ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

A Review on Software Fault Injection Methods and Tools

K. Umadevi [1]

Department of Computer Science and Engineering, Bharath University, Chennai, India[1]

ABSTRACT: Fault injection is a commonly used technique to test the tolerance of a software system methodically. All the real time systems tend to have faults which leaks through testing cycles. In order to have highly reliable systems it is imperative to have faults under the tolerance limit. The goal of the testing cycles such as integration, system and acceptance testing activities are to remove errors and their impacts from the computational system before a failure occurs in the field. Fault injection is being practiced for a long time using various methods. This review intends to give an overview about some of these common techniques practiced, tools developed and research studies carried out in the context of fault injection methods used over the last few years.

KEYWORDS: Fault injection; SWFI; software fault; fault tolerance; fault injection tools, fault injection techniques.

I. INTRODUCTION

Failure is deviation of the component or system from its expected delivery, service or result that is due or expected. Unexpected system failures can happen due to inadequate testing practice or unavailability of data to bring out certain errors in the code. In order to achieve highly reliable systems, the fault life cycle areas to be considered are classified into four areas namely fault prevention, fault removal, fault tolerance and fault forecasting [1]. Fault prevention and removal are primarily handled in the pre-release phases of an application mostly by review and testing processes. Fault tolerance is about providing the ability to the system to withstand failures. Fault forecasting helps us understand the remaining faults in the system and also the likelihood of future occurrence.

Studies have revealed that fault injection methods increases the test coverage of a software [2]. Fault acceleration is the process of injecting defects into the system intentionally and analyzing the effects. This helps getting through the lines of code that are not reached in the normal flow. Error handling code is an example of unreachable code in most of the testing scenarios. Fault injection activity helps assessing the system behaviour at various error conditions, helps achieving increased test coverage and measure the efficiency of fault tolerance mechanisms in place.

There are two major categories of fault injection types known as compile time injection and run time injection. Compile time injection involves the program instruction modification before the program execution is loaded and executed. Faults are inserted into the source code or assembly code of the program under study to observe the effect of defects. Modified code brings changes in the target program instructions, resulting in fault injection. Modified software image is generated by the injection when the system executes the fault image, it results in fault activation. The effect of fault is built into the code, can be used for simulating permanent faults. Run time injection involves software triggers to inject a fault into the system during run time. Commonly used triggers are time based or interrupt based. The simplest run time injection technique is time out based method. An interrupt is produced when a time out happens on a predetermined interval causing an unanticipated fault effect and system behaviour. The exception may be based on hardware or software timer. The fault produced is transient in nature. Another method is to insert a software trap that injects a fault when a defined criteria is met. Code insertion technique inserts certain fault injecting code into the target program at runtime without changing the original code.

Section II provides an abstraction of injection techniques used in studies of various research activities listed in Section IV. Section III gives a list of Off the Shelf tools and frameworks available for fault injection. Section IV gives a brief on research activities carried out on software fault injection.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

II. FAULT INJECTION METHODS

To perform fault injection some amount of code instrumentation is always necessary and though this can be added manually it is usually performed by a tool to carry out testing with reasonable volume. Code that is added to the program for the purpose of either simulating errors or detecting the effects of those errors is called instrumentation of code. Instrumentation code can be placed on top of input or output interfaces to the software or directly into the logic of the software. Instrumentation can be added into a variety of code formats: source code, assembly code, binary object code, etc. Any code format that can be compiled, interpreted or that is ready for execution can be instrumented.

There are two key approaches for simulating errors i) Directly changing the code that exists this is referred to as code mutation or ii) Modifying the internal state of the program as it executes. The methods will fall with one of these two but differ and what is being modified and how.

A. *SWIFI*

The early approaches for the injection of fault effects have originated from in the context of studies on hardware faults through Software-Implemented Fault Injection. SWIFI aims at reproducing the effects (i.e., errors) of hardware faults such as CPU, bus, and memory faults by perturbing the state of memory or hardware registers through software. SWIFI is a low cost and easy-to-control approach for injecting hardware faults, that overcomes several problems associated with physical fault injection techniques, such as controllability and repeatability of experiments. This approach has been adopted for Software Fault Injection with the assumption that injected errors are representative of errors generated by software faults. SWIFI approaches primarily replaces the contents of a memory location or register with a corrupted value.

In order to perform this operation, three aspects need to be defined i) What to inject: This aspect is related to the kind of errors to be injected, by replacing a correct value with an incorrect one. SWIFI tools modify the contents of an individual bit, byte, or word in a memory location or register. Several error types have been defined from the analysis of errors generated by faults at the electrical or gate level. One common error type is the replacement of a bit or byte with a fixed value or with the opposite value (inverted faults). ii) Where to inject: There are many locations in memory or register banks that can be targeted by SWIFI. Errors injected in memory typically target random locations, due to the large amount of potential memory location. The selection of memory locations can be focused on specific memory areas such as stack, heap and global data or user-selected locations such as a variable in memory. Errors injected in registers can target those registers that are accessible through software (e.g., data and address registers). iii) When to inject: The instant in which an error is injected can be time or event dependent. In the former case, an error is injected after a given experiment time is elapsed, where the time is selected by the user or through a random distribution. In the latter case, the error is injected when a specific event occurs during execution, such as at the first access or at every access to the target location. These approaches are adopted for emulating three types of hardware faults, respectively transient (i.e., occasional), intermittent (i.e., recurring several times), and permanent faults.

B. *Interface error injection*

Real time systems consists of numerous individual systems, which are organized to achieve a common business. Such systems are prone to errors that propagate across system boundaries. Data that is corrupt in the message traffic may cause a system to perform an unexpected system operation. For example, flipping a single bit might result in many flaws and data corruptions. Sufficient interface testing is necessary to see whether systems are having sufficient tolerance on receiving corrupted data. A technique used to test at the system level is Interface Mutation. Like mutation testing, this method is designed to create mutants by making changes to entities. Only those entities that reside on interfaces between components are mutated in order to limit the number of mutants to execute and analyse. The entities between interfaces include: function calls, function return values, and global data shared by two or more functions. Thus, mutating entities between interfaces can stimulate errors. A function directly relating to two components can be called with either a missing or corrupted parameter.

C. *Reflective Programming*

The Java programming language supports a Reflection API that enables the ability of a program to introspect its own behaviour. In other words, the program will be able to discover information about any Java class, including its set of methods and corresponding parameterized types. Java Reflection API has been extended to allow for a program

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

to alter its own behaviour. This extension is commonly known as behavioural reflection. Using both a meta-object7 protocol and behavioural reflection can dynamically capture an operation or method invocation, alter it, and execute it. The runtime system invokes a meta-object method that is associated with a particular operation. The logic of the meta-object method is pre-instrumented by the developer to reflect a changed behaviour of an operation. For example, a meta-object method, subtract, is executed whenever an add function is called. This capability makes reflection a well suited mechanism for fault injection.

D. *Code Mutation*

Mutation testing is the injection of faults in the code of a program for defining and running test cases. The mutations adopted are targeted at the control flow, array boundaries, mathematical expressions, and pre/post increment/decrement operations. Although the adoption of mutations broadens the scope of fault injection to software faults, the representativeness of mutations with respect to real software faults is an area less explored, which is required for achieving realistic results. Mutation testing tools usually assume the availability of the source code, which is likely the case since mutation testing is adopted on the software being developed. However, the absence of source code is a limitation for fault injection, since fault injection should be applicable to third-party software for which the source code is not available, such as COTS and third party components.

E. *Assertion Violation*

Assertions are boolean expression constructs that specify a program's expected behaviour. Examples of such constructs include pre-conditions, post-conditions, and class-invariants. Specifically, an assertion about the program's current state must be true before, during, and after a function is invoked. Thus, certain boolean conditions must be satisfied before an operation can be carried out. A true assertion statement ensures that a function is executed correctly; whereas, a false assertion statement guarantees a fault. To simulate a fault, an assertion is made false during program execution in an automatic fashion. This allows for the modelling and simulation of faults. Examples of such faults include assignment, function, and initialization faults. Furthermore, invalid assertions can cause a chain of other assertion violations in the code. In effect, this increases test coverage by exercising the assert mechanisms.

F. *Perturbation Functions*

Perturbation functions are used to forcefully override the current internal value of a variable, thereby simulating errors. A random function generator is a simple example of a perturb function. It may be written with varying complexity levels. Perturbation functions are usually applied at a source code level. To avoid being intrusive to the application source, these functions can be applied at the byte-code level. The perturb function may be written as a separate function, existing in a separate program, may be linked with the targeted program's executable.

III. FAULT INJECTION TOOLS AND FRAMEWORKS

There are many fault injection tools available both open source as well as commercial tools. In this paper some of them are given a brief. The list is neither exhaustive nor based on any specific solution oriented selection criteria. We have tried to give a representative list having a mix of technologies and techniques.

A. *Jaca*

JACA is a Software Fault Injection tool [3] [15] written in Java. It uses Javassist technology to inject faults inside the target application. It performs a golden run and a campaign execution to produce log outputs that can be compared and analyzed. JACA can perform high level fault injection in object oriented systems. It can inject faults which can be configured in UI related to parameters and functions of Java objects. JACA can also be easily extended to perform low-level fault injection, affecting Assembly-language elements such as CPU registers, buses, etc. JACA is platform independent, it runs on every platform running the Java Virtual Machine. Code instrumentation necessary for fault injection and monitoring purposes are introduced at byte code level.

Jaca provides an user-interface also which provides menu-driven access to the tool's functions. Latest version of Jaca can generate reports based on CVS file format. These reports allow statistical analysis and easily interpretable results. Jaca has monitoring abilities provide detailed information to observe abnormal behavior during the execution of fault injection campaign.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

B. *Xception*

Xception [7] provides an automated test suite that helps in injecting realistic faults. It injects faults without any intrusion on the target system. No software traps are inserted and hence program can be executed in normal speed. Xception provides a set of triggers, including temporal and spatial fault triggers. It is capable of injecting faults into C or Ada source code also injects in binary mode.

C. *Doctor*

A few tools do exist to emulate the occurrence of faults in distributed applications. One of those tools is DOCTOR [6] (integrateD sOftware fault injeCTiOn enviRonment), that allows to inject faults in real time systems. It supports processor, memory faults and communication faults. The tool can inject permanent, transient, or intermittent faults. The fault scenarios that can be designed uses probabilistic model. While this suits small quantitative tests, repeatable fault injection capabilities are required for more complex fault scenarios.

D. *Safe*

SAFE [5] fault injection tool allows to automatically generate and execute fault injection tests. SAFE injects software faults or defects in C and C++ software, in order to force a software component failure, and to evaluate the robustness of the system as a whole. Injected faults are designed to realistically reproduce the real defects that hampers software systems, including issues affecting data initialization, control flow, and algorithms. Testing team can easily know how vulnerable the software is and fix it. The SAFE tool lets users customize which faults are injected.

E. *Byteman*

Byteman [4] is a byte code injection tool developed to support Java code testing using fault injection technique. It is also very useful for troubleshooting and tracing Java program execution. Byteman provides a functions library which helps generating simple error conditions to complex error flows. Almost any Java code can be injected into the application in scope at the injection point. POJO (plain old java object) can be plugged in to replace built in functions. Byteman works by modifying the bytecode of the application classes dynamically at runtime.

IV. WORK DONE

Acantilado, N.J.P et al.[8] have developed a tool called SIMPLE that can be used to assess reliability, robustness and performance of a system as a whole. The aim of SIMPLE is to facilitate testing of Java classes used in safety critical applications. SIMPLE employs fault acceleration to test a system's fault tolerant capabilities. An object-oriented analysis of the system and several case studies, using software fault injection on selected systems to assess SIMPLE's effectiveness are presented.

Jeff Freschl et al.[9] has applied SWIFI technique to test the tolerance of Linux system and demonstrated the system's high sensitivity to single bit data structure corruption errors. This helped them suggesting a solution using redundancy repository that helped preventing corruption.

Thomas Naughton et al.[10] presented a general framework for fault injection in distributed systems. The modular design helps using it against various fault tolerance systems. A preliminary study had been carried out to help research of tolerance in high performance computing systems. The tool design caters to application level and system level failures.

Regina et al.[12][13] have proposed an architecture based strategy for Interface based fault injection. Faults are injected using a previously developed tool, Jaca that has the ability to inject faults into Java objects' attributes and methods. One of the key issues in component-based systems is its architecture, not only for development but also for testing. Another important issue is the selection of components to be injected and monitored. A risk-based strategy is proposed, in order to prioritize the components for testing which represent higher risks for the system. In this way, test costs can be reduced without undermining the system's quality.

Hashim, N.L et al.[14] uses wrapper technique for weaving an injection layer around the code. Aspect Oriented Software Development (AOSD) has been used without modifying the target code and also without using any separate tool. Interface faults are inserted into the system under test to evaluate the quality of the test cases. It ensures to detect errors due to the interactions between components. They also handle exceptions raised when interface faults

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

are triggered. Examples are presented demonstrating how interface faults can be triggered. This technique is suggested to be extended by adding counters in the interface faults code to monitor how frequently an interface service is called and subsequently the associated components invoked for the interface.

Henrique Madeira et al.[16] presents an experimental study on the software faults emulation using fault injection. Experiments were carried out to understand the effectiveness of SWIFI Tools. Xception is used to inject faults and results were compared with field data. Effort was made to emulate all the field failure. Results revealed that about 44% of the field failure could not be emulated. It has been emphasized that the field error data should be used as the main driver for designing the fault triggers to make best use of the tool. In situations where field data is not available software metrics could be used. Extensive study has been done on the fault triggers used, they seem to be the cause for the observed strong impact of the faults in the target system apart from other aspects such as code size, complexity of data structures and execution sequence.

Janos Olah et al.[17] presented an eclipse pluggable framework that provides a model-based approach and an user interface to configure and run fault injection campaigns. These experiments can be monitored at run time for their results. The framework uses javaassist technology and implements the required modifications for fault injection. It is proven to provide good support for dependability assessment testing of java components.

Cotroneo, D et al.[18] evaluated a framework G-SWFIT proposed by [20] which does binary level fault injection and experimented the validity in real world complex system. More than 12 thousand binary-level faults in the OS and application code of the system and highlighted the pitfalls and improvements on the tool. G-SWIFT technique consists of finding key programming structures at the machine code-level where high-level software faults are emulated.

The thesis [19] on a fault emulation tool emulates bugs that can happen during development and external faults such as a database disconnection. They are emulated by corrupting a program. Since many systems contain Components Off-The-Shelf (COTS), source code is not always available the tool works best on the Java byte code. The thesis also includes some utilities, designed for a better instrumentation of the Java byte code, that have been used for tool design and implementation.

V. CONCLUSION AND FUTURE WORK

Fault injection is a testing technique used for the evaluation of reliability, safety and fault testing coverage of the target system. Fault injection tools injects fault into the system and monitor the system to determine its behaviour in response to the fault. In this paper, research work on fault injection techniques and off the shelf tools available for this purpose has been discussed. Selection of such technique is driven by factors such as the injection testing goals, technology platform, criticality of the system, adaptability and size and complexity of the system. Though some of olden day systems use source code level fault injection, it has become obsolete nowadays. Future work may be to do an experimental comparative study to compare the suitability of the framework and tools for different types of systems.

REFERENCES

1. Michael R. Lyu, ' Software Reliability Engineering: A Roadmap', Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, 153-170, 2007
2. Bieman, J.M.; Dreilinger, D.; Lijun Lin, 'Using fault injection to increase software test coverage', Software Reliability Engineering,. Proceedings., Seventh International Symposium on , vol., no., pp.166,174, 30 Oct-2 Nov 1996
3. http://www.ic.unicamp.br/~eliane/JACA.html
4. http://byteman.jboss.org/
5. http://www.critiware.com/index.php?option=com_content&view=article&id=8&Itemid=109
6. S. Han, K.G. Shin, and H.A. Rosenberg, 'Doctor: An Integrated Software Fault-lnjecnon Environment for Distributed Real-T'ime Systems', Proc. Second Annual IEEE Int'l Computer Performance and Dependability Symp., IEEE CS Press, Los Alamitos, Calif., pp. 204-213, 1995
7. http://www.criticalsoftware.com/en_US/products/p/xception
8. Acantilado, N.J.P. and C.P. Acantilado, 'Simple: A Prototype Software Fault Injection Tool', Unpublished MSc Thesis, Naval Postgraduate School, Monterey, California, December 2002.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

(A Monthly, Peer Reviewed Online Journal)

Visit: www.ijmrsetm.com

Volume 6, Issue 8, August 2019

9. Jeff Freschl and Di Xue, 'SWIF-IT: A Tool for Memory Fault Injection and Protection' , Computer Sciences Department University of Wisconsin, Madison May 10, 2005
10. Thomas Naughton, Wesley Bland, Geoffroy Vallee, Christian Engelmann, and Stephen L. Scott, 'Fault injection framework for system resilience evaluation: fake faults for finding future failures' , In Proceedings of the workshop on Resiliency in high performance (Resilience '09). ACM, New York, NY, USA, 23-28, 2009
11. Moraes, R.L.O. and Martins, E., 2003. 'Jaca – A Software Fault Injection Tool' , Proceedings of the Intl. IEEE Conf. on dependable Systems and Networks, IEEE CS Press, p.667, 2003
12. Moraes, R., Martins, E.: 'An Architecture-based Strategy for Interface Fault Injection', In: Proc. of the International Conference on Dependable Systems and Networks, Firenze, Italy, 2004
13. Regina Lúcia de Oliveira Moraes and Eliane Martins, ' Fault injection approach based on architectural dependencies' , In Architecting Dependable Systems III, Rogério Lemos, Cristina Gacek, and Alexander Romanovsky (Eds.). Springer-Verlag, Berlin, Heidelberg 300-321.
14. Hashim, N.L.; Schmidt, H.W.; Ramakrishnan, S., Interface faults injection for component-based integration testing,' Computing & Informatics, 2006. ICOCI '06. International Conference on , vol., no., pp.1,6, 6-8 June 2006
15. Regina Lúcia de Oliveira Moraes and Eliane Martins, ' Jaca - A Software Fault Injection Tool' , Proceedings of the 2003 International Conference on Dependable Systems and Networks 2003
16. Madeira, H.; Costa, D.; Vieira, M., 'On the emulation of software faults by software fault injection ', Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on , vol., no., pp.417,426, 2000
17. Olah, J.; Majzik, I., 'A Model Based Framework for Specifying and Executing Fault Injection Experiments', Dependability of Computer Systems, 2009. DepCos-RELCOMEX '09. Fourth International Conference on , vol., no., pp.107,114, June 30 2009-July 2 2009
18. Cotroneo, D.; Lanzaro, A.; Natella, R.; Barbosa, R., 'Experimental Analysis of Binary-Level Software Fault Injection in Complex Software', Dependable Computing Conference (EDCC), 2012 Ninth European , vol., no., pp.162,172, 8-11 May 2012
19. Natella, Roberto, Santonu Sarkar, and Antonio Ken Iannillo. 'A Fault Injection Tool For Java Software Applications'. (Thesis)
20. Duraes, J.A.; Madeira, H.S, 'Emulation of Software Faults: A Field Data Study and a Practical Approach', Software Engineering, IEEE Transactions on , vol.32, no.11, pp.849,867, Nov. 2006
21. http://link.springer.com/chapter/10.1007/0-306-48711-X_8#page-2