stage of software development. This process is based on converting the requirements for the software product into test cases before the software is fully developed. An example of a TDD development cycle is shown in Fig. 2.

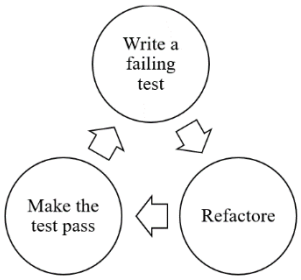

Fig. 2 Test-driven development cycle

It is when writing auto-tests and at the refactoring step that the problem of different understanding of test design is clearly visible, by developers who strive for the highest percentage of code coverage [5] and by testers who are responsible for the quality of the product and think through test scenarios to test the main functionality [6]. To solve this problem, STED specialists resort to using pre-developed tools-frameworks, with a ready-made architecture and the structure of automated tests. However, re-development of the framework and training the specialists involved in writing tests is very expensive work, which requires a huge amount of time and effort. To solve this problem, the article describes

a framework, the feature of which is the use of templates based on design patterns in test automation [7].

III. Test Automation Framework

Currently, testing automation is treated as a project under development, which should easily adapt to several products by any software company the company, for this it is necessary to consider all the nuances of testing software products, to this end, ISTQB (International Software Testing Qualifications Board) has described the general architecture of the test automation tool [8], which are shown in Fig. 3.

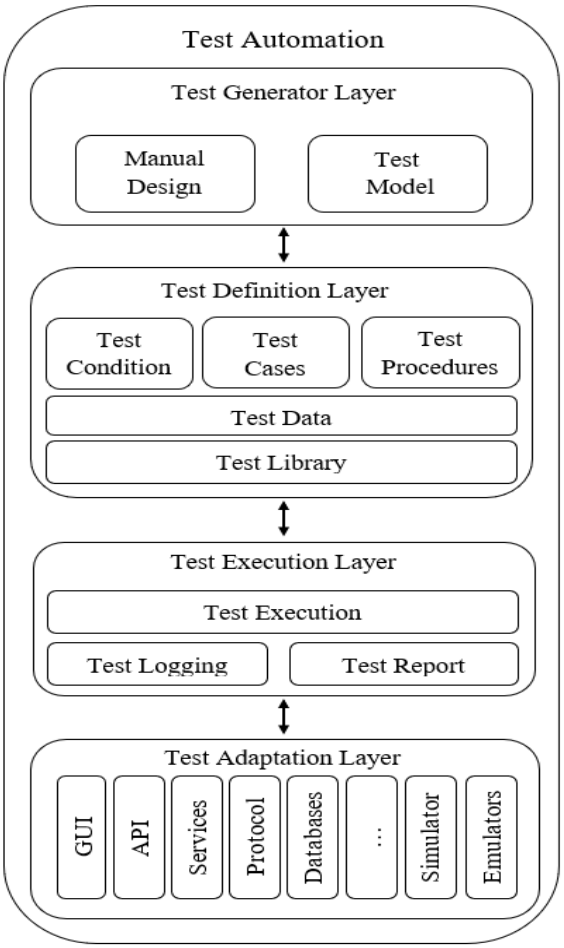

Fig. 3 General test automation architecture

- Test General Layer – a layer that supports manual or automatic design of test suites and test cases. Within this level of test automation, the language for writing tests (Python/Java/Gherkin) is selected.
- Test Definition Layer – the layer responsible for implementing the test, supporting specific test suites and test cases, for example, it describes templates or test principles.
- Test Execution – the layer responsible for finding and running tests, collecting reports and logs, and integrating with external tools.
- Test Adaptation Layer – the interaction layer with the system under test which includes:
    - Abstraction over GUI technologies: Web, Mobile, Desktop, Image Recognition.
    - Abstraction over communication protocols: HTTP, AMQP, SSH, JDBC, etc.

In this article, special attention is paid to the layer that includes testing reporting (Test Execution) and the layer responsible for implementing tests (Test Definition).

IV. Analysis of Existing Testing Auyomation Tools

To solve the problems mentioned earlier, the most popular tools for testing automation [9-12] where analyzed, the results of the analysis are shown in Table 1.

TABLE I. Results of the Analysis of Tools

Tools	Selenium	Katalon Studio	UFT	Test Complete
Positive aspects	1. Support for a large number of programming languages 2. The tool is provided free of charge 3. The tool is cross-platform.	1. Simplified script writing 2. The tool is free of charge 3. CI Support 4. Structure of the auto test project	1. A wide range of tested applications 2. Simplified test cases writing	1. Support for a large number of programming languages 2. A wide range of tested applications
Negative aspects	1. Support for Web Apps only 2. Lack of templates/structure	1. Support for a small number of programming languages 2. Lack of detailed reports	1. The tool is paid for 2. Support for a small number of programming languages 3. Lack of detailed reports 4. The tool is not cross-platform	1. The tool is not cross-platform

Among the main disadvantages of the considered tools, the most important ones can be identified:

1. Problem with automatic test report generation.
2. Lack of cross-platform compatibility.
3. Support for a small number of languages.

We attempted to design and implement a new testing automation tool to overcome problems mentioned above.

None of the considered tools provide testing across the entire test pyramid, which is a significant disadvantage because these tools are designed for use only by the testing team.

V. Implementation Description

The developed tool for testing automation is based on the following scheme, shown in Fig. 4.

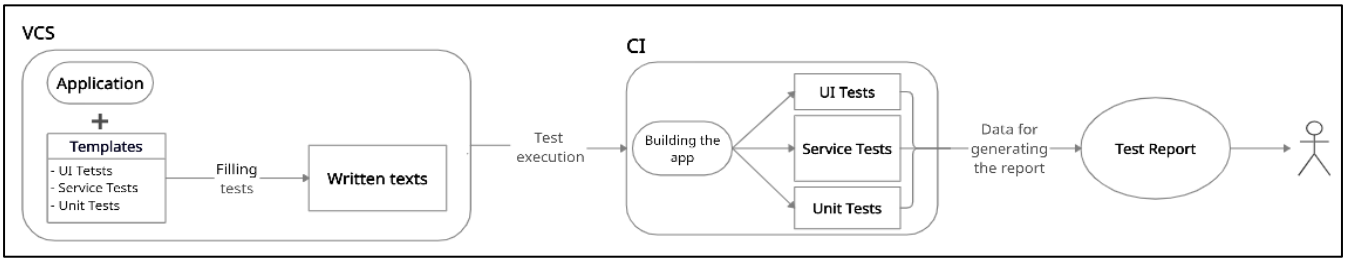

Fig. 5 The general scheme of the developed framework

The developed framework integrates the structure and templates necessary for writing tests into the application source code, which is then filled in by team members. At the stage of building the application, automated tests are performed, as a result of which reports on all types of testing are generated based on the received data. The report is deployed to a web server, and a link to view it is sent to the project team.

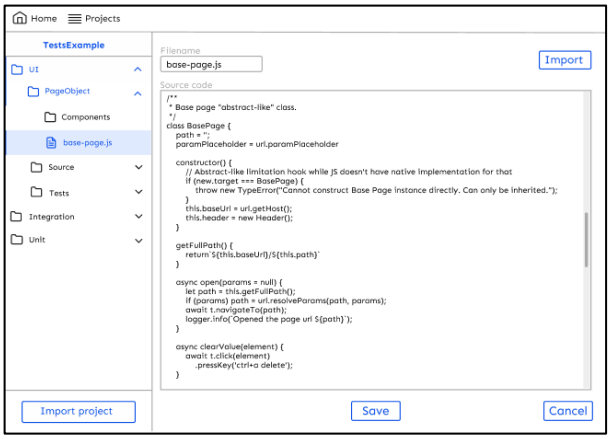

Fig. 4 Example of an interface for creating or editing templates

At the initial stage, a person with knowledge and skills in all areas of development, testing, and the product domain creates a project in the user interface of the tool, views and makes changes to standard automated testing templates stored in the database, or imports pre-prepared templates into the project (Fig. 5.). Templates include the following:

TABLE II. CONTENT OF AUTOMATED TESTS TEMPLATES

Unit Tests	Service Tests	User Interface Tests
1. Automatic generation of test structure. 2. Connecting the main libraries and framework needed for writing tests. 3. Arrangement of annotations in the test source code. 4. Specifies which methods to test.	1. Automatic generation of the test structure. 2. Connecting libraries and frameworks. 2. General functions required for tests (helpers). 3. Creating data in the application database for tests. 4. Environment variables for the test environment. 5. A class with query methods (a wrapper for the API). 6, Arrangement of annotations in the test source code.	1. Automatic generation of the test structure. 2. Classes with common methods for simulating users. 3. A class that describes WebDriver (a wrapper over Selenium [9]). 4. Arrangement of annotations in the test source code. 5. Example Page Object Model [7]. 6. Create data in the application database for tests. Environment variables for the test environment.

After filling in the templates, the framework generates a test structure and creates a project to automate testing. The scheme of the project generator is shown in Fig. 6.

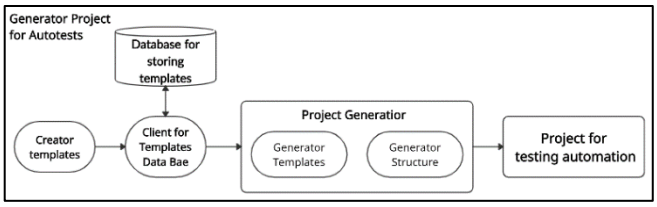

Fig. 6 Project generator diagram for test automation

In the generated project, a team of developers and testers write tests according to prepared rules and run them on the desired part of the development process through Test Runner – bash scripts that are responsible for running tests on different environments and with different settings. Through the API of the test base-Client for Test Base, test plans are automatically analyzed and checked with automated cases by name. Later, information about the test coverage is stored in temporary files.

During the test run, the data about the passing of the tests is also stored in temporary files, Generator Reports generates a report on all the collected information. At the final stage, Worker Reports raises the web server and deploys the report on it. The link to the report is sent as an email to the email address specified in the mailing list via Mail Sender. A detailed implementation diagram of the module for running tests with report generation is shown in Fig. 7

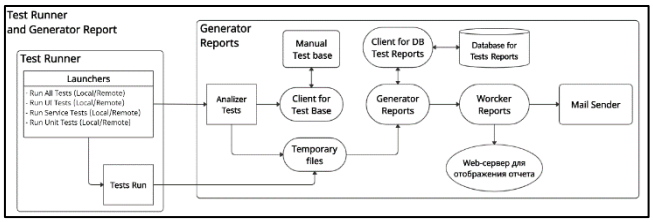

Fig. 7 Diagram of the module for running auto-tests and generating a test report

To create reports on the coverage of automated testing, use the test database, which contains the test IDs or test names. Using the API (Application Programming Interface) provided during the development of the test tool, we get the number of all test cases in the test plan and the number of automated test cases. Based on the data obtained from the test database, we calculate the percentage of test coverage, for example, for the user interface. For unit tests, coverage information can be taken from temporary files using the flag to evaluate coverage when running a test run. To display the report, we used the most popular Allure framework [13], which can be adapted to your own needs. When creating a report, a pre-prepared json

file with coverage data is used [14]. The data from the file is then simply inserted into the original report template. An example of the report is shown in Fig. 8.

Fig. 8 Example of a report with coverage assessment

VI. CONCLUSION

In conclusion, it is worth noting that this tool is designed for easy integration into the application with the possibility of further refinement of reporting and test templates. The main advantage of the tool is its flexibility and the ability to use it within different projects with different processes, due to the ability to develop automated tests not only by the testing team, but also to provide an opportunity for developers to participate in the testing process. Using this framework will reduce the development time of automated tests at an early stage of software application design, eliminate the problem of frequent code refactoring, and provide complete, automatically collected product quality reporting to all stakeholders.

In the future, we plan to provide an opportunity to write automated scripts to specialists who do not have programming skills. Adding modules to the tool to bind the prepared templates to specific commands. This feature will speed up the testing process by allowing manual testers, analysts, and managers to write and edit test cases themselves.

REFERENCES

[1] H. Vocke, "The Practical Test Pyramid", February 26, 2018 [Electronic resource]. Available in: https://martinfowler.com/articles/practical-test-pyramid.html.

[2] Mike Cohn "Success with Agile: Software development using Scrum (Addison-Wesley Signature Series)" - Moscow: "Williams", 2011. - p. 576. - ISBN 978-5-8459-1731-7.

[3] A. Xenakis, F. Foukalas «Cross-layer aware TDD Frame adaptation for FDD/TDD carrier aggregation in LTE-A System» ACM International conference proceeding series, DOI: 10.1145/3139367.3139370

[4] Y. Wang, «Test Automation Maturity Assessment», 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), 2018.

[5] Voinov N. V., Drobintsev P. D., Kotlyarov V. P., Nikiforov I. V., Selin I. A., Test set generation based on control flow, Systems and computer science tools. 2015. Vol. 25. No. 1. pp. 54-73.

[6] Voinov N. V., Drobintsev P. D., Kotlyarov V. P., Nikiforov I. V., Analysis of the coverage of the UCM model with test scenarios, systems and computer science tools. 2015. Vol. 25. no. 1. pp. 74-88.

[7] N. Adimenkov, "Design Patterns in Test Automation," in HEISENBUG, 2017.

[8] ISTQB «Cert certified tester advanced level syllabus – test automation engineer», 21 OCT. 2016 Available: https://www.istqb.org/downloads/send/48-advanced-level-test-automation-engineer-documents/201-advanced-test-automation-engineer-syllabus-ga-2016.html

[9] Documentation "The Selenium Browser Automation Project" [Electronic resource]. Available: https://www.selenium.dev/documentation/en/.

[10] Documentation "The Katalon Studio" [Electronic resource] Available: https://docs.katalon.com/katalon-studio/docs/overview.html

[11] Documentation "Unified functional testing" [Electronic resource]. Available in: https://admhelp.microfocus.com/uft/en/15.0-15.0.2/UFT_Help/Content/User_Guide/Ch_UFT_Intro.htm

[12] Documentation "The Test Complete" [Electronic resource] Available: https://support.smartbear.com/testcomplete/docs/

[13] Documentation "Allure" [Electronic resource] Available: https://docs.qameta.io/allure/

[14] Eroshenko A., "Allure 2: New generation test reports" [Electronic resource] Available: https://habr.com/ru/company/jugru/blog/337386/