SILESIAN UNIVERSITY OF TECHNOLOGY PUBLISHING HOUSE

SCIENTIFIC PAPERS OF SILESIAN UNIVERSITY OF TECHNOLOGY 2020
ORGANIZATION AND MANAGEMENT SERIES NO. 144

ANALYSIS OF EXPLORATION TESTING APPROACH AND CONCLUSIONS FROM IMPLEMENTATION IN SOFTWARE DEVELOPMENT

Michał KĘDZIORA[1*], Karolina KAŁWAK[2], Ireneusz J. JÓŹWIAK[3], Michał SZCZEPANIK[4]

[1] Wroclaw University of Science and Technology, Faculty of Computer Science and Management, Wroclaw; michal.kedziora@pwr.edu.pl, ORCID: 0000-0002-7764-1303
[2] Wroclaw University of Science and Technology, Faculty of Computer Science and Management, Wroclaw; karolina.kalwak@pwr.edu.pl
[3] Wroclaw University of Science and Technology, Faculty of Computer Science and Management, Wroclaw; ireneusz.jozwiak@pwr.edu.pl, ORCID: 0000-0002-2160-7077
[4] Wroclaw University of Science and Technology, Faculty of Computer Science and Management, Wroclaw; michal.szczepanik@pwr.edu.pl, ORCID: 0000-0001-9801-992X
* Correspondence author

Purpose: The main purpose of the research is to examine the suitability of exploratory tests in the software testing process.
Design/methodology/approach: An experiment, carried out for the sake of this study, consisted of two parts. First, a test was performed, and in the second part a survey was conducted, which allowed for the comparison of exploratory and test-based tests.
Findings: The results of the tests indicated a slightly lower effectiveness of the exploratory approach, which may have been caused by the conditions of the experiment: the choice of the tested software, short duration of test sessions, participants lacking knowledge about the investigated software and experience in performing exploratory tests.
Originality/value: Despite the weaker results obtained, the exploratory tests proved useful, as evidenced by the detection of distinctive errors, not found during tests based on test cases. In the survey, 90% of respondents confirmed the use of formalized test approach, based on test cases, while just over a half (57%) indicated having experience in conducting exploratory tests. Testers considered both approaches useful, addressing greater need for conducting formalized tests using test cases. Results included in the research allowed to indicate the qualities and shortcomings of the exploratory approach to software testing.

Keywords: software testing, software quality, exploratory testing, software development.

Category of the paper: Research paper, technical paper.

1. Introduction

Initially, the testing of programs was identified with their debugging (an activity aimed at finding, analyzing and removing the causes of a failure). The period characterized by this approach lasted until about 1957, when Charles L. Baker considered testing and debugging as separate activities in his paper (McCracken, 1957). In the following years, focus was placed on confirming software compliance with the requirements, and the division between testing and debugging was preserved. In 1958, the National Aeronautics and Space Administration (NASA) created the first team of testers to implement a manned space flight project. Three years later, Gerald Weinberg and Herbert Leeds devoted the entire chapter of their book to the discussion about the objectives of software testing (Leeds, and Weinberg, 1961). In 1967, William Elmendorf argued about the need for greater formalization and discipline in the conduct of testing (Elmendorf, 1967). Two years later, for the first time, one of the basic principles of testing was formulated, which stated that: "Testing reveals the presence of defects, not the lack thereof", which was authored by Edsger Dijkstra. In 1970, Winston Royce focused on describing the cascading (sequential) model of software development, also paying attention to the need to test it (Royce, 1970). In 1977, T. Gilb focused on the metrics used to evaluate the software (Gilb, 1977). The trend in testing has turned towards understanding testing as a tool for determining software quality. In 1982, G. Weinberg described the iterative method of production and testing (Weinberg, 1982). Then, the iterative approach was used in 1993, when Scrum was first produced and, 3 years later, Extreme Programming. Earlier, in 1988, the term "exploratory testing" was used for the first time (Kaner, 1988). In 2001, the Agile manifesto was published, which is a set of common rules for agile software development methodologies (IEEE, 2008; ISO/IEC/IEEE, 2013; IEEE, 2014; ISO/IEC, 2011). A year later, the International Software Testing Qualifications Board organization was established (Muller et al., 2005). Current Software Engineering and Knowledge Engineering approach also describes testing and software quality assurance as one of the challenges along with the current trends (Xu, 2015; Hurley, 1995). Another area is real-time automated testing (Guo et al., 2018; Bures et al., 2018) or special software testing architecture and business models (Vukovic et al., 2018; Jihyun et al., 2018).

2. Exploratory Testing

Exploratory testing is one of the possible approaches to software testing currently used. It is based on a simple, quite intuitive assumption of continuous and parallel learning, creating and performing tests and reporting their results (Whittaker, 2009). This definition, although understandable, is also very short and general. The structure of such an approach largely depends on the context and conditions of a specific programming project, which may give the impression that exploratory testing is not structured and has no specific rules. According to J. Bach (Bach, 2003), this is one of the reasons why, in the era of automation, scripting and creating test cases, exploratory testing can be depreciated, considered impractical, or only useful in special cases (Itkonen et al., 2009; Shoaib et al., 2009; Tinkham et al., 2003; Agruss et al., 2000). Researchers have investigated the effectiveness of exploratory testing (Itkonen, and Rautiainen, 2005; Itkonen et al., 2007) and compared it with other approaches of software testing. Literature research, done by the authors, indicated that the subject of exploratory testing and its efficiency compared to other approaches is relatively poorly described in the scientific literature, which was one of the factors taken into account when choosing the topic of the work, as well as the experiment planned as a part of it. The research problem at work was the assessment of the usefulness of exploratory tests in the study of the produced software, as well as the comparison of their effectiveness with formalized tests, based on test cases. Based on the research problem, the purpose of the research was to design, execute and analyze the results of the experiment that, in practice, tested the effectiveness of both approaches to software testing. This allowed not only to examine the usefulness of exploratory tests, but also to compare them with tests using test cases.

Exploratory tests are a type of software testing techniques that focuses on, in the simplest terms, parallel learning, creating and performing tests and reporting their results. This approach is quite different from tests based on the creation of test cases, and therefore automatic tests. However, in comparison with informal test techniques, it is often more structured and disciplined (Hellmann et al., 2011; Pfahl et. al., 2014; Patton, 2002; Myers et al., 2005). For this reason, in a certain context, exploratory tests can be considered an intermediary link between formal and informal testing.

In contrast to TCT-based tests, exploratory testing is not intended to “mechanize”, formalize and standardize the testing process. The test method depends, to a large extent, on the tester or the team performing it, which is why their skills are an important factor. Also, software usability is a factor that should be considered (Weichbroth, 2018). Knowledge and experience are of great importance, helping in guessing mistakes, as well as selecting and focusing on those elements of the tested software, where the appearance of defects is very likely. During the preparation of exploratory tests one should focus on their basic factors: test sessions last for a certain time, e.g. 1-2 hours, during which the tester or the team performs manual tests by interacting with running, tested software, in order to achieve a specific test purpose, e.g. examining a specific group of functionalities of a given application or checking the correctness of data integration between elements of the tested system, in order to finally report test results. The main factors will, therefore, be:

- time and length of testing,
- tester, or a group of testers selected for testing,
- tested product – application, program, system or its element,
- the purpose of the tests – guidelines indicating what area/characteristics of the developed software should be tested,
- reporting of results – a description of found defects, comments, suggestions regarding changes in the program, tips for carrying out further tests.

The essence of exploration tests is based on a large freedom of their implementation, combined with limited constraints to support their structuring. For example, each test session has a specific duration and goal to achieve. Often, test sessions begin with the presentation of only the main purpose of the test on the so-called test card, with the addition of any proposed ideas supporting testing. During the test session, the testers create and implement test scenarios themselves, often without long preparation, but if they recognize that project documentation (e.g. its requirements), record of results of previous test sessions or tests carried out using other techniques may be useful, they can prepare for the upcoming session by analyzing those. However, it is a quite rare practice. In the event that the behavior of the program at a certain point attracts their attention, they may, at any time, modify the test and examine the element that interests them. Thanks to the parallel execution and analysis, the tester learns to perform tests, as well as the operation of a given program, which allows them to create further test cases. At the moment when the tester determines that a given software element is tested well enough, or the tests take too long, they may stop the task and focus on, for example, other software functionalities. If the tester is dealing with relatively new software, or one that is poorly documented, the lessons learned from the tests may be useful in creating test cases for more formalized forms of testing in the future. In practice, the most useful traits of an exploratory tester are:

- ability to create, analyze and modify tests in the current mode, which systematically allows for effective software exploration,
- ability of careful observation – in contrast to focusing only on the expected results of the test case, the tester is supposed to observe and identify any unusual behavior of the program that may indicate defects,
- ability to think critically – the tester should be able to assess and explain the logic of their reasoning, also by looking for mistakes in their approach,
- ability to create a variety of ideas – an experienced tester generates more qualitative ideas regarding, for example, the methods of finding subsequent defects or the nature of tests carried out,
- ability to use available resources – using tools and devices supporting testing, sources of knowledge about the project, tests etc., expanding knowledge and know-how.

3. Methodology of research

As part of the work, an experiment was carried out, which was divided into two parts: empirical research, involving the execution of software tests, and conducting a survey among research participants.

The first part of the research was aimed at conducting a practical study, comparing the effectiveness of two testing methods: exploratory and based on test cases. During the planning of the empirical study, the next two stages of the experiment were distinguished. The first involved a selected research group, tasked with conducting exploratory tests in short 15-minute test sessions on the software provided for testing. The second part of the research focused on the tests based on a previously prepared list of test cases, conducted by the same group of participants. The length of this type of test was also determined to be 15 minutes. Details of the exact course of empirical research are described in the further part of the paper.

The second part of the experiment was to conduct a survey among the participants. It was decided to include questions in the survey, which were supposed to focus on the current experience in software testing, opinions on exploratory testing techniques and the ones based on test cases, assessment of individual elements, as well as lessons learned from the experiment. For obvious reasons, the survey was to be filled only after the practical part of the experiment.

The results obtained on the basis of empirical studies and questionnaires for participants of the experiment were presented in the further part of the paper for further analysis. For some of the questions contained in the survey, requiring a subjective assessment from the respondent, a five-point Likert scale was used (Likert, 1932). The scale was originally introduced to research in the field of psychology, as early as in the 1930s, by the American psychologist and sociologist Rensis Likert. He created a bipolar interval scale, allowing for the examination of subjective feelings of the respondent. At both ends of the scale, there are opposite values, and its interval character means that the distance between successively ranked points of the scale is equal – values in subsequent points of the scale are graded. The Likert scale was chosen due to its advantages, including the ease of use and flexibility, enabling quick comparison of the obtained results and facilitating further analysis of results, segmentation and factor analysis.

3.1. Determining the research group

The first and most important element of the experiment was the selection of a research group, i.e. participants engaged in the study. It was decided that only people who have professional experience related to software development would take part in the research. Additionally, due to the nature of the experiment (verifying the effectiveness of testing techniques), it was decided to include only those people, who professionally dealt with software testing. In some companies, the implementation of specific projects involves the use of programmers, project managers, "user experience" creators in testing etc. However, they usually have much less experience in testing, so it was decided that only people who work as testers would take part in the study. This allowed for the experiment to be carried out, as well as an analysis to be performed on the results obtained for the group of people potentially interested in the comparison of the tested methods of software testing. Due to the limited human resources to participate in the study, it was decided to invite testers from two software companies operating in Wroclaw. At the beginning of the experiment, both companies employed more than 1,000 employees, of which over 400 were software testers. Thirty testers – 15 persons from each company – were selected to take part in the research. The group included testers aged between 25 and 40, with professional experience from less than one year to 11 years as a tester.

3.2. Testing scenarios

A set consisting of 3 programming products, characterized below, was selected for testing. The features of individual projects are summarized in Table 1.

Table 1.
List of Projects' Features

Feature	Project A	Project B	Project C
People involved in the project	< 20	> 100	< 30
Product type	Application used for transmission and playback	Browser mechanism dedicated to the needs of smart TV devices	Service software for technical testing of smartphones
Users	Customers	Web application developers, indirectly clients	Employees of dedicated technical service
Estimated number of product users	> 1000	> 100,000	< 1000

Source: own work.

Project A – an internet application that broadcasts the most popular sports events, dedicated for certification and publication in a specific app store. This store is available globally as part of the smart TV service on a huge collection of devices, including televisions and set-top boxes manufactured by major manufacturers, including Sony, Samsung, Vestel, Tivo, Hisense, Sharp etc. The application has been passed for testing by an external developer, who does not have physical access to the devices, on which the application was intended for publication, along with modest documentation. The application had 2 modes of action: free, for unregistered users, and premium for those paying a subscription. To use premium content, authorization with login and password was required. The multimedia content of the application consisted in video records of sports events, including American football, football, tennis, table tennis, basketball, hockey etc. divided into separate categories. The application test card contained an examination of the basic functionalities of the application, i.e. logging in to the premium account, playing the multimedia content from all categories, testing application settings and video search tool on one device selected for testing.

Project B – a built-in browser mechanism, with an extended API, based on the Chromium project, implementing a set of international and industry standards for downloading and rendering websites, as well as launching and playing their content, especially live video and audio content. This mechanism was used on devices supporting Smart TV services, i.e. TV sets and set-top boxes from producers such as Hisense, Sony and Samsung as a tool enabling maintenance of the online application store mentioned earlier in the description of Project A. At the time of the experiment, the software was in the final phase of production. It is worth adding, that it was the next version of the software, several previous implementations of which were successfully built into hundreds of thousands of TV devices available around the world. Due to the complexity of the project, the test card limited the tests only to check the functionality related to the support of video and audio playback.

Table 2.
List of averaged results in each parameter for each project

Parameter	Project A	Project B	Project C
Defects (TE)	2.3	1.6	3
Defects (TCT)	4.6	1.8	4.8
Defects found only in TE	0.9	0.3	1.3
Priority	1.1	0.0	0.4

Source: own work.

Project C – a new version of the program, which is subject to frequent changes, being in the advanced phase of production, used to test the so-called points-of-care on smartphones made by a popular manufacturer of mobile phones. The program, after running in a special environment on a standard office computer and via a wired connection with a telephone, is used to test the correct operation of a wide range of phone functionalities, including display function, speakers, Bluetooth connectivity, Wi-Fi etc. One of the available phone models was selected for the tests, and the test card suggested testing the functionalities related to the display.

For the needs of the project, clients and in order to comply with copyrights, all names and information that could directly indicate a particular product have been removed from the work or replaced and made available only to testers participating in the experiment. The proposed goals, included in the test sheets for experimental tests, also had a corresponding list of test cases, together with specific test scenarios, ready to be carried out. Lists of test cases exceeded the possibility of manual testing within 15 minutes, for obvious reasons, therefore, they were appropriately shortened to give the possibility to perform tests in full. The fragment of the test list used in Project A was placed in work attachments. None of the testers participating in the study had to deal with the selected software before, thus, before the tests, a short description of the tested software was given and it was presented in practice. Each of the projects was tested by 10 testers. The group was randomly divided into 3 subgroups, the task of each was to test a single project.

The research group recorded the test results (comments and a description of the errors found) on the computer in a simple file with .txt extension. Next, they were analyzed by the authors of the research. The selected parameters, describing the results of the empirical tests, included the number of defects found, the number of defects impossible to reproduce using the list of test cases and the ratio of priority defects of all identified parameters.

Defects found and described in the course of the research, including the scenarios allowing for their replication, were then subjectively assessed by the authors, taking into account their actual occurrence, intelligibility and correctness of the described scenarios. The purpose was to remove erroneous or incomplete descriptions of defects and to assign priorities to individual errors.

3.3. The experiment

The practical part of the experiment was carried out within 5 days at the headquarters of one of the programming companies in Wroclaw (Poland). Defects found by testers were described by them in a simple file with .txt extension, in a way typical for each of them, there was no established concept. During the tests, the authors of the research were present on the spot as experts, assisting the participants of the experiment and watching over the correct course of tests and surveys. A single test session for a tester was carried out according to the following scheme: Stage 1 – a 10-minute introduction to the subject of the randomly selected project (A, B or C) and the presentation of the software and equipment needed to carry out the tests, as well as the available project documentation. Stage 2 – a 15-minute session of exploratory tests, conducted in accordance with the test card. Stage 3 – a short break. Stage 4 – a 15-minute test session, based on the prepared list of test cases. Stage 5 – filling out the questionnaire form. The use of the survey in the work was decided in order to gain access to the conclusions and experiences of a wide group of testers taking part in the survey. The survey project assumed the acquisition of information characterizing individual testers in the context of professional experience and experiment.

4. Analysis of results

This part presents the results of the tests carried out, both the test part and the questionnaire. In order to organize and facilitate further analyzes, the data is presented in tables and depicted graphically. In individual places, the results of the research on the conducted exploratory tests were marked with the abbreviation TE, while the tests based on test cases were referred to using the abbreviation TCT. Lists of averaged results in each parameter for each project are shown in Table 2. The results of the part of the experiment devoted to software testing are shown graphically in diagrams (Figures 1, 2, 3).

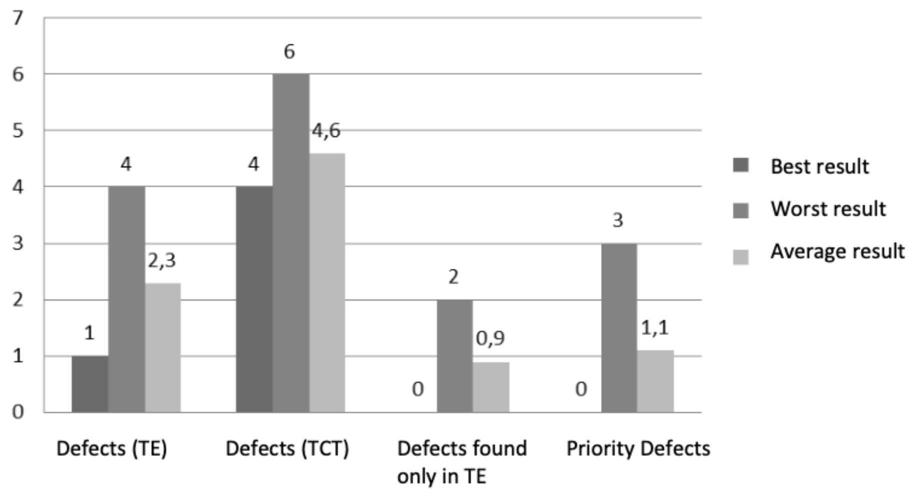

Figure 1. The results of tests carried out in Project A. Source: own work.

The research part, consisting in testing software from 3 different projects, yielded different results, which was in line with the expectations, due to the diversity of programs. The highest number of defects in a single test session was detected in Project C (6 in TCT tests and 5 in TE). In Project B the number of defects found was relatively smaller and, in the best case, it equaled to 3 errors for TE and 2 for TCT. The number of errors found only by exploratory tests for each project did not exceed 2, and the total number of priority defects ranged between 0 and 3, the highest result was recorded for the test session conducted on the application tested in Project A, while in Project B, none of the respondents were able to detect any single error. These facts may be due to differences in the production phases of the product. In the case of the application from Project A, it was one of the first versions of the software, while in Project B, the testers studied software that was in advanced development, regularly subjected to the process of automatic regression testing. Attention was paid to the weakest results of the test sessions. In all projects, there were cases in which, during individual exploratory test sessions, the tester was unable to find any, or only one error in the software they tested, which could be caused by the lack of adequate experience in conducting such tests, as well as short duration of the session.

The analysis also took into account average results for all testers within one project. Attention is paid to the tendency regarding the number of defects identified for each project, which was greater for tests based on test cases. For Projects A and C, testers reported about 1.5 defects on average. Only in the case of Project B, the difference was small and amounted to 0.2 errors, but, again, it is worth noting that none of them was considered a high priority error. The part of the research following the test consisted in conducting a questionnaire among the participants, containing 26 questions. The first part of the questions referred to the professional experience of the respondents. Among the persons participating in the study, the longest working experience as a tester equaled to about 11 years, while the shortest one was below one year. Among just over a half of people, the experience did not exceed 2 years, and one third of the testers worked in the profession between 2 and 5 years. The number of projects, in which the participants of the test were involved, was from 1 to 8. Most often, in case of 80% of the respondents, the number of projects did not exceed 3 (results for 1, 2 and 3 projects were balanced).

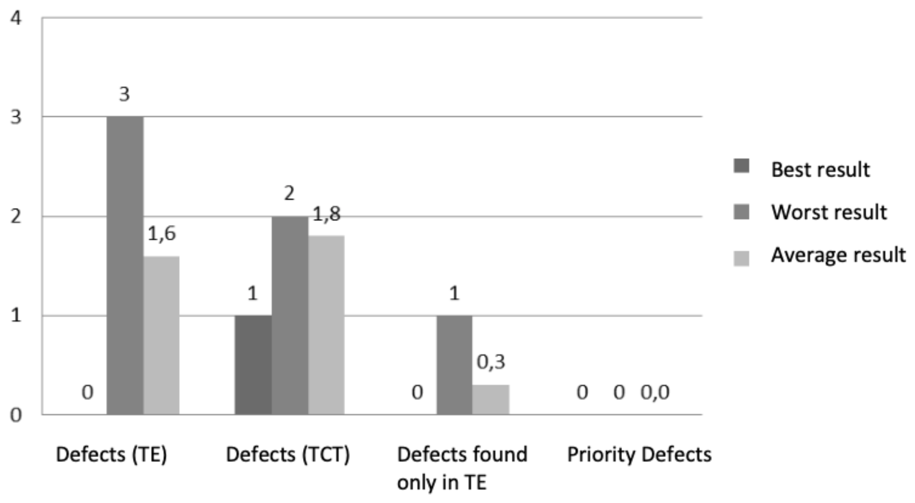

Figure 2. The result of tests carried out in Project B. Source: own work.

A significantly smaller number of testers worked on 4 or more projects (only 6 people – 20% of respondents). Each of the testers also answered several questions about the experience in the use of exploratory tests, as well as those based on test cases. The vast majority, 90% of respondents (27 people), used the TCT method at work in every previous project, while 25 people implemented it from the very beginning of their professional career, and only 2 people less than a year shorter. It is worth noting, that three people, who did not have experience with TCT tests, worked in specific projects, where the vast majority of tests were based on exploration techniques, related mainly to the tester's experience. In the case of exploratory tests, over half of the respondents (17 people) indicated their experience in the implementation of those. The vast majority (12 people) used them in every project. Four people did not use TE in any of the projects, which they worked on, and only one respondent pointed to two such projects. In the study group, both tests – based on test cases and exploratory tests – were widely used approaches to testing, but more formalized TCT tests were used more frequently, as can be seen from the experience of almost every participant in the experiment.

The next set of questions focused on the evaluation of individual elements of the testing experiment using a five-point Likert scale. The first question concerned the intelligibility of the introduction to the tests. Most of the testers rated it as sufficient and comprehensive. Average scores for individual projects were relatively similar and oscillated around 4, and only 5 people rated the level of intelligibility and adequacy of the introduction at 3. The level of complexity of the tested software was the subject of the next question. The differences between the projects were definitely greater, which corresponded with the nature and description of individual projects. In the case of the web application from Project A, the level of complexity was estimated at 2.7. The diagnostic program dedicated to phone services (Project C) received an average mark of 3.2, and software from Project B – 4.1. Testers were also asked to briefly verify and evaluate the design specification. For Project A, the testers pointed the largest gaps in the specification (average score 1.8), Project C also received a relatively low score (2.7 on average), which was mainly due to obsolete documentation in the program, which has changed frequently, as described even within the basic design requirements. Only Project B was rated relatively high (4.2 on average), because the project was accompanied by extensive documentation, mainly concerning its current version. Question 11 concerned the evaluation of difficulty of implementing TCT tests in the tested software (Table 3).

Table 3.
List of averaged results in each parameter for each project

Question no.	Project A	Project B	Project C
7	4.30	4.00	3.90
8	2.70	4.10	3.20
9	1.8	4.20	2.70
10	2.10	2.50	2.20
11	2.70	3.40	3.50
12	3.80	4.00	3.60
13	3.10	2.50	3.60

Source: own work.

In all projects, it was considered relatively easy. Testers were able to carry out this type of tests without major problems, thanks to the prepared test scenarios and short introduction, average ratings between projects ranged between 2.1 (Project A) and 2.5 (Project B). Testers were also asked for a similar assessment for exploratory tests. In this case, the results were more varied, ranging from 2.7 (Project A) to 3.5 (Project C). A clear difference was observed indicating an increase in the difficulty of conducting tests between TCT and exploratory tests. The reason for this fact could be the lack of test scenarios in TE, which forced the tester to constantly conduct and create tests based on their own experience and ongoing test results. This situation requires more experience from the tester and could be more difficult for less experienced people. In the next two questions, respondents estimated the usefulness of the tests performed.

In the case of TCT, all of the testers involved were relatively consistent in evaluating them as valuable (average ratings fluctuated between 3.6 for Project C and 4.0 for Project B). In the question about TE, testers from Projects A and B indicated a clearly lower usefulness of tests (3.1 and 2.5 respectively), while the noteworthy fact was the average rating achieved in Project C, which was 3.6, i.e. exactly the same as in the case of TCT tests. This may have resulted from the effectiveness of conducted exploratory tests, which, in the case of Project C, yielded the best results – the average number of defects found was the highest (3), also the percentage of errors detected only in TE was the highest (only in Project C the average result exceeded the value of 1 error). The next set of questions in the survey examined the general opinion of participants on exploratory tests and those based on test cases in programming projects. The testers received two questions regarding both approaches, the first concerned the assessment of suitability, the second concerned the use of the approaches mentioned.

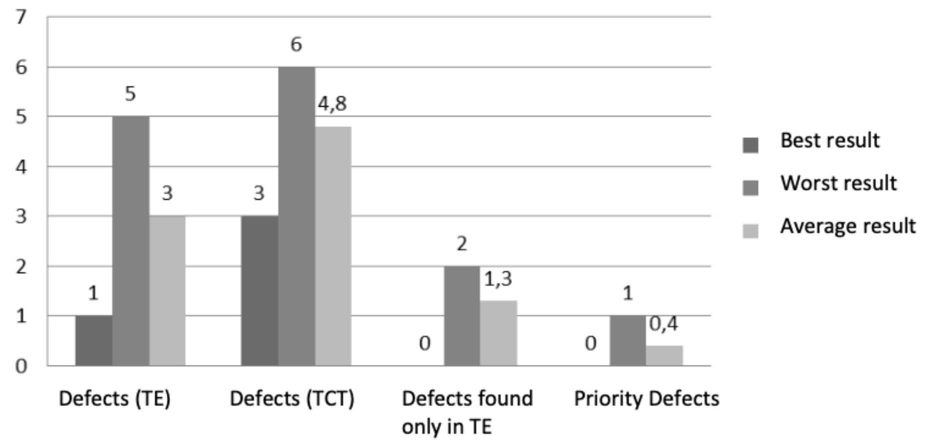

Figure 3. The result of tests carried out in Project C. Source: own work.

The results for each question within one approach strongly correlated with each other, and the average values of responses for the whole group of testers were very similar (4.47 and 4.6 for TCT tests, as well as 3.9 and 3.83 for TE). This could indicate that testers identify them with the characteristics of suitability and the need to conduct a given type of tests. It should be noted, once again, that testers turned to a more formalized approach to testing, as the results obtained for TCT were clearly higher. In the next question (No. 18), which was open-ended, testers were asked about general opinions about exploratory tests (advantages, disadvantages, essential features).

The results are summarized in the table. The largest number of respondents indicated the usefulness of TE in case of limited time to conduct tests, the ability to conduct them despite the lack of design documentation and the importance of the tester's experience in the context of their effectiveness. Respondents also exchanged features related to saving time associated with the preparation of tests and indicated a low level of formalization and a random approach to the creation of tests. Two persons mentioned the following features: TE usefulness in the case of frequently changing design requirements, limited resources, the need to conduct end-user-oriented tests; the fact that it complemented other testing approaches was also indicated, while the documentation of test results was negatively evaluated, as it was often lacking and incomplete. In the next question, the testers indicated that participation in the experiment did not change their position in the context of exploratory tests (90% of respondents). The other three people pointed out in the next question the way their opinion changed: two of them considered TE to be potentially useful in their current project, and one acknowledged the non-accidental process of creating subsequent tests, wide application and the need for experience and great attention from the tester. In the last part of the survey, the testers were asked about the use of exploratory testing in their current projects. Out of 30 participants, almost 2/3 indicated that exploratory tests are used in their current projects. When asked about the phase of the project, in which the exploratory tests were implemented, none of the respondents indicated the initial or final phase, the most (11 people) indicated the middle stage of work on the project. Of the remaining group, the testers found moderate use in the introduction of exploratory tests in their current projects (average rating of 3.73), while rated the need for introducing this type of tests lower (2.82 on average), which could be due to the opinion on the difficulties in introducing TE (average score 3.73). It is worth noting, that the testers found the TE to be equally useful as it is difficult to implement in their projects.

5. Summary and future work

As part of the research, an experiment was carried out, consisting of two parts: a test and a questionnaire. The test part has been simplified, due to limited resources, and the test results should be treated as illustrative. It would be useful to broaden the research on the subject of exploratory testing, especially with the use of a larger group of testers, coming from a larger number of companies and institutions, which would help in obtaining objective research results. An important factor would also be a change in the testing conditions, e.g. the length of the test session should be extended to the commonly used one (about 1-2 hours). It would also be worth examining the influence of external factors on the effectiveness of exploratory testing as a field requiring focus and attention from the tester (e.g. the impact of the so-called office noise on the effectiveness of exploratory tests).

The test results indicated a slightly lower efficiency of exploratory tests, which is not consistent with the studies in (Itkonen et al., 2017). Lower efficiency of tests was influenced by such factors as: selection and character of the tested software, very short test sessions (in the case of TE alone, it is usually from 1 to 2 hours, in the case of the experiment, due to limited resources, the time of a single session has been shortened to 15 minutes), lack of knowledge about the software produced in a given project (the testers had no previous contact with the program being tested), in the case of some of the testers, also lack of experience in conducting exploratory tests. Exploratory testing has positive aspects in terms of saving time, since there is no need to prepare test cases and the creation of documentation of test results, as well as the ability to conduct tests despite the lack of defined test cases. Exploratory testing is highly useful in case of limited resources, e.g. in terms of time to conduct tests using a formal approach, in programming projects containing insufficient documentation, in case of frequent changes to the software or design requirements, in end-user-oriented tests. It is also suitable for a diverse approach to testing (e.g. in combination with TCT tests, to complement them). In addition to the above-mentioned features, authors conclude significant advantages of exploratory tests, like usefulness in cases when it is necessary to quickly learn how the software works, when a quick feedback is needed on the defects found. It is also worth paying attention to the indicated disadvantages of exploratory tests, like the need for disciplined approach to creating documentation of test results, experience, the ability to focus and creativity on the part of the tester.

In summary, exploratory tests are an approach that has many advantages, which make them convenient and suitable for use in many programming projects. It requires the involvement of experienced testers, proper planning of test objectives, focus and creativity during test sessions, however, well-made exploratory tests allow for quick understanding of the software operation, dynamic test creation, provide high efficiency in defect detection and can be a valuable complement to formal tests.

References

1. Agruss, C., and Johnson, B. (2000). *Ad Hoc Software Testing: A perspective on exploration and improvisation.* Florida Institute of Technology.
2. Bach, J. (2003). *Exploratory testing explained*. Satisfice, James Bach
3. Bures, M., and Filipsky, M., and Jelinek, I. (2018). Identification of Potential Reusable Subroutines in Recorded Automated Test Scripts. *International Journal of Software Engineering and Knowledge Engineering. January,* pp. 3-36.
4. Elmendorf, W. (1967). *Evaluation of the Functional Testing of Control Programs*. London: IBM.
5. Gilb, T. (1977). *Software metrics*. Winthrop Publishers.
6. Guo, S., et al. (2018). Crowdsourced Web Application Testing Under Real-Time Constraints. *International Journal of Software Engineering and Knowledge Engineering. June,* pp. 751-779.
7. Hellmann, T.D., and Maurer, F. (2011). *Rule-based exploratory testing of graphical user interfaces*. Agile Conference. IEEE.
8. Hurley, W.D. (1995). *Software engineering and knowledge engineering: trends for the next decade*. New York: World Scientific.
9. IEEE 730 (2014). *Standard for Software Quality Assurance Processes*. IEEE.
10. IEEE 829 (2008). *Standard for Software and System Test Documentation*. IEEE.
11. ISO/IEC/IEEE 29119 (2013). *Software Testing Standard*. ISO/IEC/IEEE.
12. ISO/IEC 25010 (2011). *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, IEEE.
13. Itkonen, J., and Mantyla, M.V., and Lassenius, C. (2007). *Defect detection efficiency: Test case based vs. exploratory testing.* First International Symposium on Empirical Software Engineering and Measurement ESEM. IEEE.
14. Itkonen, J., and Mantyla, M.V., and Lassenius, C. (2009). *How do testers do it? An exploratory study on manual testing practices.* 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009. IEEE.

15. Itkonen, J., and Rautiainen, K. (2005). *Exploratory testing: a multiple case study*. International Symposium on Empirical Software Engineering. IEEE.
16. Jihyun, L., and Kang, S., and Keum, C. (2018). Architecture-Based Software Testing. *International Journal of Software Engineering and Knowledge Engineering. January*, pp. 57-77.
17. Kaner, C., Falk, J., and Nguyen, H.Q. (2000). *Testing Computer Software*. Dreamtech Press.
18. Leeds, H., and Weinberg, G. (1961). *Computer Programming Fundamentals*. New York: McGraw-Hill Book Company.
19. Likert, R. (1932). *A technique for the measurement of attitudes*. Archives of Psychology. New York: The Science Press.
20. McCracken, D.D. (1957). *Digital computer programming.* John Wiley & Sons, Inc.
21. Muller, T. et. al. (2005). *Foundation Level Syllabus.* International Software Testing Qualifications Board (ISTQB) materials. ISTQB.
22. Myers, G.J. et al. (2005). *Sztuka testowania oprogramowania*. Gliwice: Helion.
23. Patton, R.J. (2002). *Testowanie oprogramowania*. Warszawa: Mikom.
24. Pfahl, D. et al. (2014). *How is exploratory testing used? A state-of-the-practice survey.* Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM.
25. Royce, W. (1970). *Managing the Development of Large Software Systems*. IEEE WESCON.
26. Shoaib, L., and Aamer, N., and Aisha, A. (2009). *An empirical evaluation of the influence of human personality on exploratory software testing*. 13th International Multitopic Conference, INMIC 2009. IEEE.
27. Tinkham, A., and Cem, K. (2003). *Learning styles and exploratory testing*. Proceedings of the Pacific Northwest Software Quality Conference.
28. Vukovic, V., and Djurkovic, J., and Trninic, J. (2018). A Business Software Testing Process-Based Model Design. *International Journal of Software Engineering and Knowledge Engineering. May,* pp. 701-749.
29. Weichbroth, P. (2018). Delivering Usability in IT Products: Empirical Lessons from the Field. *International Journal of Software Engineering and Knowledge Engineering, July*, pp.1027-1045.
30. Weinberg, G. (1982). *Rethinking Systems Analysis and Design*. Dorset House Publishing.
31. Whittaker, J.A. (2009*). Exploratory software testing: tips, tricks, tours, and techniques to guide test design.* Pearson Education.
32. Xu, H. (2015). Future research directions of software engineering and knowledge engineering. *International Journal of Software Engineering and Knowledge Engineering 25.02,* pp. 415-421.