Section 2. Technical sciences in general

https://doi.org/10.29013/AJT-23-3.4-6-10

Bukarev Anton,
National Research University of Electronic Technology, Russia

IMPROVING MOBILE APP QUALITY THROUGH OPTIMIZED TESTING STRATEGIES

Abstract. This article explores the complexities and challenges associated with the testing process on many devices, particularly in the context of mobile applications. It highlights the importance of optimizing the testing process to ensure high-quality products and reduce time to market. The article also discusses alternative approaches to address these challenges, such as utilizing cloud services, automated testing, and integrating with CI/CD systems, which can improve efficiency and product quality.

Keywords: testing process, mobile applications, optimization, cloud services, automated testing, CI/CD integration.

Introduction

In the present study, the development and implementation of an optimized method for initiating automated tests utilizing Remote Procedure Call (RPC) is examined. The objective of this research is to establish an efficient and versatile system for automated software testing, which will facilitate a reduction in time and resources expended on test execution, while ensuring enhanced scalability and adaptability to diverse testing platforms and libraries.

Peculiarities of the Testing Process on many Devices

Ensuring high product quality in the market is inextricably linked to the proper organization of the testing process. Consider the case of a mobile application designed for warehouse management that utilizes a device's camera for recognizing product barcodes. There exists a multitude of mobile device manufacturers that differ in both hardware characteristics and software components. Each hardware and software combination may affect the camera and sensor performance; therefore, the developed application must be adapted for each case [1; 2].

Conducting software testing on all available devices and hardware-software combinations is impractical. As a result, the most popular devices in the market are typically selected and tested first. However, even such a sample may comprise 50–100 devices.

Performing local tests on such a quantity of devices entails several complexities. Local testing on multiple mobile devices requires reliable connection and management provisions. Connecting and managing many devices can prove to be a complicated and fragile process, necessitating substantial efforts and time for setup and maintaining stable operations. Furthermore, compatibility issues may arise between various devices and operating systems, complicating the connection and management process.

Under local conditions, using multiple mobile devices may result in device failures due to hardware malfunctions, power issues, overheating, or improper usage. Recovering devices after such failures can

be a complex and costly process, requiring additional expenditures on repair or equipment replacement.

Addressing the problems associated with local testing on multiple mobile devices may demand considerable time and financial resources. This may include equipment repair or replacement costs, technical support expenses, and time spent diagnosing and resolving issues. All these factors reduce testing efficiency and may lead to delays in software product development and release [3].

Given the complexity and costs involved in addressing the problems, developers and testers should consider alternative approaches to organizing the testing process. Possible solutions may include using cloud services for conducting testing on remote devices, applying automated tests, and integrating with CI/CD systems. These approaches can alleviate the team's workload, ensure flexibility, and reduce the time spent on testing, ultimately enhancing software product quality, and expediting its market launch.

Optimization of the Testing Process through the Application of RPC

To enhance the testing process, the following technical solution has been proposed. A task scheduler (figure 1) has been developed to control the testing process on remote devices. This solution enables the application of existing automated testing approaches without the need to make significant changes to the existing infrastructure. The task scheduler incorporates a module for remotely initiating tests and facilitating communication with them. In addition, a module for data exchange with the task scheduler is installed on cloud devices [4; 5].

In a test cloud, several devices of the same model may be present. In this context, the task scheduler is capable of uniformly distributing the set of test scenarios among devices of the same model or devices with identical hardware and software configurations.

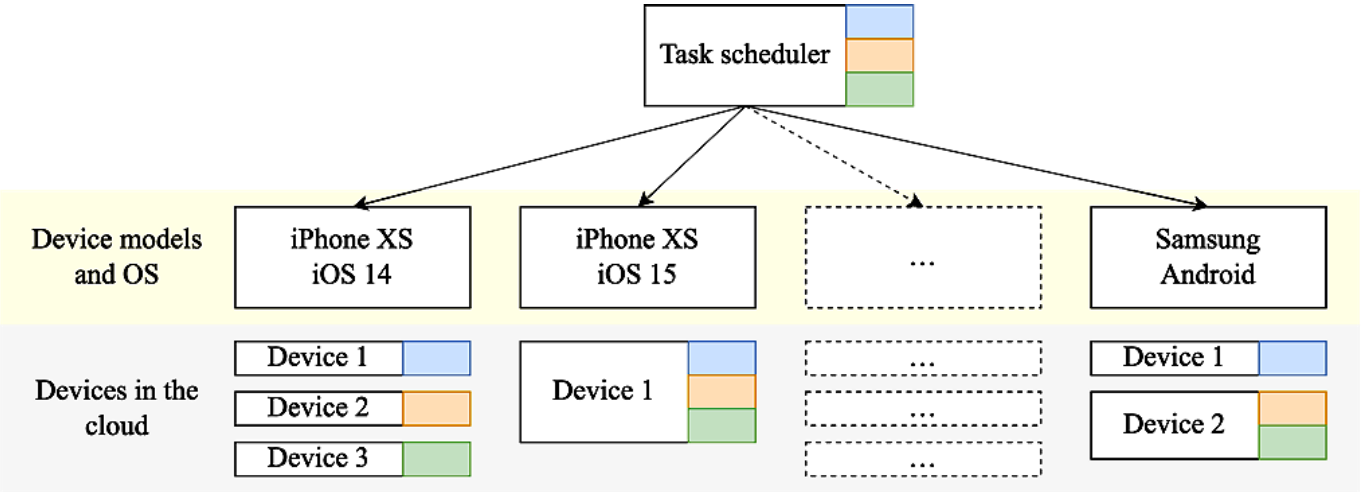

Figure 1. Test cases distribution between cloud devices

The test initiation module and the wrapper (figure 2) for the tested library are generated automatically. Consequently, the system creates a compatible interface for each library, significantly saving time in preparing the testing process, as this step occurs almost instantaneously. The proposed solution is cross-platform, allowing developers to describe test scenarios once, which are then executed uniformly across all tested platforms. Otherwise, it would be necessary to create a prototype application with user interface components for subsequent development of automated platform-dependent tests.

It consists of three main stages: initialization of the testing process, test execution, and generation of testing reports. These steps are discussed in more detail below.

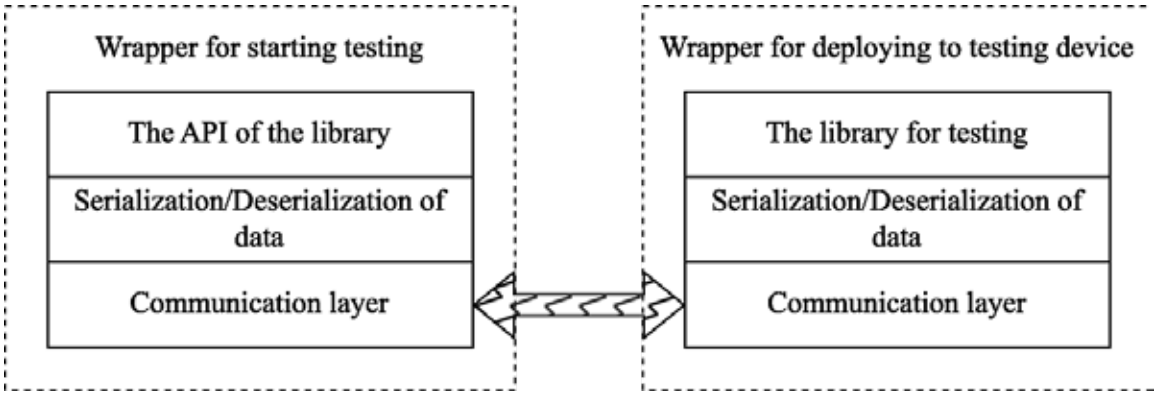

Figure 2. High level architecture of testing wrappers

The interaction process of the system components for initiating tests is represented in the diagram (figure 3).

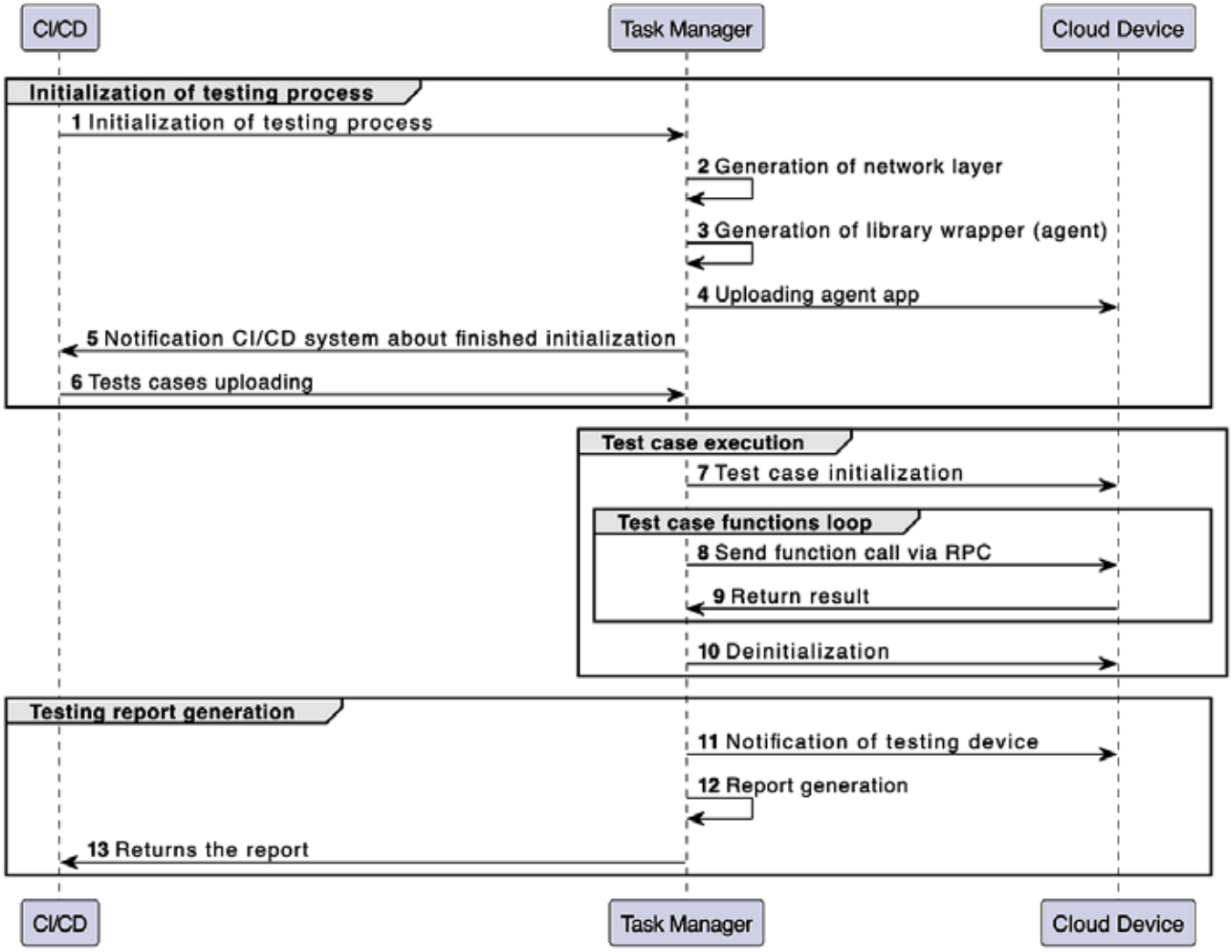

Figure 3. Diagram of communication process between CI/CD, Task Manager and Cloud device

During the testing process initialization stage, the CI/CD system activates a procedure by contacting the task scheduler. At this point, data on the tested library, including its software interface and binary code, are

transmitted. Based on the received information, the network layer and agent application containing the tested library and network layer for interaction with the scheduler are generated. The agent application is then loaded onto the test devices. Although only one device is depicted in the diagram, in practice, the number can be arbitrary. Ideally, no more than one hundred devices should be assigned to one scheduler instance. At the conclusion of this stage, the task scheduler informs the CI/CD server about the completion of the initialization process and readiness for the next stage [6].

The test execution stage involves receiving control commands from the CI/CD system and transmitting calls to agent applications in the cloud. Initially, the device prepares for the execution of the test scenario, for which the task scheduler sends corresponding initialization commands to the agent. Subsequently, a cycle of remote procedure calls begins, with the results being returned to the controlling server for analysis. Upon completion of this stage, the environment is deinitialized after the scenario has concluded. The system is ready for testing subsequent test scenarios.

The final stage is the test completion process. At this stage, the devices on which testing was conducted are deinitialized, followed by the generation of a test report and notification of the CI/CD service about the completion of the testing process.

During the testing process, results were obtained, as presented in the chart (figure 4).

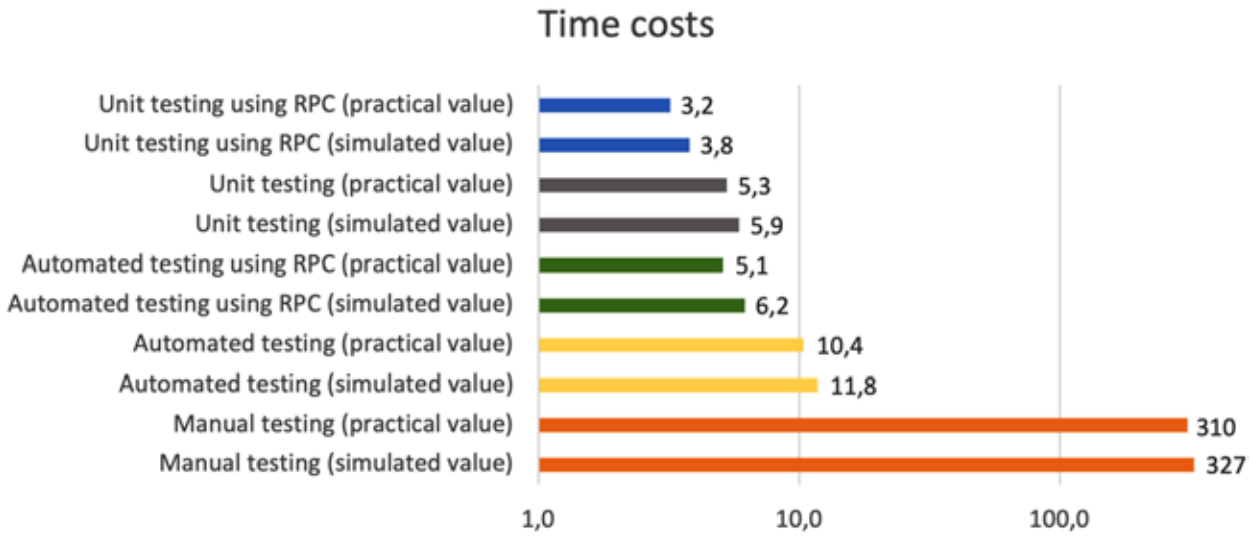

Figure 4. Time costs of testing methods

Time expenditure modeling was conducted for various testing methods, including manual testing, unit testing, automated testing, and enhanced versions of unit and automated testing using the RPC method. As expected, manual testing consumes the most time, due to the inherent characteristics of this approach. Measuring the time spent on manual testing enables an evaluation of the model's accuracy. According to the obtained results, the modeling error was approximately 16%. This can be attributed to the influence of human factors and technical factors previously discussed in the study [7; 8].

As for unit and automated testing, the difference between practical results and modeled values was not as significant. This attests to the high accuracy of the developed model and suggests that these results can be applied for further optimization of the testing process.

Now, let us focus on the results obtained using the RPC method and the conventional approach. Based on the graph, testing with the RPC method requires, on average, 45% less time for the given test environment configuration. However, it should be noted that results may vary for other testing

purposes and test environments. In this approach to testing, the complexity of the tested library's functionality and its volume play a significant role.

Conclusion

In conclusion, it can be stated that, throughout the practical experiment, a task scheduler was successfully developed and implemented, effectively managing the testing process on remote devices. This approach allows for the integration of existing automated testing methods without the need for making significant alterations to their structure. This ensures the optimization of the testing process and a substantial reduction in time expenditure.

References:

1. Gao C., Jiang S., Rong G. Software process simulation modeling: preliminary results from an updated systematic review, Proceedings of the 2014. International Confernce Software System Process – ICSSP 2014. URL: https://doi.org/10.1145/2600821.2600844
2. França B., Travassos G. Are we prepared for simulation based studies in software engineering yet? CLEI Electronic Journal, – 16. 2013.– 9 p.
3. Saremi R. A hybrid simulation model for crowdsourced software development, Proceedings of the Fifth International Workshop on Crowd Sourcing in Software Engineering, 2018.– P. 28–29.
4. Nassal A general framework for software project management simulation games, Proceedings of the Conference on Information Systems and Technology, 2014. URL: https://doi.org/10.1109/CISTI.2014.6877074
5. Lin C. T., Li Y. F. Rate-based queueing simulation model of open source software debugging activities, IEEE Trans. Softw. Eng. (2014). URL: https://doi.org/10.1109/TSE.2014.2354032
6. Uzzafer M. A simulation model for strategic management process of software projects, J. Syst. Softw. (2013). URL: https://doi.org/10.1016/j.jss.2012.06.042
7. Mahanti Rupa, Neogi M. S. and Bhattacherjee Vandana. "Factors Affecting the Choice of Software Life Cycle Models in the Software IndustryAn Empirical Study," Journal of Computer Science (Science Publications),– P. 1253–1262. ISSN1549-3636, 2012.
8. Trivedi Prakriti, Ashwani Sharma. "A Comparative Study between Iterative Waterfall and Incremental Software Development Life Cycle Model for Optimizing the Resources using Computer Simulation". Information Management in the Knowledge Economy (IMKE), IEEE, 2013.– P. 188–194.