Strategic Factors Model for Successful Fault Injection Testing

Nor Shahida Mohamad Yusop, Wan Faezah Abbas, Maslina Abdul Aziz, and Syahmi Iskandar

Abstract—**Software Fault Injection Testing (SFIT) technique can determine common error conditions through the behavior of observations; discover the interaction weaknesses and reveal how the system react when abnormalities or fault is being injected. In a nutshell, SFIT is a process of building defensive mechanism to prevent unwanted consequences emerges from the system and it is widely considered as an important technique of developing robust system. This paper offers and empirical knowledge of SFIT specifically on the testing practices in Malaysia and factors influencing the success of SFIT process. Data is collected using semi-structure qualitative interview approach. This research discovered that there are three main factors influencing the SFIT process which are Software Tester Knowledge, Software Tester Experience and Test Management of FIT Process for a successful SFIT.**

Index Terms—**Fault injection, software fault injection testing, software testing.**

I. Introduction

Software testing is an important process throughout the life cycle of software development in order to support and enhance the reliability and the qualities of the system developed. Studies estimate that more than 50% of the development cost is devoted to the testing [1].

National Institute of Standards and Technology (NIST) in collaboration with Research Triangle Institute (RTI) have conducted a survey in 2002 in order to estimate the economic impact of inadequate software testing method and tools that resulted USD 59.5 billion annually (Research Triangle Institute, 2002) [2]. This shows that a proper testing procedure is very crucial in software development and more research in software testing area should be done in order to benefits the software development process.

Besides increasing software development cost, inadequate testing procedures will lead to low quality of the system developed, dissatisfied users, increasing the maintenance costs and might produce unreliable and inaccurate system (Srivastava, Kumar, Singh & Raghurama, 2011) [3]. The key issue surrounding the software testing paradigm is the effectiveness of testing technique to find the hidden defect or bugs. Aaron and David (2001) [4] claimed that besides faults through the system codes, the external part of the system where human interaction accounts for roughly half of the system outages.

Manuscript received July 16, 2012; revised September 2, 2012. This work was supported by the Fundamental Research Grant Scheme (FRGS) under Contracts 600-RMI/ST/FRGS 5/3/Fst (235/2010).

The authors are with the Faculty of Computer and Mathematical Sciences University Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia.(e-mail:nor_shahida@tmsk.uitm.edu.my,wfaezah@tmsk.uitm.edu.my, maslina@tmsk.uitm.edu.my, syahmi.iskandar@yahoo.com).

Recent studies had shown that coding is not the main factor that caused software defect. External factors such as improper interaction with the system can also be a contributor [4]. One of the proposed testing technique is to encounter this software defect is using Software Fault Injection technique.

The scope of this research is Malaysian organizations involved with system development process which encircles on the testing phase of the System Development Life Cycle (SDLC). The testing technique scopes are Fault Injection Testing (FIT) and Software Fault Injection Testing (SFIT). This research provide the empirical study on the real SFIT practices done in Malaysian organization and produced the SFIT model that can assist software testers in implementing the process.

II. Literature Review

A. Software Testing in Malaysia

In 2005, a survey with 41 organizations in Malaysia shows that only 41.5% of the organizations have awareness on software testing and implement formal software testing at the end of their coding phase [5]. At present, Malaysia has more than two thousand MSC local-status companies in the fields of software development, creative multimedia, support services, hardware design and others. It is forecasted that software testing services to be worth USD18.3 billion (about RM58.5 billion) in 2013, compared to USD12.6 billion in 2009; growing at a Compound Annual Growth Rate (CAGR) of 9.8 per cent over the period [6].

With the Malaysia aspiration to become a high-income nation by the year 2020, Malaysian Software Testing Board (MSTB) as the authority in Malaysian software testing industry has identified software testing services as a new source of economic growth under the Tenth Malaysia Plan (10MP). It is well recognized that a trained, skilled and well-educated workforce is critical in enhancing work and economic performance and sustaining competitiveness as Malaysia transforms into an ICT-driven and knowledge-based society [7].

B. Fault Injection Testing (FIT)

Modern applications are full-fledged complex systems that are often heavily technology dependent and error-prone application, poses a new challenges to quality assurance and testing. Fault Injection Testing (FIT) is a value added testing technique that is recommended to be performed in order to access the system behavior. It is a fault-base technique that aims specifically to break the system functionality. In this manner; the weakness of the interaction can be discover, and how the system reacts can be revealed [8].

Fault injection techniques (FIT) can yield seven benefits:

an understanding of the effects of real faults, assessment the efficacy of fault tolerance mechanisms, forecasting of the faulty behavior of the target system, estimating the failure coverage and latency, exploring the effects of different workloads, identifying weak links in the design and finally studying the system's behavior in the presence of faults [9].

The earliest work for fault injection can be traced back to Harlan Mill from IBM in early 1972 where the original idea was to estimate the reliability based on an estimate of the number of remaining faults in a program. The benefit of doing this testing is to learn how badly the system can behave when things go wrong.

Most studies in FIT focus more on hardware system validation like Simulation based FIT, Hardware-Implemented FIT (HWIFI) and Software-Implemented FIT (SWIFI). Simulation based FIT had been proposed for dependability evaluation. In this approach, faults are injected into a simulation model of the system which allows testers controlling the timing, the type of fault, and the affected component in the model [10].

Basically all these fault mechanisms are categorized as internal fault where software testers are required to have basic programming skills. It is important to note that that software-based fault injection has drawbacks for robustness testing since injecting faults via software might allows different parts of a system to be targeted.

C. Software Faults Injection Testing (SFIT)

The Software Fault Injection Testing (SFIT) is one of known solution to address the software fault problem. Software FIT is different from traditional black-box software testing where the testers are required to know the system process. This technique allows software tester to determine software robustness by feeding irregular input events. This is something that traditional software testing typically fails to address [11].

By using SFIT, a tester can predict whether that the test cases are able to detect faults. If SFIT process is replayed with enough test cases and if the test profile frequently propagates anomalies, testers should be able to assess a higher reliability score for the software compared to the score which is strictly based on a single test case [11].

By using SFIT approach in robustness testing of component-based systems raises an issue whether the faults injected at interface level to represent possible consequences of residual software bugs in preceding components. Thus, forecasting the faulty behavior of the target system must include the measurement of the coverage provided by the fault tolerance mechanisms [12].

One of the approaches for the emulation of software faults is the actual modification of the target code in order to inject software faults according to the most frequent types of real software faults found in field studies. In Malaysia; few researches on SFIT were conducted. Zamli et. al (2007) [13] believed in optimized test cases using Java unit testing tool called JTst and combined with fault injection strategy to test the robustness of the Java classes in the system code. They used t-way combinatorial test cases that can be used to locate faults.

Despite injecting the faults through the code, software tester might break the system by injecting external faults. Aaron and David (2001) [4] claimed that half of computer system outages were actually caused by external faults. Their survey revealed that no research interest in addressing human error. Thus, the behaviours of the human operator need to be addressed in testing strategy to increase the testing coverage.

Having SFIT as a testing principle not only helps reduce defect slippage to customers but also helps to maintain and subsequently increase the quality of the product if right fault injection strategy is implemented [14]. This can contribute to the quality of the product and the test team themselves.

SFIT tries to determine whether the response of the system matches its specifications in the presence of a defined space of faults. Normally, faults are injected in the perfectly chosen system states and points determined by an initial system analysis.

Software Fault Injection Testing (SFIT) technique is a flexible approach of injecting faults compared to hardware injection, but it has disadvantages like its incapability to inject faults into locations that are inaccessible to software and might disturb the workload running on the target system. Careful design of the injection environment can minimize perturbation to the workload.

III. Research Process

Generally, this research is categorized under exploratory research where it studies the current practices and the factors influencing the software fault injection testing in Malaysia. This research used a case study as the strategy inquiry mechanism to get more extensive multitude of SFIT data from the industry. Semi-structured qualitative interview was chosen as data collection method due to its ability to provide complex textual descriptions.

This research used non-probability sampling, which is Purposive Sampling for the participant selection process. Purposive sampling used specific predefined groups which involved assembling sample of persons with known or demonstrable experience and expertise in some area [15]. We have identified and selected five respondents who work in software testing industry and involved with SFIT process. Each respondent had more than three years' experience in software testing and responsible with a different type of project or system testing.

A. Research Design

The research design is divided into three phases where each phase consists of several activities, objectives, method used and its deliverables. First phase consists of initiation of research where research plan are devised. Phase 2 consists of pilot study where pilot data collection is done to get initial data analysis. Finally, the third phase is the empirical study is done after the data from the pilot study have been refined and analyzed.

B. Data Collection

Semi-structured face-to-face interview was applied during this data gathering process. SFIT methodology research model were used as a guideline for the researcher to lead the interview. During the interview, the research background is briefed to respondent and conversations were recorded using a digital recorder as reference.

IV. Analysis of Data

In the following section, we will describe the results of the analysis. First, the software tester experience and their testing environments are depicted. Then we describe the SFIT processes that explain the similarities practices between the software testers. After that, processes together with affecting factors describe the successful practice of SFIT.

A. Description of Software Tester Background

Tester 1 is a Senior QA Engineer with CTFL certification and worked in software testing for more than four years. SFIT is not a standard practice in his organization, but it is performed on the initiative of the testers. The white-box testing is used for SFIT purpose.

Test 2 is the Lead Test Engineer that involved in software testing for more than eight years. She works with telecommunication equipment vendor and currently responsible with testing the Unix-based telecommunication system. Using white-box testing method, she wrote Korn shell scripting to inject wrong data conditions and ineffective data.

Tester 3 is a Software Test Engineer who involved with software testing for seven years. She works for international telecommunication system provider and responsible on testing several telecommunication Network Management Products. She used scripting and simulation for SFIT process while testing on different data type like CSV, HTML, SNMP and few more.

Tester 4 is a Software Test Engineer that involved with software testing for seven years. She works for international telecommunication system provider and responsible on testing Unix-based client-server system. SFIT process is not a standard testing practice for the product testing. It is only done base on request by software architect and developer.

Tester 5 is a Software Engineer who involved with software testing for eight years. He works for international telecommunication system provider and responsible for testing the web-based Tivoli products such as ITNM (IBM Tivoli Network Manager), TBSM (Tivoli Business Service Manager) and Tivoli OMNIbus. SFIT is a standard practice used in his organization. Since most the products consist of multicore functionalities, SFIT helps in evaluating the system interactions by focusing on the attack point of the core functions to break its functionalities.

B. Common SFIT Processes Among Malaysian Software Tester

Software Fault Injection Testing (SFIT) offers a way to measure the effectiveness of the target-system by purposely inserting faults at a particular location which intention to cause the target-system to fail. This technique allows the software testers to monitor how the system behaves when something goes wrong and to find solutions to improve the quality of the system. In most test organizations, SFIT is not a standard practice but rather to a complementary testing strategy to increase a number of faults found during system test.

Based on our findings, there are similarities based on five software tester testing practices. The first process, "fault identification" shows different type of faulty data used in preparing SFIT test cases. Tester 1 focuses on field limitation on filed limitations and restrictions; tester 2 focus on wrong data conditions and ineffective data; tester 3 focus on negative scenario or faulty processes; tester 4 focuses on positive and negative scenarios and tester 5 focus on injecting negative values, data beyond valid range and outbound scenarios. It also identifies the attack point where and when the faulty data need to be injected.

The process "test case design" denoted how the SFIT test cases are created and planned. This includes the testing environment setup and pre requisite test setup. In "test execution" process, tester will run the execution based on the defined test cases using manual or automation. A manual testing meant that the tester will inject the faulty data through the user interface while automation testing require tester to write test script to inject the faulty data. Then "behavior observation" process will observe the defect occurrence and "test report preparation" process will analyze the defects found and documented.

The common SFIT processes are divided to three main stages; a preparation stage, an execution stage and an evaluation stage (Fig 1).

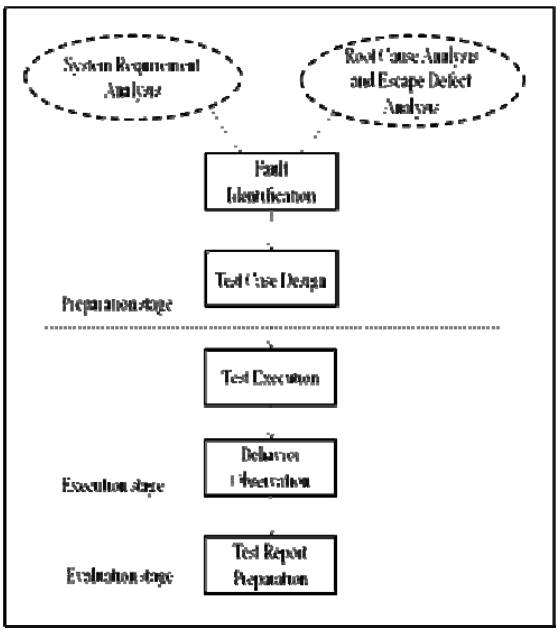

Fig. 1. Common SFIT process among malaysian software tester

In the preparation stages, SFIT strategy is identified from two main sources; system requirement analysis and Root Cause Analysis (RCA) and Escape Defect Analysis (EDA).

The input from the system requirement analysis serve as a benchmarks and control boundary while the input of RCA and EDA historical data will give the common problems and possible scenarios that may be considered in the test strategy. Software tester studies the function of the target system, process flow, system parameter and field, system parsing and time handling in order to gain in-depth knowledge before conducting the SFIT process. Firstly, fault identification process will identify the fault type based on field restriction and limitation, faulty process, data beyond valid range and outbound scenarios. Then identify the attack point to insert the faulty data. Usually, this step requires knowledge and experience of the tester to pinpoint the location that have high potential interaction fault. Once the fault type and locations are determined, tester will start creating the test cases that

incorporated positive and negative scenarios. Sometimes, the test cases will also include hacking strategies like manipulation of the process flow, sequence violation, module penetration and database penetration such as SQL injection. Other than that, interrelation between modules is considered to observe interactions behavior.

In the execution stage, tester will start the test execution either using automation or manual injection according to the fault type and attack location identified. Manual fault injection usually involves with user-input testing by inserting faulty data through the system user interface. While automation injection is done using test script, which focus on data-driven input using real customer data in order to simulate the real system environment. During the test execution, tester will monitor the system behavior and the database reaction towards the injected faults or errors. Sometimes, the neighboring module would be impacted by the injected faults. By right, when a faulty data is injected to the system, the system will stop and issue a warning to notify users. The system is reported as a defect when the system still resume its operation even a fault is being injected.

The execution stage involved with fault reporting. If errors persist during the execution, tester is required to perform RCA and write result evaluation report. The evaluation process will analyze the log files and identify the defect severity. The report is used by the developer for bug-fixing process.

C. Factors Influencing the Success of Software Fault Injections Testing (SFIT)

In order to accelerate high reliability of software fault injection, the system should undergo all sort of testing which require inserting various types of software faults. SFIT typically can be applied with white-box and black-box testing methodology. Whitebox testing requires knowledge of the internal structure of the target-system with the aim of seeking errors that are difficult to detect while black-box testing generally compares the application's behavior with requirements. Thus, a combination of white-box and black-box testing techniques (grey-box testing method) is very useful when developing test case designs or models for SFIT.

This study has classified three categories of factors which are testing management, tester knowledge and tester experience. These factors are identified based on the success practices of the tester running SFIT. Table I shows the categories of factors that have been defined.

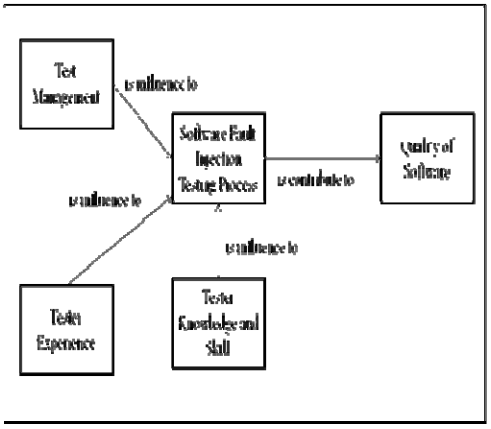

Fig. 2. Factors model of successful fault injection testing process

Fig. 2 shows the most essential factors that influence the success of SFIT.

The SFIT process was associated with the software quality. Software quality becomes dominant objective for software testing process. The focus was in producing a system which has six quality attributes; functionality, reliability, usability, efficiency, maintainability and portability. The success of SFIT process contributes to the effectiveness of software quality is determining faults with regards of the fault severity.

Tester knowledge is the most important affecting factor that determined factors of SFIT process. Tester knowledge includes system domain, system familiarizations and tester skill. According to the tester, system familiarizations are obtained by continuously doing the SFIT process on the target system and detail observation on the system behaviour during the testing period. This will help the tester to determine the type of fault to be injected and which location to attack. Tester with high knowledge on system behaviour and the process has an advantage to manipulate the system to suit their testing purpose and later predict the outcome from the test. Apart from that, several essential skills for software tester such as programming skills and knowledge on products such as Java, Paros, Jmeter, Jbadboy, Oracle, Unix, Linux and other related products and tools related to system development process are essential in performing SFIT. The SFIT process as also associated with tester experience. The tester experience is based on education background and working experience. Eventually experience come a long way and hard to be achieve by level entry software tester. The tester experience was classified to tester total working experience and experienced on the system-in-test and. The more years a tester involves in the testing industry the more competent and confident they do the testing.

Based on the survey, tester with more than 5 years working experience dealt with fewer problems in SFIT. According to testers (more than 7 years working experience), they need about two or three weeks to fully understand the new system developed. While inexperience tester (less than 7 years) said they need at least a month. Experience with system-in-test also helps tester do the SFIT since they have knowledge of overall functions, system interactions, faults to inject and location to attack. Other than that, experienced tester find SFIT is a challenging testing job to discover defects from a different angle. The test management was also influenced the SFIT process. Involvement of test planning was seen as a means to increase the SFIT strategy by improving test design and documentation. Most of the testers practiced documenting the test cases in early testing process. This will eventually help on testing schedule and prioritizing testing tasks.

V. Conclusion

This paper contributes to new knowledge on software fault injection testing (SFIT) specifically in Malaysia perspective. A framework for SFIT has been developed based on software tester practices in Malaysian context. This framework highlights three major factors influencing SFIT which are Tester Knowledge, Tester Experience and Test Management of FIT Process.

This research found that SFIT is not a compulsory practice for every type of system testing. It depends on the type of

system and type of data the system operates. It is also found that the practices between different organizations are almost similar.

For future research extension, the study of tester knowledge and experience in Software Fault Injection Testing (SFIT) can be broaden and identify what knowledge have the highest impact in influencing the SFIT process.

TABLE I: CATEGORIES OF FACTORS FROM RESEARCH

Tester	Success Practices of SFIT	Category of Factors		
		Testing Management	Tester Knowledge	Tester Experience
1	Create test planning	√		
	Understanding system environment and prerequisite requirements		√	
	A good communication skill among development team		√	
2	Understand product perspective such as business rules, system logic and domain knowledge		√	
	Design associated test cases	√		
	Incorporated skill such as a programming language, Oracle, Unix and etc			√
	Involved with experienced tester only			√
3	Perform test cases review	√		
	Leads by experienced tester			√
	Understanding n system requirements		√	
4	Follow the requirement	√		
	Manipulating user interaction base on tester logic thinking		√	
	Understanding process flow		√	
5	Focus attack on core functionality			√
	Insert data beyond requirement range		√	
	Test different scenarios on each fields	√		

ACKNOWLEDGMENT

This work was sponsored by the Fundamental Research Grant Scheme (FRGS) under Contract 600-RMI/ST/FRGS 5/3/Fst (235/2010).

REFERENCES

[1] B. B. Beizer, *Software Testing Techniques*, 2nd ed. Van Nostrand Reinhold Co. New York, NY, USA, 1990

[2] *The Economic Impact of Inadequate Infrastructure for Software Testing*, National Institute of Standard and Technology (NIST). Research Triangle Institute, 2002.

[3] P. R. Srivastava, S. Kumar, A. P. Singh, and G. Raghurama, "Software Testing Effort: An Assessment Through Fuzzy Criteria Approach," *Journal of Uncertain Systems*, vol. 5, no. 3, pp. 183-201, 2011.

[4] B. B. Aaron and P. A. David, "To Err is Human," in *Proceedings of the First Workshop on Evaluating and Architecting System dependability* (EASY '*01)*, Göteborg, Sweden, 2001.

[5] F. Baharom, A. Deraman, and A. R. R. Hamdan, "A Survey on the Current Practices of Software Development Process in Malaysia," *Journal of ICT*, vol. 4, pp. 57-76, UUM Press, 2005.

[6] MSTB, EPU to Hold Forum for Public Conusltation on Software Testing," Bug Free, vol. 1, 2010
[7] A. A. R. Zainol, "National Workforce Transformation," in *Proceedings of the 4th National Conference on the Civil Service*, Kuala Lumpur, 1999.
[8] S. Manaseer, F. A Masooud, and A. A. Sharieh, "Testing Loaded Programs Using Fault Injection," in *World Academy of Science, Engineering and Technology*, no. 3, pp. 86-89, WASET , 2005
[9] H. Ziade, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," *The International Arab Journal of Information* Technology, vol. 1, no. 2, 2004.
[10] J. Carreira, H. Madeira, and J. G. Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units," in *Dependable Computing And Fault Tolerant Systems*, vol. 10, pp. 245-266, USA: Springer-Verlag, 1998.
[11] J. M. Voas and C. Norman, "Marrying Software Fault Injection Technology Results with Software Reliability Growth Models," in *Fast* Abstracts *ISSRE 2003*: Chillarege Press, 2003.
[12] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, New York: Kluwer Academic Publishers, 2003
[13] K. Z. Zamili, N. A. M. Isa, and M. F. J. K. A. S. N. Azizan, "Designing a Combinatorial Java Unit Testing Tool," in *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology (ACST'07)*, pp. 153-158, Thailand: ACTA Press, 2007.
[14] V. Suma and T. R. G. Nair, "Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels," in International *Conference on Software Engineering (ICSE)*, vol. 42, Singapore: WASET. 2008
[15] W. Neuman, "Social Research Methods: Qualitative and Quantitative Approaches 7th Edition," *Boston: Pearson Education*, Inc.2011

Nor Shahida Mohamad Yusop is a Lecturer at Universiti Teknologi MARA, Malaysia. . She holds a master degree in Software Engineering, from CASE Universiti Teknologi Malaysia. She worked as a Software Engineer at Motorola Multimedia, Cyberjaya. Her research interests are in software testing and requirement engineering.

Wan Faezah Abbas is a Lecturer at Universiti Teknologi MARA, Malaysia. She holds a Msc in Information Systems Engineering from UMIST, Manchester, United Kingdom. She previously work at Motorola Cyberjaya as an Application Engineer in testing field. Her research areas are in Software Testing and Enterprise Systems.

Maslina Abdul Aziz is a Lecturer at Universiti Teknologi MARA, Malaysia. She holds a M(E) in Software from University of Queensland, Australia. She worked as an Executive at Multimedia Development Corporation (MDec) in Cyberjaya. Her research interest are in Systems and Software Engineering and Cloud Computing

Syahmi Iskandar Bin Mohd is currently working as IT Consultant at the eCEOs Sdn Bhd. He holds a MSc in Information Technology from University Technologi MARA (UiTM), Malaysia. He previously works as Testing and Integration Engineer at XYBase Sdn Bhd which involves testing the web-base and java-base applications.