International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi :https://doi.org/10.32628/IJSRCSEIT

Software Testing Best Practices in Large-Scale Projects

Mouna Mothey
Independent Researcher, USA

Article Info
Publication Issue :
Volume 3, Issue 6
July-August-2018

Page Number : 712-721

Article History
Received: 12/07/2018
Accepted: 19/08/2018
Published: 30/08/2018

ABSTRACT
Innovative and long-lasting testing techniques are required as the difficulties of testing software programs rise in tandem with their complexity and scope. —A crucial step in the software development process is software testing. Unfortunately, despite testing efforts, defects still plague many projects, and testing still consumes a large amount of time and money. Software testing offers a way to lower the system's total cost and mistake rate. To improve software quality, a variety of software testing approaches, strategies, and tools are available. Given its importance in both the earlier and later creation phases, software validation is an essential component of the life cycle of software development. Should be supported by improved and effective processes and procedures. This article offers a brief overview of software testing, including its goals and fundamentals. Additionally it also responds to inquiries concerning the fundamental abilities needed for software testers, or those who want to pursue a career in testing. Focuses on the fundamentals that are considered while creating test cases and planning. Writing effective test cases is another topic covered in this article. This is among the crucial elements in testing.

Keywords :- Software Development Lifecycle, Software Applications, Development Process, Software Quality, Automation Testing, Test Driven, Test Data, Test Optimisation, Test Cases.

I. INTRODUCTION

The growth of large-scale applications has brought about a new age of complexity and creativity in the ever-changing field of software development. The testing process faces several difficulties as these applications develop and becoming more complicated, necessitating a long-term and strategic strategy. With an emphasis on test automation, this study aims to investigate and tackle the particular testing difficulties that come with large-scale software programs, emphasizing the need for comprehensive options in the areas of examination methodologies, tools, techniques, environments, and challenges [1]. Large-scale applications provide complexity that goes beyond the conventional limits of testing techniques. It takes a deep comprehension of complicated circumstances [2] to ensure the security, performance, and dependability of such large-scale systems. By analysing the unique issues that come up while testing applications on a large scale and putting out a comprehensive plan for sustainable test techniques, this study seeks to add to the body of knowledge already in existence [2, 3].

An essential component of the software production lifecycle is testing [2, 3]. Testing is done to make sure a software product satisfies both functional and non-functional criteria. Because errors may have major effects on the economy, software testing has become even more important as software has grown in the dimensions and complexity [3, 4]. Test cases are essential to testing, and developing and keeping track of them takes a lot of effort from practitioners. However, it is often seen that problems arise in applications despite testing efforts. Furthermore, when systems change, testing efforts for many projects remain high [3, 4]. These raise concerns about the quality of test cases and lead us to look at what constitutes excellent test cases. Numerous empirical investigations have examined a range of testing topics, including automated test generating tools, coverage, mutants, and issue identification [4]. The majority of test cases nowadays are made by hand, even if many of these research investigations take into account test cases that are produced automatically - c.f [5].

The testing domain is divided by two complementary test forms: checking and exploring. Similarly, explain how automated testing and exploratory testing are complementary:

"Automation and exploratory testing complement one other rather than conflict. The exploratory testers may test anything the team didn't consider before developing since automation takes care of the daily repeated regression testing (checking) [5, 6]."

We created a test methodology for preliminary evaluation of massive operations systems in our earlier work. We demonstrated that exploratory testing contributes to the continually evolving delivery and integration pipeline for a huge-scale software program (with automated evaluation and exploratory testing complimenting one another) [6] based on a research that included both quantitative and qualitative data [6, 7]. The exploration test teams in the case study generated more issue reporting than other test teams, according to quantitative data gathered for the research, demonstrating the effectiveness of exploratory testing as a test methodology [6, 7].

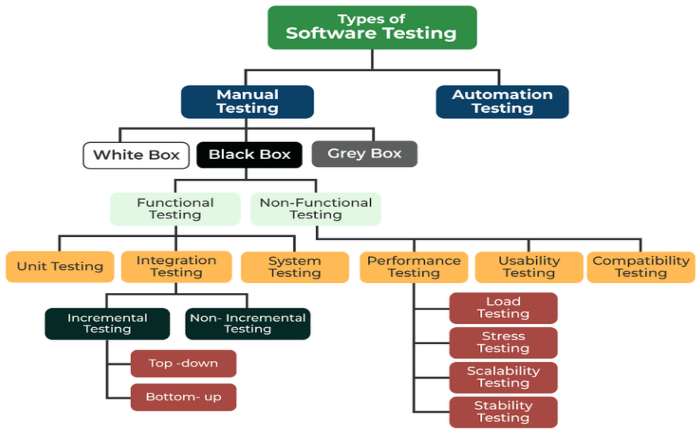

Fig. 1 Each project of software has an ideal amount of testing.. [6]

- **Need of Software Testing**

To show that the program is doing its intended functions and that it is not performing its intended functions [6, 7].

— **Fundamental objective:** The main objective of testing is to find and correct defects as soon as possible; this indicates that the purpose is not to patch the code but rather to search for and uncover program or code flaws as soon as possible [7, 8]. Either the testing process or the cases for testing need to be modified if a flaw cannot be identified in the code. It is the tester's duty to create accurate test cases in order to find any hidden defects in the software or program [8].

1. **Bugs:** Prior to comprehending the principles of identifying flaws or faults in code, we need to grasp how these issues arise in software [8, 9]. The following causes lead to software bugs:
2. **Unclear requirements:** Either the client is unclear about what they need or desire, or occasionally they are unable to express what they need [9].
3. **Programming errors:** Additionally, it sometimes occurs that developers build software incorrectly, either because they have poor programming

abilities or because they make incorrect assumptions [9, 10].

4. **Poor documentation:** Lack of transferring knowledge exacerbates the issue when developers quit their present project in the middle of their work; in these situations, new developers are allocated halfway through the project [10].
5. **Communication gap:** There is often a communication breakdown between the software developers and the people handling the requirements [10,11], which leads to project developers having the erroneous idea.
6. **Changing requirements:** Any modification becomes a burden for the whole team, which leads to incorrect implementation of requirements or the omission of some from the code [11].
7. **Defect:** Anything that we must eliminate from our code via testing is considered a defect. A defect might be described as something that software development should not do but that the product specification requires it to do [11]. Anything the program does that the product description states it shouldn't do is considered a defect. Anything that software performs that is not included in the product specification is considered a defect [11, 12]. Anything that software should perform but doesn't and isn't included in the product specification is considered an imperfection [12].

• SOFTWARE TESTING PROCESS

Testing is done to find code flaws and ensure that the program is error-free [12]. One must adhere to a systematic testing approach in order to implement testing correctly and get better outcomes.

- **Testing process includes**

— **Test planning:** This section outlines the test objective, test schedule, and test approach, which means that before beginning any testing, the tester must properly plan, specify the testing goal (i.e., what we want to test), and then choose the appropriate timing or schedule (i.e., when testing will be finished) [12]. This exam plan also includes responses to questions like:

— What we have to test?

— What are the test case's pass and fail criteria?

— Which software and hardware environments are required?

— Which features need to be tested—all of them or just part of them—must be determined in the test plan?

— Which features are we not interested in testing? [12],

— Given the large number of participants, what are the roles and duties of the persons and organizations working on the project?

To some degree, effective planning will lead to good test outcomes, thus there are some principles to create the test plan, such as:

- Start early.
- Maintain the test plan's flexibility.
- Examine the test plan often. [11],
- Make sure your test plan is brief, clear, and easy for others to read.
- Compute the planned activities [12].

— **Test Design:** One must develop the various test cases and rank them as soon as the timetable is determined. These are reviewed by an inspector, reviewer, or higher-ranking official, such as the test manager, once they are designed [11, 13]. This requires the tester to create quality test cases. A test case is a collection of inputs, execution requirements, and anticipated results created for a specific goal. According to IEEE Standard 610 (1990), a test case is a collection of test inputs, execution circumstances, and anticipated outcomes created for a specific goal, such exercising a certain program route or confirming adherence to a particular requirement [13].

For designing some essentials are:

— All of the key characteristics should be covered when we create the test cases [13].
— Normal, aberrant, and boundary test scenarios are all balanced.
— Testing techniques, such as black box and white box testing, functional and non-functional testing, static and dynamic testing, and so on, are balanced [14].

Some characteristics of Good test cases are:

- An accurate, cost-effective, repeatable, traceable, self-standing, and self-cleaning test case is essential [14, 15].
- An accurate test case is one one will be tested in accordance with its description.
- Only required actions that accomplish the goal are included in a good test case; extraneous processes are avoided [15].
- A good test case is self-standing, meaning that regardless of who tests it, it will always provide the same findings.
- A successful test scenario will be suitable for the environment and testers [15].

Above all, these are the elements that demonstrate excellent test cases; however, writing good test cases is another matter. Some crucial elements in writing effective test cases include:

- **Improve testability of the test cases:** Testability refers to the ease of testing [15]. Since the link or steps should be obvious and intelligible to anyone who performs the test cases, test actions will be written in current cases.
- **Improve accurately:** When testers adhere to the instructions, the outcome—whether successful or unsuccessful—will be accurate [15].

Test Cases common mistakes:

- Avoid creating lengthy test cases or combining two or more test cases into a single test case [15]. It's feasible that a single test case might verify several criteria and include numerous outcomes.
- Testers get confused by incomplete or incorrect test cases.
- Aim to avoid omitting any steps in test cases since doing so will make it difficult for testers to execute the test cases because they may not know how to finish them [15].
- Occasionally, it's unclear whether the system or tester takes any actions that put the tester in a precarious position and leave them unsure of what to do next [16, 17].

Test Case Design stages:

Test cases must be properly designed before being conducted in order to ensure that the testing process runs smoothly [15].

— **Identify the test resources:** It is advised that the procedure be developed after determining the degree of assets allotted for the test's information tool [16].
- As previously stated, determine which test conditions must be employed in the test cases; the testing a matrix is used for this purpose.
- To determine which conditions will be examined first, the test conditions will be ranked. The condition we want to test is chosen based on ranking, and at this point, the requirements should be quite detailed [16].
- Every test case should be assigned a distinct number.

— **Test Implementation:** Depending on the needs, we may also use a variety of automated technologies to construct test scripts that will run the test cases that we generate for testing [16, 17].
— **Test Execution:** To do this, run the test cases or scripts in the test environment and use defects reporting applications to inform developers and management of any issues so that the code is free of errors [17].
— **Test Analysis:** Key indicators are calculated using project and test metrics [17, 18].
— **Test Review:** It is utilized for future perspective, which involves talking about the

lessons gained from the difficulties that have been phased in and figuring out ways to avoid them in the future [17, 18].

- **Test Initiation Criteria**

The question of when to begin testing is another one. Therefore, timing is crucial for this [18] because we can begin testing as soon as we have software requirements or baselines. This is because improper requirements lead to incorrect design and implementation, and once implementation is complete, it becomes extremely difficult and costly to identify and fix any flaws. Therefore, the goal of testing is to identify requirements-related flaws as soon as feasible [17, 18].

— **Test complete criteria:** Since full testing and completely defect-free software are not achievable, we must take into account some metrics or conditions before we can halt the testing process. Many factors are taken into consideration when determining when testing should conclude, such as project testing deadlines, release deadlines, test case completion with a certain % passed, the conclusion of the Alpha and Beta testing periods, or when testing reaches its peak level. The graph indicates that although the number of problems continues to decrease over time, costs are also increasing exponentially [18, 19]. The intersection of these two lines or charts provides us with the ideal amount of testing. Figure 2 shows that anybody may cease testing at this stage in order to balance the risk and expense of testing [19].

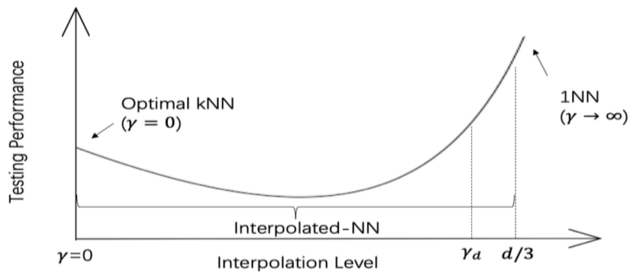

Fig. 2 Diagram for optimum level of testing. [20]

1. **Participants in testing:** Upon further examining the testing procedure, we find that almost every member of the software development group, in addition to users or clients, participates in the testing procedure [20, 21]. Even though their degree of participation may vary, their presence is just as crucial to the testing process' effectiveness [21, 22]. For example, clients may participate in the analysis stage, users in alpha and beta evaluation, developers in unit examination, managers in the release of problems, and auditors in the verification of various software testing policies and procedures [22, 23]. Testing difficulties: Given the many difficulties in the testing procedure, such as:

- Testing is seen as a late project.
- The requirements cannot be tested.
- It is challenging to gauge test progress [23, 24].
- It is not feasible to do thorough testing.

Therefore, there may be certain guiding variables to help avoid such issues. For example, when the software team begins the project, test scheduling and preparation may begin [25]. We can test tiny code as we write it, rather than waiting until the last minute to test the full product or code [25, 26]. Additionally, organize how one integrates and tests the system using repeatable procedures [27].

- **Skills for Testers**

To become an effective tester and pursue a career in software testing, one needs a variety of abilities, including the ability to read [27, 28], ask questions, and communicate.

- **Reading skills:** Knowing what you read is a prerequisite for this competence [28, 29]. Some things to think about while reading include asking questions about information collection and looking for answers, classifying the information that is found, and then reading it again.
- **Questioning skills:** Asking questions like who my client or customer is, what problems this product is intended to solve, [28], what problems this product could create, whether

you are the privilege person to answer the question in question, whether anyone else can provide more pertinent and additional information, whether I missed something or wanted to ask something, and whether I can follow up with more questions later are all necessary to become a good tester [28, 29].

- **Communication skills:** This is a necessary ability for every tester. This ability will be required while interacting with clients, customers, developers, etc. Poor or ineffective communication leads to miscommunication or conflict.
- **Other skills:** In addition to the aforementioned abilities, testers also need to be able to think negatively in order to create negative test cases, make decisions for testing, which calls for good judgment, multitask by reading documents, execute tests, and build teams, among other things.

Software is a collection of computer-executed instructions created and intended to carry out certain tasks. Phases such as requirement analysis, design, coding, testing, deployment, and software maintenance are all part of the software development process [29, 30]. Software testing is one of the most crucial stages of the software development process. Testing is done on the program before it is deployed. Figure 1 any discrepancy between the predicted and actual results might be considered a fault, which is what software testing is all about. The practice of examining or running software to look for errors is known as software testing.

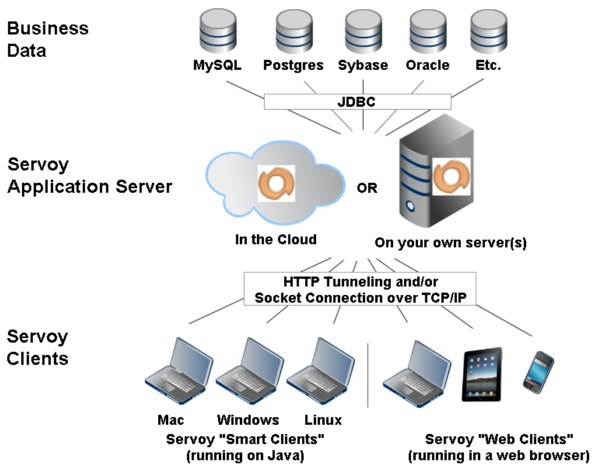
Business Data
MySQL
Postgres
Sybase
Oracle
Etc.
JDBC
Servoy Application Server
OR
In the Cloud
On your own server(s)
HTTP Tunneling and/or Socket Connection over TCP/IP
Servoy Clients
Mac
Windows
Linux
Servoy "Smart Clients" (running on Java)
Servoy "Web Clients" (running in a web browser)

Fig. 3 Deployment and maintenance. [30]

Testing is the process of determining whether or not a certain system satisfies the initial criteria. It is primarily a validation and verification procedure that determines if the produced system satisfies the user-defined criteria [28, 29]. As a consequence, the actual and predicted effects of this action vary. Program testing is the process of identifying defects, mistakes, or requirements that are missing from a created system or program. Accordingly, this study gives the stakeholders precise information on the product's quality [29, 30]. Another way to think about software testing is as a risk-based activity.

The most crucial aspect of the testing process is for software testers to know how to reduce an enormous amount of tests into a manageable test set and make informed judgments about which risks are worth testing and which are not [30, 31]. Figure 1 illustrates the correlation between testing costs and mistakes. Figure 1 makes it abundantly evident that the expense of evaluating both functional and non-functional kinds increases significantly. When choosing what to test or cutting down on testing, many problems may be missed. Doing the ideal number of tests to minimize additional testing effort is the aim of effective testing [31, 32].

The importance of software testing can be seen from life-critical applications (e.g., flight control) testing, which can be very costly due to the risk of schedule delays, price overruns, or outright postponement, and more [24, 28]. Figure 1 [23, 25] illustrates the importance of software testing as a component of computer software quality confidence.

- **ENHANCEMENT IN TESTING PROCESSES**

Test Suite Prioritization uses Combinational Criteria to improve the testing process. Converting weblogs into test suites pertinent to the user experience and then putting it down in an XML format is the main mechanism behind this test case prioritization. The level of coverage calculated using combined test suites should precisely prioritize the algorithm utilized for this method. Furthermore, in order to analyse the efficacy of the particular application and its associated test suites, empirical research have to be conducted [28, 29]. One tool employed in this context is called C-PUT, which effectively transforms web application logs into XML-formatted test suites. It is then utilized to provide the capability for prioritizing these tests [29]. The effectiveness of these test suite prioritization strategies in improving the fault detection ratio is still being investigated [29]. Another improvement in the testing process is the use of Genetic Algorithms (GAs) to generate test data automatically for application testing. Previously, dynamic methods of test data generation were a major problem in software testing, so using Genetic Algorithm-based testing is an efficient way to generate test data [29, 30]. It can also handle data generation in accordance with the complexity of the program [30, 31].

- **Test Automation**

Test automation, which involves using specific software to conduct the testing process and comparing actual results with predicted outcomes, is the result of significant improvements in the testing process [30, 31]. Because it eliminates the need for time-consuming manual testing, the test automation approach is time-effective. Both the implementation and testing phases of the SDLC include test automation. Automating test procedures is being used globally in place of manual testing since it saves a significant amount of time and completes the testing procedures faster [30, 31]. By eliminating the requirement for human testing and revealing the quantity of mistakes and shortcomings that manual testing is unable to detect, test automation has supplanted it [31, 32].

When done by hand, regression testing, one of the main testing kinds, takes a lot of time. Usually, it checks to see whether the program or application functions correctly after any bugs or defects have been fixed. Because sometimes the error or bug ratio of the code or program increases even more after the error has been fixed [32, 34]. In order to reduce the amount of time required for regression testing, a collection of automation test suites is created specifically for this purpose. Additionally, test automation helps in identifying issues much sooner, which saves a ton of money and effort on later stages of change [35].

The environment that supports a word is usually known as the Testing Framework, which is used to execute automated testing. The testing framework is primarily in charge of carrying out the tests, specifying the language for expressing expectations, and reporting the findings. Testing Framework's application independence is its most notable attribute, which contributes to its broad applicability across a variety of disciplines globally. There are many types of testing frameworks, such as hybrid, data-driven, modular, and keyword-driven. The foundation of the Modular Testing Foundation is the abstraction concept, which calls for writing distinct scripts for each of the software or application's modules that need to be tested in order to separate each and every element from a higher level.

The automatic test suites can be scaled and maintained more easily thanks to this modular split. Additionally, it becomes simple and quick to create various driver scripts for various test kinds if the library has the capabilities [31, 34]. The main drawback of this kind of framework is that it embeds data; thus, when the test data has to be amended or upgraded, the whole test script code must be changed [34]. This was the primary reason for the creation of the Data Driven Test System. A single driver script may run all test cases with several sets of data thanks to this sort of framework, which ideally stores the test data and anticipated outcomes in separate files [34, 35]. This kind of framework minimizes the quantity of code needed to generate test cases and test scripts, and it provides more flexibility in fixing faults or flaws.

- **Test Driven Development (TDD)**

It is a methodology that uses automated unit tests to force the decoupling of dependencies and drive software architecture. Testers often discover one or more flaws or faults during the standard testing procedure [35], but TDD provides a clear indication that something went well when the test subsequently fails, increasing the degree of assurance that the system satisfies its fundamental requirements. A significant amount of time that may be lost during the debugging process can be saved by using the TDD technique [35]. Behaviour-driven development, or BDD, is essentially an extension of development driven by tests that focuses on the system's behaviour rather than its implementation-level features. As a result, the testing procedure is more efficient as it is apparent what the system is meant to achieve. As a result, BDD is mostly Test-driven Development combined with Approval Testing, which generally means testing to see whether the program or product satisfies the requirements [35]. It is called User Acceptance Testing if it is carried out by the target user or client.

TESTING METRICS

- **Prioritization Metrics**

Utilizing test metrics is crucial as it may significantly increase the efficacy of the testing procedure. They function as a crucial gauge of the effectiveness, accuracy, and analysis of specified metrics [29, 30]. Along with the next course of action or action that must be followed to eradicate it, they may also assist in identifying the areas that need improvement. Test metrics serve as an umbrella for the ongoing enhancement of the whole procedure for testing rather than being a single stage in the STLC [30, 31]. Software testing metrics, which are divided into two categories—process quality metrics and product quality metrics—focus on the aspects of quality that are pertinent to both the process and the final product. Both metrics seek to improve both the testing procedure and the quality of the final product [31].

- **Process Quality Metrics**

A process is the most important component as it can provide a high-quality result in the shortest amount of time and at the lowest possible cost [32]. This is the main reason why businesses all over the globe have focused on improving the performance of their processes, and this is precisely where the demand for metrics arose since it is necessary to effectively assess the process from a variety of angles [33]. The primary indicator of process quality is process efficiency, which includes metrics such as the test progress curve, which shows how the testing phase is expected to go according to the test plan [34].

CONCLUSION

Since the ultimate product delivery depends on testing, it is the most important phase of the software production lifecycle. Since it is a laborious and time-consuming procedure, better techniques and creative approaches are needed. This makes it necessary to include automated testing as well as other test metrics both before and throughout the testing process. It may improve the current testing procedures in terms of time efficiency as well as the production of a dependable and effective end product that not only satisfies the standards but also offers the highest operational efficiency.

These days, testing is commonly utilized to assist developers in creating software that is free of defects. Even if there are many different testing methods, it is still crucial to properly organize the testing process. A high-quality software or product devoid of defects will be the outcome of careful test preparation. Test case design is also crucial for effective planning. It ought to cover every stage of the process. The success of the testing process depends not only on preparation but also on self-skills, meaning that testers must do the testing with the intention of not finishing the assignment but also from the bottom of their hearts.

II. REFERENCES

[1] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in ISSRE, 2014, pp. 201–211.

[2] X. Xia, D. Lo, P. S. Kochhar, Z. Xing, X. Wang, and S. Li, "Experience report: An industrial experience report on test outsourcing practices," in ISSRE, 2015, pp. 370–380.

[3] X. Xia, D. Lo, J. Tang, and S. Li, "Customer satisfaction feedback in an IT outsourcing company: a case study on the Insigma Hengtian company," in EASE, 2015, pp. 34:1–34:5

[4] X. Xia, D. Lo, F. Zhu, X. Wang, and B. Zhou, "Software internationalization and localization: An industrial experience," in ICECCS, 2013, pp. 222–231.

[5] R. Opdenakker, "Advantages and disadvantages of four interview techniques in qualitative research," Forum: Qualitative Social Research, (Last accessed on March 9, 2016), vol. 7, no. 4, 2006.

[6] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in FSE, 2014, pp. 155–165.

[7] A. Zaidman, B. V. Rompaey, A. van Deursen, and S. Demeyer, "Studying the coevolution of production and test code in open source and industrial developer test processes through repository mining," Empirical Software Engineering, vol. 16, no. 3, pp. 325–364, 2011.

[8] J. Cleland-Huang and J. Guo, "Towards more intelligent trace retrieval algorithms," in RAISE, 2014, pp. 1–6.

[9] G. Meszaros, XUnit Test Patterns: Refactoring Test Code. Prentice Hall PTR, 2006.

[10] B. V. Rompaey, B. D. Bois, S. Demeyer, and M. Rieger, "On the detection of test smells: A metrics-based approach for general fixture and eager test," IEEE Transactions on Software Engineering, vol. 33, no. 12, pp. 800–817, 2007.

[11] M. Greiler, A. van Deursen, and M.-A. Storey, "Automated detection of test fixture strategies and smells," in ICST, 2013, pp. 322–331.

[12] F. Palomba and A. Zaidman, "Does refactoring of test smells induce fixing flaky tests?" in ICSME, 2017, pp. 1–12.

[13] Guide to the Software Engineering Body of Knowledge, Swebok, A project of the IEEE Computer Society Professional Practices Committee, 2004.

[14] E. F. Miller, "Introduction to Software Testing Technology", Software Testing & Validation Techniques, IEEE, 1981, pp. 4-16.

[15] M. Shaw, "Prospects for an engineering discipline of software," IEEE Software, November 1990, pp.15-24.

[16] D. Nicola et al. "A grey-box approach to the functional testing of complex automatic train protection systems." Dependable Computing-EDCC 5. Springer Berlin Heidelberg, 2005. 305-317.

[17] J. A. Whittaker, "What is Software Testing? And Why Is It So Hard?" IEEE Software, 2000, pp. 70-79.

[18] N. Jenkins, "A Software Testing Primer", 2008, pp.3-15.

[19] Luo, Lu, and Carnegie, "Software Testing Techniques-Technology Maturation and Research Strategies', Institute for Software Research International-Carnegie Mellon University, Pittsburgh, Technical Report, 2010.

[20] M. S. Sharmila and E. Ramadevi. "Analysis of performance testing on web application." International Journal of Advanced Research in Computer and Communication Engineering, 2014.

[21] Siripong Roongruangsuwan, Jirapun Daengdej" Test Case Prioritization Techniques"Journal of Theoretical and Applied Information Technology" (2005 – 2010).

[22] Shivkumar Hasmukhrai Trivedi "Software Testing Techniques "Volume 2, Issue 10, October 2012.

[23] S. Sampath and R. Bryce, Improving the effectiveness of Test Suite Reduction for User-Session-Based Testing of Web Applications, Elsevier Information and Software Technology Journal, 2012.

[24] B. Pedersen and S. Manchester, Test Suite Prioritization by Costbased Combinatorial Interaction Coverage International Journal of Systems Assurance Engineering and Management, SPRINGER, 2011.

[25] S. Sprenkle et al., "Applying Concept Analysis to User-sessionbased Testing of Web Applications", IEEE Transactions on Software Engineering, Vol. 33, No. 10, 2007, pp. 643 – 658.

[26] C. Michael, "Generating software test data by evolution, Software engineering", IEEE Transaction, Volume: 27, 2001.

[27] A. Memon, "A Uniform Representation of Hybrid Criteria for Regression Testing", Transactions on Software Engineering (TSE), 2013.

[28] Kuhn, A.: On extracting unit tests from interactive live programming sessions. International Conference on Software Engineering, pp. 1241-1244 (2013).

[29] Mihindukulasooriya, N., Rizzo, G., Troncy, R., Corcho, O., García-Castro, and R.: A two-fold quality assurance approach for dynamic knowledge bases: The 3cixty use case. CEUR Workshop Proceedings (2016).

[30] Rashmi, N., Suma, V.: Defect Detection Efficiency of the Combined Approach. Advances in Intelligent Systems and Computing 249 VOLUME II, pp. 485-490 (2014).

[31] Schaefer, C.J., Do, and H.: Model-based exploratory testing: A controlled experiment. 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014, pp. 284-293 (2014).

[32] Ghazi, A.N., Garigapati, R.P., Petersen, K.: Checklists to support test charter design in exploratory testing. Lecture Notes in Business Information Processing 283, pp. 251-258 (2017).

[33] Sviridova, T., Stakhova, D., Marikutsa, U.: Exploratory testing: Management solution. 12th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2013, pp. 361 (2013).

[34] Ghazi, A.N., Petersen, K., Bjarnason, E., Runeson, P.: Levels of Exploration in Exploratory Testing: From Freestyle to Fully Scripted. IEEE Access 6, pp. 26416-26423 (2018).

[35] Raappana, P., Saukkoriipi, S., Tervonen, I., Mäntylä, M.V.: The Effect of Team Exploratory Testing - Experience Report from F-Secure. International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2016, pp. 295-304 (2016).