StoreApp: A Fault Injection Testing Tool for Software Using Web Service

Dissertation submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science and Applications

Submitted By
Rajbir Singh Deher
(Roll No. 601203019)

Under the supervision of
Mrs. Sunita Garhwal
Assistant Professor (SMCA)

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS
THAPAR UNIVERSITY
PATIALA – 147004

July 2014

Certificate

I hereby certify that the work which is being presented in the dissertation entitled, **"StoreApp: Fault Injection Testing Tool for Software Using Web Service "**, in partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Application submitted in School of Mathematics and Computer Applications (SMCA), Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Mrs. Sunita Garhwal** and refers other researchers' work which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for award of any other degree of this or any other University.

Rajbir Singh

(Rajbir Singh)
601203019

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Sunita 18/7/14

(Mrs. Sunita Garhwal)
Assistant Professor
SMCA

Countersigned by:

(Dr.Rajesh Kumar)
Head, SMCA
Thapar University
Patiala

(Dr.S.K.Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

I would like to express my sincere gratitude to my mentor and supervisor Ms. Sunita Garhwal for her immense help, guidance, stimulating suggestion and full time encouragement. She always provided a motivation and enthusiastic atmosphere to work with, it was a great pleasure to do dissertation under her supervision.

I am also thankful to Dr. Rajesh Kumar, Head, School of Mathematics and Computer Applications and Dr. Singara Singh, P.G. Coordinator for their constant support and encouragement.

I would like to thank all the faculty members and staff of the department who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of this work.

I express my thanks to my family for their love, support and enthusiastic encouragement without which I could not complete this dissertation. I also want to thank my friends for optimism and moral support throughout the completion of this work. Finally I thank the Almighty who gave me the strength to complete this work.

(Rajbir Singh)
601203019

Abstract

Fault injection is a phrase covering a variety of techniques for inducing faults in systems to measure their response to those faults. In particular, it can be used in both electronic hardware systems and software systems to measure the fault tolerance of the system. For hardware, faults can be injected into simulations of the system, as well as into implementation, both on a pin or external level and, recently, on an internal level for some chips. For software, faults can be injected into simulations of software systems, such as distributed systems, or into running software systems, at levels from the CPU registers to memory to disk to networks. Fault injection is best used as a means for measuring the fault tolerance or robustness of a system, especially for stress testing a system that may experience faults too infrequently for normal testing. While the theory behind fault injection is still being developed, the mechanisms are well understood. For an embedded system designer attempting to measure the degree to which his design is resistant to faults, fault injection can be a useful technique for quantifying this aspect of design.

For this dissertation learning approach has been followed. The primary objective of this dissertation is to make a test cases that has high possibility of finding yet undiscovered errors. The Fault injection testing has been used for this purpose. The pros and cons of using Fault injection testing is also discussed. Fault Injection Testing is very helpful in finding errors .

Table of Contents

List of figures

List of Tables

CHAPTER 1
INTRODUCTION

This chapter include basic introduction of software testing and fault injection testing. It also consists a brief introduction of types, conditions, examples and application of fault injection testing.

1.1 Software Testing

Software development life cycle comprises of a number of phases and software testing [1, 2, 3] is the most important among them. Testing is the last point where any error can be detected and rectified. Executing a program is easy as compared to testing. Software testing involves a lot of complexities such as maintaining record, inspection, reviewing etc. all these are time consuming and cumbersome. Moreover a lot of investment is involved in all this. IT companies keep vigil on fault detection and removal process. Main reason behind their attention is that if less time and labor is involved the software will be prepared within the estimated budget. Hence a good amount of money will get saved. Best way is to start the testing phase as early as possible for better results of the project. Human-rated software generally is supposed to be more costly as compared to the other activities that are related to software engineering. All the above written facts provide us an understanding that testing is quintessential and needs to be done with great precision.

1.2 Objectives of Software Testing

Software testing [2] objectives are written below:

- Ensures the solution meets the business as well as user requirements.
- Process to catch errors like defects or bugs.
- Good test case has high possibility of finding out unknown errors.
- Testing determines user acceptability.
- Ensures that system is ready for use.
- Assures that the software works.

1.3 Testing Process

Managing a testing process follows certain protocols. The IEEE829 standard has devised a framework which helps in governing and managing the testing process. Members of the testing project are able to communicate with each other easily with the help of this project. In addition to this it outlines the documents that are required to be part of a compliant testing process [4].

- **Test Plan:** For any test plan the resource, approach and schedule of the testing activities are required to be described. Essential features that are supposed to be tested, items that are tested, tasks that are performed, personnel who are allotted a particular task and the risks involved are required to be identified.
- **Test Design:** Detailed test cases plus results that are expected and test pass criteria are the main parts of a test design.
- **Test Case:** It involves specifying test data for use in running the test cases identified in the Level Test Design.
- **Test Procedure:** Gives description about running each test, establishes pre-requisites and steps that are supposed to be followed.
- **Test Log:** Provides chronological order of details related to execution of tests. Recording which test cases were run and who ran it. Say for example in what order the test was done. Moreover information about whether the test was a success or failure.
- **Anomaly Report:** Testing process can witness any sort of result and preparing document for that is important. Such a document is known as anomaly report. Generally an anomaly report can involve issue, trouble, defect, test incident etc. Discrepancy between actual results and the expected results can occur. Basic function of this report is to provide information about any fault if present. Even though the fault needs to be found out still then the name anomaly is given because discrepancy between actual and expected results can occur for various reasons other than some fault in the system. Possibility of expected results being wrong also exists, running of the test incorrectly, or requirements that are not consistent which results in more than one interpretation, The report comprises of all details of the incident such as expected and actual results, when it was unsuccessful, and any evidence that will prove to be helpful in its resolution.

- **Test Status Report:** Gives a summary of the interim results of designated testing activities optionally to provide evaluations and recommendations based on the results for the specific test level.
- **Test Report:** This report forms the gist of all the activities that are undertaken. It can be used by any organization. All the important aspects involved in testing are covered in this report say for example statistics obtained from the anomaly reports, software system quality, testing effort quality assessment etc. Primary importance of final document is to check whether the software system is serving the desired purpose and has it been prepared in accordance with the criteria set by the stakeholders.

1.4 Software Testing Principles

Below written are the basic principles involved in Software Testing [2]:

- Main goal of any system testing is to find errors. Reason behind this thinking is that these errors only hinder the normal functionality of the program and ultimately lead to failure of the project. All the tests should match requirements of the customer.
- Plan for testing should be done in advance. It can begin as early as the requirements model gets complete. Elaborated definition of test cases can be started with the design model being prepared. Hence, we can plan for all tests even before the code generation.
- Testing should start small and gradually move towards large. Primary test that is planned and run, generally lays stress on individual program modules. With the progress in testing finding errors becomes the first priority
- Feasibility of exhaustive testing is null. Number of path permutations for even a medium sized program is very large. Due to this, executing every combination of paths is not possible.
- Testing conducted by a third party is considered to be more effective. Logically thinking any person who works on a project has some mind set with which he develops the project. But we need software that suits the requirements of a number of customers, so a third party tests the project taking into consideration all the requirements which makes it acceptable to customers.

1.5 Characteristics of a "Good" Test

Written below are characteristics of a good test [2] :

- Finds an error efficiently with accuracy. For doing so a tester is supposed to understand the software and visualize the reason as to why a software can fail.
- Redundancy is not a feature of a good test. Time allotted for a test and resources available to conduct a test do have limitations. Conducting a test that serves same purpose as some other test is of no value. So the main objective for every test should be different.
- Most preferred of all the tests is the one that covers a larger chunk of errors. Benefit of such a test is that a number of tests having the same intent need not be undertaken because their purpose gets solved by the said test.
- Tests that are neither too simple nor too complex are considered good. Although combining a series of tests into a single test case is possible, but side effects related with this approach can result in errors. In general, it is better to execute each test separately.

1.6 Types of Software Testing

Software testing is of many types. Below written are the types of software testing:

1.6.1 White Box Testing

White box testing [5] is a program level testing technique. Tester in such type of a testing is familiar with the internal structure/design/implementation of the item under test. The tester selects inputs so as to create channels through the code and evaluate the outputs. Technicalities involved in programming should be known to the tester. Other names given to White Box Testing are Open Box Testing / Clear Box Testing/ Glass Box Testing. For example, a tester, studies the code of a particular field on a webpage, finds all legal (valid and invalid) as well as illegal inputs and verifies the outputs and compares them with the expected outcomes, which is a determined by a study of the code.

1.6.2 Black Box Testing

Black Box Testing [5]**,** is also called as Behavioral Testing. In this method implementation/design/internal structure of the item under test is not known to the software tester. Tests of such type can be functional or non-functional, though mostly functional.

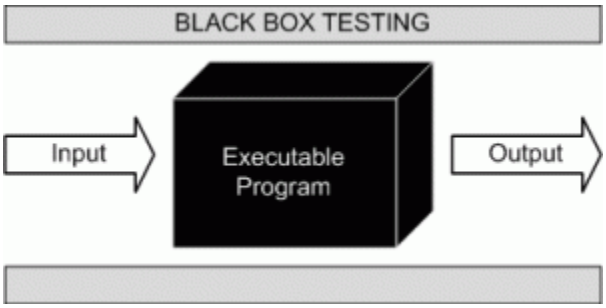

Fig 1.1: Black Box Testing [5]

The name black box testing is given because is named so because the program appears to be like a black box; through which one cannot see. Say for example, a tester, without basic knowledge of the structure (internal) of a website, performs the test with the help of a browser; providing inputs (e.g keystrokes, clicks) and checking the outputs against expected outcome.

1.6.2 Gray Box Testing

A combination of Black Box Testing method and White Box Testing method is called Gray Box testing [5]. Gray Box Testing requires the knowledge of the internal structure to some extent, like access to algorithms and internal data structures so as to design the test cases.

Gray Box Testing is called so because the program appears like a semi-transparent box/gray; through which one can see partially. For example, Gray Box Testing comes in to play when codes for two modules/units are studied.

1.7 Fault Injection Testing

A technique that helps in expanding area of a test by introducing faults to test code paths is called fault injection. This is done in some particular error handling code paths that are rarely followed. This technique is often used with automated unit testing and stress testing. It is considered to be an essential part of making robust software. Robustness testing (also called as Syntax Testing, Fuzz testing or fuzzing) is a type of fault injection used to test communication link vulnerabilities such as command line parameters and protocols, or APIs. Executing, a fault leads to an error, which is not a valid state in a system boundary. Any error results in further errors, so each new error acts as a fault, it may spread to the system boundary and become observable. These visible error states are called failures. Figure 1.2 shows a fault-error-failure cycle.

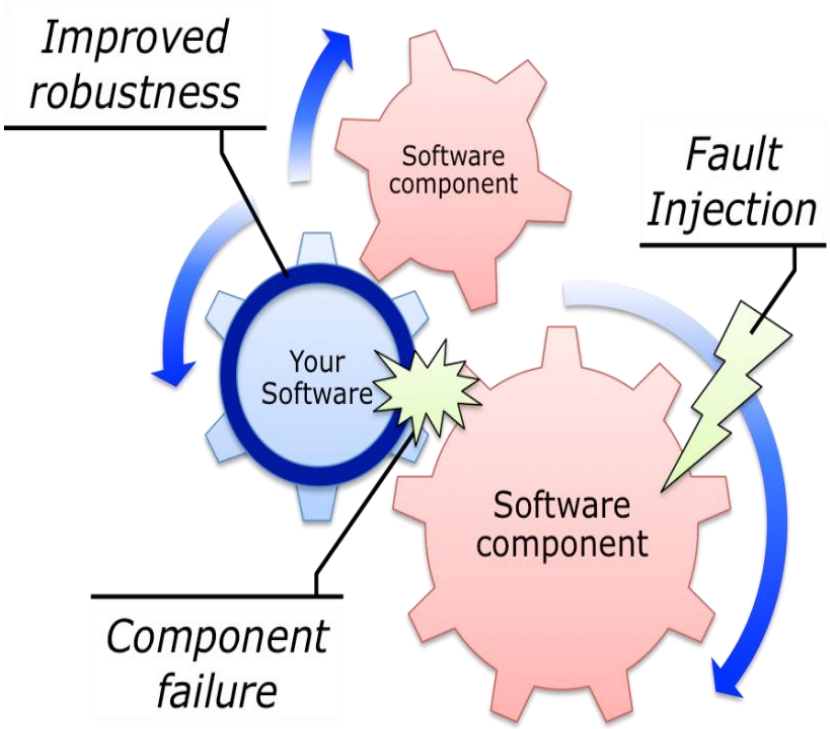

Fig 1.2 Fault Injection Testing [6]

1.8 Types of Fault Injection Testing

There are two types of faults injection testing as discussed below:

1.8.1 Hardware Fault Injection

If we want to add faults into hardware and witness the effects then Hardware Fault Injection testing is employed. This testing is done on VLSI circuits generally at transistor level, because of the fact that circuits are very dense and warrant characterization due fault injection as compared to performance scope, these are the best known basic faults in those circuits. Transistors are typically given faults like transient faults, bridging and stuck-at and results are observed. Such faults can be injected into circuits that are cut from the wafer software simulations of the circuits.

1.8.2 Software Fault Injection

It is used for injecting faults into the operation of software and examining the effects. This is mostly used on code that has cooperative or communicative functions so that fault injection becomes useful. Faults of all sorts can be injected, like memory and register faults, to replicated network packets, to erroneous flags and error conditions. These faults can be injected into complex systems simulations where interactions are understood and effects are examined.

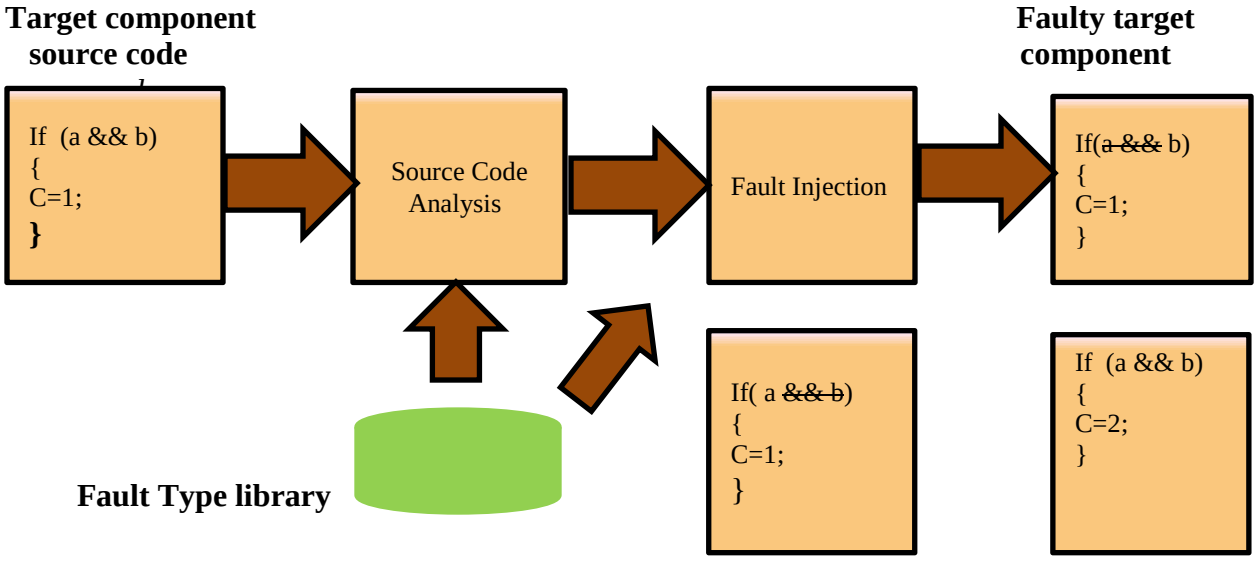

Fig 1.3 Software Fault Injections [18]

1.9 SFIT (Software Fault Injection Testing)

There exist three lines of defense against software faults [8]:

- It is achieved during the coding, design and specification process of the software.
- Eliminating faults is responsibility of the testing processes.
- Fault tolerance is one of the properties of the deployed software. It is designed to withstand survive anticipated possible failures.

Fault tolerance, serves as complementary to fault elimination and fault avoidance. It is very important for telecommunications software. It provides graceful service degradation in case of system failures. Testing a large system for fault tolerance is a difficult task. So a clear methodology for testing need, supporting tools and concept of testing adequacy is required for completion of this task

.

1.10 Methods of Fault Injection

Written below are methods of fault injection:

1.10.1 Compile Time Injection

If you want to inject faults at compile-time [9], the program instruction needs modification before the program image is loaded and executed. Errors are injected into the source code or assembly code of the target program to observe the effect of hardware, software and transient faults. Modified code brings changes in the target program instructions, resulting in fault injection. Erroneous software image is generated by the injection when the system executes the fault image, it results in fault activation. It does not cause any perturbation to the target system during execution. Fault effect is hard-coded; hence it can be used for emulating permanent faults. Compile-Time Injection is a technique in which source code is modified to inject simulated faults into a system. Following is a simple example of this technique:

x = x + 1 to x = x – 1

An advantage of this technique is that we can use it to simulate both hardware and software faults. Since the faults are coded into the executable form so it is possible to transient faults as well as permanent faults. This system is very simple to implement.

Drawbacks

- Modification of the actual source code is required. Hence the test team should have the code available with them. Such may not be the case for all the systems.
- As the code has been modified therefore there is possibility of unintended faults being introduced.
- As a result of the source code being changed, it cannot be used as a part of certification processes because the system under test will be different to that which is shipped.

1.10.2 Runtime Injection:

If you intend to trigger fault injection a mechanism is required so as to inject faults runtime [9]. This mechanism include: time-out, exception or trap, and code insertion. Time-out employs a timer expires at a predetermined time, triggering injection. The time-out event generates an interrupt to invoke fault injection. For exception or trap, a hardware exception or a software trap transfer control to the fault injector. Exception or trap can inject the fault whenever certain event or condition occurs. In code insertion technique, instructions are added to the target program that allows the fault injection to occur before particular instructions.

Runtime Injection techniques use a software trigger to inject a fault into a running system. Faults can be injected via a number of techniques. Triggers can be implemented in a number of ways. The main types are:

- **Time based Triggers:** After reaching a specified time an interrupt is generated and the interrupt handler that is associated with the timer can inject a fault. Intermittent and transient faults within a system are simulated with these triggers.
- **Interrupt based Triggers:** Hardware exceptions along with software trap mechanism to cause interrupt at a particular place in the code or on specific event within the system, e.g. access to a particular memory location. This method is capable of injecting a fault on a specific event and it does not require any modification to the system code. Quite a number of Runtime Injection mechanisms are based upon debugging features that exist in most modern processors [6]. This implementation method provides low latency triggers and allows fault injection tools to be written in a noninvasive way, but has the disadvantage that it ties a tool to particular machine architecture.

1.11 Categorization of Software Faults, Errors and Failures

Categorization [10] of software faults, errors and failures can attain the following benefits:

- Set of errors, failures and inherent faults, errors and failures can perform as a partial tolerability checklist of difficulties faced ordinarily against fault handling.
- User requirements remain the same irrespective of the difference in the system. They are vulnerable to failures or errors and inherent faults.
- Realistic inputs can be obtained by interacting with the fault manager through faults, failures or errors.
- Prior to categorization it provides help in knowing relationship between failures, faults and errors.

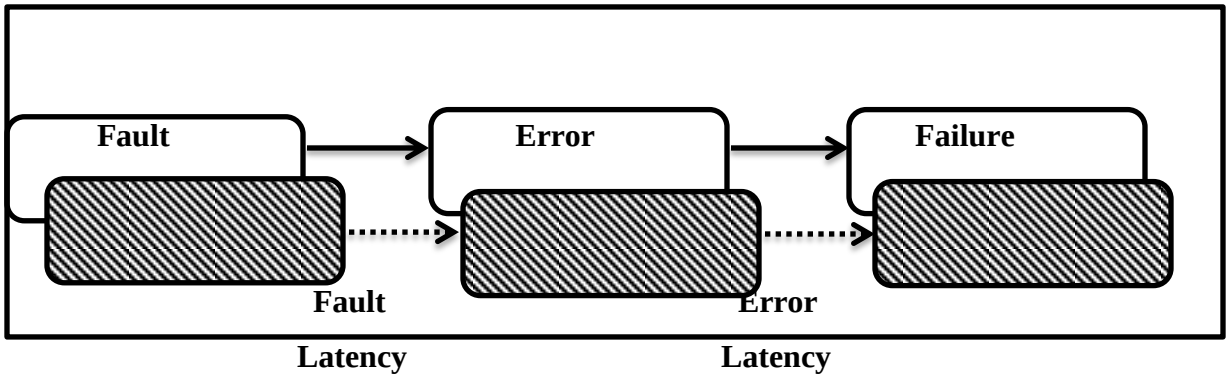

Fig 1.4 Correlation between fault, error and failure [10]

1.11.1 Fault Category

Inherent fault types [11] comprise of a number of sources. Primary source for fault categories is derived from system architecture evaluation. For example, the system architecture evaluation may view that in certain conditions the execution sequence of two actions is necessary. So, the system might have recovery plan for handling the task of disordering the implementation of the actions. Thus a fault that is responsible for upsetting the order will test this method. Error coding of the system forms the second source of fault categories. Generally, a list of error code is available for any system software. Main purpose of such code is to specify which errors can be faced and how to preserve these errors. Such errors were projected at the time of formation of the software thus bring forth the designer's view of errors that are likely to happen.

Third source of fault categories is by root cause examination of empirical data. Field data on failures and updating requests from the customers showed that these faults were not projected by

the software or its designers.

Following is a trial set of common single fault categories from various sources.

- **Memory faults -** transient, intermittent or perpetual errors in the memory.

Table 1.1 Memory Faults [1]

Fault Type	Perturbation	Inter-Arrival Time	Injection Location
Single Bit Compensating Single Byte Multi Byte	Set Reset Toggle	Deterministic Exponential	Code Global variables Heap OS kernel area

- **Processor faults**-failures in functional units of the CPU. Table 2.3shows some fault examples.

Table 1.2 Processor Faults [1]

Fault Type	Perturbation	Inter-Arrival Time	Injection Location
Single bit Compensating Single byte Whole word	Set Reset Toggle	Deterministic Permanent	Data registers Address registers Stack pointers Program counter

- **Communications Faults -** lost, altered or delayed messages. Table 2.4, shows some fault examples.

Table 1.3 Communication Faults[1]

Fault Type	Inter-Arrival Time	Injection Location
Lost outgoing messages	Deterministic	Header
Lost incoming messages	Exponential	Data
Lost all messages	Permanent	
Alter messages		
Delay messages		
Duplicate messages		

A complex system can have a number of software faults. Faults must be processed systematically or improved so as to signify most fault categories. New fault types must be inherent and genuine independent to the fault types in the collection.

1.11.2 Error Category

Error is produced by faults that occur under certain constraints. A number of facts exist that inspire the use of state injection of errors about the code injection method. Thus, injection of errors is a form of fault injection if we know the error rank in the state injection method. Further, injection of a particular error can focus on a particular operation of the fault manager as compared to fault insertion. Finally, in a scattered environment, some complicated error mode is obtained as a result of communication of a number of errors. When error [12] injection acts as replacement for faults, the fault expectancy gets reduced. For example, in order to design processor failure, sending failure message is easy as compared to introducing faults into methods to actually fail the processor. Written below are standard set of common error types based on

field experiences.

- Process errors, like process looping, process hung, deadlocks, live locks and using many system resources.
- Message errors, like duplicated messages, lost message, lost message, corrupted message, and time-out waiting for messages between components.
- Memory/Storage errors like corrupted data and structure value in a database, any storage unit or memory.
- Network errors, like routing error, network congestion (traffic over-flow) and link oscillation.
- Operational procedure errors, like wrong maintenance procedures and incorrect data input from the operator.

1.12 Applications of SFIT

Fault tolerance is critical for systems like a typical telecommunications system. Therefore, it needs to be tested thoroughly in systematic and controlled way before deployment. The SFIT [13] method nowadays is considered an important technique for at least two reasons:

- **Failure acceleration -** Waiting for occurrence of failures in a system in the field is time consuming and undesirable. Inserting faults intentionally to invoke the fault tolerance capability, we can achieve more by testing in a controlled environment and within a desirable time frame.
- **Systematic testing –** Systematic analysis is required to ascertain the faults that can be activated in the field and the system components that will respond to error caused by the fault. Functionality can be tested in a testing environment by inserting faults that are designed to invoke some specific fault tolerance capability. At Ericsson, fault insertion helps in verifying whether redundancy works as intended or not. If some portion in a sub-system does not work normally, a redundant sub-system should take over.

CHAPTER 2

Literature Survey

2.1 Introduction to Web Services

Web services [10] nowadays have become a limitless technology and recent thing to merge as a standard prototype for communication between programs over the internet. It involves less cost to make communication between the system and heterogeneous applications. As a result of this, Web services are used for building all type of distributed systems for wide areas: security, multimedia, business.

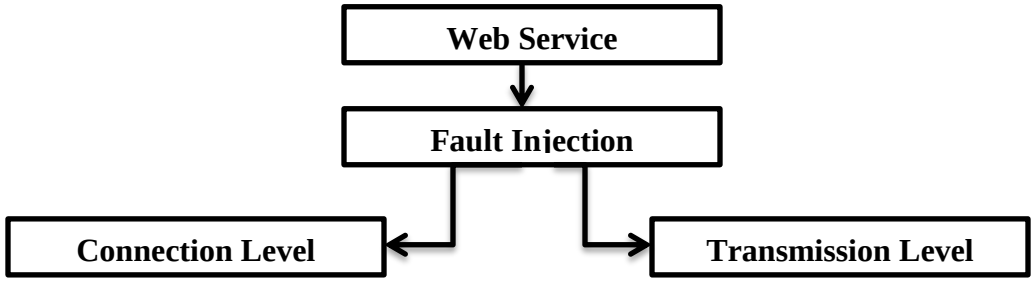

Fig 2.1 Faults Injected by Web Service

Insertion of faults in web services can be done at connection as well as transmission levels. Faults at the connection level affects actions like SOAP messages areas and input/output constraints by data or assigning constraint values that are invalid or altering data. On the other hand, SOAP messages are assumed as black box by faults at transmission level. We can delay the forwarding of messages instead of making changes in carried data. This can also be done by modeling a connection loss. Two types of faults are important for testing Web service communications. They can be used either to trouble communication between a Web service and client application or between service partners having same service configuration. In addition to this, combining interaction and connection faults results in more complicated errors and allows deeper fault insertion tests.

WS Inject is a Web service fault introducer used for testing Web services (simple and collected). Depending on user conditions it is used and possesses the ability to insert both interaction and connection faults. Condition-action prototype is followed by its cursive language and it provides an easy and powerful way for generating various types of faults. An experimental study has been presented that is based on a set of injection experiments as conducted on a case study of Web services.

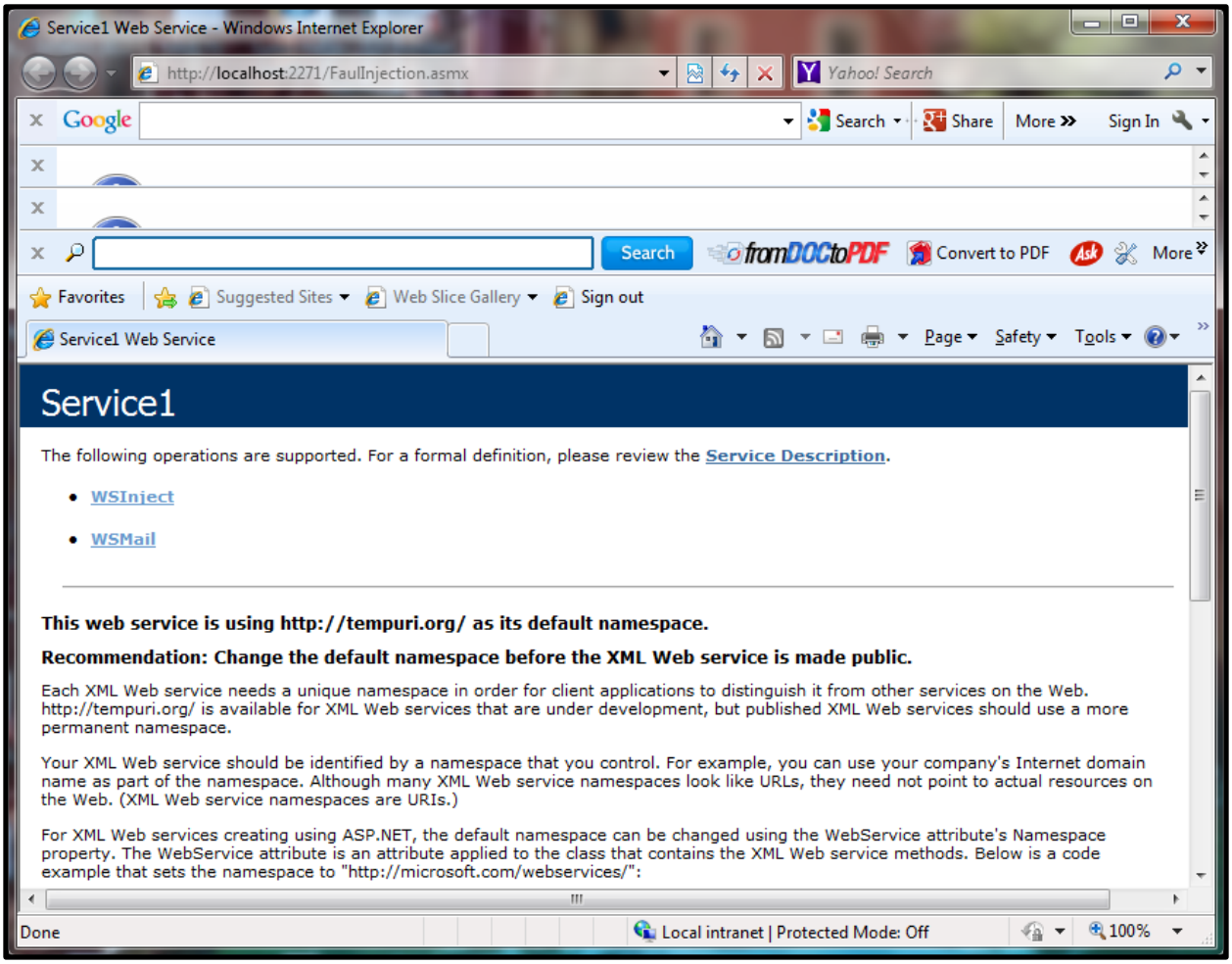

Fig 2.2 WSInject and WSMail Web Service Method [14]

In this experiment we enter some parameters and with respect to those parameters we get error description in the form of table. While running on any server this web service does not need any parameter to be entered. Results are automatically given by the server..

2.2 Definition of Web Service

A mode of communication between two or more electronic devices over network is called a web service. Actually it is function software that is available over the web at a network address.

Web service is defined as a:

"A software system proposed to support interoperable machine-to-machine messaging over network. It has a boundary described in a machine-processable format (specifically WSDL). Other systems communicate with the Web service in a way prescribed by its explanation using SOAP messages, typically carried using HTTP with an XML serialization in union with other Web-related standards."

2.3 Importance of Web Services

A number of software systems are used for handling work by different organizations. Different software systems need to exchange important data, and web service [15] serves as mode of interaction between software systems for doing so over the internet. Software system that makes request some service is termed service requester, whereas the software system that processes that request and facilitates that service is termed as service provider.

Different software can be developed with the help of different programming languages, so there is a need for some mode for exchanging data that doesn't rest on a particular programming. Quite a number of software can, however, interpret XML tags. Thus XML files can be used by web services for data transfer.

Rules and regulations for doing so need to be defined

- One system can request data from another system.
- A data request requires some specific parameters.
- Generally swapping of data is done in XML files format; the structure of the XML file is justified against .xsd file.
- Error messages need to be shown, when a definite rule for communication is not perceived, this is done to make troubleshooting easier.

Almost all of these rules required for communication are defined in WSDL (Web Services Description Language).

2.4 Web Services Architecture

The service provider transmits a WSDL file to UDDI. Service requester in order to catch on who is the provider of the required data consults the UDDI. After that it contacts the service provider by SOAP protocol. Service provider validates the request and sends data in an XML file with the help of SOAP protocol. This XML file is again validated by service requester using an .XSD file. Defining which software systems needs to be contacted for which type of data is done by UDDI (Universal Description, Discovery and Integration). Therefore, when some software system requires a particular report/data, it takes help from UDDI so as to find out the other system that can be contacted for receiving that data. Software system can contact with UDDI making use of SOAP protocol. Service provider system will validate the data request by referring to WSDL file, and then process the request and send the data with the help of SOAP protocol.

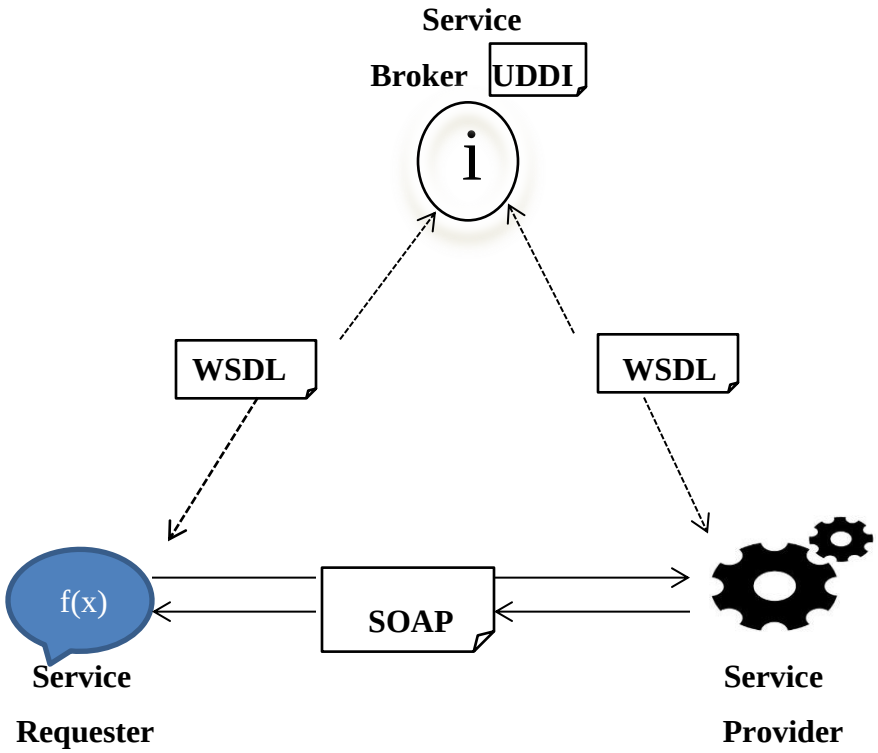

Fig 2.3 Web Service Architecture [10]

2.5 Applications of Fault Injection Testing

Mutation testing can be applied on several applications. Some of them are following:

2.5.1 Research of memory Fault Simulation and Fault Injection Method for BIT Software Test

Jun Xu, Ping Xu developed a fault injector as virtualization software based on QEMU. Advantages of using QEMU are described below.

It is open-source software that allows modification of the codes of the device for the addition of the fault injection. QEMU can emulate a number of hardware devices. So it may treat various hardware faults in the guest Operating System. Modifying QEMU helps in expanding its function with three modules: Analyzer, Fault Injector, and Controller. Modified simulator can simulate decoder faults and memory cell array faults. Figure 2.2 shows the structure of the fault injection tool.

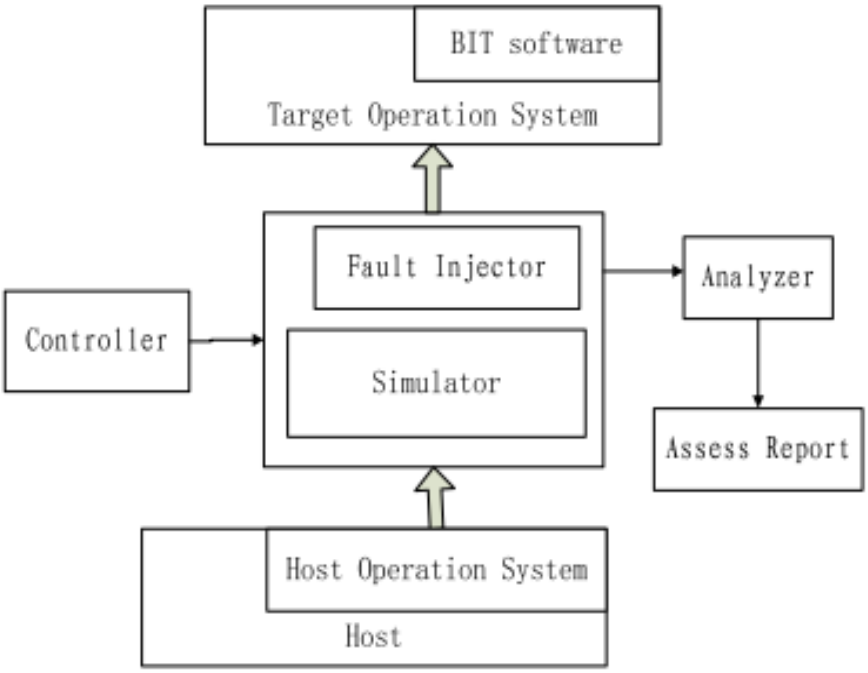

Fig 2.4 Structure of Fault Injection Tool [20]

Fault model for the experiment is configured by controller. Download the XML file into Fault Injector. Once fault model is received by fault injector, it will analyze it to the fault sequence

So as to match the desired conditions like bits, address. Simultaneously BIT software is tests the system. If BIT detects fault it will report. When analyzer gets the fault report it will compare the report with the fault model. Thus we can provide an assess report about fault isolate rate of the BIT and fault detect rate.

2.5.2 A Fault Injection Method for Testing Dependable Web Service Systems

A fault injection toolkit[10] developed by Khaled Farj emulates a WAN within a LAN environment between composed service components and offers full control over the emulated environments. In addition to this it has the ability to inject application specific and network-related faults. Background workloads are also generated on tested system for producing more realistic results. It describes an experiment that has been carried out to test the impact of fault tolerance protocols deployed at a service client by using our fault injection toolkit.

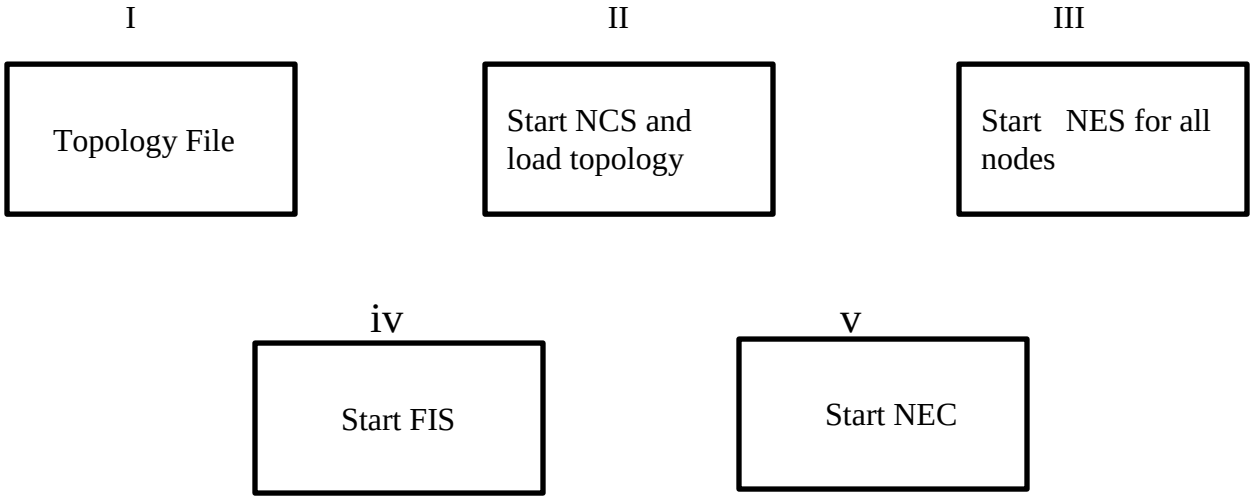

Fig 2.4 Steps to set up the tool [10]

Primary stage comprises of preparing a description of the target network using a topology file and to describe the traffic load to generate on all network nodes. Next stage involves starting the load topology and NCS. Third step is to start the NES's for all the network nodes, after that FIS for every node is to started that will generate a proxy service for each service to be called, and finally, to order the NEC to start the emulation and then start the system to be tested. Topology file is a simple configuration XML file and it gives description of the target network topology. Nodes in the network together with their configurations are listed. A trace file is required by each node describe its traffic load. It shows packet counts per unit time and

can either be captured from real traffic traces or created by hand. It can also be generated using

network traffic modeling algorithms. After this NCS (Web service) is started. It is used by NES's to provide locations of neighboring NES's and node configuration parameters. One NES and one FUS represent each node of the emulated network. XML file that contains URL(s) of the Web service(s) under test is required by each FIS at the client side. For generating a Web service proxy client needs to call it and will be called by the client.URL of the NES emulating the same node is present in the XML file. This tool does not need any modifications to the system under test, the only work to be done by the client is to start calling the proxy service generated by the FIS.

2.5.3 Fault Injection Testing for Distributed Object Systems

It is a technique proposed by Sudipto Ghosh. Interface-based fault injection testing (IFIT) is used to assess the fault tolerance of distributed object systems. IFIT [8] makes use of description of an object's interface for generating application dependent faults. IFIT presents shortcomings of the fault recovery mechanisms that exist in the application. The app1ication of IFIT for different distributed object systems is described.

2.5.4 The Fault-Tolerant Design and Fault Injection Test for Embedded Software

The test system [12] is made up of monitor computer, Trace32 ICE, monitor instrument for fault injection instrument and the output. Test case is used to verify the behavior of fault-tolerance and judge the validity of fault-tolerant design of hardware and software, and it includes actual result, expected result, verification method, test method, device and instrument, test content, test case name etc. The result shows that the fault injection test system can verify the fault-tolerant design well.

Main function of on board data handling software include dual-computer switch, Internal and external memory management, software running state management, system running mode management, fifo data management, etc. Generally functions are designed for fault-tolerant.

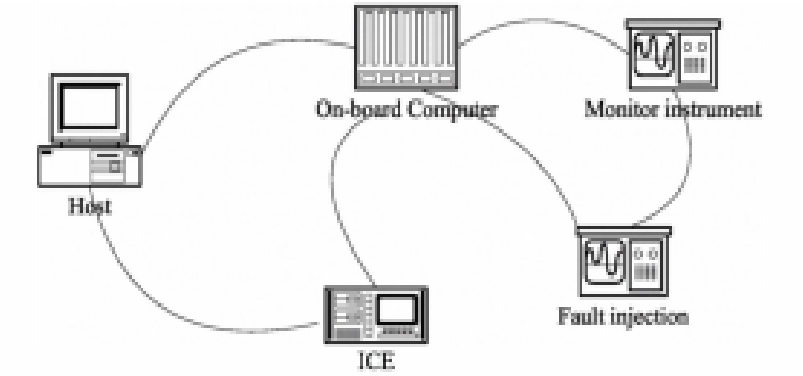

Fig 2.5 Fault Injection Test System[12]

Fault injection test system comprises of four parts which are shown in figure, and they are fault injection instrument, monitor instrument for output, Tr.ace32 ICE, and monitor computer. The OBDH software do not only control the running of software, but also redundant hardware component, and it becomes essential to verify the control instruction, In order to observe the output the monitor instrument for output is employed.

OBDH software run in host computer, TRACE32 ICE simulate the running of CPU, and then we can debug the software, such as modifying parameter, setting memory value, tracing program, setting breakpoint etc.

2.5.5 Radiation and Fault Injection Testing of a Fine-Grained Error Detection Technique for FPGA

Gabriel L. Nazar, Paolo Rech, Christopher Frost, and Luigi Carro present the experimental evaluation of a fine-grained hardening approach that exploits underused and abundant resources found in state-of-the-art SRAM-based FPGAs [14] to detect radiation-induced errors on configuration memories. The technique's main goal is to provide the benefits of fine-grained redundancy, namely improved diagnosis and reduced error latency, with a reduced area overhead. Neutron experiments, validated with fault injection campaigns, demonstrate the proposed technique's efficiency when compared to the traditional dual modular redundancy.

2.5.6 A novel fault Injection Algorithm for Safety Analysis

Novel fault injection algorithm [19] was proposed by Wang Xi and Xu Zhong-wei and it consists of the main algorithm sub-algorithm "Result Analysis", Fault Verify()", sub-algorithm "Modelextend", sub- algorithms "multiFault".

```
FaultVerify()
   { int rds= FaultDB.size();
    For i=1 to rds do
   {pickout(Name, No, FaultModel) from FaultDB[i];
    faultRd[1]=< Name, No, Fault Model>;
    Modelextend(FSP_sys, faultRd[1]);
       If(! ExFSP-| SafetyProperty )
      {then {Result.add(Name, No, 1, "Yes, violate system's SafetyProperty");
           FaultDB[i].Label=true;}
       Else ResTemp.Add(No);}  // end If(! ExFSP -| SafetyProperty);
    } // end For i=1 to rds do
    Int n=FaultDB.count(false);
    While(RestTemp ≠ NULL) do
     multiFault(ResTemp,n);
     ResultAnalysis(Result);
    Print Result;
}  // end FaultVarify()
```

Fig 2.6 Main Fault Injection Algorithm

In the above figure line 1 is counts the number of records present in FaultDB; The third line, (FaultModel, No, Name) are selected from FaultDB, where fault's behavior is denoted by FaultModel that is expressed in finite state machine model; in the fourth line, (Name, No, FaultModel) are stored in faultRd[1]; in fifth line calling sub-algorithm "Modelextend" to extend system mode "FSP_sys" with faultRd[1]; sixth line tells whether the extended mode "ExFSP" violates safety property or not; seventh line, for adding name of the fault, identification number, the number 1 for single fault injected, and the result of safety verification in Result; in line 8, the value of label in FaultDB[i] is changed to true; in line 9, adding fault's identification number in

ResTemp; in eleventh line, counting number of faults having a false label in FaultDB; in thirteenth line, calling sub-algorithm “multiFault” to do with fault injection for multiple faults; in fourteenth line, calling sub-algorithm “ResultAnalysis” to get minimal combination of multiple faults, that violate system’s safety property; in fifteenth line, outputting the results.

2.6 Techniques of Fault Injection Testing

Techniques of fault injection testing are written below:

2.6.1 Hardware-Based Fault Injection

It is done at the physical level, laser fault injection or modifying the value of the pins of the circuit, injecting voltage sags on the power rails of the hardware (power supply disturbances), disturbing the hardware [21] with parameters of the environment (heavy ion radiation, electromagnetic interferences, etc.),

6.2 Software-Based Fault Injection (software implemented fault injection)

Main aim of Software based fault injection [21] technique is reproducing at software level the errors that get produced due to faults that occur in the hardware. These faults [21] are most likely considered to be major cause of system outages. Fault injection method is a feasible way to assess the results of bugs that have not been seen. Traditionally, it includes the software modification the system under study so as to provide potential to modify the system state corresponding to modeling vision of the system.

2.6.3 Simulation-Based Fault Injection

Simulation Based fault injection [21] involves injecting faults in high-level models (most often, VHDL models). Early evaluation of the system dependability is allowed, in case only one model of the system is available. After that it addresses different abstraction levels by making use of distinct languages (description). Coherent environment is needed to favor interoperability between the successive abstraction levels and to integrate validation in design process.

2.6.4 Emulation-Based Fault Injection

It serves as an alternative solution for reducing time spent during simulation-based fault injection campaigns. It depends on the exploration of Field Programmable

Gate Arrays (FPGAs) for speeding-up fault simulation and exploits FPGAs[14] for effective circuit emulation. It allows the designer to study actual behavior of the circuit in the application environment, taking into account real-time interactions. However, when an emulator is used, the initial VHDL description must be synthesizable.

2.6.5 Hybrid Fault Injection

Hybrid approach [21] is a combination of two or more fault injection techniques so as to fully exercise the system under analysis. For instance, performing software-based or hardware based fault injection experiments can prove to be beneficial in terms of time for performing fault injection experiments, and can reduce the initial amount of setup time before beginning the experiments, and so forth. The hybrid approach involves combination of accuracy of hardware monitoring and versatility of software fault injection. Hybrid approach sis well suited for measuring extremely short latencies.

2.7 Tools of Fault Injection Testing

Performance of fault injection testing is enhanced by automated tools. These tools are considered to be best for testing practically as they involve experimental results and concepts. Moreover, the effectiveness of any approach is presented by that tool. A number of fault injection testing tools have been developed for supporting automated fault analysis. There exist 36 implemented mutation tools of all categories.

2.7.1 GOOFI

GOOFI (Generic Object-oriented Fault Injection) is a new fault injection tool which can perform fault injection campaigns using different fault injection techniques on different target systems. Main purpose of this tool is to provide user-friendly fault injection environment with a graphical user interface and underlying generic architecture that assists the user when adapting the tool for new fault injection techniques and new target systems. The GOOFI' [22] tool is highly portable since the tool was implemented using the Java programming language and all data is saved in a SQL compatible database. Furthermore, an object-oriented approach was chosen which increases the extensibility and maintainability of the tool.

2.7.2 Jaca

Jaca is a reflective fault injection tool based on patterns. Jaca [23] is a software fault injection tool that validates OO applications written in Java. Jaca's major goal is to inject faults using high-level programming features during runtime by corrupting attribute values, methods parameters or return values. Jaca's design was based on a set of patterns - the Fault Injection Pattern System. This pattern describes a generic architecture defined from recurrent design aspects present in most fault injection tools. The objective was to reduce tool development time while enhancing qualities such as portability, extensibility, reusability, efficiency and robustness. The paper presents the pattern set and its use in Jaca's development. An extension of Jaca to consider injection at assembly level is also presented to show how easy it is to add new features to the tool.

2.7.3 DOCTOR

DOCTOR is an Integrated Software Fault Injection Environment for Distributed Real-time Systems which is capable of generating synthetic workloads under which system dependability is evaluated, injecting various types of faults with different options, and collecting performance and dependability data. A comprehensive graphical user interface is also provided. The software-implemented fault-injection tool supports three types of faults: memory faults, CPU faults, and communication faults. Each injected fault may be permanent, transient or intermittent. A fault-injection plan can be formulated probabilistically, or based on the past event history. The modular organization of tools is particularly designed for distributed architectures. DOCTOR [24] is implemented on a distributed real-time system called HARTS, and its capability has been tested through numerous experiments.

2.7.4 J SWFIT

SWFIT is a generic technique proposed by Durães and Madeira [25]. This technique provides the ability to emulate software faults. It consists in modifying the compiled binary code of software modules by introducing specific changes, which correspond to the code that would be generated by the compiler if the intended software fault were in the high level source code. The emulation is performed based on an emulation operator’s library. Each operator in this library consists of two binary code instructions. The first one represents a pattern corresponding to a specific fault and the second one represents the necessary changes for

the adequate injection of the fault. The authors also present a field data study to compose a faultload, whose faults were identified in applications developed in C language. The most frequent software faults from the faultload were injected in C applications to validate the G-SWFIT [26] technique. Although this technique is generic, the study was developed using the C language and additional investigation was necessary to extend it to other programming languages, which was done by a previous study. Both studies were essential to define the fault load that J-SWFIT comprehends.

2.7.5 ORCHESTRA

ORCHESTR is a Probing and Fault Injection Environment for Testing Protocol Implementations This tool was developed for testing dependability and timing properties of distributed protocols. ORCHESTRA [26] is based on a simple yet powerful framework, called script- driven probing and fault injection. The emphasis of this approach is on experimental techniques intended to identify specific "problems" in a proto- col or its implementation rather than the evaluation of system dependability through statistical metrics such as fault coverage. Hence, the focus is on developing fault injection techniques that can be employed in studying three aspects of a target protocol: i) detecting design or implementation errors, ii) identifying violations of protocol specifications, and iii) obtaining insights into the design decisions made by the implementers.

2.7.6 GCDEFI

GCDEFI (Generic Component Dynamic Environment Fault Injection) adopts environment fault injection based on API interception technology. Faults can be injected by GCDEFI without the source code of target applications under assessment, nor does the injection process involve interruption. To evaluate our tool, we conduct several environment fault injection testing experiments. The results show that our tool is stable and effective.

Chapter3
Problem Statement

This chapter contains the problem statement and objective of the dissertation.

3.1 Problem Statement

In growing complexity of information system (IS) projects, there are an increases number of potential fault sources which will affect the projects. Fault injection is not really suited for debugging and improving the system so much as it is suited for testing the fault tolerant features of the system and in Fault Injection, the good test coverage of good quality is not a easy task. The principal assumption of test coverage metric is that each branch and statement should be tested. However, it does not determine whether it detected any errors or faults in each branch and statement.

3.2 Dissertation Objectives

- We propose an approach in which Web Service strikes the application again and again checks the results for the injected faults.
- Our approach reduces the total time required for executing all faults. Thus, it improves performance of fault injection testing.
- Our approach is used for C# language.

Thus we will use Web Service Applications, which automate the fault injection testing technique and reduce the cost and time consumption.

Chapter4

Proposed Work

This chapter includes the details of work that has been done to for solving the problem that is stated in problem statement. The proposed technique helps to solve the problem of large time and cost consumption.

4.1 Working of Fault Injection Testing

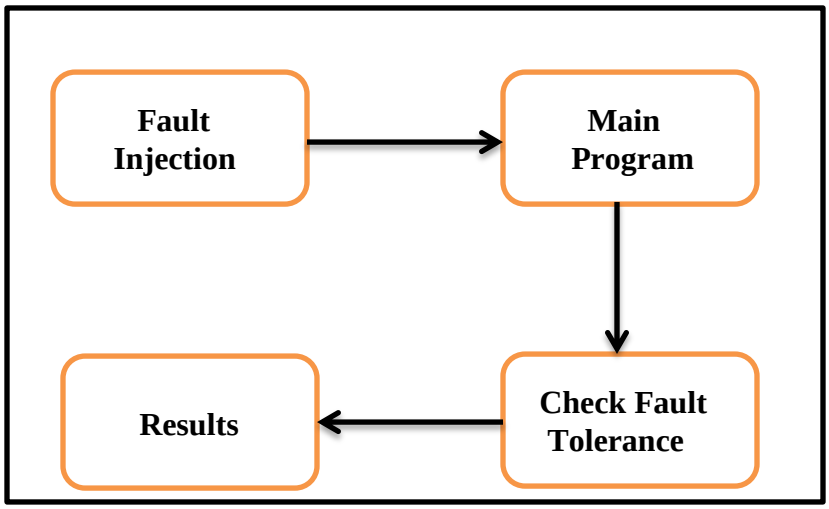

Fig 4.1: Working of Fault Injection Testing

For performing fault injection testing initially faults is injected into the main program. These faults are introduced into the code on hypothesis basis. After fault injection, all the test cases are executed related to entire faults and results are compared with the very first program's outcome. If the main program's output is different from the produced output, then that fault is occurred else that the test is said to be failed.

4.2 Steps for Fault Analysis

For achieving Fault analysis three steps are executed.

- **Injection of faults**

  In this step multiple copies are created with small syntactic changes in the main program which are called fault sets.

- **Execution of faults**

  In this step, entire test cases are executed corresponding to the main program and whole faults are injected and results are detected.

- **Result analysis**

  The result of test cases whether they are passed or failed on the basis of predefined conditions are mentioned .

4.3 Overview of Experiment

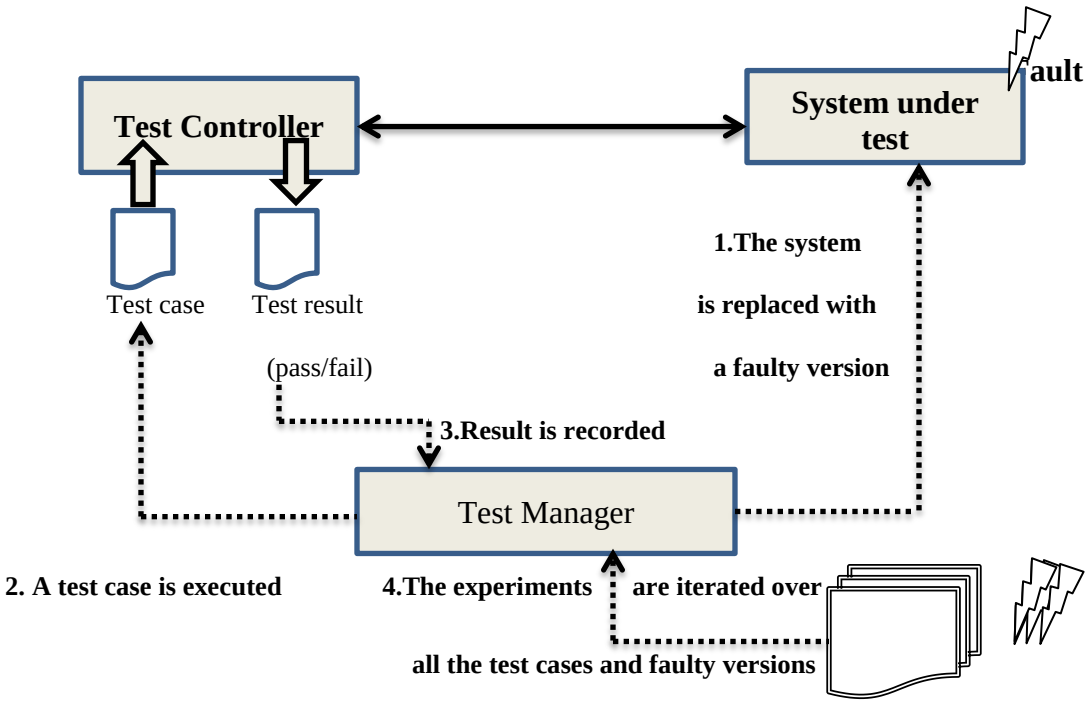

Fig 4.2 Overview of Experiment [13]

The experimental setup[30] is shown in Fig. 4.2. In every experiment, the Test Manager run a test case on a faulty version and collects the test result. Since we are interested in whether the test case is able to detect a given fault (i.e., to cause a failure), we only need a simple failure result (i.e., a pass/fail outcome).

4.4 Proposed Technique

In the proposed technique, a tool StoreApp for fault injection testing is developed using Visual studio 2012 that significantly reduce the total execution time of faults. This too gives faster performance than other previous systems. And StoreApp tool works in C# language. The faults are injected on the basis of following categories:

Input Injections:

- Inputs of various parameters into the methods.
- Return value fault.
- Faults based on fault rule.

Functionality Injections:

- Fault injection based on conditions.
- Injecting faults by fault rule.
- Signature parsing.

Memory Leak Detection From Injections:

- Throw exception fault.
- Call stack.
- fault scope

Run time Fault Injections:

- Run time fault.
- Return value runtime fault.

4.5 Proposed Algorithm

This section explains how this algorithm works. We are providing a pseudo code which explains the working of this algorithm. Input programs are applied on StoreApp tool and give a reduction in time for execution of faults and effective results.

Table 4.1: Proposed Algorithm

Input: Code of faults in C# and conditions mentioned **Output:** Occurrence and execution time of faults. *Algorithm: StoreApp_algo*
Step 1: Data is stored in database at the backend. **Step 2**: Create faults using StoreApp tool. **Step 3:** StoreApp web service hit the data stored in database according to the faults. **Step 4:** Alter, delete and modification **o**perations performed on database. Web service will hit and if operation gives right output, then there is no fault. **Step 5:** Web Service hit for each fault one by one. **Step 6:** Finally fault is injected and result is displayed. **Step 7:** The results include occurrence and execution time of each fault.

The proposed algorithm works on the basis of injection of faults on database of software and check whether the fault occur or not and execute the faults. Faults injection is based on some conditions and fault rule. The concept of signature parsing is used to detect the fault. The Fault injections occurs at run time and also return some value at runtime.

4.3 Snapshots

Fig 4.3 shows that initially when we open the StoreApp.sln File, the following window appear.

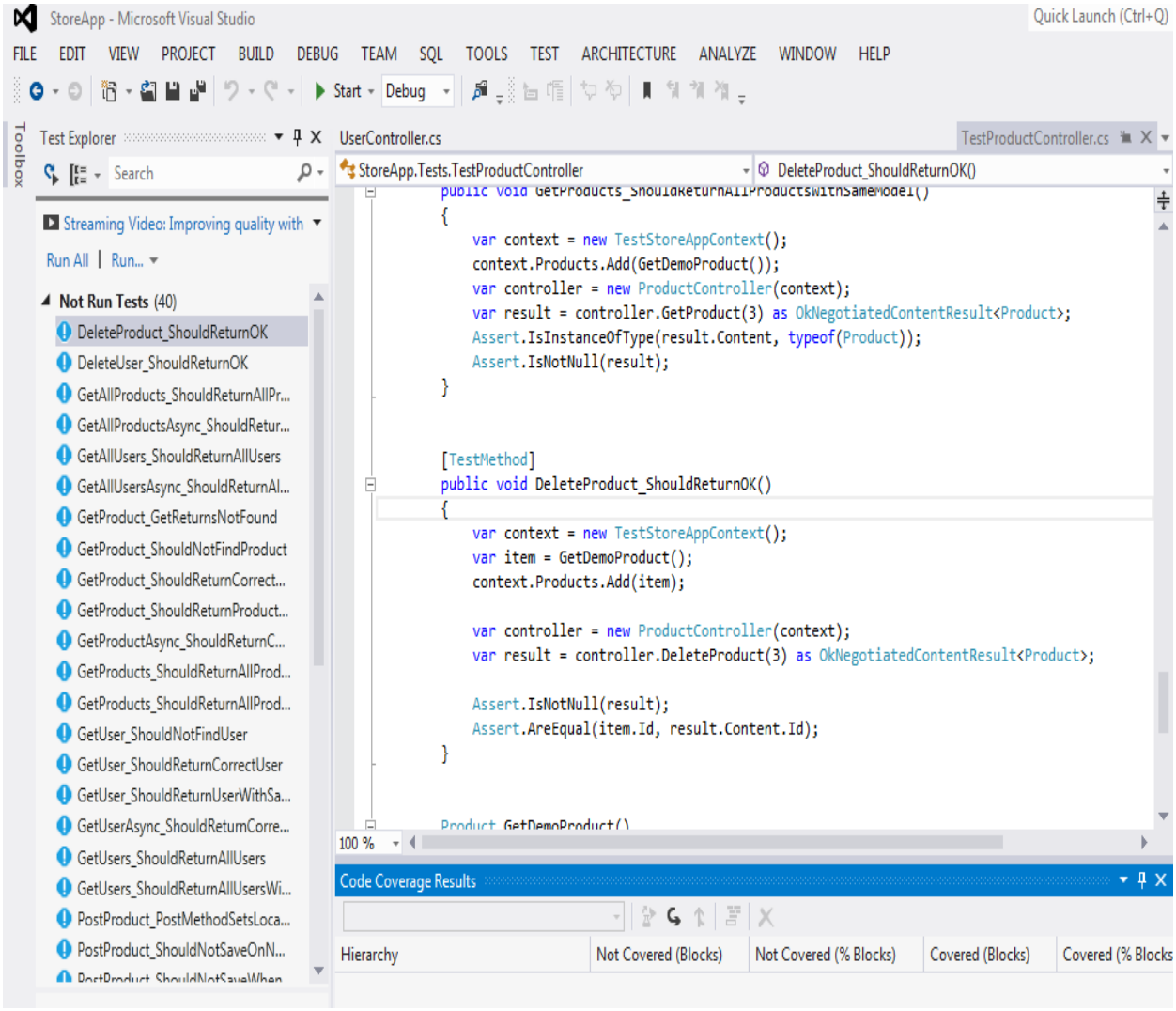

Fig 4.3 First Page

In Fig 4.4 ,there are list of test on left hand side which are performed .On double clicking any code we can see the code of that particular test.

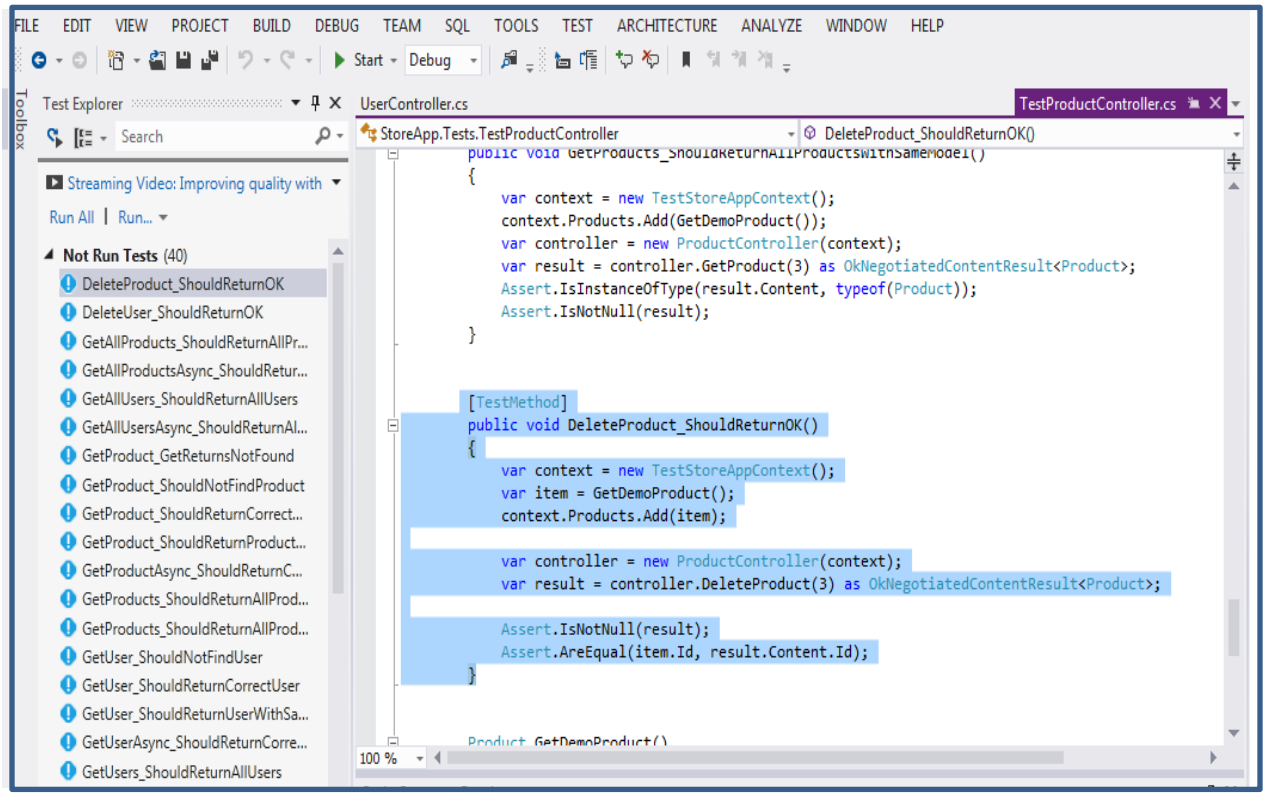

Fig 4.4 Code of Particular test

In the previous snapshots, the code is not in running state. Fig 4.3, the option is shown how we can perform the test in customize way.

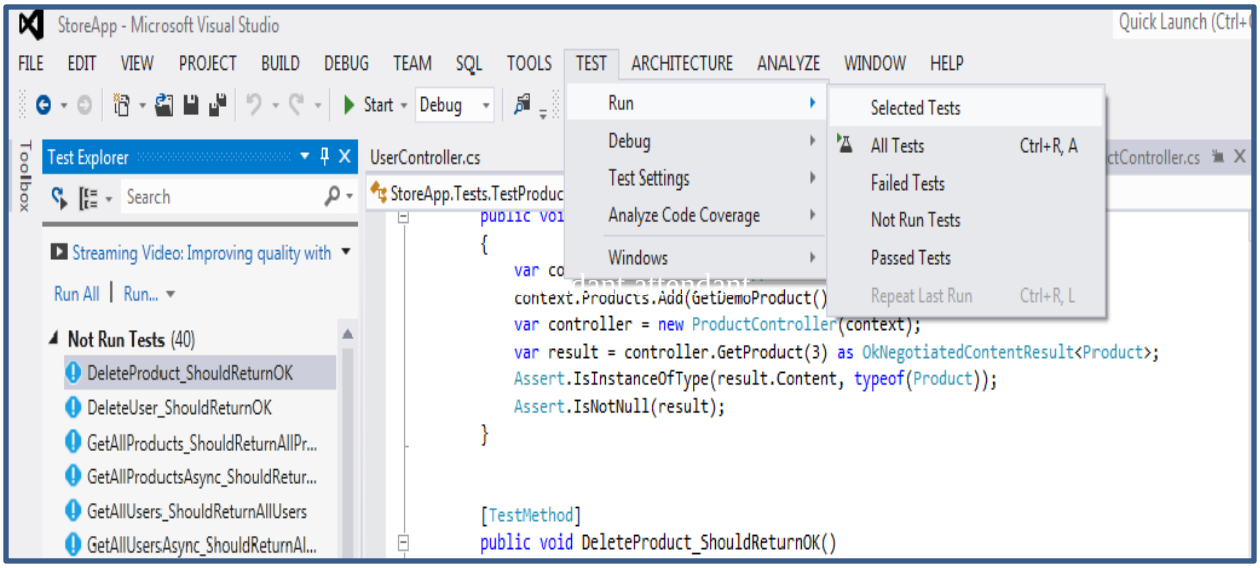

Fig 4.5 Test in Customize Way

After running the test cases, the result show on the basis of fault fail or past. In Fig 4.4 We can see the failed test and passed test sepaeratly.

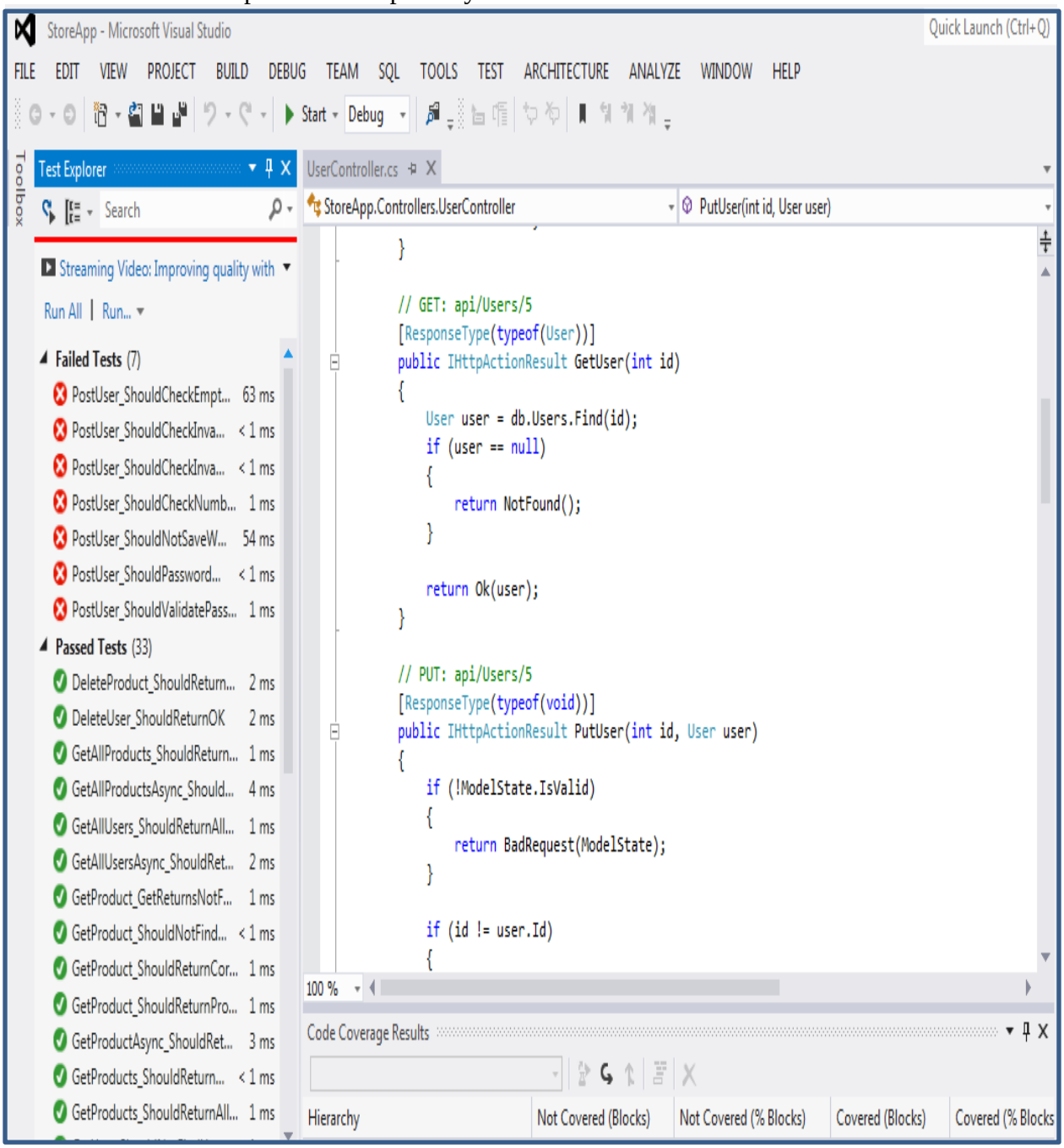

Fig 4.6 Result of Tests

In fig 4.5, the test case consume some time to perform.so the time is displayed in ms (millisec) consumed by each test case and summary also.

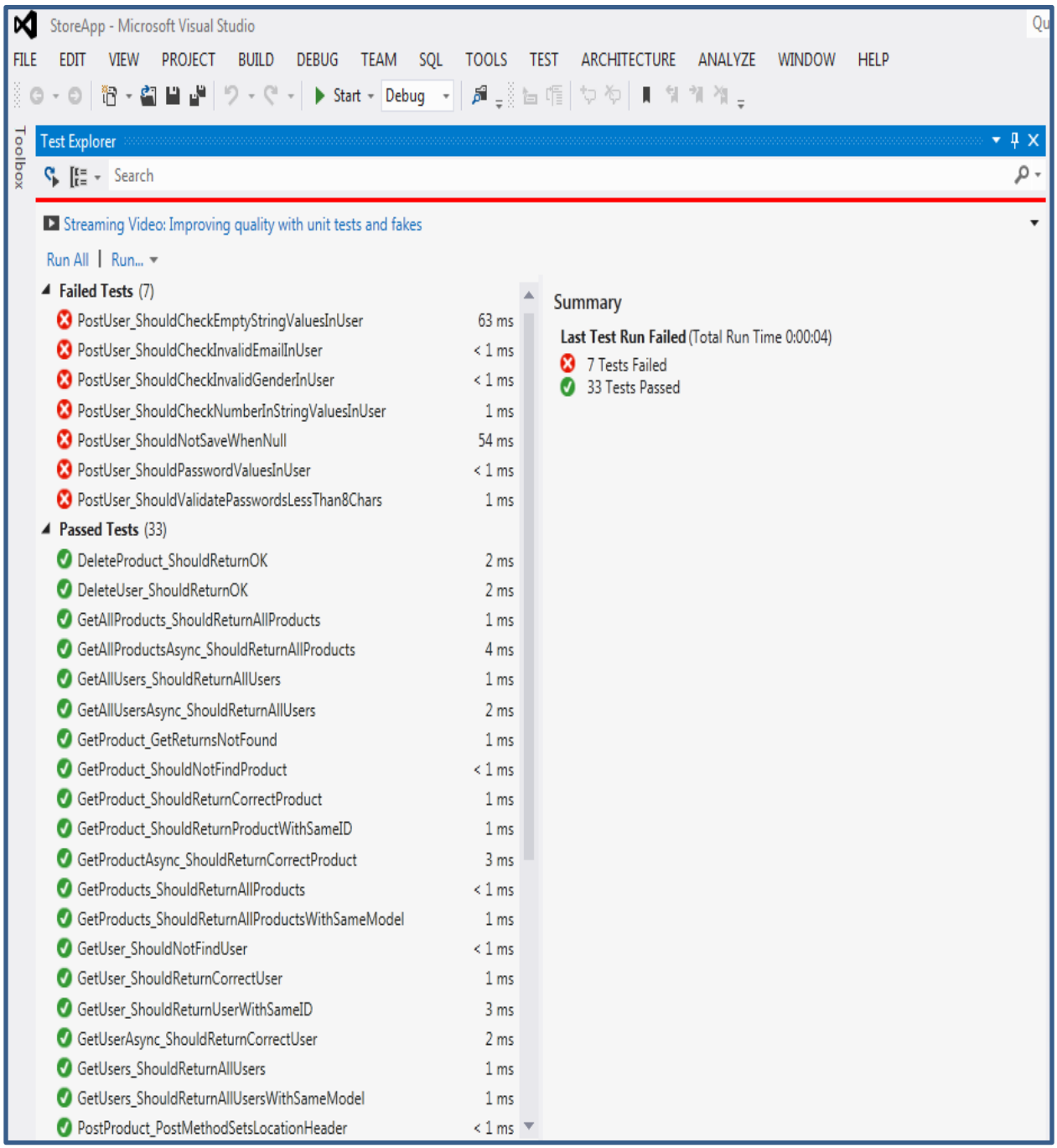

Fig 4.7 Time Consume by each test

4.2 Experimental Result

The execution time of each test is comparatively low with respect to other testing techniques.

Table 4.1 Execution Time of Passed Test

Fault Methods	No. of Lines	Actions	Execution Time(in ms)
Delete _Product	14	Return OK If product deleted	1
Delete _User	12	Return OK if user deleted	1
Get_All_Products	15	Return detail of all products	1
Get_All_User	10	Return detail of all users	3
Get_Product	11	Return detail of product	2
Get _Product_SameModel	13	Return Product with same model	1

Table 4.1 shows the total execution time of faults that are passed during execution. Moreover, it can be observed from the tables that execution time is significantly reduced using proposed algorithms.

Table 4.2 Execution Time of Failed Test

Fault Methods	No. of Lines	Actions	Execution Time(in ms)
PostUser_check_Empty	14	Check empty strings	9
PostUser_check_Invalid	10	Check invalid email user	1
PostUser _Invalid_Gender	10	Check the user gender	1
Post _NumberInString	15	Check the number in String	1
PostUser_NotSave	13	Check null value	5
PostUser _PasswordValue	15	Check the password	1
Post_Validate_Password	12	Validate password less than 8 characters	1

Table 4.2 shows the execution time of faults that are failed during execution. It can be observed from the table that execution time is reduced.

Chapter5

Conclusion and Future Scope

5.1 Conclusion

Fault Injection Testing is a fault based technique that works in conjunction with automated unit testing technique. It is done by introducing faults and observes fault tolerance behavior of the system. But it has two main problems: In which block to inject the faults and large effort, time and cost consumption.

This dissertation presents an approach for solving these problems. The proposed approach is inexpensive for mutation testing. A new tool *StoreApp* is developed that has some methods of code. In *StoreApp,* Web Service hit again and again, performs some test and record the results.

Some test failed and some are passing on the basis of conditions mentioned in code.

StoreApp tool that reduces the total execution time required to execute test and test results is shown. A test is failed if it does not fulfill the conditions defined in code otherwise it is discarded. The proposed technique helps in reducing time and cost consumption. Additionally it helps in differentiating failed or passed tests accurately.

5.2 Future Scope

In future following works can be done:

- Develop an automated technique to add new faults as the product become older.
- Proposed method can work with any type of data like numeric, alphabetic.
- New faults are not added as they occur from time to time automatically. We have to add these faults.
- Scope of integration of fault injection testing with other types of testing like and integration testing.

REFERENCES

[1] B.Beizer, ”Software Testing Techniques”, Van Nostrand Reinhold, 2nd edition,1990.

[2] R.S. Pressman. “Software Engineering: A Practitioner’s Approach” , 3rd Edition,McGraw Hill ,New York, pp. 559,1997.

[3] [Online].Available: http://tetworks.opengroup.org/.

[4] [Online].Available: http://en.wikipedia.org/wiki/IEEE_829\

[5] D. M. Blough and T. Torii, “Fault-Injection-Based Testing of Fault-Tolerant Algorithms in Message-Passing Parallel Computers *,” pp. 258–267, 1997.

[6] W. K. Chan and T. Y. Chen, “An Overview of Integration Testing Techniques for Object-Oriented Programs,” pp. 1–6, 2002.

[7] W. Chao, F. Zhongchuan, C. Hongsong, and C. Gang, “FSFI: A Full System Simulator-Based Fault Injection Tool,” *2011 First Int. Conf. Instrumentation, Meas. Comput. Commun. Control*, pp. 326–329, 2011.

[8] G. Chen-xi, C. Bai-gen, S. Wei, and W. Jian, “Study and Application of Fault Injection in HLA-based Train Control Simulation System,” pp. 1869–1873, 2011.

[9] F. Collins, “Fault Injection Testing for Distributed Object Systems,” 2001.

[9] D. Cotroneo and R. Natella, “Software Fault Injection for Software Certification,” pp. 1–17,2010.

[10] K. Farj, Y. Chen, and N. a. Speirs, “A Fault Injection Method for Testing Dependable Web Service Systems,” *2012 IEEE 15th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, pp. 47–55, 2012.

[11] M. Le, A. Gallagher, and Y. Tamir, “Challenges and Opportunities with Fault Injection in Virtualized Systems,” 2008.

[12] R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira, “On Fault Representativeness of Software Fault Injection,” vol. 39, no. 1, pp. 80–96, 2013.

[13] R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira, “On Fault Representativeness of Software Fault Injection,” vol. 39, no. 1, pp. 80–96, 2013.

[14] G. L. Nazar, P. Rech, C. Frost, and L. Carro, “Radiation and Fault Injection Testing of a Fine-Grained Error Detection Technique for FPGAs,” vol. 60, no. 4, pp. 2742–2749, 2013.

[15] N. Shahida, M. Yusop, and W. F. Abbas, “Increasing Test Coverage using Human-Based Approach of Fault Injection Testing,” pp. 287–291, 2011.

[16] N. Silva and R. Barbosa, “A View on the Past and Future of Fault Injection,” 2013.

[17] E. Software, “The Fault-Tolerant Design and Fault Injection Test for Embeded Software,”, pp. 307–310, 2010.

[18] G. Suganya, “A Study of Object Oriented Testing Techniques : Survey and Challenges”,pp. 2009.

[19] W. Xi, “A novel fault Injection Algorithm for Safety Analysis,” , pp. 1–4, 2012.

[20] J. Xu and P. Xu, “The Research of Memory Fault Simulation and Fault Injection Method for BIT Software Test,” *2012 Second Int. Conf. Instrumentation, Meas. Comput. Commun. Control*, pp. 718–722, Dec. 2012.

[21] H. Ziade, R. Ayoubi, and R. Velazco, “A Survey on Fault Injection Techniques,” *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[22] J. Aidemark, J. Vinter, and P. Folkesson, “GOOFI : Generic Object-Oriented Fault Injection Tool,” pp. 83–88, 2001.

[23] E. Martins, C. M. F. Rubira, and N. G. M. Leme, “Jaca : A reflective fault injection tool based on patterns ,” pp. 0–4, 2002.

[24] S. Han, K. G. Shin, H. A. Rosenberg, and A. Arbor, “DOCTOR : An IntegrateD Software Fault InjeCTiOn EnviRonment for Distributed Real-time Systems Fault-Injection Requirements,” 1995.

[25] B. P. Sanches and R. Moraes, “J-SWFIT : A Java Software Fault Injection Tool,” 2011.

[26] S. Dawson, F. Jahanian, and T. Mitton, “ORCHESTRA : A Probing and Fault Injection Environment for Testing Protocol Implementations,” , pp. 56, 1996.