International Journal of New Technology and Research (IJNTR)
ISSN: 2454-4116, Volume-4, Issue-6, June 2018 Pages 95-97

An Initiative to Improve Quality of Software Testing Techniques and Calculating Total Number of Failures using Bayesian Method

Vaibhav Sancheti, G. Srikari Sharma

***Abstract*— The current concern regarding quality of evaluation performed in existing studies reveals the need for methods and tools to assist in the definition and execution of empirical studies and experiments. This paper discusses those issues specific for evaluation of software testing techniques and proposes an initiative for a collaborative effort to encourage reproducibility of experiments evaluating software testing techniques . This paper also proposes the development of a tool that enables automatic execution and analysis of experiments producing reproducible research compendia as output that is, in turn, shared among researchers. There are many expected benefits from this endeavour, such as providing a foundation for evaluation of existing and upcoming STT, and allowing researchers to devise and publish better experiments. Total number of failures of a software system can help practitioners to have a better understanding of the software quality. This paper proposes a model to predict the total number of software failures in a software system by analyzing the failure data from testing using failure data and code coverage are combined in a Bayesian way. The methodology is applied to real world failure data to validate its predictability. The predictive accuracy of our model is also evaluated with different methods. The results of our experiment show that our proposed model can provide a very good estimation of the total number of failures. The estimation is stable from a very early point on.**

***Index Terms*— Bayesian Method, Software Testing.**

I. Introduction

Testing is defined as a process of evaluation that either the specific system meets its originally specified requirements or not. It is mainly a process encompassing validation and verification process that whether the developed system meets the requirements defined by user. Therefore, this activity results in a difference between actual and expected result. Software Testing refers to finding bugs, errors or missing requirements in the developed system or software. So, this is an investigation that provides the stakeholders with the exact knowledge about the quality of the product. Software Testing can also be considered as a risk-based activity. The important thing during testing process the software testers must understand that how to minimise a large number of tests into manageable tests set.

II. EXISTING TESTING METHODS

For the commencement of the Testing process, the first step is to generate test cases. The test cases are developed using various testing techniques, for the effective and accurate testing. The major testing techniques are Black box testing, White Box testing and Grey Box testing. White Box testing is significantly effective as it is the method of testing that not only tests the functionality of the software but also tests the internal structure of the application. While designing the test cases to conduct white box testing, programming skills are requisite to design the test cases. White box testing is also called clear box or glass box testing. This kind of testing can be applied to all levels including unit, integration or system testing. This type of testing is also called Security Testing that is it fulfils the need to determine whether the information systems protect data and maintains the intended functionality. As this kind of testing process makes use of the internal logical arrangement of the software hence it is capable enough of testing all the independent paths of a module, every logical decision is exercised, all loops are checked at each boundary level, and internal data structures are also exercised. However, white box testing serves a purpose for being a complex testing process due to the inclusion of programming skills in the testing process. Black Box testing is a testing technique that essentially tests the functionality of the application without going into its implementation level detail. This technique can be applied to every level of testing within the SDLC. It mainly executes the testing in such a way that it covers each and every functionality of the application to determine whether it meets the initially specified requirements of the user or not. It is capable of finding incorrect functionalities by testing their functionality at each minimum, maximum and base case value. It is the most simple and widespread testing process used worldwide.

III. ISSUES

The empirical software engineering community has been dedicating efforts in breaking down the fundamental elements of an experiment in software engineering. Despite the efforts and discussions regarding appropriate methods to conduct experiments in software engineering, evaluating STT is not easy since many information and artefacts are required in order to thoroughly evaluate the capabilities of a testing technique. First of all, software testing itself is a very multidisciplinary area, that is able to comprise distinguished and combined types of software such as standalone, web services, real time, critical systems, embedded systems, etc. Therefore, drawing a precise line to establish the scope affecting the test is very challenging. At the same time, trying to draw that line is one of the most important steps to define a good experiment, since it allows researchers to identify:

Vaibhav Sancheti, Computer Sciences, VIT University, Vellore, India
G. Srikari Sharma, Computer Sciences, VIT University, Vellore, India

context, subjects, limitations, and more importantly, which variables can/must be controlled. Why are we not seeing, over the years, a significant increase in the number of empirical findings regarding STT? There is an array of classifications, configurations and guidelines that help researchers to define, conduct and report their experiments. But manifesting those methods for constructs specific to STT evaluation is very challenging. For example, statistics often require large sets of significant data (usually hard to obtain or unavailable) to achieve conclusive results. In addition, the need for human involvement in testing makes the generation of a large number of test sets infeasible. Similarly, obtaining defect data for larger test sets is nearly impossible since detection of defects will always be, to a certain degree, a stochastic process. Availability and access to information is still limited when evaluating STT. For instance, the System Under Test (SUT) may not be ready for testing, test cases may present limited coverage, defect data is unknown, among others. Even when available, many researchers disregard the representativeness of their artefacts (e.g. choice of inappropriate programs or unreal defects) or even the minimum sample size required to claim statistical significance of results. That creates gaps and validity threats that seriously compromise credibility of results. There are existing techniques for stochastic generation of data for testing. More specifically, they rely on models to generate well-formed data. Those types of strategies enable control over automatic generation of large sets of artefacts, hence assisting experimenters who struggle with availability and accessibility of objects in an experiment. Thus, existing approaches can already be applied to overcome those issues. Besides getting data, organization and analysis of all elements in an experiment intimidates many researchers. Devising an appropriate experimental design to comply with one's hypotheses, objectives and variables is overwhelming and a poor design lead to confusing reports. Similarly, lack of experience and/or knowledge in statistics hinder researchers to harness full potential of their data causing them to either enhance or destroy credibility of conclusions. For example, many researchers recklessly use parametric tests (such as ANOVA or t-test) as a rule of thumb, without proper investigation of its assumptions, sometimes leading to erroneous conclusion. In its early start, the empirical software engineering community turned to other disciplines that have been dealing with empirical evaluation over decades (e.g. biology and social sciences) in order to overcome general difficulties in managing data, subjects, statistical analysis, etc. The software testing community seems to be performing more experiments. However, they neglect validation of existing experiments and proposal of innovative strategies to help their fellow researchers in overcoming the specific challenges of evaluating STT. That being said, validation of experiments needs to be performed through reproduction, replication or re-analysis of existing experiments with STT.

IV. METHODOLOGY

A. Model Based Testing

Model-based testing is a software testing technique in which the test cases are derived from a model that describes the functional aspects of the system under test. Generally, it makes use of a model to generate tests that includes both offline and online testing. Some advantages of model-based testing are:

1) Higher level of Automation is achieved.
2) Exhaustive testing is possible.
3) Changes to the model can be easily tested.

B. Ghost Transition Testing

These testing for part of model based testing wherein the only difference that lies between ghost transition testing and all pair testing is that the transitions which are not defined/ visible for a state (called as ghost transition) when triggered will remain in same state as the original. Such type of transitions when triggered does not affect the change in state.

C. Multiple Condition Testing

Development of tests by white box testing method in which test cases are designed to execute combinations of single condition outcomes and each of the conditions or branches are evaluated.

D. Number of Failures

It is highly plausible, that the additional information of code coverage could help us to predict the total number of failures better than with failure data alone. Here for the Bayesian computations the hyper geometric distribution is used as a prior. This method to use the hyper geometric distribution is known and used as the "Capture-Recapture Method" in biology since many decades. It goes back to P.S. Laplace. He used it in the year 1786 to estimate the population size of France. Here we use it to estimate the total number of failures. This distribution can be used if we know that the faults (the causes of our failures) are somehow spread "uniformly" all over the code. To verify this assumption we check the relationship of code coverage achieved and number of detected failures. In the data set examined in the next chapter there is an approximately linear relationship between coverage achieved and failures detected. Other people observed this relationship too (at least after some initial time).

V. ALGORITHM

For our computations we need as input:

a) number of failures detected during the test b) total number of statements (or whatever is used for coverage) c) percentage of coverage reached

$$p \quad \frac{\binom{s}{n}\cdot\binom{S-s}{N_0-n}}{\binom{S}{N_0}} \quad =$$

where

N0=Total number of failures

n:= number of failures detected

S:= total number of statements

s:= number of statements covered

VI. LITERATURE REVIEW

The initiative begins with a collaborative effort within the testing community to share and standardize methods and artefacts used to evaluate STT. Therefore, the research comprises the state of art of empirical studies in software engineering, software testing, statistics, experimental designs, among others. The outcome is the development of techniques and tools to provide support and execution of experiments. Therefore, researchers will extend the body of knowledge by proposing new methodologies to perform reproducible studies with STT. Even though replication and re-analysis are just as important as reproducibility, we believe that focusing first in reproducibility will allow researchers to quickly overcome the general lack of availability and accessibility of data required to replicate/re-analyse the existing experiments. Thus, as the initiative strengthens, we enable more replication because the community will rely on a larger repository of experiments. In summary, we intend to encourage practices similar to the initiative of reproducible software engineering but adapting them to the software testing community. We begin by defining input and output of our process. The input are the study parameters, such as the objects of study, a SUT, sets of test cases, number of subjects, a general hypothesis and goals. The output, in turn, is a compendium (i.e. a package), named Reproducible Software Testing Research Compendium(RSTRC).

The proposed approach (Bayesian Zipf-models+coverage) to predict the total number of failures of a software system can be used with failure time data or the number of test cases. In both cases it can also be used with grouped data. This approach gives excellent predictive performance with the data set of Wang et al.. The same methodology can be used to fit time or number of test cases vs. code coverage. In the future, we would like to extend our experiment to compare the estimated constant c in Zipf(c) with the shape of the usage profile (at a functional level), which can also be described via Zipf's law. It is conjectured, that there is a strong connection, which can be used for even better estimation at early levels (prior distribution for c from the usage profile). As the predicted mean value function fits the - in the future observed - failure data extremely well, the method described in and could be used very early to detect failure prone modules or irregularities during the test process (statistical process control).

VII. CONCLUSION

This paper discussed issues and goals regarding the need for more reproducible research with software testing techniques. Based on existing achievements and contributions from empirical software engineering, we proposed introduction of guidelines, artefacts and methods to encourage researchers to produce compendia targeting evaluation of STT (RSTRC). The objective is to define, execute and deploy experiments with improved description, accessibility and availability of required artefacts to reproduce, replicate and re-analyse the experimental findings. Through a collaborative effort among researchers, sharing and reproduction allows the community to identify which experimental findings are consequences of an actual cause effect relationship, rather than an artefactual result. That leads to a more reliable body of knowledge, increasing confidence in existing research and setting high standards for upcoming testing techniques.

The proposed approach (Bayesian Zipf-models+coverage) to predict the total number of failures of a software system can be used with failure time data or the number of test cases. In both cases it can also be used with grouped data. This approach gives excellent predictive performance with the data set of Wang et al. [6]. The same methodology can be used to fit time or number of test cases vs. code coverage. In the future, we would like to extend our experiment to compare the estimated constant c in Zipf(c) with the shape of the usage profile (at a functional level), which can also be described via Zipf's law. It is conjectured, that there is a strong connection, which can be used for even better estimation at early levels (prior distribution for c from the usage profile). As the predicted mean value function fits the - in the future observed - failure data extremely well, the method described in and could be used very early to detect failure prone modules or irregularities during the test process (statistical process control).

REFERENCES

[1] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad “Software Testing Techniques: A Literature Review”
[2] Jai Gaurl, Akshita Goya1, Tanupriya Choudhury and Sai Sabitha ”A Walk Through of Software Testing Techniques”
[3] Francisco G. de Oliveira Neto, Richard Torkar, Patricia D. L. Machado “An Initiative to Improve Reproducibility and Empirical Evaluation of Software Testing Techniques”
[4] Harald A. Stieber, Linghuan Hu, W. Eric Wong” Estimation of the Total Number of Software Failures from Test Data and Code Coverage”
[5] Eduard P. Enoiu∗, Adnan ˇCauˇsevi´ ,Daniel Sundmark, Paul Petersson “A Controlled Experiment in Testing of Safety-Critical Embedded Software”
[6] S. Wang, Y. Wu, M. Lu, and H. Li, “Software Reliability modeling based on test coverage,” in Proceedings of 9th International Conference on Reliability, Maintainability and Safety, Guiyang, China, 2011