Towards a Model of Testers' Cognitive Processes: A Problem Solving Approach for Software Testing

Eduard Paul Enoiu
eduard.enoiu@mdh.se
Software Testing Laboratory (STL),
Mälardalen University, Västerås, Sweden.

***Abstract*—No applicable model of testers' cognitive processes in software testing exists. As a first step in formulating such a model, a cognitive model for software testing is developed. The new model is grounded in problem solving and cognitive psychology theory. The theoretical foundation of problem solving as a reference description for exploring software testing is provided. The use of problem solving as applied to software testing is instantiated for the creation of new test cases. The practical and research implications of the new approach are discussed.**

I. Introduction

For many years, researchers have tried to create new techniques for effectively testing software. This research has answered many questions, but still many of them remain unanswered. For example, not that much research has focused on discovering how humans create test cases. The purpose herein is to develop a model of the cognitive processes involved in software testing. Software testing includes all activities associated with verifying existing software applications from test planning and test creation to test execution [1]. A significant portion of the software testing effort involves the creation of test cases based on a variety of test goals. Ammann and Offutt [1] classified the process of creating test cases in two general approaches: criteria-based test design satisfying certain engineering goals and human-based test design based on domain and human knowledge of testing. However, in practice this is an artificial distinction, since these are complementary and in many cases human testers are using both to fully test software. Itkonen et al. [2] observed testing sessions of eleven software professionals performing system level functional testing. They identified several practices for test session and execution strategies including exploratory testing (e.g., simulating abnormal and extreme situations, exploring against old functionality, feature interaction testing and defect based exploring), systematic comparison (e.g., comparing with another application or version, checking all the effects) and input partitioning (e.g., testing boundaries and restrictions, testing input alternatives, covering input combinations). It is obvious that in a domain like software testing, test goals and strategies for creating test cases seem to be less well defined than problems in other areas like physics and math. Research in software testing has identified several variables that influence the quality and performance of software testing [3], [4]. Among these are knowledge and strategies for testing, as well as external factors.

In response to this need to better understand the software testing process, this works explores the cognitive processes used by testers engaged in software testing. In this paper we use a problem solving approach to map the steps and sequence by which testers perform test activities. Finally, we outline different methods that can be used to research and refine the proposed cognitive model of software testing.

II. Background

The software testing process can be divided into four steps [1]: test design, test automation, test execution and test evaluation. These activities occur within an organizational environment and one or more persons are assigned to perform them. Many factors influence these activities such as organization structure, training, experience, testing knowledge, automation environment and testing standards. All too often, researchers focus on different technical aspects of software testing without taking into account the human aspects of different test activities. Humans involved in the creation of test cases are basically trying to solve certain problems that have been posed to them (e.g., assignments from test managers) or recognized on their own (e.g., the need for additional confidence in releasing a certain feature). Despite the diversity of these testing problems (i.e., test goals), the ways people go about solving them show a number of common characteristics to general problem solving [5]. Problem solving is a very important topic of research in the field of artificial intelligence and cognitive science. Over the years many researchers have delved into this topic and have discovered a great deal of knowledge about how humans solve problems. We recognize here the need for examining and classifying the underlying characteristics of problem solving models when applied to software testing.

III. Foundations for a cognitive model of software testing

Many models representing the problem solving processes are devised principally from Polya's phases [6]. Several psychologists have described the problem solving process as a cycle [7]–[9]. Problem solving occurs throughout life. A young child may be trying to figure out how to solve a mathematical problem. An architect may be designing an apartment building, or a software tester is attempting to find certain effective test inputs using boundary value analysis. Despite the diversity of

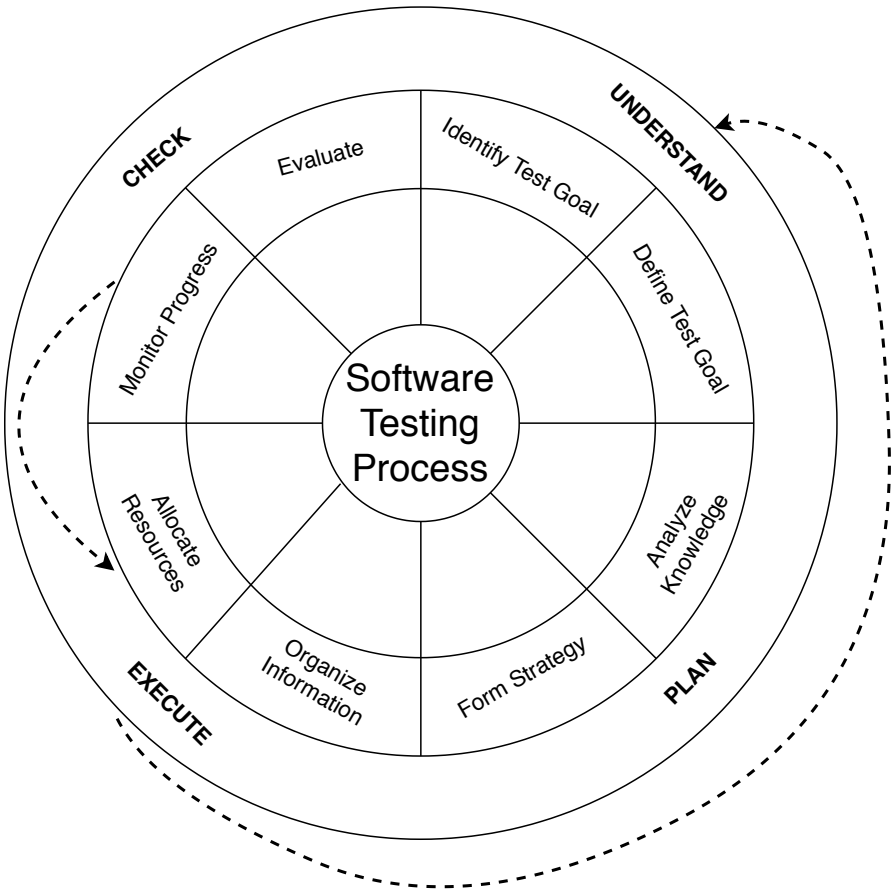

Fig. 1: The software testing process viewed as a cyclical problem solving model.

problems, examining these as a problem solving process, their commonalities and what they tell us about human thought and intelligence is important.

Traditionally, problem solving has been viewed in psychology and artificial intelligence research as determined by representation and search processes [5], [10]. In this way, a problem is represented as a problem space, consisting of states and operators. A human or a program solves a problem when it finds a path from the initial goal to the goal state. In search-based software testing [11], much of the work has focused on understanding different search strategies that might be used for guiding the search towards a test goal.

More recently, problem representation has become an area of high interest in problem solving research [9]. The representation seems to be at least as crucial in determining whether a problem is solved as how the search is performed.

Given these foundations on problem solving, we consider in the next section a cognitive model of software testing performed by human testers based on problem solving activities.

IV. A COGNITIVE MODEL OF SOFTWARE TESTING

To get a handle on the issues involved in studying the psychology of software testing performed by humans, we need a way of defining and classifying the process of human software testing. In response to the need for a better understanding of the test creation process, this work explores the cognitive underpinnings used by testers in the test design phase. The model shown in Figure 1 is centered around our observation that test design and execution can be viewed as a problem solving process. In order to precisely describe this process, we need to better understand the cognitive processes associated with test design.

The software testing cycle viewed as a problem solving process consists of the following stages (mirrored in Figure 1) in which the human tester must:

- *Identify Test Goal:* Recognize and identify the test goal as a problem that needs to be solved. According to Getzels [12], there are three kinds of problems one can identify those that are presented, those that are discovered, and those that are created. A presented test goal is one that is given to the tester directly and it is stated clearly (i.e., predefined criteria-based test goal). A discovered and created test goal, however, is one that must be recognized. In this case, testers are using exploratory strategies and seek out to discover what the test goal by intuition and experience.
- *Define Test Goal:* Define and understand the test goal mentally and what the test cases must do. The test goal definition is the aspect of testing in which the scope and

the test goals are clearly stated. Mental representations are composed of four parts: a description of the initial state of the test goal, a description of the goal state, a set of operators, and a set of constraints. A test goal may be represented in a variety of ways, for example, visually or verbally. For instance, in achieving pairwise coverage [1], one would need to define and represent the goal as the task to create all possible pairs of parameter values that can be covered by at least one test case.

- *Analyze Knowledge:* Organize his or her testing knowledge about the test goal. Everyone approaches a problem situation with a unique knowledge base. For somebody with knowledge in test design techniques, analyzing the knowledge involves creating abstract models and using them to for a strategy to create test cases.
- *Form Strategy:* Develop a solution strategy for creating the necessary test cases. The set of operations you perform to get to the goal state constitutes the test specification (i.e., how the test case should be obtained). In many cases, the operators are not specified in the test goal, but we can infer them from our prior knowledge (e.g., mathematical operators like division and multiplication, mental operators like calculations).
- *Organize Information and Allocate Resources:* Organize information, allocate mental and physical resources for creating and executing test cases. Testers are using general skills such as inferencing, case-based reasoning, abstraction and generalization for organizing the information from different steps. At an even more general level there are metacognitive skills related to motivation and allocating cognitive resources such as attention and effort. In addition, testers can use test automation for embedding the test values in executable scripts and allocate certain computing resources for executing test cases. If tests are run by hand, testers will allocate physical resources and record the results.
- *Monitor Progress:* Monitor his or her progress toward the test goal. This step monitors the results of test creation and execution. For test execution one will use test oracles embedded into scripts or manually monitor when the correct output cannot be encoded easily.
- *Evaluate.* Evaluate the test cases for accuracy. If you find that the test goal is not met, analyze the test goal and then make corrections. Follow the steps to see if the test cases really do not check the test goal.

The testers begins the testing process by analyzing the test goal, breaking it into manageable pieces, and developing a general solution for each piece called a test case. The solutions to the pieces are collected together to form a test suite that solves the original problem (i.e., the identified test goal). The testing cycle is descriptive, and does not imply that all test case creation proceeds sequentially through all these steps in this order. In practice, experienced testers are those who are flexible. In many cases, once the cycle is completed, the steps are usually giving rise to a new test goal and then the steps need to be repeated.

V. What is a Testing Goal?

Before discussing how one would investigate how people design test cases, we define what is meant by a test problem. For our purposes, a tester has a problem to solve when it wants to attain some test goal. We consider a test problem to have four aspects: test goals, assumptions, means to attain the goal and obstacles. The test goal is some state for which some criterion can be applied to assess whether the test problem has been solved. To give an example, the goal might be to check if a certain requirement has been implemented correctly or to find a way to crash the user interface.

Even if different test goals have common aspects, figuring out how to find test inputs to cover all branches in a program seems very different from figuring out how to find interface bugs in the same program. Clearly, these two test goals have many differences. A crucial activity in understanding problem solving is to analyze which test goal differences are important and which are not.

There are two classes of test goals that map directly on the generic problems solved by humans:

- *Well defined test goals* (e.g., coverage criteria, boundary-value analysis, category partitioning) have completely specified initial conditions, goals and means of attaining the goal. Many coverage criteria are well defined. For example, creating tests for covering program branches is usually well defined. The goal is some well specified coverage score. Thus, you will know when you have attained your goal. Finally, the means of attaining the goal are by exercising the different branches in the program.
- *Ill defined test goals* (e.g., stress test goals, fault-based testing) have some aspects that are not completely specified. The problem of finding faults is clearly ill defined. Even if you know how to tell whether a fault is discovered, you wouldn't know exactly what to do to try to achieve this goal in every specific situation.

Test problems differ on how well defined they are and we can consider that this categorization is a continuum of problems rather than a dichotomy.

VI. Investigating Software Testing Practices using Problem Solving Methods

In order to build knowledge on how testers are solving test problems by creating test cases, it is useful to consider the methods used in problem-solving research in cognitive science. Three methods are often used in problem-solving research [9]: intermediate products, verbal protocols, and software simulations.

A. Intermediate Products

Getting intermediate products [5] means that instead of recording and analyzing only the created test case to the problem, we observe some of the work the subject does in getting the test case. If we are interested in how people create test cases for finding logical bugs, we collect information

about the various steps they make in getting to the goal. If we are interested in testing for requirement coverage, we collect and analyze the models, equations and other information the subject writes down in the course of problem solving. The resulting intermediate products provide finer constraints and possible explanations.

B. Verbal Protocols

The second method often used in problem solving research is a verbal protocol [13]. The most common way to collect such data is to ask the subjects to think aloud as they go about solving the problem. The idea behind this measure is to provide information about the course of the problem solving. Verbal protocols can be used in software testing research as a direct evidence for some hypothesis or to generate new ideas that can be tested by other methods.

C. Simulations and Search-Based Testing

A common goal of problem solving research is to build a simulation that is meant to mimic the problem-solving process as revealed by the intermediate steps [5]. In testing, automatic search-based test generation [11] can be used to mimic how testers solve testing problems, to the extent that problem solving can be regarded as information processing.

VII. Conclusions

Problem solving provides the foundation for uniting tester cognitive processes. The software testing model developed here allows for a cyclical process that accommodates both simple and complex test goals.

For the research community, the model proposed in this works provides a reference on which to ground future investigations. In general, more questions concerning testers' cognitive processes in software testing can be posed. Specifically, the process by which a tester identifies and defines a test goal is not fully understood, nor is the process by which testers form strategies for creating test cases. The selection of search strategies could vary with testing experience, but little is known about this. Likewise, how a tester switches between test goals and executes test cases has received little attention.

The model also provides a framework from which to investigate tester knowledge and expertise and it is the first step in evaluating and refining a testers' cognitive process model for software testing.

References

[1] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.

[2] J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? an exploratory study on manual testing practices," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 494–497.

[3] ——, "Defect detection efficiency: Test case based vs. exploratory testing," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 61–70.

[4] J. Itkonen, M. V. Mäntylä, and C. Lassenius, "The role of the tester's knowledge in exploratory software testing," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 707–724, 2012.

[5] D. L. Medin and B. H. Ross, *Cognitive psychology.* Harcourt Brace Jovanovich, 1992.

[6] G. Polya, "How to solve it ny," 1957.

[7] J. D. Bransford, "The ideal problem solver," Scientific American, Tech. Rep., 1984.

[8] J. R. Hayes, "Cognitive processes in creativity," in *Handbook of creativity*. Springer, 1989, pp. 135–145.

[9] J. E. Pretz, A. J. Naples, and R. J. Sternberg, "Recognizing, defining, and representing problems," *The psychology of problem solving*, vol. 30, no. 3, 2003.

[10] A. Newel and H. A. Simon, "Human problem solving," *Englewood Cliffs, NJ*, 1972.

[11] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2011, pp. 153–163.

[12] J. W. Getzels, "The problem of the problem," *New directions for methodology of social and behavioral science: Question framing and response consistency*, vol. 11, pp. 37–49, 1982.

[13] K. A. Ericsson and H. A. Simon, "Protocol analysis," *A companion to cognitive science*, vol. 14, pp. 425–432, 1998.