CHALMERS
UNIVERSITY OF TECHNOLOGY

Understanding Problem Solving in Software Testing: An Exploration of Tester Routines and Behavior

Understanding Problem Solving in Software Testing: An Exploration of Tester Routines and Behavior*

Eduard Paul Enoiu[0000−0003−2416−4205]1, Gregory Gay[0000−0001−6794−9585]2, Jameel Esber[1], and Robert Feldt[0000−0002−5179−4205]2

[1] Division of Networked and Embedded Systems, Mälardalen University, Sweden
firstname.lastname@mdu.se
[2] Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden
greg@greggay.com, robert.feldt@chalmers.se

Abstract. Software testing is a difficult, intellectual activity performed in a social environment. Naturally, testers use and allocate multiple cognitive resources towards this task. The goal of this study is to understand better the routine and behaviour of human testers and their mental models when performing testing. We investigate this topic by surveying 38 software testers and developers in Sweden. The survey explores testers' cognitive processes when performing testing by investigating the knowledge they bring, the activities they select and perform, and the challenges they face in their routine. By analyzing the survey results, we provide a characterization of tester practices and identify insights regarding the problem-solving process. We use these descriptions to further enhance a cognitive model of software testing.

Keywords: Test Design · Problem Solving · Software Testing

1 Introduction

During software testing, test cases—sequences of input and expectations on the resulting behavior of the system-under-test (SUT)—are designed and executed as a method of determining whether the SUT is functioning correctly [15]. Testing is the most common verification technique [15], and consequently, one of the most researched topics in the software engineering field [14]. However, a significant portion of past research has focused on improving the tools that testers use—there is a lack of investigation of and, consequently, evidence regarding *human aspects of software testing*.

To that end, in previous research, we proposed a cognitive model of software testing based on how problem solving is conceptualized in cognitive psychology [5]. This model mapped software testing to a cyclical problem solving model, consisting of activities related to four major phases of the testing

* Support provided by Software Center Project 30: "Aspects of Automated Testing", H2020 under grant agreement No. 957212 and Vinnova through SmartDelta project.

process—understanding testing goals, planning testing strategy, executing tests, and checking test results.

The purpose of this study is to gain a deeper understanding of the personal routines of testers, including both their external behaviors and internal processes. While our general knowledge of software testing is vast, there is a lack of clear understanding of the personal decision-making processes of testers and developers—e.g., how they reason, which test design techniques they apply, what kind of difficulties they face, how they decide which test cases to create, and how they decide to stop testing in different testing situations.

As a step towards narrowing this knowledge gap, in this study, we utilize our earlier cognitive model as a foundation for collecting data on the testing process [5]. We have surveyed 38 developers and testers working in the Swedish software development industry, focusing on the *activities performed, knowledge utilized, and challenges encountered* during each major phase of the testing process, as defined in the cognitive model. We utilize thematic analysis of the survey results to characterize how testers approach each of these phases. In turn, we use this characterization to deepen the cognitive model.

Closing this knowledge gap has implications for both researchers and practitioners. The development of a realistic cognitive model enables the formulation of clear guidance on performing effective and efficient testing. In addition, a cognitive model can benefit future approaches to automated test generation, potentially leading to the development of more human-like generation tools [4]. This study provides a foundation for this future research on both human testing practices and human-like test generation.

2 Background and Related Work

The field of *Behavioral Software Engineering* (BSE) focuses on understanding the mental, social, and behavioral aspects of software engineering performed by individuals, teams, and organizations [12]. As an example, Hale et al. [7] created a model of the mental abilities required by programmers during software maintenance. This cognitive process model of debugging combines declarative models, such as a program understanding model, with problem-solving models, based on the idea of structural learning. The proposed model was later tested by Hale et al. [8] through a controlled experiment where participants debugged a program with an unknown fault, and their verbal protocols were analyzed.

Robillard et al. [18] studied the thought processes of developers in mixed teams comprised of engineers and psychologists to develop best practices. Thematic analysis was utilized to define cognitive behaviors. This research showed that software review involves three mental activities—review, alternative solution development, and synchronization.

Letovsky [13] delved into the cognitive processes behind program comprehension, with a focus on specific moments that occur within seconds or minutes, such as understanding the purpose behind a line of code. This investigation led to the creation of a categorization system for questions and hypotheses, along

Fig. 1. Overview of the method used for collecting data and developing the problem-solving model of test design.

with a theory of the mental images and processes that produced them. The questions were defined as procedures that evaluate the coherence and accuracy of a person's developing mental model, while the hypotheses were identified as a planning process that draws on various forms of knowledge.

Recently, Aniche et al. [1] investigated the thought processes and decision-making developers experience when manually engineering test cases. Using observations from developers and survey data, it provides a broad framework for understanding developers' approach to test case development.

Despite the diverse range of approaches, it is essential to examine software testing as part of a problem-solving process to identify commonalities and gain insights into the processes involved in problem-solving. These processes seem to vary [5, 1, 11] depending on the specific testing problem/activity and how the goal is mentally represented. As a first step towards investigating the problem-solving perspectives, we previously hypothesized [5] that *a cognitive test design model could be represented as a cyclical problem-solving process* and conducted a pilot study with five students. As an initial approach, the software testing cycle, considered as a traditional problem-solving process, contains the following phases where a human tester needs to: (i) understand the test goal, (ii) formulate the test strategy, (iii) execute the tests, and (iv) check the test results.

3 Method

In this research, we are interested in addressing the following questions: (1) *How do testers utilize cognitive resources, knowledge sources, and problem-solving processes during testing?* (2) *What are the main challenges testers face in their testing routine?*

To address these questions, we followed a mixed-methods research approach outlined in Figure 1. This approach combines both qualitative and quantitative data analysis using a survey, allowing us to develop a more comprehensive understanding of the problem-solving processes involved in software testing. We utilized survey research to explore the opinions and decisions of individuals during testing. We developed an exploratory cross-sectional survey, utilizing both qualitative and quantitative descriptive methodology, and distributed it. We then performed a thematic analysis of the qualitative data to obtain an extended problem-solving model of software testing.

Table 1. Survey questions.

Survey Question	Format
1. How many years of experience do you have with development?	Choice
2. How long have you been working with testing?	Choice
3. What is the size of the company you work in?	Choice
4. What role are you presently working in?	Choice
5. Can you summarize what you typically do in your current role?	Text
6. What programming languages do you use in your company?	Choice
7. What activities do you perform when understanding the testing purpose/goal?	Text
8. What knowledge do you bring when trying to understand the purpose/goal?	Text
9. What kinds of purposes/goals do you use during testing?	Choice
10. What are the difficulties you face when you try to understand the purpose/goal?	Text
11. What activities do you perform when planning test strategy and/or creating tests?	Text
12. What knowledge do you bring when you plan test strategy and/or create tests?	Text
13. What difficulties do you face when planning test strategy and/or creating tests?	Text
14. What test design techniques do you use to create tests?	Text
15. What activities do you perform when executing test cases?	Text
16. What knowledge do you bring when executing test cases?	Text
17. What automated tools/frameworks do you use during testing?	Text
18. What activities do you perform when checking test results?	Text
19. What knowledge do you bring when checking test results?	Text
20. What software testing tools do you use to check results?	Text
21. Provide your top three challenges when checking test results.	Text
22. What criteria are used in your projects to decide to stop testing?	Text
23. Do you agree with the purpose of this survey?	Likert
24. Were there questions that were not clear?	Text
25. Do you have any feedback on the survey topic?	Text

3.1 Survey Development

We started by identifying the related work on problem-solving models that have been developed based on Polya's phases in solving mathematical problems to represent problem-solving processes [16] on which our earlier model of test creation and execution [5] was based. Psychologists have also described problem-solving as a cyclical process, as noted by Bransford et al. [2], Hayes [9], and Pretz et al. [17]. We used the overall phases previously outlined by Enoiu et al. [5] to understand how this model of test creation happens in practice, and we focused on the overall activities, knowledge, and other human aspects of different steps. Based on these steps, we developed the questionnaire questions by concentrating on the activities that participants perform when performing software testing according to the initial problem-solving model of Enoiu et al. [5]. To start the questionnaire, a brief explanation was provided about the survey's objective. In addition, participants were informed of ethical and social considerations and were assured that all collected data would be anonymized.

The survey questions are listed in Table 1. The questions were split into several sections, starting with demographic information. It then asked about the activities that participants undertake when understanding the testing goal, when planning a testing strategy, when executing testing activities, and when checking their results. We then allowed participants to provide feedback.

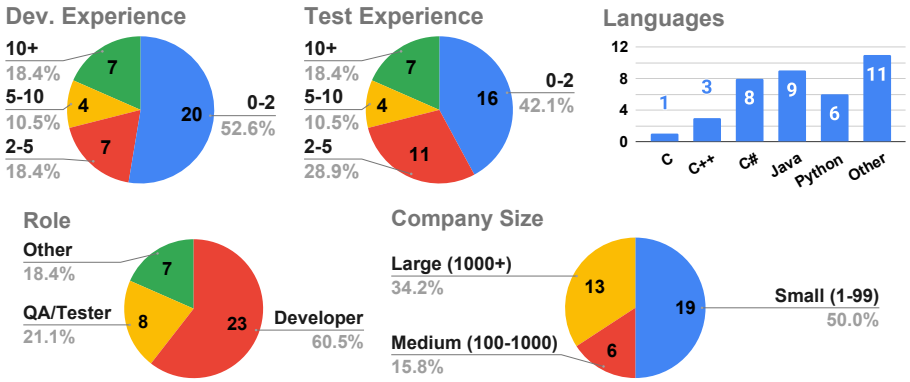

Fig. 2. Demographic information on survey participants.

3.2 Survey Population and Sampling

We targeted professionals in the software development industry. Our primary distribution method was convenience sampling. Connections with organizations were utilized to reach a large number of testers, developers, and practitioners in Sweden to gather diverse opinions and perspectives. A total of 38 responses were submitted. To ensure anonymity, we do not report the identities of respondents.

Demographic information is provided in Figure 2. The participants had a strong knowledge of software development, with all reporting at least one year of experience. The participants work for companies of all sizes, with half working for small companies and in various roles—with the majority identifying as developers. The most common programming language reported was Java, but many different languages were reported as being in use[3].

3.3 Thematic Analysis

The open-ended questions in the questionnaire were subjected to thematic analysis, a qualitative technique for analyzing data [3]. This approach involves analyzing a collection of text—in this case, participants' responses to open-ended questions—to identify common themes, patterns, and topics that arise frequently. We carried out our thematic analysis using a six-step process: becoming familiar with the data, coding the responses, identifying themes, checking the themes, naming the themes, and reporting our findings.

To begin with, we familiarized ourselves with the data by gathering and reading the responses and making preliminary notes to obtain a comprehensive understanding of the obtained data. The next step involved coding, which entailed identifying specific words or sentences in the responses and assigning them short labels or "codes" to describe their content.

After coding the responses, we reviewed and identified connections between the codes and grouped them into broader themes. This involved combining multiple words and sentences to form cohesive themes. We then verified that the

[3] The "other" languages include PHP, MATLAB, Flutter, JavaScript, and Simulink.

Table 2. Themes and sub-themes related to participants' activities, knowledge, and challenges when understanding testing goals/purpose.

Theme	Important Sub-themes
T1. Activities when understanding goals.	T1.1. Understanding software requirements. T1.2. Document analysis. T1.3. Identifying the correct behavior of the system. T1.4. Finding bugs and faults. T1.5. Inspecting the architecture of the SUT. T1.6. Using experience from previous testing sessions.
T2. Knowledge used in understanding goals.	T2.1. Documentation, Specifications, Requirements. T2.2. Code. , T2.3. Memory., T2.4. Experience. T2.5. Discussions with colleagues., T2.6. Web resources
T3. Challenges when understanding goals.	T3.1. Incomplete or unclear requirements. T3.2. Complex and highly configurable scenarios.

themes accurately reflected the data by comparing them to the responses and making modifications if necessary. Finally, we gave each theme a descriptive and concise name. These themes were then used to extend our problem-solving model of software testing.

4 Results and Discussions

This section presents the findings of the thematic analysis and introduces the extension of the problem-solving model of software testing.

4.1 Survey Results

This section summarizes the results of analyzing the survey data. The results are organized based on the sections of the survey, as explained in Section 3.1.

Understanding Testing Goals/Purpose: We asked participants to describe their activities when understanding the purpose or goals of testing. As shown in Table 2, their answers revolved around understanding software requirements and identifying the precise correct behavior of the SUT. This information was needed to understand the recognition, definition and representation of goals before creating test cases.

Our thematic analysis revealed that, during the process of understanding testing goals, participants follow a set of steps, including examining the architecture of the SUT, followed by identifying the interfaces (e.g., hardware, software, and user interfaces) and determining which levels to test them on. Finally, testers identify the responsibilities for the different test levels, if applicable.
One participant emphasized the value of conversations with colleagues:

> "Firstly, I turn to the other team members since they ... have developed the new functionality. Then if needed, I turn to code and/or documentation."

Our thematic analysis yielded several sub-themes related to the knowledge utilized during this step (Table 2). 61% of participants selected documents as a source of knowledge. However, multiple sources are often required. The majority of the same participants also chose code as another source of knowledge:

Table 3. Themes and sub-themes related to participants' activities, knowledge, and challenges when planning a test strategy.

Theme	Important Sub-themes
T4. Activities while planning test strategy.	T4.1. Identify the SUT., T4.2. Identify the test level. T4.3. Gather information from sessions in "previous" test levels. T4.4. Identify the requirements. T4.5. Identify the interfaces and create test cases. T4.6. Define the test environments., T4.7. Prepare documentation.
T5. Knowledge used in planning test strategy	T5.1. Documents (Documentation, Specifications) T5.2. Code., T5.3. Knowledge and Memory. T5.4. Experience., T5.5. Web resources.
T6. Challenges when planning test strategy	T6.1. Difficulty in coming up with edge cases/out-of-bound bugs. T6.2. Correctly selecting the test steps. T6.3. Handing ambiguous/not clear requirements. T6.4. Lack of time., T6.5. Communication. T6.6. The use of testing documents created by others. T6.7. Unstable environment. T6.8. Test automation tooling understanding.

> "I use multiple resources such as documents, code, and my memory, and consult experts whenever necessary."

A small number of participants indicated that they utilize knowledge from previous testing experience, familiarity with the software and hardware, comprehension of the implementation and testing guidelines, web resources, their memory, or conversations with colleagues. For example:

> "I rely on my previous experience, as well as discussions with architects and developers, and an inspection of the architecture and requirement specifications."

We inquired about whether test goals were discovered, created, or presented. Most participants reported a blend of options. 66% (25) discovered goals, 47% (18) created their own goals, and 61% (23) utilized test goals that were defined by someone else. One participant noted:

> "The testing goals are already pre-defined as part of the company's test strategy. When creating test cases, we apply different test design techniques such as BVA and equivalence partitioning."

When examining the challenges faced when comprehending the purpose/goals of testing, most participants identified incomplete or unclear requirements as one of the most common difficulties encountered during this phase. One participant noted that vague requirements "*cannot be developed and cannot be tested.*" Additionally, some participants reported facing challenges related to complex or highly configurable scenarios, often exacerbated by communication gaps between developers, testers, and clients.

Planning a Testing Strategy: Table 3 presents the activities involved in test strategy planning and test case creation. For example, analyzing the application before creating test cases based on experience or test specifications. Before commencing testing activities, testers strive to gain an understanding of the SUT by learning everything they can about it, obtaining detailed requirements, and comprehending the developed solution.

Regarding knowledge that participants use when planning a test strategy, we observed that most rely on documentation and the code. Additionally, testers

Table 4. Themes and sub-themes related to participants' activities, knowledge, and tool use when executing test cases.

Theme	Important Sub-themes
T7. Activities when executing a test case.	T7.1. Test environment setup. T7.2. Selecting and running test cases. T7.3. Validate the test coverage. T7.4. Continously observe and analyze outcomes.
T8. Knowledge used when executing tests.	T8.1. Documents (Documentation, Specifications) T8.2. Code., T8.3. Knowledge and Memory. T8.4. Experience T8.5. Knowledge of administering the tests
T9. Automated tools/frameworks used.	T9.1. Selenium., T9.2. Pytest., T9.3. Azure pipelines T9.4. Xunit., T9.5. IntelliJ., T9.6. Apache JMeter. T9.7. MATLAB., T9.8. Eclipse., T9.9. Ranorex T9.10. Laravel

draw on previous testing experience, knowledge of testing guidelines, and specifications from earlier versions of the SUT. Participants who had been testing for more than ten years mentioned that they preferred to use test strategy templates, knowledge of the SUT's architecture, their own experience, and regulatory requirements during the planning phase. One participant provided the following:

> "Knowledge of the software and hardware, previous testing knowledge, knowledge of the testing guidelines, and review of relevant documentation."

Regarding challenges while planning test strategy or creating test cases, many struggled with understanding complex or ambiguous requirements:

> "Lack of clear requirements is the most common difficulty."

Participants also identified limitations of testing tools, such as forced tool use leading to an unstable environment. Additionally, participants mentioned that lack of time for planning or tight deadlines were significant difficulties.

We also examined the test design techniques that participants employ. 26% of participants (10) design test cases based on specifications, 29% (11) use code as a basis for test design, and 24% (9) rely on their prior experience. 45% (17) employed a combination of experience, specification, and code. One participant provided a brief description of their creation techniques:

> "We ensure that each public interface has at least some tests, and we also consider code coverage. If a module has insufficient coverage, we add tests there. We also consider different levels of testing and aim to conduct both unit tests, module-integration tests, and system-level tests."

Executing Test Cases: Participants were asked to describe their activities during test case execution (Table 4). Responses indicated that activities include reviewing test specifications, writing test scripts, executing test scripts, and reviewing results.

Multiple tasks were performed by participants during test case execution, such as test environment setup, test case execution (including fulfilling preconditions), log-file gathering, archiving of execution and log files, documentation, and analysis of any found discrepancies. Automation of the test environment was also discussed, with one participant stating that they try to automate everything, including the setup of the test environment, running test cases,

Table 5. Themes and sub-themes related to participants' activities, challenges, and criteria when planning to check test results.

Theme	Important Sub-themes
T10. Result checking activities.	T10.1. Compare test specifications with results obtained. T10.2. Discuss results with the development team. T10.3. Modify requirements, test cases, or code-under-test.
T11. Challenges in checking results.	T11.1. Communication and interaction with other roles. T11.2. Lack of skilled testers skilled in test result analysis. T11.3. Lack of easy-to-use test analysis tools. T11.4. Lack of automation in test result checking. T11.5. Lack of historical test trends. T11.6. Difficulty understanding if the result is correct. T11.7. Challenging debugging process. T11.8. Misunderstanding of test specifications and requirements. T11.9. Incomplete historical record of test reports. T11.10. Unstable environment. T11.11. Challenging test selection based on result analysis. T11.12. Missing links between sources of documentation and logs.
T12. Completion criteria.	T12.1. When testing done on all items in the testing plan. T12.2. Coverage of edge case scenarios and "normal" scenarios. T12.3. UI functionality is covered. T12.4. All specified tests and exploratory test sessions executed. T12.5. All found discrepancies are analyzed. T12.6. Human judgment., T12.7. Experience., T12.8. Budget.

and observing the output. Some participants focused on debugging and defect identification when software bugs appeared during test execution, while others emphasized the importance of regression testing.

When executing test cases, participants often rely on documents and code to review specifications, report bugs, and document test results. However, one participant claimed that documentation is unnecessary once the tests are ready to be executed, except for instructions on how to report the results. Another participant mentioned that they only require knowledge of administering tests for automated test runs.

The most popular tools used by testers and developers during test execution were Selenium and Pytest, as they provide frameworks for automating web application testing and scalable and straightforward tests, respectively. Some participants perform tests manually, while others use custom-made tools.

Checking Test Results: Participants were asked to describe their routine for checking test results (Table 5). Their answers largely centered around comparing test specifications with the results they obtained. Testers undertake several activities while checking test results, e.g., comparing the requirements with the test results. One participant emphasized that, during this process, it is important to keep an eye on any events not specified in the test case. Some participants also discussed the results with the development team or other testers, examined test scripts, or provided feedback to the designers.

In the event of a test failure, testers and developers iteratively modify either the tests or code until achieving the desired outcome. One participant shared:

> "We rerun the test multiple times to confirm if it's a fluke. We then proceed to fix the test, the code being tested, or even the testing environment. This may involve checking for errors in parameters when setting up dockers or regenerating test data."

In terms of the knowledge utilized when checking test results, documents and code remain the most common sources. One participant emphasized the significance of documentation testing:

> "When tests fail, we almost always refer to the test case documentation. Though sometimes insufficient, we write at least one sentence about the test's purpose. Since the test cases are usually small, this is usually adequate."

However, one participant stated that the tests alone are adequate:

> "The tests created contain all the necessary information to check the results."

Participants identified the three primary challenges encountered while checking results that they would like to see addressed in software testing research. One participant highlighted some challenges that arise when tests fail:

> "1: It can be difficult to recognize that a failed test already has an open bug report. 2: Multiple failed tests may be caused by the same underlying error. 3: It can be challenging to differentiate between failing test cases due to actual software errors versus test environment issues."

Another also mentioned lack of observability into the causes of SUT behavior:

> "The primary challenge is understanding whether the obtained result is correct by chance or if the application is performing as intended."

Other challenges identified include the need for test selection (due to having too many tests to execute) and challenges that emerge from having to make this selection—e.g., the time between executions and lack of certainty in SUT correctness—visualization of test results over time, establishing traceability between documentation sources, and the difficulty of knowing who is responsible for dealing with test results (e.g., the test case creator, the feature developer, or the test environment developers).

We also asked about the criteria participants utilized to determine whether testing activities had been completed. One of the criteria that participants used was ensuring that all the tests were executed successfully and met the desired coverage levels of code and functionality requirements. Another participant stated that all planned tests must be executed without any stopping errors. Other participants mentioned budget and deadline constraints. Another participant indicated that the stopping criterion is when all test steps in the test specification have been executed and assigned a pass/fail grade.

4.2 The Extended Problem Solving Model

Based on the analysis of the survey results, we extended the basic steps of our existing test design model [5] with a more concrete process model. It operationalises the steps testers take, clarifies the multiple sources of knowledge used and the internal representations the activities are based on and updates. The extended model is depicted in Figure 3, with the new, process model in the inner circle of the original problem solving model (outer two circles). The extension can be applied throughout the problem-solving phases (mid circle) of the original model [5]. Below we provide further details, overall and per phase.

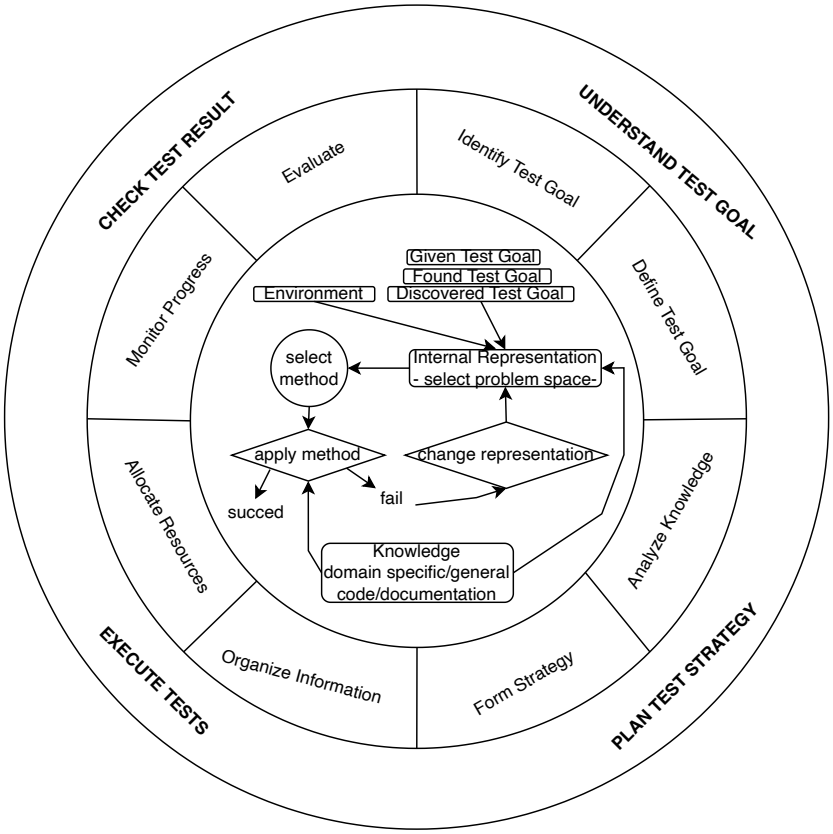

Fig. 3. The extended problem-solving model of software testing.

The Overall Problem-Solving Process: To initiate the testing process, the tester first analyzes the test goal and divides the goal into manageable components, then creates a generic solution for each component—known as a test case. These individual solutions are then combined to form a full test scenario that covers the initial test goal. In Table 6 we consider a specific test goal of security testing and go through the steps outlined in the process, providing practical examples. It is important to note that the testing cycle is not always a linear process, and skilled testers are able to remain flexible. Often, the completion of the cycle leads to the identification of a new test goal, which requires the repetition of the process itself as well as its phases.

In Figure 3, we depict the interaction between the problem solver and a testing environment. Initially, the tester inspects the environment and generates an internal representation based on the context of the test goal (e.g., a flowchart that illustrates potential SQL injection attacks and vulnerabilities). This representation involves selecting a problem space to define the test goal's representation, the context of problem-solving, and the inferences that can be drawn from it based on general knowledge and past experience (e.g., understand-

Table 6. An example of the steps outlined in the problem-solving process for a specific test goal in security testing.

Overall Steps	**Practical Examples**
Understand the Test Goal	Identify the Test Goal: Recognize potential security vulnerabilities in the login mechanism of a web application. Define the Test Goal: Design a test case that bypasses the login mechanism using SQL injection and brute-force attacks.
Plan Test Strategy	Analyze Knowledge: Analyze the login mechanism's code and infrastructure, review security guidelines and understand common attack vectors used to exploit login vulnerabilities. Form Strategy: Craft a test case that attempts to inject malicious SQL statements into the login form fields to check if the application has implemented proper input validation to prevent SQL injection attacks.
Execute Tests	Organize Information and Allocate Resources: Document the different attack scenarios, list the expected outcomes, and prepare the necessary tools, such as automated security testing tools or proxy servers, to capture and analyze network traffic during the test.
Check Test Results	Monitor Progress: Execute the test case, observe if the application properly rejects the malicious input, and monitor if any unauthorized database queries or errors occur. Evaluate: Analyze the test results, identify any successful security breaches or vulnerabilities.

ing where the system might be vulnerable and what signs might suggest a failing test case). Equipped with this representation, the tester selects a problem solving method associated with each phase of software testing. For instance, in the solution-searching phase, the tester may revisit the mental representation (e.g., fault trees, checklists), apply different methods for test design (e.g., boundary value analysis), and employ heuristic strategies to facilitate the creation of test cases. The results of these steps are then monitored, and feedback is provided, which may lead to modifications in the representation of the test goal. The survey results suggest that testers rely on multiple and varied sources of knowledge when creating and executing test cases. Among these, documentation is the most commonly used across all testing activities. Testers refer to documents such as specifications, project requirements, and testing guidelines to ensure that needs are met, defects are identified, quality and risk are assessed, confidence is established, and defects are prevented. In practice, code is the most crucial knowledge that testers and developers use during the testing process. It is used to inspect and verify the SUT and detect faults throughout the testing process. Finally, experience and skills are also fundamental sources of knowledge that testers and developers leverage in their routines.

During the testing process, testers engage in different activities at each phase specified by the mid-circle. *The process model of the inner circle thus applies throughout the phases of the mid-circle.* For example, when understanding the test goal, testers mainly focus on comprehending the software requirements and outlining the acceptable behavior of the system. Some of the primary activities they perform include inspecting the system's architecture, identifying the various interfaces (hardware, software, and user), determining the appropriate testing level, and clarifying the responsibilities for the different test levels.

Identify the Test Goal: This is the phase when a tester understands and defines the test objective as a problem that requires a solution. Getzels [6] identifies three types of problems—those that are given, those that are discovered, and those that are created or generated. A given test goal is presented to the tester (e.g., a pre-defined criteria-based test goal, such as applying particular input partitions). A discovered test goal, however, must be identified. Such a test goal exists but has not been clearly stated to the tester, which has to seek out the knowledge gap to discover what the test goal is. In contrast to given and discovered test goals, a created test goal must first be recognized and formulated. In these cases, testers may use exploratory test methods and develop new test objectives based on their knowledge, skills, and interactions with the SUT.

Define the Test Goal: This relates to how one can mentally define the test goals and what the linked tests must accomplish. The test goal definition phase of testing is when the scope and objectives of each test are established and defined precisely. A test goal presents a collection of "givens". When dealing with these constraints, a tester performs certain procedures to achieve the desired state (i.e., creating a test fulfilling a test goal). A test goal can be expressed in many ways, including graphically or audibly. For example, to achieve pairwise coverage, one must describe the objective as the task of generating all available pairs of parameter values that may be applied by at least one test case.

Analyze Knowledge: This phase structures the tester's knowledge concerning testing scenarios. Every tester addresses a particular scenario with a different set of knowledge. For example, someone familiar with test design techniques will assess their past knowledge and use various methods and representations of the test goal to clearly state the needed strategies. To develop test cases, we might have to use broad abilities such as inference, case-based logic, and generalization to organize the data gathered throughout the various processes. On a broader level, higher cognitive abilities such as inspiration and allocating mental resources such as awareness and effort must be used. Additionally, domain expertise, such as electrical, mathematics, computer science principles, programming concepts, and regulations, would be required. We discovered that testers' primary activities involve acquiring a deep understanding of the SUT, obtaining the exact requirements that led to its development, and comprehending the generated test solution. They also develop tests that cover a distinct portion of the system or algorithms.

Form Strategy: In this step, one needs to create a solution approach for generating the required test cases using certain operators. These operators are cognitive frameworks of the operations that a tester may conduct on the "givens." For instance, some computations require the use of mental operators. The activities required to reach the target state are the set of actions required to construct test cases that satisfy a particular test goal. While the operators are often not listed in the test goal, we may infer them based on our past knowledge (e.g., mathematical operators, cognitive operators).

Organize Information and Allocate Resources: After defining the test goals, the next step for the tester is to manage their cognitive and physical

resources to develop and execute test cases. Testers can use automation tools to develop test cases as executable scripts and allocate computer resources to run test cases. Alternatively, when tests are performed manually, testers allocate physical resources and document the test outcomes. We found that testers' primary activities include reviewing test specifications, documenting test scripts, and running these tests. Testers also archive test cases and log files and monitor test execution to constantly document and analyze the SUT.

Monitor Progress and Evaluate: In the end, it is crucial for testers to monitor the advancement towards the test objective(s). This phase involves tracking the results of the test generation and execution procedures. In cases where the correct output cannot be easily defined, test oracles are incorporated into scripts or results are manually monitored. This allows for the evaluation of test case quality. If testers determine that a test goal is not being met, they investigate the issue and make adjustments. Testers analyze the sequence of procedures to determine if the test cases fail to validate the test objective. Our results suggest that the testers' main activity in this step involves comparing the software requirements and specifications with the obtained test results. Testers also discuss the results with the development team or other testers to obtain feedback about the outcomes. If the test fails, testers modify the test cases multiple times to achieve the desired result. The primary activities testers and developers perform include comparing test specifications with test results, discussing the results with the team, and modifying the test cases.

5 Discussion

Based on our previously proposed problem solving model of testing we surveyed 38 professionals in the software development industry. The 25 questions of the survey focused on the activities and knowledge they use and the challenges they face in their daily test design, creation, and execution. The thematic analysis then allowed us to extend the problem solving model with a process model that can be instantiated in the phases of the overall process. While the specific method changes between the phases, the extension clarifies that an internal representation, formed based on knowledge about the environment and specific test goals, first helps select and apply a phase-specific activity which in turn leads to the internal representation being updated. Our results also clarify the information that is used in this external-action-internal-refinement loop. Specific challenges that testers face during the process were also identified.

In practice, companies can use the extended model as a basis for discussions among testers and thus create a higher awareness of both the importance of internal representations, the information and knowledge needed during the process, and how to overcome or mitigate specific challenges. Researchers can use the extended model as a basis for further data collection but also as a basis for further theoretical refinement. They can also consider how their proposed new testing technologies and methods fit with the problem-solving methods of testers and how it addresses existing challenges.

In comparison to most related works (e.g., [1, 11, 10]), our research proposes a broader perspective on software testing as a problem-solving activity, emphasizing the cognitive processes involved. The study of Aniche et al. [1] aligns with our model as it also acknowledges the presence of a mental model in test case development. They also found that this mental model is updated when failures and unexpected behavior surface during testing. However, our results also highlight that the internal representations affect not only the test case that is developed but help select the specific activity the tester uses during test creation and execution. Our extended model also places the detailed inner updating process in the context of an outer, general problem-solving process which guides which activities are appropriate.

5.1 Threats to Validity

A survey method was chosen because it allowed us to contact potential participants directly, and they could respond anonymously at their convenience. However, as the survey was conducted digitally and anonymously, we were unable to follow up with participants for clarifications or further questions regarding their responses. Consequently, it is possible that our survey participants misunderstood the questions or that we did not have a clear understanding of the testers' perspectives when formulating the questions. To mitigate this risk, an expert in software engineering reviewed the material provided and guided the respondents. To prevent confusion during the survey, we provided brief explanations for each attribute included in the questionnaire.

The survey garnered 38 responses which limit generalizability. To enable comparisons and statistical evaluation, such as based on company size and years of experience, a larger sample would be necessary. We thus focused only on general trends across the entire dataset.

6 Conclusions

This study aimed to understand the routine and behavior of software testers when performing testing, to improve software testing tools and environments to better serve their needs. We surveyed software testers with an average of five years of experience in the field and used thematic analysis to identify main themes, including knowledge, activities, and challenges related to software testing. Through this analysis, we gained insights into how testers use different sources of information and perform various activities, such as understanding software requirements, learning about the software, and discussing results with other team members to get feedback. We refined an existing test design model to show how knowledge and internal representations help select activities that develop test cases that in turn, after execution, then lead to refined internal representations. Overall, our study provides insights into the routine and behavior of software testers during testing, which can inform the development of better software testing advice, tools, and environments.

References

1. Maurício Aniche, Christoph Treude, and Andy Zaidman. How developers engineer test cases: An observational study. *IEEE Transactions on Software Engineering*, 48(12):4925–4946, 2021.
2. John D Bransford and Barry S Stein. The ideal problem solver. new york: W. h, 1984.
3. Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77,101, 2006-01-01.
4. Eduard Enoiu and Robert Feldt. Towards human-like automated test generation: Perspectives from cognition and problem solving. In *International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 123–124, 2021.
5. Eduard Enoiu, Gerald Tukseferi, and Robert Feldt. Towards a model of testers' cognitive processes: Software testing as a problem solving approach. In *International Conference on Software Quality, Reliability and Security Companion*, pages 272–279. IEEE, 2020.
6. Jacob W Getzels. The problem of the problem. *New directions for methodology of social and behavioral science: Question framing and response consistency*, 11:37–49, 1982.
7. David P Hale and Dwight A Haworth. Towards a model of programmers' cognitive processes in software maintenance: A structural learning theory approach for debugging. *Journal of Software Maintenance: Research and Practice*, 3(2):85–106, 1991.
8. Joanne E Hale, Shane Sharpe, and David P Hale. An evaluation of the cognitive processes of programmers engaged in software debugging. *Journal of Software Maintenance: Research and Practice*, 11(2):73–91, 1999.
9. John R Hayes. Cognitive processes in creativity. In *Handbook of creativity*, pages 135–145. Springer, 1989.
10. Juha Itkonen, Mika V Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *International Symposium on Empirical Software Engineering and Measurement*, pages 494–497. IEEE, 2009.
11. Juha Itkonen, Mika V Mäntylä, and Casper Lassenius. The role of the tester's knowledge in exploratory software testing. *IEEE Transactions on Software Engineering*, 39(5):707–724, 2012.
12. Per Lenberg, Robert Feldt, and Lars Göran Wallgren. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and software*, 107:15–37, 2015.
13. Stanley Letovsky. Cognitive processes in program comprehension. *Journal of Systems and software*, 7(4):325–339, 1987.
14. Alessandro Orso and Gregg Rothermel. Software testing: A research travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 117–132, New York, NY, USA, 2014. ACM.
15. M. Pezze and M. Young. *Software Test and Analysis: Process, Principles, and Techniques*. John Wiley and Sons, October 2006.
16. George Polya. How to solve it, 1957.
17. Jean E Pretz, Adam J Naples, and Robert J Sternberg. Recognizing, defining, and representing problems. *The psychology of problem solving*, 30(3), 2003.
18. Pierre N Robillard, Patrick d'Astous, Françoise Détienne, and Willemien Visser. Measuring cognitive activities in software engineering. In *Proceedings of the 20th international conference on Software engineering*, pages 292–300. IEEE, 1998.