2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) | 978-1-6654-7836-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/QRS-C55045.2021.00161

Exploring students' sensemaking of test case design. An initial study.

Niels Doorn*†, Tanja E. J. Vos*‡, Beatriz Marín‡, Harrie Passier*, Lex Bijlsma* and Silvio Cacace§
*Open Universiteit {*firstname.lastname*}*@ou.nl*
†NHL Stenden University of Applied Sciences *niels.doorn@nhlstenden.com*
‡Universitat Politècnica de València {*tvos,bmarin*}*@dsic.upv.es*
§TestCompass *info@compass-testservices.com*

***Abstract*—Testing is the most used process to assure software systems quality. With increasing complexity of software, testing is getting more important. Testing is an intellectual activity that needs to allocate multiple cognitive resources in students, making it a challenging topic to teach in computer science programs. We advocate that testing is both model-based and exploratory, meaning that we can only make useful test models for test case design once we have made enough sense about the testing problem. The latter can only be achieved through exploring, i.e. questioning, studying, observing and inferring. In this paper, we present an initial diagnostic study to understand the sensemaking used by students while creating test models. We found indications of four different approaches used by students when modelling test cases. A plan for further research is presented on how to improve teaching by taking into account the student's sensemaking approaches.**

***Index Terms*—Software Testing, Education, Sensemaking.**

I. Introduction

Test case design is one of the most challenging and important parts of testing, since the test cases you execute determine the quality of your testing. Test case design is a complex activity that needs to allocate different intellectual and cognitive skills for, among others, analysis, reasoning, exploring, modelling, design of border cases, decision making, definition of oracles [6]. Students in computer science programs struggle with the complex tasks that encompass designing test cases. This makes teaching software testing is also hard and not straightforward [16]. Identified as a complex topic to teach, testing education needs to be improved by understanding the students' misconceptions and their sensemaking when solving test problems. This way we can construct a cognitive model of learning to simulate the needed sensemaking processes that avoid misconceptions.

Several initiatives exist to improve the teaching of software testing. In [19], Scatalon et. al. present a systematic literature mapping review that comprises 293 articles published between the years 2000 and 2018. They identify different ways the articles try to improve teaching testing, such as (1) stating the importance of testing, (2) the inclusion of testing jointly with programming courses, (3) the inclusion of testing practices, (4) the use of supporting tools, (5) the creation of course materials about testing, and (6) the assessment of students knowledge about testing. However, the drawbacks to integrating software testing in programming courses are still many, and probably the reasons for it not to be done yet worldwide [19].

Looking into how testing is being taught, we find two contradictory views similar to the two paradigms for software development as a whole [18]. On the one hand there is the *analytical school* (rational paradigm) where the emphasis is on test case creation using the precision of specifications and many types of models, i.e. **model-based testing**. This paradigm has many proponents in academia. On the other hand, there is the *context-driven school* (empirical paradigm) where emphasis is on test case creation that adapts to the circumstances under which the product is developed and used through exploration and experimentation, including: questioning, study, observation, inference, etc., i.e. **exploratory testing** [1].

Both schools have been quarrelling for years without touching common ground. This has definitely not done any good to improve test education at universities. We believe this is because that both schools are right: the design of test cases should be both model-based and exploratory. Test case design requires sensemaking that we get through exploring, questioning, studying, observing and inferring. Subsequently, we can make a test model that we can use to derive test cases and have a notion of what we have (and have not) tested. In order to properly define and integrate materials, practices and guidelines that help students to do **exploratory model-based** test case design, we need to get more insights about the cognitive processes and sensemaking of students doing the design of test cases.

Sensemaking is a description of someone's thought process about how something works in the real world. It is a representation of the surrounding world, the relationships between its various parts and a person's intuitive perception about his or her own acts and their consequences. A *cognitive model* of learning should explain or simulate these sensemaking processes and show how they produce relatively permanent changes in the long-term memory of learners [14].

Testing a system requires different sensemaking than that for the programming of software [6]. This paper describes a first step in this direction of students' sensemaking of model-based test case design. It describes an experiment with master students of the Software Testing course at Universitat Politècnica de València (UPV), which main goal was to analyze the test models that they created with respect to a given problem in

2693-9371/21/$31.00 ©2021 IEEE
DOI 10.1109/QRS-C55045.2021.00161

order to have initial insights of the sensemaking process used by the students. To do that, we used the TestCompass tool [4], which allows to create flow models and automate the design of test cases.

Results show that students focus mostly on obvious functionalities and less on exploring the problem out of the box. In this paper, we identify and describe the approaches that were considered by the students to design test cases. These approaches will give us insight and information on the sensemaking and cognitive processes of the students while modelling for test case design. This is the main contribution of our work, which can be used by researchers that are interested on the improvement on teaching testing in order to develop techniques that take advantage of the sensemaking of students. Moreover, this contribution is also important for practitioners, which can take into account the cognitive process to improve the knowledge transfer of testing among teams.

The rest of the paper is structured as follows. Section II presents the TestCompass tool. Section III presents the design of the experiment and Section IV presents the execution and results obtained with a discussion of our insights. Section V presents a brief of relevant related works, and finally Section VI presents our main conclusions and future work.

II. The TestCompass tool

As a modelling tool we used TestCompass [4]. TestCompass is an early based, and easy to use, model-based testing tool in the cloud, making it easily accessible for all participants without the need to install any software on their local machines. With TestCompass, models can be created using a web based design tool with a single purpose of creating flow charts-like models and generating test cases with predefined test coverage criteria based on that models. Nodes can optionally have additional details with pre-conditions and data to enhance the information provided in the generated test cases.

TestCompass provides users with basic feedback on their models related to incompleteness with regard to the conceptual constructs (i.e. missing edges or end points), and with online help for guidance.

Being a flow chart, each TestCompass model has exactly one start-node and can have one or more end-nodes. Between those start- and end- nodes, other node types are used for representing actions, decisions, and results. All nodes are connected with edges to form a logical model. A model can be annotated with comments for clarification.

TestCompass can generate test cases based on different test coverage criteria; Node Coverage (to cover all the nodes in the model at least once), Edge Coverage (all edges in the model are covered at least once), Multiple Condition Coverage (cover at least all combinations of two consecutive test paths once) and Path Coverage (all possible paths in the model from start to end are covered). This leads to a direct relation between the model and the generated test cases. If the model is syntactically incomplete, the test cases will not be generated.

III. Design of the experiment

We performed a quasi-experiment [5] to study the students' sensemaking when modelling a testing problem with the purpose to design test cases. We used the well-known guidelines presented in "Reporting Experiments in Software Engineering" [12] for this experiment.

A. Context

The subjects are master students enrolled in a software testing course which is an elective course of the Master in engineering and technology of software systems degree at the UPV. The Master counts for 60 European Credit Transfer and Accumulation System (ECTS) credits in total. The software testing course has assigned two ECTS credits.

In this course, students get introduced in the why, the what and the how of test case design. The whole course is based on the idea that testing is both model-based and exploratory [20]. Different test case design techniques are explained with the same three steps: make a model, pick a coverage criterion and derive the test cases from the model. Different models are needed for different test case design techniques, i.e. domain models, flow models, state models, etc. Exploratory testing is used to make useful models for testing.

During the experiment, the students used the TestCompass tool, since it is lightweight and offers a relative small set of semantic building blocks to create test models with flow charts. This mitigates the threat of students not understanding the formalism of the model that could generate noise when studying their sensemaking of test case design. TestCompass models the test relevant aspects of a System Under Test (SUT), for example about different combinations of inputs that are used to generate an output, or different orders in which operations or actions can be executed.

B. Research questions

We want to get an initial insight into the students' sensemaking while analysing a test problem and creating the corresponding test models. For that we want to measure different aspects of the models and how they were designed. Thus, we defined the following research questions:

RQ1: What are the different approaches that the students use for creating test models?

RQ2: How is the use of each approach distributed among the student population?

RQ3: What is the students perception of understanding and usefulness of the created test model?

C. Description of SUT to be tested

We create a simple case that will be used by the students to create the test models. We consider that the case should not be over complicated, since the students need to understand it and also create the test model in the assigned time for a session of the course, but it must contain a non-trivial component.

We selected the following assignment.

A hardware store sells hammers (5 euros) and screwdrivers (10 euros). Over time however, their discount system has grown a bit complex. They have asked the nephew of the boss (who is studying computer science) to develop an application that can calculate the price a customer needs to pay when buying these products. They have the following discount rules:

- If the total price is more than 200 euros, then the client obtains a discount of 5% over the total;
- If the total price is more than 1000 euros, then the client obtains a discount of over 20% over the total;
- If the client buys more than 30 screwdrivers, then there is an additional discount of 10% over the total.

Design a test model in TestCompass to adequately test the application.

D. Variables

In order to answer the research questions, we have collected different data items: the students' models and questionnaires that contain information about the students' approach of the given task. We express this information in terms of variables that we can measure to look for indications or aspects of the sensemaking used by the students.

First, we measured *quantitative aspects* and properties of the models:

- Number of nodes, edges, edges to end-nodes and end-nodes that are used.
- Complexity of the model is measured by looking at the number of test cases it generates for different coverage criteria (node, edge, multiple and path).
- The time used to create the models.

Second, we measured *how the resulting models fit the given test problem*:

- Coverage of the three discount rules of the case, individually and combined.
- Evidence of testing other (non-functional) aspects of the SUT.
- Usage and misuse of the types of nodes in relation to the semantic meaning.

Finally, we measured the *perception of the students* about this whole test modelling task:

- Perception of difficulty to use the tooling.
- Perception of clearness of the functionality of the tool.
- Perception of understanding of the problem.
- Perception of the usefulness of their model for testing.

E. Tasks and materials

Prior to the experiment, three authors of this paper (including the teacher of the course and an expert testing practitioner), created several models and compared the outcomes in order to properly define the tasks of the experiment.

Our objective of this pre-study was to find out if the tooling was suitable; what problems/cases would be useful; and if the time allocated would be sufficient. After several iterations, we reached the case that students used in the experiment (presented above). We were confident that it is possible to perform it in the session assigned for the course, and we were confident that the tool can be used without to much effort to learn how to use it.

Thus, for the experiment we planned the following tasks. First, the TestCompass tool was presented to the students. Then, the students were given a description of the hardware store case. Subsequently, we asked them to analyse the case. And later, based on their analysis, we asked them to create a testing model using the TestCompass tool. The participation in the experiment does not affect the grading of the Software Testing course, nevertheless we ask the students to their best effort to create the test models.

When students finished the test models, we ask them to answer a questionnaire in order to gain feedback and as a way to upload the created model. We asked the following questions:

Q1: How difficult was it for you to use the TestCompass tool? *(Likert scale: 1-very hard, 5-very easy)*

Q2: How clear was the functionality of the tool for you? *(Likert scale: 1-very unclear, 5-very clear)*

Q3: How difficult was it to understand the specifications? *(Likert scale: 1-very difficult, 5-very easy)*

Q4: Did your understanding of the specifications improve by using modelling? *(Likert scale: 1-not at all, 5-very much)*

Q5: Do you think your model covers all parts of the specifications? *(multiple choice: Yes, my model is complete, No, my model is not complete, I am not sure if my model is complete, I don't understand this question)*

Q6: How much time did you spent on modeling (in minutes)? *(Open end question)*

Q7: Feedback or remarks (optional but much appreciated) *(Open end question)*

Table I
NUMBER OF NODES AND EDGES OF THE CREATED MODELS

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Avg	SD
Nodes*	9	17	17	10	13	15	14	18	18	14,6	3,2
End nodes	2	5	1	3	1	2	2	4	3	2,6	1,3
Decision nodes	3	7	7	3	6	6	6	6	7	5,7	1,5
Edges	13	25	24	14	20	22	21	25	25	21,0	4,4
Edges to end-nodes	2	5	5	3	2	2	2	4	3	3,1	1,2

* The nodes count does not include the start and end nodes.

IV. EXECUTION AND RESULTS OF THE EXPERIMENT

Our experiment was executed on the 10^{th} of May 2021 with nine participating students (P1 to P9).

The quantitative properties of the models are shown in Table I. These properties correspond to the number of nodes and edges used together with the average and standard deviation of those values.

To get an additional grasp on the complexity of the models, we generated the test cases per model for each criteria TestCompass has available. The results on how much test cases were generated per model are shown in Table II.

Table II
NUMBER OF TEST CASES GENERATED BY TESTCOMPASS FOR DIFFERENT CRITERIA.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Avg	Std
Node	2	7	5	3	2	3	2	5	3	3,6	1,6
Edge	3	8	5	3	4	4	5	6	4	4,7	1,5
Multiple Criteria	6	8	7	5	6	7	9	9	8	7,2	1,3
Path	6	8	23	5	22	19	17	9	37	16,2	9,8

Table III
TEST CASE INFORMATION IN THE MODELS. 'Y' AND 'N' ARE USED TO INDICATE WHETHER THE MODEL COMPLIES WITH THE INDIVIDUAL DISCOUNT RULES AND THE COMBINED RULES OR NOT.

	$price > 200$	$price > 1000$	$screwdrivers \geq 30$	combined
P1	Y	Y	Y	Y
P2	N	N	Y	N
P3	Y	Y	Y	Y
P4	Y	Y	Y	Y
P5	Y	Y	Y	Y
P6	Y	Y	N	N
P7	Y	N	Y	N
P8	Y	Y	Y	Y
P9	Y	Y	Y	N

As we can see from these numbers immediately, and as we expected, the nine students created different test models. P1 used almost half of the nodes that were used by P2, P3, P8 and P9. There were two students that only used one end-node, whilst the others used multiple end-nodes up to five. We can also see that although quantitatively P8 and P9 have similar numbers in Table I, yet they induce a significant different amount of path covering test cases in Table II. This can be explained by the fact that the model of P9 uses conditional nodes connected with each other, which results in much more test cases. Note that this information does not give any insights into the usefulness of the models not the quality of the test suites generated. Note that there are different ways to correctly design a test model for this case, as it is widely known that different models can be correct for a case depending on the view-point of the analyst [15].

The core functionality of the test problem is the correct application of the individual and combination of the three discount rules. If we generate the test cases using the path coverage criteria from TestCompass, we can see if they are modelled according the given description. Combining this information of all created models in regard to testing of the case, we get the information as shown in Table III.

If we look at the functional requirements of the assignment, we see that all students are able to model one or more discount rules correctly. Most students, six out of nine, are able to create a model that meet all the discount rules individual. Five out of nine are also able to model the combination of the discount rules correctly. There is variation in the created models, but we can see a clear group of nodes recurring in all the models to handle the discount rules. These groups are made up of three conditional nodes that check for the discount rules, together with nodes that apply the discount rules if the conditions are met. Figure 1 and Figure 2 are typical examples of such groups.

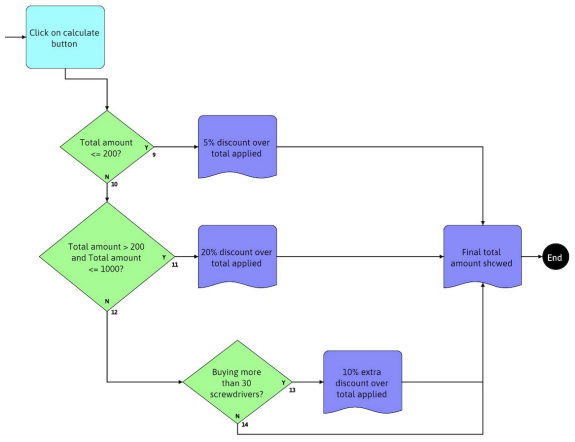

Figure 1. Example of a typical group of three conditional nodes to apply the discount rules in the model of participant 2.

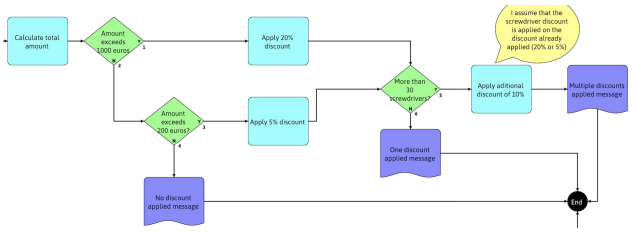

Figure 2. Another example of a group of three conditional nodes to apply the discount rules in the model of participant 3

Since the most complex algorithmic part of the assignment was the application of the different discount rules in the correct order, one might expect that mistakes in this regard would be common, but it was only made by one student, as shown in Figure 3. A more common mistake while modelling the test problem, made by three students, is with the thirty screwdriver discount rule, if this extra discount is applied before the discount of the price is applied, the total price is higher than it should be. An example is shown in Figure 4. As you can observe, if the thirty screwdriver discount rule is applied before the other discount rules, the resulting price is lower than it should be. Only five out of nine students were able to model the correct application of the combined discount rules.

In two of the nine models we see the application of wrong node types, specifically the remark node. In the model of which an excerpt is shown in Figure 5, we see the usage of remark nodes to model test cases. The remark nodes have no semantic meaning that translates to test cases by the tooling. Also we saw the same usage of remarks in Figure 8, the remarks contain valid and useful test cases.

To gather evidence of whether, and how, exploration was done by the students we studied all the models in detail to find

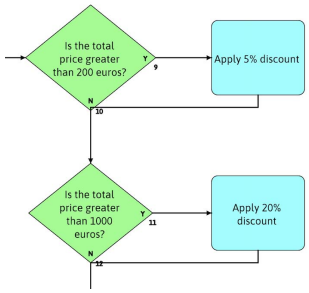

Figure 3. Example of wrong order to applying the discounts in the model of participant 7.

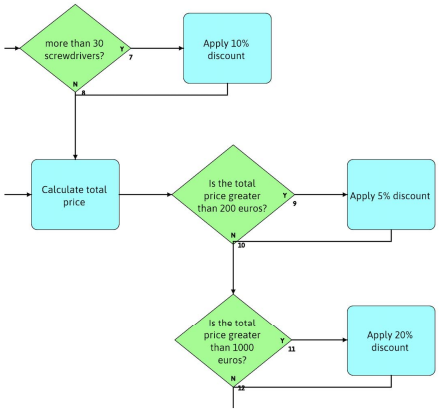

Figure 4. Example of mistake in the thirty screwdriver discount rule in the model of participant 7.

whether the model considers testing for robustness, testing for non-functional aspects and thinking outside of the box. We found such evidence with three of the models created.

An example of an exploratory test goal is the checking if the store is actually open for the customer, in Figure 6 we see an excerpt of such a model. The students assumes a context of a physical store, which is not given in the case, to create a test case.

Another example is a student who assumes that a web shop

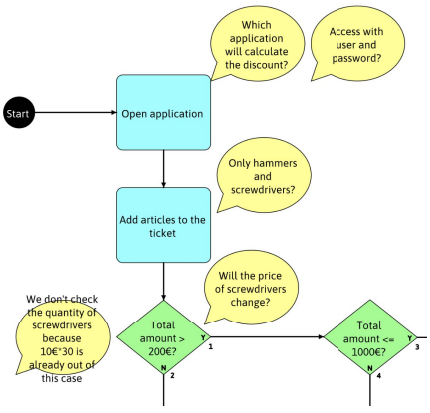

Figure 5. Example of wrong use of remark nodes by students to model test cases. Found in the model of participant 4.

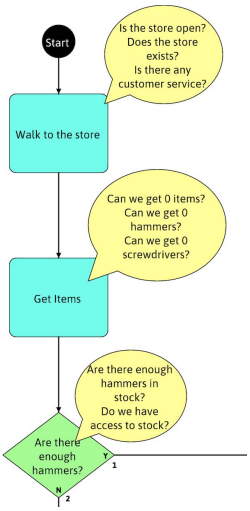

Figure 6. In this example, the model of participant 8, the student who assumes that that the store is physical and explores that context.

is used with a shopping cart. This can be seen in the excerpt of the model presented in Figure 7.

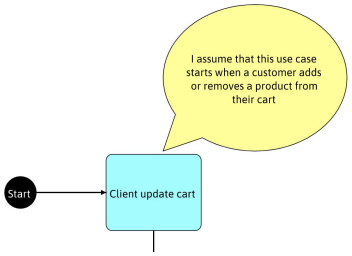

Figure 7. In this model of participant 3, the context of a website with a shopping cart is assumed and explored.

Another example is a test goal to see if the inventory contains enough items to sell to the customer. We can see excerpts of two of such models in Figure 8 and 9.

In Figure 10 we see a model of a student who added a remark node used as assertions to check if the discount is actually applied. This is an indication that the student is using coding constructs while modelling.

A last example is a student that checks whether the client has enough money to pay for his purchase (see Figure 11).

The results of the questionnaire that measures the perceptions of the students during the modelling task are presented in Table IV. As you can observe, for Q1, seven of the nine students are totally agree that the use of TestCompass was very easy. None of the students find the tool hard or very hard to use. This means that these students do not perceive that the use of the tool could affect the creation of test models. Based on Q2, we can see that all students agree that the functionality of the tool was clear to them. There was one student who provided feedback on the tool usage as can be seen in Table V. Q3 shows that eight students found the assignment clear to understand.

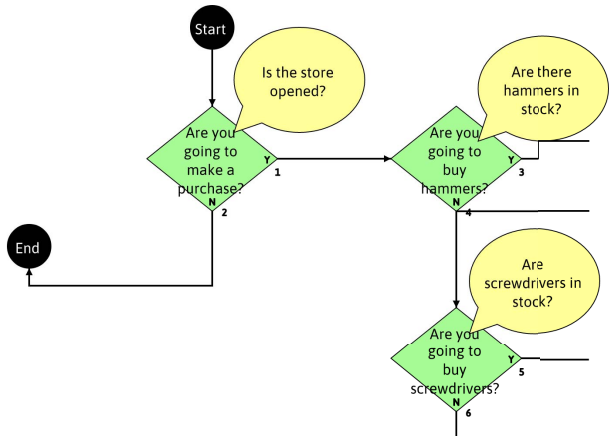

Figure 8. This excerpt of a model created by participant 9, contains three conditional checks in the remarks, one to see if the store is opened, and two to check the stock of the hammers and screwdrivers is enough to proceed with the purchase.

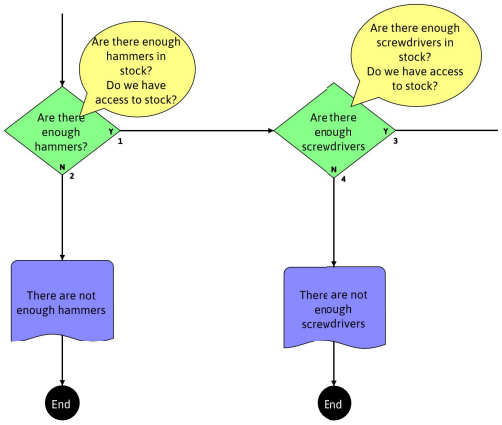

Figure 9. In this excerpt of a model created by participant 8, we see a check for the stock of screwdrivers and hammers.

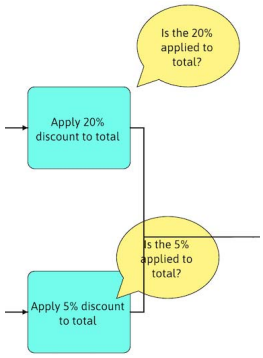

Figure 10. Remark nodes asserting the applied discount in the model of participant 8.

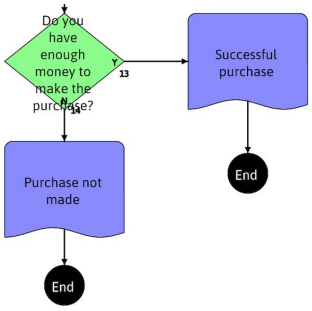

Figure 11. Condition node to check whether client has enough money from participant 9.

Table IV
THE RESULTS OF THE QUESTIONNAIRE

Participant	Q1	Q2	Q3	Q4	Q5	Q6
P1	5	4	5	5	Yes	15
P2	5	5	4	4	Yes	25
P3	5	5	5	5	Not sure	30
P4	5	5	4	4	Yes	30
P5	3	4	4	4	Not sure	30
P6	5	5	4	5	Not sure	approx. 60
P7	4	5	5	5	Yes	120
P8	5	5	3	4	No	65
P9	5	5	5	4	Not sure	90
Average	4,7	4,8	4,3	4,4		
Std.dev.	0,7	0,3	0,6	0,5		

When asking the students if they believe their model is complete in relation to the assignment, only four students believe this is the case. Four are not sure about this and one student (P8) knows for sure that the model is not complete. Oddly enough, the model of P8 was considered to be complete by the researchers. slay, give him his roses

The indicated time spent on modelling ranges from 15 minutes up to 90 minutes. We see that model on which more time is spent contain more nodes and edges, but are not necessarily more useful. However, we see that models on which more time is spent, do contain more information about testing. For example, as shown in Figure 10, the model of participant 8, who spent 65 minutes on modelling, contains information about post-condition checks.

The participants were encouraged to leave feedback, and three of them did so. Table V shows their comments. This feedback shows that we can further improve this experiment by paying attention to real life usage, different platforms and extending the experiment to gradually work on models during a longer period.

A. *Pre-study models compared to the models of the students*

The model created by a expert testing practitioner during the pre-study, as shown in Figure 12, is clearly different from the models created by the students.

It is also different from the models created during the pre-study by the researchers with a software engineering background, one of which is depicted in Figure 13.

Table V
FEEDBACK FROM THE PARTICIPANTS

Participant	Q7 feedback
P4	See and work on a simple real life developed case would clarify more some aspects of this tools usage.
P5	In MacOS I have problems when try to eliminate the components.
P8	It would have been awesome if we could contact with the lecturer to make multiple models. I mean, we could submit multiple models, each version as an improve of the previous. It was kinda frustrating that I felt like the model was incomplete.

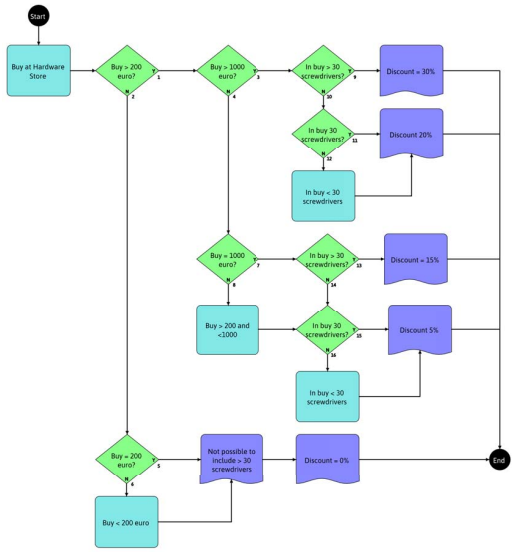

Figure 12. The hardware store modelled by an experienced expert testing practitioner during the pre-study.

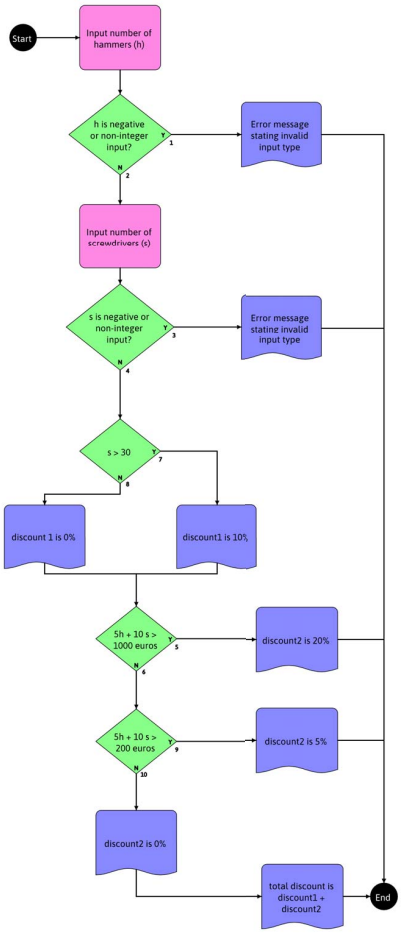

Figure 13. The hardware store modelled by a researcher with a software engineering background during the pre-study.

One of the noticeable things in the differences between the two models as shown in Figures 12 and 13, is the creation of similar nodes by the expert testing practitioner. In software engineering, we try to avoid duplication, this could well be the reason that software engineers, such as the students and the researchers, try to avoid this in creating test models as well. In none of the models created by the students, who all have a software engineering background from their master study, did we see any duplication of nodes. This is an indication of possible differences between the sensemaking of a expert testing practitioner and a software engineer.

B. Answer to research questions

Regarding *RQ1: What are the different approaches that the students use for creating test models?* We identify that students use a mixture of different sensemaking approaches. A *happy path approach* [2] means that students create the models focused only on the fulfillment of the requirements and do not take into account the less unhappy paths taken when certain conditions are not fulfilled. Some students who used a happy path approach tend to create overhead by using extra conditional nodes that are merely there to confirm the steps of the happy path. An example of such overhead can be seen in Figure 14.

A *development approach* means that the students think about how they would implement the problem rather than test it, their models strongly resemble data-flowcharts, and satisfy typical programming restrictions such as avoidance of duplication.

An *exploratory approach*, where elements are added to the models that are not mentioned directly in the requirements, but have been added by the students through exploration and trying to think outside of the box. Although exploration is desirable in exploratory model-based testing, some of the examples in our experiment (e.g. Walk to the store 6) coincide with the *student approach* [13] since the students had no chance to check the correctness of their alternative conceptions that led to additional aspects they wanted to test.

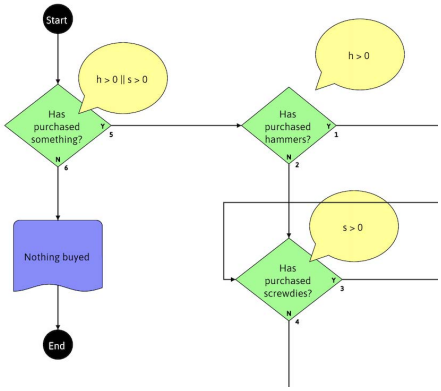

Figure 14. This excerpt of the model of participant 6 shows extra steps in the happy path.

With respect to *RQ2: How is the use of each sensemaking approach distributed among the student population?* We identify that from the six students in total that have applied a happy path approach, three students (P1, P2 and P4) created pure happy path models without any overhead. None of the created models contain any duplication. In this respect, all students have used a development approach. We see the exploratory approach in three models, P3, P8 and P9. Elements of the student approach are found in the models P3, P4, P7, P8 and P9.

Finally, with respect to *RQ3: What is the students perception of understanding and usefulness of the created test model?* The most clear indication for an answer for this question is that only four students believe their model is complete and the other students are doubtful or know for sure this is not the case, despite having no problems with understanding the tooling, they clearly perceived clear the functionality of the tool, and also they perceived that they did understand the problem description. Nevertheless, the feedback of one of the students suggest that it is difficult for them to relate to the usage and real world application of test models.

C. Discussion

This has been a small and tentative experiment from which no firm conclusions may be drawn but only hypotheses emerge. The number of participants was low (9), but it considers the total number of students that were currently enrolled on the elective Software Testing course. Because only the finished products were available, the students' sensemaking approaches could only be inferred indirectly. Techniques for more direct observation would include recording of screens, thinking aloud sessions or verbal protocol encoding [7]. Thus, we plan to perform more empirical studies taking into account these improvements in order to validate the insights obtained with our experiment.

The criteria for assessment of the result were not announced because we want to explore how they use their intrinsic sensemaking approaches to design the test model, for that reason, the solutions produced varied considerably in granularity of case analysis.

A threat of our study is that the students were not familiar with the tool to be used. In order to tackle this threat, the first activity in the experiment was to present the tool, the conceptual constructs and how to use it. Nevertheless, after the experiment we appreciate that the meaning of node types in the models was not explained sufficiently precisely to exclude differences in interpretation.

We observe that in computer science education we seem to be particular good in getting students to become professionals who are passionate about the creation of software, but less so about the testing of software. It also seems We observe that professionals with a focus on testing software produce different kinds of test models. This observation might be an indication that the creation of test cases requires a different sensemaking approach than the creation of software.

Moreover, on the basis of our results, we formulate the following research questions that require further investigation:

- What is the relationship between different sensemaking approaches and the resulting test cases?
- In what way does the sensemaking used to create test cases differ from the sensemaking used to develop software?
- Is the sensemaking of test case design affected by the personal characteristics of students (such as gender and born year)?
- Do students have a alternative standard for correctness regarding test models?
- Are there modeling notations which are more suitable to design test cases?
- Is there a correlation between the marks obtained by the students during the course and the quality of the test models created?

V. Related Works

Many publications advocate the necessity and benefits of improving testing in computer science education as can be seen in recent literature reviews [10, 19]. There seems to be a lot of consensus on the need of early testing and the early introduction of testing in computer science programs. Some are more specific and state that emphasis for exploratory testing in education is needed [11]. However, we found little research on the sensemaking and cognitive processes of students when testing software, and even less literature on changing these sensemaking approaches in education.

Buffardi and Edwards, who studied the influences on student testing behaviors [3], do note the need to change the sensemaking approaches of students. They concluded that the sensemaking of students during testing is focused on fixing problems rather than proactively avoiding them. This is something we concluded in a previous study as well [2].

Galster and Angelov studied the mental processes required for software architecture, they see the understanding of these mental processes as the foundation for devising new teaching and learning approaches. Software Architecture is a different

part of software development then software testing, but one thing they concluded is the expectation of students of well formulated problems with clean solutions [9]. This is relevant for software testing as well, since we want students to be able to explore testing beyond the description of the problem. They believe this originates from the often thought of connection between testing and debugging.

The expectations of students is in contradiction with one of the three activities software engineers have to do during software development which is making sense of an ambiguous context [17]. This indicates not only a gap between testing in education and in the industry, but also a gap between software development in education and the industry. During our literature scan, we also found a publication about the possibilities to combine exploratory and model based testing [8], supporting our idea about testing should be model-based and exploratory.

Looking at the literature, we see support for many of the ideas we have about testing, the combination of exploratory and model-based testing, the idea to shift testing to the left both in practise and in education, and the notion that the sensemaking approaches are important to study how to found educational changes on.

Many attempts have been made to improve testing education and the mindset of students. We believe more effort should be made into looking into the sensemaking approaches of students and in ways to improve this. With this study we contribute with a first step in this direction.

VI. Conclusions and further work

In this paper we describe our initial step in unravelling the sensemaking of testing and the underlying cognitive processes. We believe that ultimately this problem needs to be solved by evidence based design changes in computer science programs. To be able to do this, we use this initial diagnostic study to determine the next steps.

The most salient of our findings are the four sensemaking approaches students reveal during their tasks of modeling test cases. We advocate that in order to improve test education, these four sensemaking approaches need to be studied in more detail such that it becomes clear how we can adjust our teaching towards strengthening those part of the approaches that are needed for exploratory model-based testing.

Based on these findings our hypotheses is that test case design is a complex problem requiring a different sensemaking approach than the creation of software, and subsequently it needs to be addressed in a different way in computer science programs.

We aware that our findings were obtained with a small number of master students, but it is demonstrated that there exist different sensemaking approaches that students follow when think in testing. This pursue us to continue performing experiments to better understand the sensemaking approaches, and to validate and further generalize our initial insights. For these next experiments, we plan to invite undergraduate and master students in order to be able to identify and understand the sensemaking approaches from the beginning of the computer science education of participants. It would be worth while to also undertake more direct observation through recordings of the software use and thinking-aloud sessions with students, in order to obtain insight into the cognitive processes underlying the finished results. Moreover, we want to investigate the further research questions raised on the discussion section. We want to use these steps to move towards designing evidence based test case design education based on a process theory of test case design practice.

Acknowledgements

This work was funded by the Erasmus+ project QPeD under contract number 2020-1-NL01-KA203-064626.

References

[1] James Bach. *Exploratory testing explained*. 2003.

[2] Lex Bijlsma, Niels Doom, Harrie Passier, Harold Pootjes and Sylvia Stuurman. 'How do students test software units?' In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 2021, pp. 189–198.

[3] Kevin Buffardi and Stephen H. Edwards. 'A Formative Study of Influences on Student Testing Behaviors'. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA, 2014, pp. 597–602. ISBN: 9781450326056.

[4] Silvio Cacace. *TestCompass, the early based and easy to use Model Based Testing tool in the cloud*. 17th Aug. 2021. URL: https://www.compass-testservices.com/.

[5] Thomas D. Cook, Donald Thomas Campbell and William Shadish. *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin Boston, MA, 2002.

[6] Eduard Enoiu, Gerald Tukseferi and Robert Feldt. '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)*. 2020, pp. 272–279.

[7] Karl Ericsson. 'Protocol Analysis'. In: Aug. 2017, pp. 425–432. ISBN: 9780631218517.

[8] K. Frajtak, M. Bures and I. Jelinek. 'Model-Based Testing and Exploratory Testing: Is Synergy Possible?' In: *2016 6th IEEE International Conference on IT Convergence and Security (ICITCS)*. Los Alamitos, CA, USA, Sept. 2016, pp. 1–6.

[9] Matthias Galster and Samuil Angelov. 'What Makes Teaching Software Architecture Difficult?' In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. Austin, Texas, 2016, pp. 356–359. ISBN: 9781450342056.

[10] Vahid Garousi, Austen Rainer, Per Lauvås and Andrea Arcuri. 'Software-testing education: A systematic literature mapping'. In: Journal of Systems and Software 165 (2020), p. 110570. ISSN: 0164-1212.

[11] J. Itkonen, M. V. Mantyla and C. Lassenius. 'Test Better by Exploring: Harnessing Human Skills and Knowledge'. In: *IEEE Software* 33.04 (July 2016), pp. 90–96. ISSN: 1937-4194.

[12] Andreas Jedlitschka, Marcus Ciolkowski and Dietmar Pfahl. 'Reporting Experiments in Software Engineering'. In: *Guide to Advanced Empirical Software Engineering*. Springer London, 2008, pp. 201–228. ISBN: 978-1-84800-044-5.

[13] Yifat Ben-David Kolikant. 'Students' alternative standards for correctness'. In: *Proceedings of the first international workshop on Computing education research*. ACM, 2005, pp. 37–43.

[14] H. Chad Lane. 'Cognitive Models of Learning'. In: *Encyclopedia of the Sciences of Learning*. Boston, MA: Springer US, 2012, pp. 608–610. ISBN: 978-1-4419-1428-6.

[15] Beatriz Marín, Giovanni Giachetti, Oscar Pastor, Tanja E. J. Vos and Alain Abran. 'Using a Functional Size Measurement Procedure to Evaluate the Quality of Models in MDD Environments'. In: *ACM Trans. Softw. Eng. Methodol.* 22.3 (July 2013). ISSN: 1049-331X.

[16] Luana Martins, Vinicius Brito, Daniela Feitosa, Larissa Rocha, Heitor Costa and Ivan Machado. *From Blackboard to the Office: A Look Into How Practitioners Perceive Software Testing Education*. 2021. arXiv: 2106.06422 [cs.SE].

[17] Paul Ralph. 'The Sensemaking-Coevolution-Implementation Theory of software design'. In: *Science of Computer Programming* 101 (2015). Towards general theories of software engineering, pp. 21–41. ISSN: 0167-6423.

[18] Paul Ralph. 'The two paradigms of software development research'. In: *Science of Computer Programming* 156 (2018), pp. 68–89.

[19] Lilian Scatalon, Jeffrey Carver, Rogério Garcia and Ellen Barbosa. 'Software Testing in Introductory Programming Courses'. In: SIGCSE '19. ACM, Feb. 2019, pp. 421–427. ISBN: 145035890X.

[20] Tanja E. J. Vos and Nikè van Vugt-Hage. *Software Testing*. Open Universiteit The Netherlands, 2019. ISBN: 978 94 92739 59 9.