An Investigation into Qualitative Human Oracle Costs

Sheeva Afshan[1] and Phil McMinn[2]

Department of Computer Science, The University of Sheffield
sheeva@dcs.shef.ac.uk, p.mcminn@sheffield.ac.uk

Abstract. The test data produced by automatic test data generators are often 'unnatural' particularly for the programs that make use of human-recognisable variables such as 'country', 'name', 'date', 'time', 'age' and so on. The test data generated for these variables are usually arbitrary-looking values that are complex for human testers to comprehend and evaluate. This is due to the fact that automatic test data generators have no domain knowledge about the program under test and thus the test data they produce are hardly recognised by human testers. As a result, the tester is likely to spend additional time in order to understand such data. This paper demonstrates how the incorporation of some domain knowledge into an automatic test data generator can significantly improve the quality of the generated test data. Empirical studies are proposed to investigate how this incorporation of knowledge can reduce the overall testing costs.

Keywords: Test Data Evaluation, Testing Costs, Human Oracle.

1 Introduction

Software testing is a laborious, expensive and time consuming process that consumes approximately 30 to 60% of the software development budgets [8]. The process of software testing involves establishing a set of testing requirements, generating test cases that satisfy these requirements, execution of the test cases and determining whether the corresponding outputs are as expected. This process is performed in two essential phases; test data generation and test data evaluation. Test data generation focuses on identifying the test cases. The evaluation phase concerns with verifying the success or failure of the generated test data. Today, various stages of the software testing are still performed manually. Manual testing usually requires a large number of human testers to perform the expensive and time-consuming test procedures by hand. Considering the complexity of advanced software applications today, manual testing is a very expensive operation.

Automation of software testing promises to effectively verify software systems while saving a considerable amount of time [12]. Several techniques for automating the test data generation have been proposed over the last decade; including symbolic execution [3], concolic execution [1] [11], and search-based testing [4]. While these techniques have achieved great success in the test data generation, they have paid no attention to the evaluation phase. As a result, in many real cases, the system behaviour is evaluated manually by a human tester. On the other hand, the automatic test data generators have no domain knowledge of the program under test, and therefore the test data they produce are arbitrary-looking values that are hardly identified by human testers. One of the benefits of such data is that they can force the program to experience unusual yet possible input values and thus can produce unexpected outputs which reveal faults. However evaluation of this type of data is suspected to have longer cognition time. This forms a significant cost, frequently referred as the human oracle cost.

A potential solution to this problem is to incorporate knowledge about the program's input domain into the automatic test data generators [6]. Prior to application of this solution, it is required to ensure whether the arbitrary-looking test data have additional cognition time. To investigate this, we aim to recruit human testers and ask them to evaluate different types of test data. The length of the time each tester spends on evaluating the test data is a key factor in this experiment. The next step is to examine whether the 'natural' test data are less effective in detecting the software faults in comparison to the 'arbitrary-looking' test data. To inspect this, the fault-finding capability of the test data will be computed. If the test data with the lower cognition time turns to have lower fault-finding capability, the test data generator should be

guided to consider the two possible constraints. This study is currently at preliminary stages, and the initial experiments are under progress. Full details about the potential solutions and experiments are presented in Section 3. Prior to this, a background to the test data generation techniques and the oracle problem is given in Section 2. Finally Section 4 concludes the study.

2 Background

The principle of software testing is to apply a number of inputs (test cases) to the program under test, observe and check whether the program response is as expected (see figure 1).

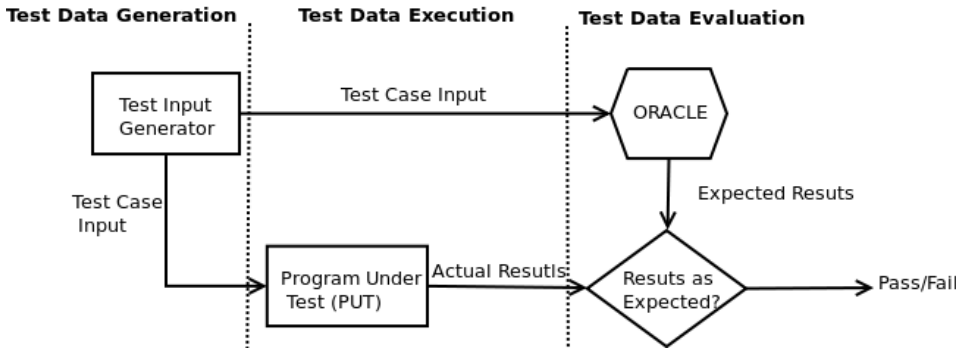

Fig. 1. Different Phases of Software Testing

Test data evaluation is a critical component in software testing. Without evaluating the test data according to an existing oracle, testing does not achieve its goal of revealing failures or assuring correct behaviour in a practical manner. Automatic evaluation is based on existence of a testing oracle which can determine the correctness of the system's behaviour against the a set of test cases. There are various types of oracles, such as Heuristic Oracle [2], Specification-Based Oracle [9] and Pseudo Oracle [2]. An oracle generated from the specifications will only produce correct results if the specifications are correct. A pseudo oracle is as an independently implemented program, that aims to perform the same task as the original but using a different approach (i.e. computational algorithm, compiler, etc) [7].

For most software systems, the existence of an automated oracle that can accurately evaluate the system's behaviour is seldom available. This problem is referred as the oracle problem, and is a major concern for the entire field of software testing. Due to this problem, in many real cases, the system's performance is evaluated by a human tester. When the test data are generated automatically, the manual evaluation is not satisfactory for several reasons. One of the reasons is that the volume of the test data is often overwhelming and therefore requires a large number of human testers to concentrate for arbitrarily long periods of time. This forms a significant cost which is referred as quantitative human oracle costs. To decrease costs, reduction techniques [13]have attempted to reduce the size of the test data by identifying a representative set from the original test suite, that satisfies all the testing objectives. These techniques have used various reduction algorithms and have achieved great success in reducing the volume of the test data [10].

Another basis for dissatisfaction in manual evaluation is the "quality" of the automatically generated test data. The test cases produced by automatic test data generators are usually of poor "quality" in terms of matching the program's input domain. This issue mostly occurs when the program makes use of human-recognisable values for input variables such as 'email', 'url', 'date', 'name' and so on. The scenarios that such data represent, are suspected to require additional cognition time. This is referred as qualitative human oracle costs and contributes to the overall testing costs. The quality of the test data is usually assessed based on coverage, fault-finding capability and human recognisability criteria. The majority of test data generators have

focused only on the coverage and fault-finding capability criteria and have not considered the human recognisability factor. For this reason, the test data they produce, are often strings that appear to be random sequences of characters or arbitrary-looking values rather than meaningful pieces of data. A sample of such data for is shown in Table 1.

Branch	Different random seed used to generate the starting point for the test data search for each branch		Same random seed used to generate the starting point for the test data search for each branch		Supplied test case (1/1/2010, 1/1/2010) used as the starting point	
1T	-4048/-10854/-29141	3308/-25426/-11998	-1247/-17004/9006	3305/6393/-10930	0/1/2010	1/1/2010
1F	4091/-31366/-23576	-9671/1283/-29866	15136/-17004/9006	3305/6393/-10930	1/1/2010	1/1/2010
2T	10430/3140/6733	-14884/-8416/-18743	15136/-17004/9006	-790/6393/-10930	1/1/2010	0/1/2010
2F	-31846/-3340/4891	7021/-24358/13435	15136/-17004/9006	3305/6393/-10930	1/1/2010	1/1/2010
3T	3063/31358/8201	9560/32094/-23160	15136/-17004/9006	3305/6393/-10930	16/1/2010	1/1/2010
3F	-2459/13917/984	6289/31510/-21766	-1247/-17004/9006	3305/6393/-10930	1/1/2010	1/1/2010

Table 1. Shows automatically generated test data produced a search-based tool for a method called "days_between".

The method "days_between", is a part of a C program called Calender, which calculates the number of days between two given dates. The search-based tool used, is commonly referred to as Iguana, developed by McMinn [5]. In the first column of Table 1, T and F refer to the true and false branches respectively. The second column shows test cases that are generated for each branch using a different random starting point. The subsequent column shows similar test data, where the same random seed is used to initiate the search for each branch. In the final column, a human supplied test case (1/1/2010, 1/1/2010) was used as the initial point for the search. The majority of values shown in Table 1 correspond to obscure dates (i.e negative numbers or dates that are several millennia in the past or future). This is because the initial input values are selected from the program's input space by random and thus poorly match the program's input domain. The input space for this method is a set of values that range from -32,768 to 32,767 (which is the range of short int type). However the test cases provided in the fourth column are of higher quality in terms of conforming to an accurate data value. This is because the human supplied test case provides the search mechanism with some domain which leads to production of realistic test values for the date variable. This demonstrates providing the search with some realistic initial values can significantly improve the quality of the resulting test cases.

3 Proposed Solutions

A possible way of improving the quality of the test data is to supply the test data generators with reasonable amount of information about the program's input domain. As discussed in section 2, providing a search based test data generator with even a small amount of information about the program's input domain can dramatically produce simplified and more recognisable test data (see Table 1). Prior to developing techniques that can automatically provide the search mechanism with some domain knowledge about the program under test, confirming the following hypothesises is essential.

To examine whether the arbitrary-looking test data have additional cognition time, this project aims to recruit software developers from Genesys company (an internal software company in Sheffield). The software developers are both employed programmers and students (both undergraduate and postgraduate) who work on real software development projects. They will be asked to evaluate two types of test data; one that is generated from a random initial seed, and one that is generated from a human supplied initial seed. The length of time each student takes to evaluate each set of test data is the main factor in this part.

To examine whether the proposed approach may have an effect on the fault-finding capability of the resulting test data, mutation analysis should be performed on the test data. This part

can assess and compare the capability of both 'natural' and 'arbitrary' test data in detecting faults.

The initial experiment is currently being run in the form of an online questionnaire. The questionnaire is designed in two phases. The first phase is aimed to gather human provided seeds for a number of methods. In this phase, the user is asked to provide a test case for each method together with its corresponding return value. Each method is a segment of Java program, which takes at least one argument and has a return value. The are 15 methods in total, 6 of which have primitive input types, the rest make use of string input types. These methods are selected as they cover the majority of primitive input types, they are simple enough for users to compute the corresponding terurn values, yet complex enough with regards to the number of branches. The users are primarily asked to generate an initial simplified test case for each method (see figure 2).

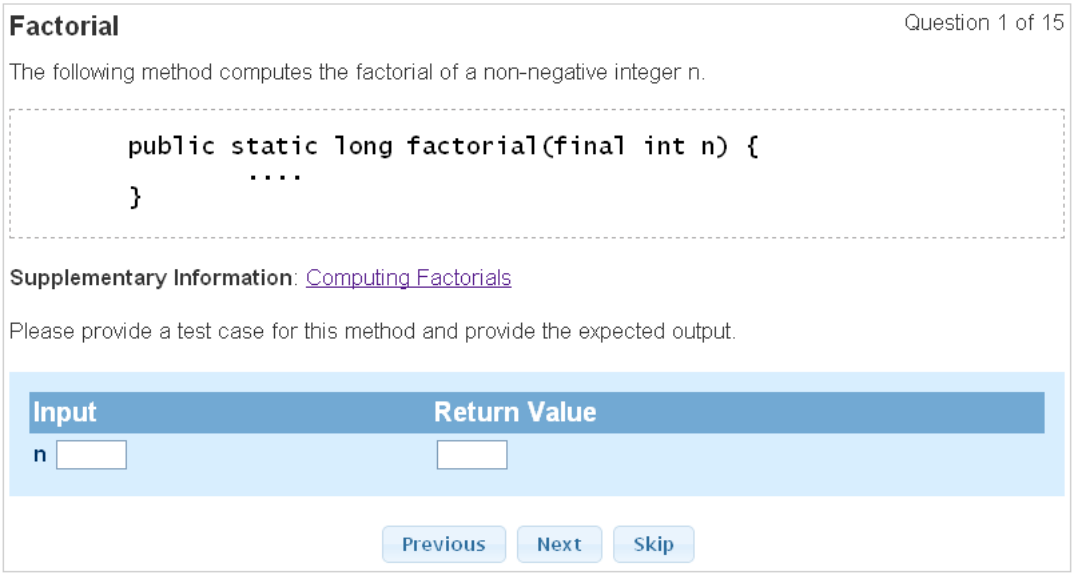

Fig. 2. A screen shot of the online questionnaire for the first stage. For each method the user is asked to provide a test case with the expected output.

Next, the data the users provide are passed to the Iguana tool. The search mechanism employed in Iguana, uses these human provided seeds to initiate the search and thus provides additional test cases that are close to the initial seeds in terms of matching the program's input domain. In the second phase of the questionnaire, the user will be asked to take the role of a human oracle by evaluating the test data achieved from the previous stage (see figure 3). The time each student takes to evaluate each type of the test data will be noted, the complexity of each method, and the programming ability of each student will be considered.

This experiment is currently under progress. 15 people have answered the first questionnaire to date. The questionnaire will be send to more people to obtain more data. As an alternative, Mechanical Turk can be used if additional data are required. The Amazon Mechanical Turk (MTurk) is a internet marketplace that allows computer programmers to ask workers to perform human intelligence tasks by fulfilling the provided questions.

4 Conclusions

This report has reviewed the oracle problem and investigated how quantity and quality of the test data can effect the overall testing costs. While a number of techniques have attempted to alleviate the quantity issues, there has been been seldom work focusing on the quality aspects

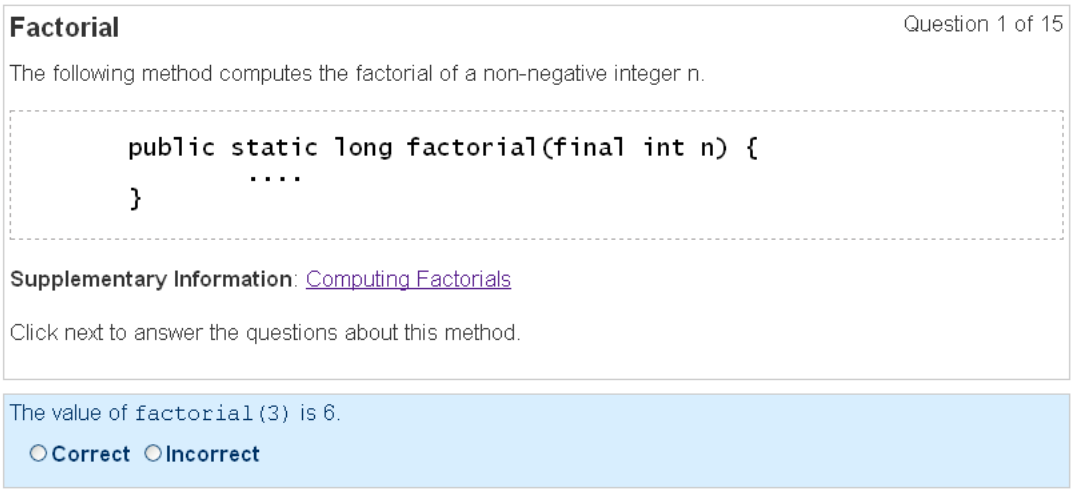

Fig. 3. A screen shot of the online questionnaire for the second stage. Theser is timed while answering each question.

of the generated test data. This is an important problem which needs to be resolved in order to increase the reliability of software system while saving a great deal in time and budget. Reduction of qualitative human oracle cost can significantly contributes to software testing and therefore it is an essential matter.

5 Acknowledgments

Sheeva Afshan is funded by EPSRC grant. Phil McMinn is supported by EP/I010386/1 (RECOST: REducing the Cost of Oracles in Software Testing).

Bibliography

[1] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. *SIGPLAN Not.*, 40:213–223.

[2] Douglas Hoffman. Heuristic test oracles. *Quality Engineering Magazine*, 1999.

[3] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19:385–394.

[4] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[5] P. McMinn. IGUANA: Input generation using automated novel algorithms. A plug and play research tool. Technical Report CS-07-14, Department of Computer Science, University of Sheffield, 2007.

[6] P. McMinn, M. Stevenson, and M. Harman. Reducing qualitative human oracle costs associated with automatically generated test data. In *STOV 2010: Proceedings of the 1st International Workshop on Software Test Output Validation*, 2010.

[7] Phil McMinn. Search-based failure discovery using testability transformations to generate pseudo-oracles. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1689–1696, New York, NY, USA, 2009. ACM.

[8] S.P. Ng, T. Murnane, K. Reed, D. Grant, and T.Y. Chen. A preliminary survey on software testing practices in australia. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 116 – 125, 2004.

[9] M. Núñez, I. Rodríguez, and F. Rubio. Specification and testing of autonomous agents in e-commerce systems. *Software Testing, Verification and Reliability*, 15:211–233, 2005.

[10] A. J. Offutt, Z. Jin, and J. Pan. The dynamic domain reduction procedure for test data generation. *Software - Practice and Experience*, 29(2):167–193, 1999.

[11] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE 2005*, pages 263–272. ACM, 2005.

[12] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, 2001.

[13] Hao Zhong, Lu Zhang, and Hong Mei. An experimental study of four typical test suite reduction techniques. *Information and Software Technology*, 50:534 – 546.