Unit Testing and Automate Security Testing

Roxana PRUTEANU
Faculty of Electronics, Telecommunications and Information Technology,
University POLITEHNICA of Bucharest, Romania
roxana.pruteanu@stud.etti.upb.ro

Abstract

In the current context, technology plays a crucial role in our lives, from the moment we wake up until the end of the day we interact directly or indirectly with this new world. Since it appeared, its purpose has been to come to the aid of humanity, to evolve in an efficient and effective way and with all that, it also represented an open door for people who used technology in an obscure way. The number of cases of cyber-attacks has increased exponentially, from data theft to the integrity of critical sectors (health, transport, energy, financial), every possibility was tried to be exploited, leading to serious consequences. Awareness is the first step towards safety, and further it is important how to use technology in order not to be the target of attacks, but also to stay informed and to become better persons in daily activities. This paper presents an analysis of automated testing for software applications, what it is, how automated testing is divided, the benefits brought by it, as well as unit testing details and some examples. Finally, automatic security testing is discussed, the most emerging web application security risks, suggestions about Android security testing tools and some automation frameworks. The focus is on creating an overview, differentiating between the terms used and exemplifying them.

Index terms: Automated Testing, Unit Testing, Development Testing, Dynamic Testing, Automated Security Testing

1. Introduction

To ensure the reliability of the software produced, a crucial term appears in software development: testing. Testing as part of Software Development Life Cycle is the phase where all activities that address the implications of producing quality products, work according to customer requirements and, although it does not add quality to the final product, testing contributes to a high-quality product. The smallest part that builds the software quality testing base is unit testing, and from this point a good quality program is built, saving time and costs. But unit tests can be written/executed manually as any regular program, however, living in the age of speed, unit tests acquired a new, more advantageous form, that of automated tests. In the following topic, automatic testing will be explained to understand its role at present, the advantages it comes with, but also the influence in the other test blocks. Then, unit testing will be analyzed, steps in a process of running unit tests, an example of a piece of the production code and an example of a false unit test.

Last but not least, a rather important topic in the current context, which grabs attention is security automation and testing. In this section, security testing is presented as a vital part of the quality assurance process, the terms are exemplified to create an improved overall image, the top ten web application security risks were presented, suggestion of Android security tools and the automation frameworks that may help in various kinds of security testing scenario. And finally, the paper ends with conclusions in Section 4.

2. Automated testing for software applications

Software testing is one of, if not the most critical phase of the Software Development Life Cycle and in this phase are measured the quality assurance criteria and the Key Performance Indicators already set for releasing software in the production environment. Software testing, as part of the Software Development Life Cycle, has seen exponential growth in both directions of types of testing and volumes of testing. Quality is often sacrificed in order to release software as quickly as possible. Testing has many benefits, including the confidence in the delivered code quality and increased application reliability [1]. All the activities of software testing can be conducted by two means: automated testing and manual testing. Manual testing is considered the fundamental software testing and is done without using automation tools or scripting [2].

One of the most dynamic parts of software testing is automated testing. Automation testing reduces both regression time and the overall costs. The most used automated testing types are Automated Unit Tests, Automated Functional and Non-Functional Tests, Automated Regression Tests, Automated Deployments. The automated testing will provide the following benefits: faster release cycles, extended test coverage, better code quality, improved reliability. Automated testing must be realized using proper tools that provide advanced capabilities for optimization, execution, and reporting [1]. The testing tool executes the test plan in order to assess both functional and non-functional attributes for the software iteration under test. The aim of automated testing is to reduce repetitive tests for a redundant action and to gain time, but is not a replacement for a manual testing.

Organization's success strongly rely on the quality of their products. To develop a product of good quality and without any critical/major defects within the cost and time constraints have become critical and implementing such products is a difficult task. Software testing is performed to evaluate correctness and functionality of software for assuring fulfillment of user requirements and the expected quality. "IEEE defines software testing as the process to evaluate the system or its components manually, or by automated means to determine whether it fulfills the user requirements, or to find the difference among actual result and expected result" [2]. Therefore, the purpose of software testing is to execute a software to identify defects or any missing features that were expected by the user requirements. If it is executed appropriately, software testing results in improved quality and effectiveness of the software system. The reduction of maintenance costs is done by detecting the defects in a software and removing those defects before the release of software.

Test cases describe the complete test scenario in terms of actions to be performed during testing, but when the automatic testing is performed, there is no manual navigation through the different parts of the application, the testing is conducted through some testing tool. Initially, only manual testing was performed, which did not ensured a good quality of software systems, and few defects may be ignored or unidentified through manual testing because of human errors. This situation led to the evolution of automatic testing, being useful in quicker testing process, recently many testers prefer to use automated testing for the variety of software systems. "The basic element behind automated testing is the automated testing tool that is used to conduct the tests" [2].

Due to the popularization of automated testing in software industries, the testing process become more effective, this one helps in easily executing various tests like performance testing and regression testing and many difficult testing activities got easier than before, as the automated testing evolved and improved. Automated software testing conducts the tests for various datasets and the tests can be executed repeatedly without human involvement, it can be performed in various phases (e.g. preparation of test plan or developing the test cases, selecting the testing tool, creation of the test script and finally executing the test by using the automated testing tool and the script). Testing automation results in improved efficiency, the main objective of automating software testing is to reduce the testing effort, costs and time, and to reduce human involvement in the testing process as

much as possible. "Automated testing supports the reusability of test scripts, using the testing tool, for different upgrades of the system under test. Automated testing has the following benefits:

- Tests are repeatable and reusable,
- Simplified regression testing,
- Reduces time and costs,
- Performance testing is possible due to simultaneous testing.

Automated testing has the following drawbacks:

- It is more expensive than manual testing,
- All areas cannot be automated,
- Manual testing cannot be fully discarded" [2].

2.1. Unit Testing

Viewed in the light of developer testing activities, testing is an activity performed to ensure correctness and quality of software (we can verify its functionality, measure progress while developing it) and this process start with unit testing. Unit testing is an integral part of development [3], and this type of testing is based on testing the smallest piece of code that can be logically isolated in a system, it is the easiest, fastest, and most consistent way to verify developers' assumptions about the code they produce [6]. According ISTQB (International Software Testing Qualification Board), unit tests are the first level of dynamic testing in the software testing process [3], is usually performed by developers, testers and QA engineers (they are written by developers and often involves a collaboration between these roles) and it helps uncover early bugs and flaws in application code [9]. Productivity is lower for software supported by tests, but it's kept constant over time, when they plot productivity versus time for software with and without tests. Initially, for software without unit tests, productivity is higher, but it plummets after a while and becomes negative [6]. According to the test pyramid, the higher the level, the higher the involved costs. This effect is multiplied because the involved cost increases involve:

- Setting up the tests,
- Maintenance of existing tests,
- The cost of test execution,
- Test result waiting time,
- Analysis and resolution time when a bug is detected [3].

The process of running unit tests consists of four steps:

1. Creating test cases: this stage requires writing multiple test cases for web applications,
2. Review and re-write: written test cases are reviewed and re-write if there are any mistakes,
3. Baseline: it is checked if each line of code is in a manner or another,
4. Execution: it is performed test execution using an online Selenium Grid [9].

To create a clearer picture, we have the following example which shows a unit test for a piece of the production code [10]:

```
import pytest
#Production Code
def str_len( theStr ):
   return len(theStr)
#A Unit Test
def test_string_length():
   testStr = "1"                    // Setup
   result = str_len(testStr)        // Action
   assert result == 1               // Assert
```

"The production code is the function that returns the length of past in string and the unit test is a single positive test case that verifies the length of one has returned for a string with one character in it. The test string length call is the unit test for the string length production code." This structure represents a common structure that all unit tests should follow being composed of three steps: **a setup step** where it creates a test string, **an action step** where calls a production code to perform the action that is under test, and the last one **an assertion step**, where the test validates results of the action [10].

After the unit tests are performed, the "higher-level tests" follow (integration tests, system tests, system integration tests, solution tests, acceptance tests) [6], but there is a family of tests that shares some characteristics with unit tests and some with higher-level tests, which tends to cause confusion. A common trait of such tests is that they're not unit tests although they seem, due to the fast execution time - in the range of 1 to 2 seconds - these tests they look deceivingly simply and are fast, but they're often integration tests, or even system tests. How do they make it into the unit test suite? The most plausible reasons are: they are not classified (if no attention is paid to how and when different tests are done, these tests end up in unexpected places), the test suite is small (if the test suite consists of a small number of unit tests it doesn't matter if some of them take a few extra seconds to run), hurry (for example, in the beginning of a project it is wanted to prove that the product has the potential to be commercially viable and in this stage, fast medium tests may live in the unit test suite). The easiest way to recognize these tests is to see a concrete example that have been appearing rather consistently in developers' projects over the years, named tests using In-memory databases.

Various SQL-compliant in-memory database implementations exist that are very fast, which not just only do they perform reads/writes much faster than databases that make use of disk storage, but they're also easier to set up, because they require virtually no installation and provide programmatic APIs for configuration. Databases are an important component for tests that require a data source and that validate vendor-agnostic functionalities. Let's take the case of the authentication because it is complicated and this is quite a good case, AuthenticationManager class uses the database correctly and that password hashing seems to work as expected, but nevertheless it is not a unit test. It loads classes, starts a database, and establishes a connection to it, at the time of writing, it ran in less than a second [13].

```
private Connection conn
def setupSpec() {
    Class.forName("org.hsqldb.jdbc.JDBCDriver")
    conn = DriverManager.getConnection("jdbc:hsqldb:mem:db", "SA", "")
    Sql.newInstance(conn).execute(
            "CREATE TABLE users(id BIGINT IDENTITY, " +
                    "name VARCHAR(45), "+
                    "password_hash VARCHAR(45))")
}
def "Authenticate user"() {
    given:
    Sql.newInstance(conn).execute("INSERT INTO users " +
        "(id, name, password_hash) VALUES (NULL, 'regular_user', '%ReG^7@R')")
    expect:
    new AuthenticationManager(conn).authenticate("regular_user", "secret")
}
```

Any application can host a multitude of opportunities to create tests similar to unit tests by running just as fast, but which in essence are not, they are part of the category of false unit tests. The least common denominator of these tests is that they all start a server somehow, but server took relatively little time to start, so waiting has been acceptable. One in-memory database test takes one second, ten of the same tests 2 seconds and combining this with other almost unit tests, the unit test suite will start getting sluggish. If the latency will pass a certain threshold, the tests will no longer be

executed, but leaving that aside, running these tests as unit tests is not a good idea, they make test suite more brittle and sensitive to environment settings [13][14].

Unit tests, even in the absence of test-driven development (TDD), are vital when writing new code (because they contain the highest amount of detail), their presence ensures that the code is testable, and they serve as specifications [6]. In those analyzed previously, the first safety net for catching bugs are unit tests, tests made in the production code, they build and run in the development environment [10].

3. Security automation and testing

Security testing is an integral part of the testing process, which verifies the software's vulnerability to cyber-attacks and tests the impact of malicious or unexpected inputs on its operations. These tests are evidence that systems and information are safe and reliable, and unauthorized entries are not accepted [4]. Returning to what was previously presented, about the dynamic part of software testing and its most used parts, automated functional and non-functional testing [1] represent subdivisions that cover different requirements regarding software testing. Dynamic testing includes functional testing, which focuses on what the software does, if software's functions are working properly, and non-functional testing which focuses on the design and correct configuration of the application (targets a solution's quality attributes, these being usability, reliability, maintainability, security, portability, etc. [6] [14]), security testing being part of non-functional testing [4].

For example, to differentiate functional from non-functional testing:

- A functional unit test: is considered a unit test of a sorting algorithm who verify that the input is indeed sorted,
- A non-functional unit test: the previous sorting algorithm it is subject to a unit test that times it to make sure that it runs within a specified time constraint.

Probably, this example seems not to be part of the present chapter, but this is aimed at differentiating terms according to context, security testing was described as part of non-functional testing, but the example can be extrapolated to security too. For example, a functional security test may be about a user who logging in nonprivileged, and attempting to do something in the system that only users with administrative privileges are allowed to do [14]. This example was created for a better understanding, security testing will be used further as part of non-functional testing.

The purpose of security automation is to reduce repeated manual testing and increase testing coverage in an efficient manner. According to OWASP, they presented top 10 web application security risks. "Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within organizations into one that produces more secure code" [8].

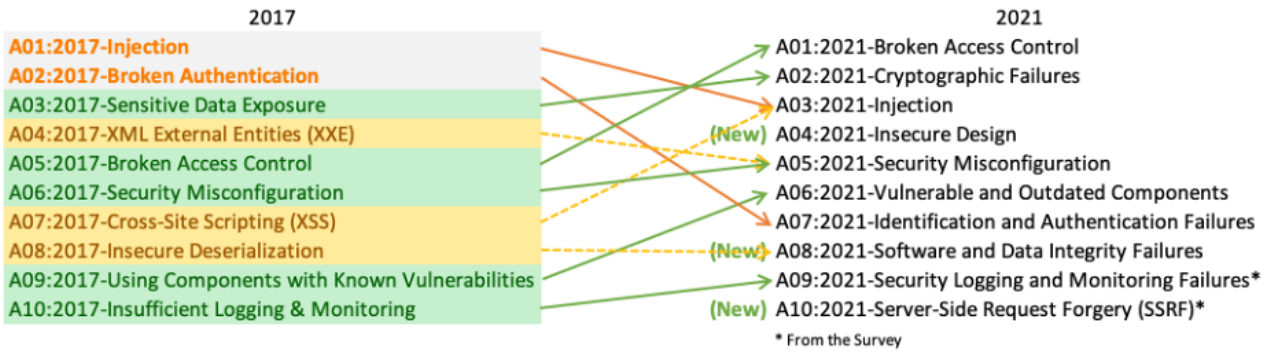

Fig. 1. Top 10 web application security risks [8]

Some categories have changed from the previous installment of the OWASP Top Ten, but it takes time to integrate these tests into tools and processes, and in the figure presented previously is a

high-level summary of the category changes. Top ten is an awareness document, it is the bare minimum and just a starting point for a coding or testing standard [8].

Next is presented a suggestion of Android security testing tools:

Table 1. Suggested Android security testing tools and approach [5]

Scanning approach	Automated tools	Description
Secure code scanning	Fireline	Static Java source code scanning. It's a light-weight secure code scanning tools, but it may require the Java source and the reverse of APK.
Privacy and sensitive information scan	Androwarm	The focus is on privacy and sensitive information scanning of any given APK. Static analysis of the application's Dalvik bytecode, represented as Smali for PII and sensitive information leakage or exfiltration such as telephony identifiers, geolocation information leakage, audio/video flow interception, and so on.
Light-weight all in one APK security scanning	Quick Android Review Kit (QARK)	It's a Python program that can automatically do security scanning of any given APK.
All in one security scanning	Mobile Security Framework (MobSF)	The MobSF is similar to QARK. In addition, MobSF supports Android, Windows, and iOS applications. It not only does the static security analysis, but also does dynamic runtime behavior analysis.

Android application security testing techniques include: the source code scan, privacy information inspection, reverse engineering of an APK, the adoption of automated security testing frameworks (for example QARK or MobSF) [5]. There are some key considerations when is required selecting security automation tools and these selected may depend on the integration of the existing automation testing framework [7]. In general, a security automation framework includes security testing tools, a web service, testing results, an automation framework (for example Robot Framework), automation scripts, and security payloads. "Security testing tools are in charge of testing for specific security vulnerabilities, such as cross-site scripting (XSS) and SQL injection, and also analyze HTTP responses for security issues, security testing tools may provide initial testing reports and testing results can be further integrated by either a testing framework" [12].

Automated UI testing may only cover the scenarios from a user perspective, while the system testing may cover more business logic and user expectations. The following table shows the automation frameworks that may help in various kinds of security testing scenario:

Table 2. Automation frameworks [11]

Types of automation frameworks	Usages
Web UI automation (Selenium or Robot Framework)	*User registration* flow *Authentication/authorization* flow Shopping cart and *Payment* flow *Forget password* flow PII (Personally identifiable information) – sensitive operations, such as *Profile update*
API testing (JMeter, Postman)	RESTful API testing with injection payloads
Behavior-driven development (BDD) testing (Robot Framework or Gauntlt)	When a BDD framework is applied to security testing, the purpose is to enhance cross-team communication and enable a non-security team to understand how security is covered
Fuzz Testing	Security payload testing with various injection and buffer-overflow testing
Data-driven testing (DDT)	DDT testing is normally done with fuzzy testing DDT is normally included in the unit testing framework of the programming language

"The Selenium Web UI framework is used to walk through the UI flow for security tools to inspect security issues, JMeter can be used with security payloads to do RESTful API security testing. Robot Framework can be integrated with ZAP to introduce BDD testing into the security testing cycle. Robot Framework is a common keyword driven testing acceptance automation framework, and Gauntlt is a purpose-built for security BDD framework in Ruby" [11].

To produce secure software systems, all aspects of the development environment must be in symbiosis, including the organizational culture, the tools used, the use of libraries and code repositories, the standards and processes used by the organization. The development environment can be considered a major source of potential vulnerabilities, so it is necessary that the development environment to be well-managed and secure in order to be able to produce secure, quality software systems, the end product being the image of the environment in which it was produced. The tools used to develop secure software systems need to be reviewed carefully to ensure that using the tool does not insert new vulnerabilities into the development process. The final results are directly affected by the quality of the tools used in the development process, and the process of choosing the necessary tools must be carefully approached [7]. Manual security testing are still important in creating a completely secure software (such as full penetration tests or security audits), but organizations must automate security testing and perform these for the most part, preferably with every change to applications or computing infrastructure. Security must be incorporated into every part of the development process to ensure a continuous integration (CI) and to reduce compliance costs [4]. Security requires a large part of our attention in the development process, being a key aspect in the quality assurance process.

4. Conclusions

In this paper were introduced notions such as automated testing, the most used types of automated testing, the benefits of automated testing, included in section 2 of this study. The automation process is in full development, it monopolizes most fields and especially in the creation of high-quality software is present. It was wanted to explain these notions as clearly as possible, and after that the basis of software development will be discussed in detail: unit testing. The notion of unit testing represents the key to starting software testing with purpose to ensure correctness and quality of software. In this section the purpose was to exemplify why it is important this type of test, how such a test is composed and how it differs from other higher-level tests. And the same with unit tests, which ensure quality in the testing environment, security testing aims with maintaining quality high. This process evidence that system and information are safe and reliable, and unauthorized entries are not accepted. In section 3 the subject is discussed at length, differentiates from the previously presented notions and some suggestions of Android security testing tools, a suite of automation frameworks and top ten web application security risks. The research will be continued by constructing more advanced cases scenario and an implementation of the notions discussed.

References

[1]. Flaviu Fuior, "An overview of some tools for automated testing of software applications," in Romanian Journal of Information Technology and Automatic Control, Vol. 29, No. 3, 97-106, 2019.

[2]. Haneen A., Maham K., Zainab S., Muhammad I.B., Saima C., Furkh Z., Muhammad J., Summiyah S., Shahid N.B., "A Comparative Analysis of Quality Assurance of Mobile Applications using Automated Testing Tools" in International Journal of Advanced Computer Science and Applications,Vol. 8, No.7, 2017.

[3]. Alexander Aubert. (2020, June 25th). "Why invest in unit testing?". Available: https://blog.atinternet.com/en/why-invest-in-unit-testing/.
[4]. Oliver Moradov. (2022, May 29th). "Security Testing: Types, Tools, and Best Practices". Available: https://brightsec.com/blog/security-testing/.
[5]. Tony Hsiang-Chih Hsu, "Android Security Testing" in Practical Security Automation and Testing, 2019, ch.7, Available: https://learning.oreilly.com/library/view/practical-security-automation/9781789802023/1f29016b-1353-4061-81c2-1a82a45d1ea6.xhtml.
[6]. Alexander Tarlinder, "Developer Testing Activities" in Developer Testing: Building Quality into Software, 2016, ch. 1, pp. 1-8.
[7]. Erik Fretheim, Marie Deschene, "The Development Environment", in Secure Software System, ch. 13, pp. 221-227.
[8]. Andrew van der Stock, Brian Glas, Neil Smithline, Torsten Gigler. (2021 September 24). "OWASP Top 10:2021", Available: https://owasp.org/Top10/.
[9]. LAMBDATEST, "Unit Testing Tutorial: A comprehensive Guide With Examples and Best Practices". Available: https://www.lambdatest.com/learning-hub/unit-testing#n.
[10]. Richard Wells. (2018, June 6th). "What is unit testing?", Available: https://www.linkedin.com/learning/unit-testing-and-test-driven-development-in-python/what-is-unit-testing?autoplay=true&u=2037052.
[11]. Tony Hsiang-Chih Hsu, "Automating existing security testing" in Practical Security Automation and Testing, 2019, ch. 2, Available: https://learning.oreilly.com/library/view/practical-security-automation/9781789802023/f3a726d8-f7a4-4cd0-a0c4-fe694e30bb69.xhtml.
[12]. Tony Hsiang-Chih Hsu, "Project Background And Automation Approach" in Practical Security Automation and Testing, 2019, ch. 10, Available: https://learning.oreilly.com/library/view/practical-security-automation/9781789802023/34cbdd6a-f26f-4ca0-8c83-a2f125615967.xhtml.
[13]. Alexander Tarlinder, "Almost Unit Tests" in Developer Testing: Building Quality into Software, 2016, ch. 11, pp. 151-157.
[14]. Alexander Tarlinder, "The Testing Vocabulary" in Developer Testing: Building Quality into Software, 2016, ch. 3, pp. 21-36.