Collecting cognitive strategies applied by students during test case design

Felix Cammaerts
felix.cammaerts@kuleuven.be
LIRIS
Leuven, Belgium

Monique Snoeck
monique.snoeck@kuleuven.be
LIRIS
Leuven, Belgium

Ana C. R. Paiva
apaiva@fe.up.pt
Faculty of Engineering of the
University of Porto & INESC TEC
Porto, Portugal

ABSTRACT

It is important to properly test developed software because this may contribute to fewer bugs going unreported in deployed software. Often, little attention is spent on the topic of software testing in curricula, yielding graduate students without adequate preparation to deal with the quality standards required by the industry. This problem could be tackled by introducing bite-sized software testing education capsules that allow teachers to introduce software testing to their students in a less time-consuming manner and with a hands-on component that will facilitate learning. In order to design appropriate software testing educational tools, it is necessary to consider both the software testing needs of the industry and the cognitive models of students. This work-in-progress paper proposes an experimental design to gain an understanding of the cognitive strategies used by students during test case design based on real-life cases. Ultimately, the results of the experiment will be used to develop educational support for teaching software testing.

KEYWORDS

software testing, test case design, students' cognitive strategies

ACM Reference Format:
Felix Cammaerts, Monique Snoeck, and Ana C. R. Paiva. 2023. Collecting cognitive strategies applied by students during test case design. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3593434.3593954

1 INTRODUCTION

Software testing is an important, however often neglected, part of the software development lifecycle. Software testing checks if the software system behaves as expected, and helps to prevent failures of the deployed software system. In fact, a report written in 2020 has estimated the cost of poor software quality at around $2.08 trillion for the US alone [6]. This cost arises from the loss of users due to software errors, as well as developers having to spend time fixing those errors, rather than developing the features for the next release.

EASE '23, June 14–16, 2023, Oulu, Finland

ACM ISBN 979-8-4007-0044-6/23/06…$15.00
https://doi.org/10.1145/3593434.3593954

Both in industry and academia little attention is spent on software testing. Even when programmers theoretically understand the value of testing, they still do not adequately test the software they develop [1, 10]. Many computer science curricula have insufficiently integrated software testing, allowing students to graduate without adequate software testing skills [14].

Despite the benefits of incorporating testing in academic programs, such as improved programming performance, timely feedback, objective assessment, and a better understanding of the programming process, there are still significant gaps in how software testing is taught. The current teaching initiatives often focus on testing as a separate topic, without considering the training needs of students, and they are often positioned too late in the curriculum. This results in a disconnection between theory and practice, reduced student interest, and a negative attitude toward testing. Additionally, students may lack confidence in their testing skills due to a lack of performance measurement, leading to a reluctance to engage in testing activities. Finally, instructors may feel overburdened with the additional workload required to include separate testing topics.

So, even though academia has made several efforts to integrate software testing into curricula, testing remains a complex learning task, difficult to teach. To better address the learning needs of students, it is important to understand their cognitive models and cognitive strategies when designing test cases to understand the way in which software testing should be introduced to those students [5].

This paper thus seeks an answer to the following research questions:

RQ1. What are the cognitive processes applied by students during test case design?
RQ2. How does previous experience in the field of software testing affect these cognitive processes?
RQ3. Does the provision of different types of software artifacts (requirements, models, code) impact the cognitive processes?

This ongoing-research paper introduces an experimental setup to understand the cognitive processes applied by students during test case design. Section 2 discusses related work on understanding students' cognitive processes during test case design. Section 3 introduces the experimental design for this ongoing research. Finally, section 4, provides further work.

2 RELATED WORK

Several approaches to teach software testing in education have already been proposed. These approaches consist of software testing tools, gamification approaches, and pedagogical approaches.

Examples include CodeDefenders [9], Marmoset [11], Web-CAT [4], iLearnTest [8], and TILES [15].

Currently, there is no systematic approach available to understand the cognitive processes of students when it comes to learning software testing, despite the existence of various methods. In a survey conducted by Bijlsma et al. [2], students were found to base their test cases on programming code even when none was provided. Meanwhile, Doorn et al. [3] used TestCompass, a model-based testing tool, to explore the sensemaking of students' test case design in an introductory programming course. The researchers identified four approaches that students use to create test models: happy path, development, exploratory, and student approaches. However, as the students used TestCompass to design their test cases, they were limited by its functionality and familiarity. Moreover, all students received the same software artifacts, which raises questions about how students would approach test case design if they were not constrained by tools and were given different artifacts to work with. Further research is needed to explore these aspects of student sensemaking in software testing.

3 EXPERIMENTAL DESIGN

3.1 Goal

The goal of this research is to collect the cognitive strategies applied by students during test case design. We aim to identify how students apply their (informal) knowledge on test case design to new situations. During the experiment, students are asked to think out loud while they solve test modelling tasks that we give to them. This will enable us to collect data about the cognitive strategies students use while they design test cases (using modelling and exploration). We aim to identify both key cognitive strategies (how they do it) and the mental model (concept understanding) of students [13]. The cognitive strategies refer to how students address problem-solving and may refer to elements such as problem formulation, research, interpretation, communication, and precision/accuracy [7]. The mental model refers to a student's domain knowledge [13] and can be studied by considering the words used to phrase cognitive strategies.

3.2 Participants

The target participants are students that have had at least one session on Software Testing before the experiment takes place. We will further distinguish these target participants into two distinct groups. The first group consists of students who have been taught a limited number of topics on software testing (e.g. unit testing and/or acceptance testing) as part of a non-software testing course. The second group consists of students who have done a full course on software testing and hence know about the landscape of software testing. Students will partake individually in this experiment and will be recruited on a voluntary basis. To ensure the students do not feel any pressure in the experiment, the experiment will not be made part of a course. Furthermore, students' grade will not be influenced by their participation or performance in the experiment. We will run a pre-experiment questionnaire (Table 4) to make sure all participants fit the required profile thus mitigating the threat to validity related to uncertain background.

3.3 Experimental setup

We aim to perform an experimental simulation [12] where we ask participants to test the target functionality of a given software by creating a collection of test cases to the best of their abilities while thinking aloud, being videotaped, and being allowed to write notes. For that, we will setup a contrived setting where participants must develop test cases for testing functionality inspired on a real scenario.

The participants receive two exercises that they can perform in any order. To guarantee relevance for real software settings (as opposed to educational settings), both exercises are based on real web stores, from which textual requirements have been reverse-engineered. Additionally, for the second exercise, a flowchart has also been reverse-engineered. The first exercise is inspired by the Amazon web-store[1], the second exercise is based on the Nike web-store[2]. Both web stores allow users to browse items and buy them online.

The first exercise is a requirements-based exercise. The participant is given a small scenario where different variables are used to calculate a total checkout amount via rule-based calculations, whereby different rules use different boundary values for certain variables. For this rule-based exercise, all participants will be given textual requirements. These textual requirements can be found in Table 2.

The second exercise is a model-based exercise, consisting of functionally testing an application. For this exercise, we will provide different artifacts: textual requirements, a software model, and the working application. The software model consists of a flowchart describing the behavior of the website, while the working application is the Nike website.

3.4 Protocol

To investigate the impact certain artefacts have on a students cognitive strategy, the artefacts will be provided to the students in different combinations. The participants are split into four different groups accordingly (A, B, C, D), each with different artifact types as listed in Table 1.

The setup consists of a workstation with screen recording software installed. For each scenario, students are given a blank sheet of paper on which to write down the test cases. They are also given pens, markers, and paper to take notes, which are collected at the end of the experiment. In addition, in Scenarios C and D, the students are also given a laptop to browse the Nike website.

The tasks have to be completed in a limited amount of time. The participants' behaviour is recorded during the experiment. To minimise the influence of the live audio and video recording of the participants' behaviour, the camera is set so that the participants' faces are not recorded. Audio recordings will also be used to record the participants' thinking aloud. The camera is positioned to record the paper and notes taken by the participants. Screen capturing software can be used to record the computer screen. Participants are asked to clap at the beginning of the experiment to synchronise all recording devices.

[1]The Amazon web-store can be found at: https://www.amazon.com.be/
[2]The Nike web-store can be found at: https://www.nike.com/be/en/

Table 1: The four different scenarios that can be given to participants.

Scenario	Textual requirements	Software model	Working application
A	X		
B	X	X	
C	X		X
D	X	X	X

At the beginning the participants are given information about the aims of the experiment. Participants will be given a small exercise to get used to the "thinking out loud" protocol and to allow the researchers to technically check the camera and sound setup. Participants will then be given the artefacts from which to build test cases. There will be some time to clarify doubts about requirements, models, and code. We do not want to restrict the way students complete the task. They are free to use different techniques and approaches within the given time limit. However, we do ask students to verbalise their thoughts so that we can transcribe and analyse them afterwards. At the end, we distribute a questionnaire to collect additional data, e.g. about difficulties encountered, their own assessment of the quality of the designed set of test cases, etc. Finally, we collect the handwritten notes, the screen recording, the video recording, and the answers to the questionnaire.

3.5 Exercise 1

For the first exercise, the participants are given the requirements of a software system and are asked to write down their test strategy and the test cases designed to test the software system. They are also asked to think aloud. The assignment and requirements list are given in Table 2. When the time is up or the participant feels satisfied with the test cases written down, the paper notes are handed in. Written test cases are analysed according to equivalent class partitioning and boundary value analysis black-box coverage criteria.

3.6 Exercise 2

In the second exercise, the participants receive artifacts depending on which scenario they have been assigned to (as shown in Table 1). The assignment of participants to scenarios is random. This selection of scenarios is only used for the second exercise.

The textual requirements are given in table 3. The textual requirements are given to participants who are doing either scenario A, B, C, or D. Also some guiding questions have been setup that can be used by the researchers conducting the experiment. They can be used to guide participants who do not fully understand the task at hand, to help them understand what is expected of them.

A flowchart (see Figure 1) is given to the participants doing either scenario B or D. The participants doing either scenario C or D are given the working application, i.e., the link to the Nike website. These participants use the provided computer on which the actions performed are recorded using screen capture software.

3.7 Metrics

Several quantitative metrics can be used to assess the results, such as the coverage achieved by the participants and the time taken.

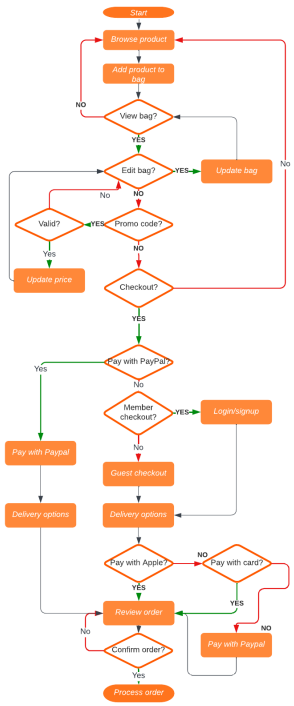

Figure 1: Flowchart diagram provided as software model in exercise 2.

For the first exercise the coverage can be measuring the number of boundary values correctly identified by the test cases. For the second exercise the coverage can be measured by measuring the amount of covered path in the flowchart. Awareness of the quality of the test cases designed and the difficulties encountered can also be gathered from the post-experiment questionnaire.

The notes taken also help to provide insight into the cognitive strategies of the participants. For example, whether participants create a diagram on their own, whether they add annotations to the flowchart or textual requirements. Mental models can also be analysed by looking at the words participants use when describing their work (during the think-aloud) and how they formulate relationships between different concepts. These qualitative measures can be used to identify the cognitive processes applied by students during test case design using grounded theory (RQ1). From these cognitive processes a cognitive model can be build. Taking into consideration the responses to the pre-experiment questionnaire and the combination of used artifacts, also allows to formulate an answer to the other research questions.

Table 2: Assignment and requirements given for the first exercise.

Ghamazon, a major online retailer, has written down several requirements for their website.	
They are asking for your help to test whether all requirements are met.	
Explain your strategy to develop a set of test cases to test the application to satisfy the above requirements.	
Requirement	Description
1	Member customers get free shipping and free 30-day returns.
2	Purchases higher than 100 euros get free shipping, otherwise the Estimated Delivery & Handling is 5 euros.
3	If you have a promotional code and the purchase is higher than 150 euros, the final value is reduced according to that promotion.
4	If you use a gift card and the purchase value is higher than 100 euros, the value of the gift card is deducted from the total final price.
5	The output is the price to be paid.

Table 3: Requirements text for exercise 2.

The Nike website allows customers to order Nike products online and have them delivered to their home.
To help customers find the right item, the Nike website has a search bar that can be used to search for specific items, as well as filters to help customers who are not sure what they are looking for.
Customers who are logged in can also add an item to their favourites to make it easier to find next time.
When a customer views their shopping basket, they are shown an overview of the items in their basket, the subtotal, any promo codes applied and the estimated price for postage and packing.
Once an item has been added to the basket, it is still possible to change the size (e.g. 5.5 to 14 for shoes) as well as the quantity. Any promo codes the customer has must also be added to the bag.
At checkout, the customer cannot change the quantity, size or add any promo codes.
However, the customer can still make changes by going back to their shopping bag.
The customer can choose to have the order delivered to a collection point or to their home address.
If the order is intended as a gift, a gift message can be added. A gift bag can also be added for an additional £4.
After filling in the information required for delivery, the customer is offered several payment options.
There are currently three payment options available, namely Apple Pay, Credit/Debit Card and PayPal.
Users must be logged in to pay with Apple Pay or Credit/Debit Card.
Paying with PayPal can also be done before adding delivery information.

Table 4: Pre-experiment questionnaire questions.

Question	Answer type
I have a lot of previous knowledge on flowcharts.	5-point Likert
I have a lot of previous knowledge on programming from a (previous) course.	5-point Likert
I have a lot of previous knowledge on testing a software from a (previous) course.	5-point Likert
Years of programming experience (if applicable)	Number
On average, I use computers (laptops, desktop, tablet) per day	Number (in hours)
I have followed a course completely devoted to software testing.	Yes/No
I have followed a course partially devoted (only some lectures) to software testing.	Yes/No
I am familiar with following topics on software testing.	Testing approaches

4 FURTHER WORK

The next steps in this research will be to pre-test the current experimental setup. This will allow any necessary modifications to be made before the experiment is run with students. This pre-test aims to check the suitability of the test case design exercises. We will then address any potential problems identified during this pre-test. The participants involved in this pre-test cannot be part of the experiment in which the results will be analysed. After this pre-test, the actual experiment can be carried out with the students. The results of the experiment can then be analysed to answer the research questions.

ACKNOWLEDGMENTS

This paper is being funded by the ENACTEST Erasmus+ project number 101055874.

REFERENCES

[1] Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. 2020. A study on challenges of testing robotic systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 96–107.

[2] Lex Bijlsma, Niels Doorn, Harrie Passier, Harold Pootjes, and Sylvia Stuurman. 2021. How do Students Test Software Units?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 189–198. https://doi.org/10.1109/ICSE-SEET52601.2021.00029

[3] Niels Doorn, Tanja EJ Vos, Beatriz Marín, Harrie Passier, Lex Bijlsma, and Silvio Cacace. 2021. Exploring students' sensemaking of test case design. An initial study. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 1069–1078.

[4] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[5] Eduard Enoiu, Gerald Tukseferi, and Robert Feldt. 2020. Towards a model of testers' cognitive processes: Software testing as a problem solving approach. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 272–279.

[6] Herb Krasner. 2021. The cost of poor software quality in the US: a 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* (2021).

[7] Allison R Lombardi, David T Conley, Mary A Seburn, and Andrew M Downs. 2013. College and career readiness assessment: Validation of the key cognitive strategies framework. *Assessment for effective intervention* 38, 3 (2013), 163–171.

[8] Ana C.R. Paiva, Nuno H. Flores, André G. Barbosa, and Tânia P.B. Ribeiro. 2016. iLearnTest – Framework for Educational Games. *Procedia - Social and Behavioral Sciences* 228 (2016), 443–448. https://doi.org/10.1016/j.sbspro.2016.07.068 2nd International Conference on Higher Education Advances,HEAd'16, 21-23 June 2016, València, Spain.

[9] José Miguel Rojas and Gordon Fraser. 2016. Code defenders: a mutation testing game. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 162–167.

[10] Lilian Passos Scatalon, Jeffrey C Carver, Rogério Eduardo Garcia, and Ellen Francine Barbosa. 2019. Software testing in introductory programming courses: A systematic mapping study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 421–427.

[11] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K Hollingsworth, and Nelson Padua-Perez. 2006. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. *ACM Sigcse Bulletin* 38, 3 (2006), 13–17.

[12] Klaas-Jan Stol and Brian Fitzgerald. 2020. Guidelines for conducting software engineering research. In *Contemporary Empirical Methods in Software Engineering*. Springer, 27–62.

[13] Jeroen JG Van Merriënboer and Paul A Kirschner. 2017. *Ten steps to complex learning: A systematic approach to four-component instructional design*. Routledge.

[14] DR TANJA EJ VOS. 2017. Zoeken naar fouten. *Op weg naar een nieuwe manier om software te testen. Open Universiteit* 1 (2017).

[15] Tanja E.J. Vos, Niels Doorn, Beatriz Marín, Beatriz Marín, and Niels Doorn. 2022. Test Informed Learning with Examples. In *Proceedings of the 10th Computer Science Education Research Conference* (Virtual Event, Netherlands) *(CSERC '21)*. Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/3507923.3507924