Resilience Assessment of Software Systems via Software Fault Injection

Johannes Günthör

University of Stuttgart
Institute of Software Technology (ISTE)
70569 Stuttgart, Germany

Abstract. It is utterly impossible to create fault-free software. Engineers and researchers have developed different techniques to evaluate software stability and flexibility. Fault injection is one option to test software on its stability and resilience. This paper aims to give a comprehensive overview of the current software fault injection tools. Furthermore, the current state of the art of fault injection in large companies is presented.

1 Introduction

At the beginning of the 21st century a lot of research on the matter of fault injection had been done[16]. For example the three different types of fault injection (Injection of Data errors,Interface errors and code changes where made) were Differentiated. The researchers abandoned the subject since they were finished researching software fault injection. In recent years the subject was lifted again because new software environments evolved and and the requirements for software had changed. Huge cloud-based software programs need to scale with user demands. If more users are online the system capacity needs no grow on its own to manage the expanding traffic. These systems run in a cloud environment that is mostly a black box for the developer because another company operates the cloud service. The scalability and the unknown hardware result in new bugs that can not be sorted out with conventional testing methods. Therefore researchers take a new approach at fault injection and see what it can do to cope with the new challenges.
First of all this paper focuses on fault injection in general. What has been the status when the subject was abandoned, what can be done with fault injection and what techniques are out there.
In section three we focus on the a subset of tools that where developed to conduct a fault injection. The section contains information about these tools and what capabilities they offer. These are the main questions we try to answer in the third section.
In section 4 we discuss the state of the art of fault injection in today's environments. Large companies have also seen that conventional unit testing is not that useful in a cloud based environment. They started developing their own

tools to conduct a fault injection. The fourth section contains information about the most commonly known fault injection techniques that are used today in a cloud-based environment.
The fifth section is a documentation of a fault injection experiment conducted by the author of this paper. The theory on how a fault injection should be done was tested in a real environment with the theoretical knowledge that this paper summarizes.
At last we summarize all the described sections to come to a conclusion. "Where is fault injection today and where should it continue in the future?" is a question this paper tries to answer.

2 Fault Injection - What Techniques are Out There?

Starting of with the simplest question of all: What exactly is fault injection? Many papers describe this in different and complicated ways [11, p. 12] or ignore it completely by assuming that the reader already knows. Fault injection is simply the art of controlled insertion of bugs in a system to assume its stability. Sounding very simple there has been a lot of research done on how to conduct an injection. First of all, we have to differentiate between software and hardware fault injection.

2.1 Hardware Fault Injection

Since this paper focuses on software fault injection this section will be quite short but it would be irresponsible not to mention hardware fault injection at all. Hardware fault injection focuses on corrupting the actual hardware of a system. Therefore it can not be done with software tools. Hardware fault injection uses additional hardware like for example a radiation unit to irradiate a CPU during the run time of a system [2, p. 7-8]. The radiation will corrupt the output of the CPU and produces incorrect results. A random fault was injected into the system.

2.2 Software Fault Injection

Software fault injection is testing software on a higher level. Testing is a part in every good software development process and very important. Software fault injection is conducted by one or multiple tools to discover errors in a system. Different parts of the software are manipulated to produce corrupted data or to malfunction to see how the system behaves with the wrong data, how dangerous it is to have false data and how to correct errors that can potentially occur during the run time of a system.

2.3 Software Fault Injection Techniques

Over the last decades engineers and researchers developed several different techniques on how to conduct an injection. Each of these Techniques have something

in common. During the planning phase of a fault injection experiment is necessary to evaluate these three overlapping topics [11]:

- What to Inject?
- Where to Inject?
- When to Inject?

Injection of Data Errors a method created to mirror hardware faults by perturbing the state of memory. The first question that should be answered is always "Where to Inject" into the System? Since this method focus on hardware faults we identify the location where hardware faults can be located e.g, in the memory register. This can either be a random location or a user selected one. "What to Inject?" is very simple. Hardware faults are usually bit flips or opposite values and since this method tries to mirror hardware faults select one of these. The last issue that should be solved before an injection is conducted is "When to Inject". It has proven useful to start the injection at a certain time after the system was started or after a specific event has been called [11, p. 15]. After all how useful is the injection of data errors? This method brings a lot of limitations with itself. First of all many injected faults do not cause errors since the changed data is irrelevant for the source code anyway. These leads to a low efficiency of the procedure and a high rate of injections. This method also needs field data to be used and tested. Gathering lots of field data is either very expensive or the injection takes place during the run time of the system which is obviously out of the question for most systems [11, p. 17-22].

Injection of Interface Errors This method is emulating faults outside a component or system. Corrupting the data of an input or output shows how a system can deal with unknown data. "What to inject?" can be quite different. One approach may be the Fuzzing approach. Replacing a correct value with another. The new value can either be randomly generated or chosen by the Fuzzer. The name already hints on "Where to inject?". Selecting an interface can be done by either using a user interface or by looking at the architecture of a system and deciding which interface should be tested. "When to Inject" a corrupted data field into an interface is basically the same as in the "Injection of Data Errors". The time can either be chosen by waiting a specific amount of time after the system is started or counting the number of calls of a specific service [11, p. 23]. Now it is to clarify how useful this method is. "Injection of Interface Errors" is actually used in several real world applications like mission-critical systems, server applications, user applications and cloud computer systems [11, p.34-35]. The high usage of this method can be explained by the fact that it is commonly known that wrong inputs are happening in real software environments. Therefore, testing inputs on correctness is usually a part of testing during development (e.g., (J)Unit Tests). Limitations are also given. Discovering an input that causes a system or component to fail is mostly luck dependent if the tester has no knowledge of the code. This leads to a high number of tests, but it can never be known on how resistant an interface actually is until all possibilities

are tested which is impractical and utterly impossible. Therefore an injection of interface errors can only improve confidence in a code but it can almost never proof that an interface is 100% save to use.

Injection of Code Changes Like the name states this method focuses on changing the source code of the targeted system. "What to Inject" is the most important part of this technique. It is very important to change only a small part of the code like an operator, a definer or adding a NOP. "Where to Inject" a change of the source code is a binary option. The change can either be done by corrupting the source code or by changing the binary executable. The subject "When to Inject" has changed over the last decade. Older studies recommend to change the code before the execution of a system to ensure that the error can be reproduced [16]. Newer studies on the other hand recommend changing the code during the run time of a system to increase the possibility of a failure [11, p. 35-37]. A downside of this technique is the accessibility of the code. A program is mostly sold as an executable and it is way harder to modify the executable then the source code itself. Studies have shown that tools like G-SWIFT that are specially made to perform a code change injection, on a binary, sometimes fail. The reason for this is the complexity of objects created by modern programming languages. These objects are hard to identify by just looking at the binary code. A closer look at G-SWIFT is taken in Section 3. Another problem is that even if a code is correctly injected so that a fault would be the result of the method call, the method has to be called first. Therefore, after a code injection forces the tester to do a complete system test to see all results. Studies have also shown that accidental redundancies in the code cover injected fault errors [11, p. 38-42].

3 Tools

Injecting one or two faults manually is obviously not very useful. The data gathered by this will not be useful at all if the person who is conducting a injection don't want to do hundreds of injections by hand. Using the right tool is quite important because each technique uses a different set of tools. It is also important to know what the capabilities of the tool are, to make sure engineers are trying to achieve a goal with a tool that was actually made for that purpose. This section explains a couple of tools that exist for each technique.

3.1 Tools for Injection of Data errors

Fault Injection Based Automated Testing Environment (FIAT) is one of the tools that contribute to a test environment to mirror hardware faults [11, p.16]. FIAT is manipulating workloads of a real-time dependable system [13, p.2] by using an attached fault library. An Attribute extractor is extracting attributes that can the be loaded with values from the attached library [13, p.2]. This tool was already mentioned in papers written in 1990 and therefore it has a minimal age of 26 years [3].

Generic Object-Oriented Fault Injection Tool (GOOFI) uses a graphical user interface to ensure an easy interaction between the fault injector and the software. On the other hand adding new injection techniques or adapting to new systems requires a programmer to increase the code at certain points by using interfaces that the program provides [1, p.3] The tool highly compatible because of the programming language Java and it stores information in a SQL friendly database [1, p.1]. This tool was also mentioned in a paper that was written in 2001 [1] and must therefore be at least 15 years old.

Remaining tools There a few more tools that can provide the ability to conduct a injection of data errors. Most frequently mentioned tools are:

- ORCHESTRA [6]
- DOCTOR [11, p.16]
- XCEPTION [11, p.19]

3.2 Tools for Injection of Interface Errors

Simian Army is an entire set of tools that where developed by Netflix. Netflix is a large company that only uses a cloud-based infrastructure to offer their services to the public. It is the core of the current state of the art analysis at Netflix and will be focused at in Section 4. The Simian Army was developed a few years ago and it is running since then on Netflix applications until today.

DICE Fault Injection Tool DICE is similar to the Simian Army a tool that was developed to conduct fault injection in a cloud-based environment [14, p.1]. The tool was developed to overcome the Chaos Monkey (explained in Section 4) by making it platform-independent and giving it more power than just killing instances [14, p.1]. The DICE fault injection tool was developed after a few flaws in Chaos Monkey where detected. The tool is therefore younger than the Chaos Monkey and was probably built in the last one or two years.

Burp Suite Burp Suite is a Fuzzing tool originally created to do security testing of web applications [12]. Burp Suite is capable of catching HTML requests and replace values with different ones chosen by the Fuzzer. Since this tool offers the same functionality that we need to conduct an interface fault injection this would also a good choice to use as a tool for injection of interface errors. The benefit of this tool is that is uses a graphical interface and does not require additional scripts to work.

Remaining tools There are more tools that can possibly be used to conduct an interface fault injection. The most commonly mentioned tools are:

- Ballista [11, p.24-25]
- PreFail [6]

3.3 Injection of Code Changes

Generic and Representative Fault Model (G-SWIFT) changes small parts of the code by adopting generic fault models of common bugs from different languages. The initial model only had C programming bugs from different Sources [11, p.38]. The tool identifies small parts of the binary code like simple operators and changes them. This leads to a faulty algorithm or program execution. Therefore a fault is injected [11, p.39]. The tools earliest mention comes from the year 2002 but research on it was done since then and it still up to date. The tool was reportedly used in investigation in real world problems in the year 2013 [11, p.42].

Remaining tools More tools that are used to conduct an "Injection of Code Changes".The most commonly mentioned tools are:

- FAUST [11, p. 36]
- FINE [11, p. 36]

3.4 Conclusion about tools

Most of the mentioned and existing tools are quite old and are not up to date. The older and outdated tools are not really user friendly and hard to get. Since fault injection would be a part of testing during the development, it is highly problematic that there has to be additional script writing and learning of complex tools in a time during the development that is mostly considered to have not enough time.

4 Software Fault Injection Today

How does fault injection look today? This section focuses on answering this question. Information about a the current state of the art analysis is rarely described in scientific papers. Therefore information about this section is mostly extracted form blog records and videos of engineers in high positions at large companies like Netflix, Microsoft and Google.

4.1 Netflix

The most important source is most likely Netflix. Netflix is using the Amazon cloud: AWS (Amazon Web Services) to offer their services to the public. Netflix is a service which most people use in their free time. A simple conclusion that Netflix has made is that their service must offer a high scalability to run smoothly and not waste hundreds of server racks idling at 7 am in the morning. This is based on the fact that most people have a similar daily routine and therefore free time in the evening. The Netflix data distribution that has be publicated, which is used by Netflix to approximate the required computing power, can be seen in Figure 1[7].

High scalability and elasticity in a cloud is a relative new property that software has to master. Netflix invented the Simian Army (a tool set) to continuously inject faults to see what happens to the system. The Simian Army is a collection of tools that can inject faults in a cloud based environment. Since the Simian Army needs a large amount of data to show potential errors, the entire tool set is used in the actual live environment of Netflix services. Obviously an injected fault can actually cause an error and might not be corrected by the system itself. Therefore the Simian Army is only working on non-holiday weekdays between 9am and 3pm when engineers are working regularly to fix potential errors [5]. The most important members of the Simian Army and their field of use in the following.

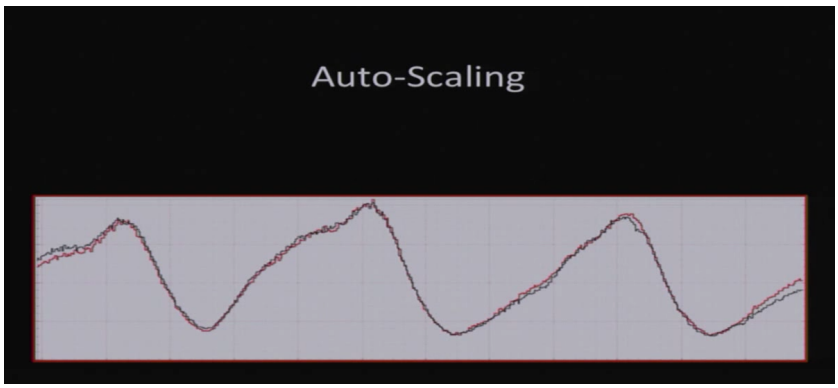


Fig. 1. Netflix Data Graph

Chaos Monkey The most commonly known member of the Simian Army is the Chaos Monkey mentioned in the previous section, which simulates failures of single instances. This means the Chaos Monkey is killing Virtual Machines (VM) [14]. The Chaos Monkey operates on such a small level that besides the monkey itself errors are quite common and well known.The Chaos Monkey is running permanently in the above described time frame [5].

Chaos Gorilla The bigger brother of the Chaos Monkey would be the Chaos Gorilla. It simulates the outage of a entire data center. Entire data center outages are more dramatic and require an human oversight because if a Chaos Gorilla actually causes an error many users affected and real harm could be done to the brand Netflix or their wallets. Because of the potential dramatic effects the Chaos Gorilla is not permanently active. Chaos Gorilla is a planed event to simulate disaster recovery and to test fail over of data centers with people actively looking into the current state of the system to learn something form the potential outage [7]. This functionality exists as a tool but it has to be started manually.

Chaos Kong The Chaos Kong is like the oldest brother of the crew. Chaos Kong simulates the outage (or unavailability) of entire clusters of data centers like in the previously seen attack of the IoT [8]. He works similar to the Chaos Gorilla and consequently the same rules apply. It does not run permanently in the background and is more of a planned event where people are on alert and looking actively into it [7]. Chaos Kong as the same code base has Chaos Gorilla that was expanded [4].

Latency Monkey Induces delays in the server client communication to detect partial errors that occur when the servers are under high utilization but no complete outage is caused. This is really useful to test the fault-tolerance of a system on its dependencies since those are needed to run the system [15][7].

Janitor Monkey This is basically a garbage collector on a complex level. Searching for unused instances and resources and trashes them [15].

Doctor Monkey Tapping into systems to check if they are running properly. Terminates or reports unhealthy instances [15].

Conformity Monkey If anything is not running to previously agreed conditions the "Conformity Monkey" will detect this, shut it down and inform the owner of that service [15].

Security Monkey An extension of the "Conformity Monkey" that is looking for violations of predefined security rules like for example by looking if SSL and DRM certificates are up to date. It also kills instances that are not working properly and alerts the responsible engineer [15].

After all why did Netflix even invent all these new tools and did not completely stick to the conventional testing methods like (J)Unit testing? The answer is simple. Netflix does unit testing but it does not show the same results as the real environment since the test environment is just a small fraction of the real one. The huge scale of the real world applications is simply to complex and sees permanent updates so there can't be a single version which is fully tested [4].

4.2 Microsoft

Microsoft hosts its own Cloud-Computing-Platform called "Microsoft Azure". "Microsoft Azure" has the same business model as AWS by selling cloud computing servers. Inspired by the accomplishments of the Simian Army, developers at Azure Search created their own Chaos Monkey and called it "Search Chaos Monkey". Search Chaos Monkey was inspired by the original "Chaos Monkey" working on a low level. The tool has multiple functionalities to interrupt services and has three potential failure outputs. These failure outputs are categorized into low, medium and high chaos failures. The tool runs permanently and has different outputs for each kind of failure it encounters. Low chaos failures are reported

as a bug but the system can recover by itself and no further action has to be done immediately. Medium chaos failures can also be recovered from but the users of these services are impacted by performance or availability degradation. A Medium chaos failure already alerts engineers on call. High chaos failures will interrupt services immediately and often require manual fixes. With these chaos levels there is an iterative system in place that constantly improves the infrastructure for the services that are affected by "Search Chaos Monkey". There is one more chaos level called "Extreme Chaos". This describes a state in which data is lost and instances fail without causing alerts to the people in charge. Since the results of these kinds of failures are to extreme to justify them the "Search Chaos Monkey" does not have the ability to cause them [10].

4.3 Google

Either Google does not publish what their approach on fault injection is or there is nothing big happening in that direction. Either way not much is known from Google. What is known is that they already do tests that mirror the behavior of "Chaos Gorilla" and "Chaos Kong". Google is causing outages at data centers manually by redirecting the traffic of up to two out of their four data centers [9, p.6]. Google calls these tests "DiRT" and they are overseen by two Teams that expect failures at any given time during their tests [9, p.4].

5 Experiment

5.1 Planning the Experiment

After researching this topic a small experiment was conducted based on the knowledge gained by the research. Before conduction the experiment it had to be planned and some questions had to be answered:

- What system should be tested
- What injection technique to chose
- What where and when to inject
- What tool to use to inject the fault

The System to Test The chosen system is a student project called MuVi that is currently in development at Fraunhofer Stuttgart. This system was chosen because there is development knowledge of the tester on how the system MuVi works and where it might be vulnerable. The System was originally developed by engineers and a student (2009-2015) and is now expanded as a student project (StuPro) with 14 students. The project MuVi is a Multi Display Wall with 36 HD displays that stream information inserted in a web interface and displays the information like it is one big screen. The project currently contains about 20k lines of code and is big enough to contain a lot of unknown bugs errors and faulty behavior.

Injection Technique "Injection of Interface Errors" was the chosen technique for this experiment. This technique was chosen because the system is designed to take multiple inputs from users that have no idea how the system works and potentially use it wrong.

What, Where and When What to inject was quite simple. A predefined text file which contains hundreds of potential fault sources for injection into a web interface. The file named "All_Attack.txt"[1] contains different injection strings starting at true,false ranging over to some JavaScript. Where to inject was a more interesting question. Chosen was a WYSIWYG (What You See Is What You Get) editor that was developed by students. This part of the system was chosen because it seems promising to contain many errors and faults because it is one of the most vulnerable interfaces in the entire system. When to inject seems quite irrelevant for this case but since it is recommended to pick a specific point in time one was chosen. The point was defined as right after a new system start up.

Tool The tool chosen to conduct this experiment is called "Burp Suite". This tool was chosen for multiple reasons. First of all it is a professionally sold tool that still gets updated. The tool also specializes in injection of web pages which was perfect since the Editor WYSIWYG editor is embedded into a web page. The last reason why "Burp Suite Professional" was chosen is simply that it was one of the available and "easy" to access tools. Most of the mentioned tools are either old, hard to find or for specific purposes that did not match this kind of injection.

5.2 Conduction the Experiment

First of all the target was selected and a change was made so that an HTTP request was sent through the system that could be catched by the tool as stated in Figure 2. After targeting the right locations in the request the injection was started. The payload per injection was one line in the above stated text file. During the injection a few interesting events happened. During the first few thousand injections nothing happened expect occasional Java exceptions in the console of the server that receives the data. At some point the server started to no longer respond to the incoming HTTP requests by giving a 500 error to the injection tool like Figure 3. Later in time the system "recovered" and answered correctly again. But it is notably that after about 3000 injections the server responded with a large delay and it took about 30 min to do 5616 injections. Another fringe thing was the fact that the data that is transited, contains an XML file that actually contains the information for the system what to display on which screen. At some point in time the XML file duplicated itself and therefore a text file containing two copies of the XML was sent between server and tool.

[1] `https://github.com/xmendez/wfuzz/blob/master/wordlist/Injections/All_attack.txt`

```
POST /manager/newLargeText.do HTTP/1.1
Host: localhost:3636
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: */*
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:3636/manager/wysiwyg.jsp?scenarioId=150&sceneNumber=1
Content-Length: 378
Cookie: JSESSIONID=wmic8x5xtthz3k8gz60xfbff
Connection: close

scenarioId=150&sceneNumber=1&editMode=insert&largeTextNumber=0&text=tes&style=backgrou
+normal%3B+font-style%3A+normal%3B+text-decoration%3A+none%3B&start=1&width=6&height=1
```

Fig. 2. Catched Expression

After all 5616 injections the XML that usually contains something like 200 or 300 characters was blown up with duplicates to over 150000 characters seen in Figure. While this was happening the server console started to post the XML file on what seems like every injection Figure 4. After all 5616 injections where completed the system was still running smoothly with the exception that an opening or editing function in the injected scenario was not working anymore. From this point on every action taken to open or edit the scenario caused the web page to display a Java exception and post the XML file in the web browser Figure 5.

6 Conclusions

Fault injection seems to be an older topic that has been abandoned because research on it was done. Some techniques like injection of interface errors made it a little bit into real developments environments (Unit testing) but other techniques did not make the cut for non system critical developments. This may come from the small amount of tools that require a large amount of work [6, p.2] and are not really convenient to use. Fault injection has come up again since the established testing methods are not enough work in a cloud based environment. Netflix has proven to be the innovation leader on this matter because their services run exclusively in a cloud environment. Therefore they needed new ways to test the reliability of their systems. The successful results at Netflix seems to be the right direction for groups and companies that are also running large cloud applications. For the future it would be helpful to have an extended amount of members in the Simian Army to monitor even more information because a cloud service is mostly a black box. The Simian Army is open sourced but only usable for AWS. It would be really useful to have the ability to use the Simian Army or similar tools independent from the host of the service.

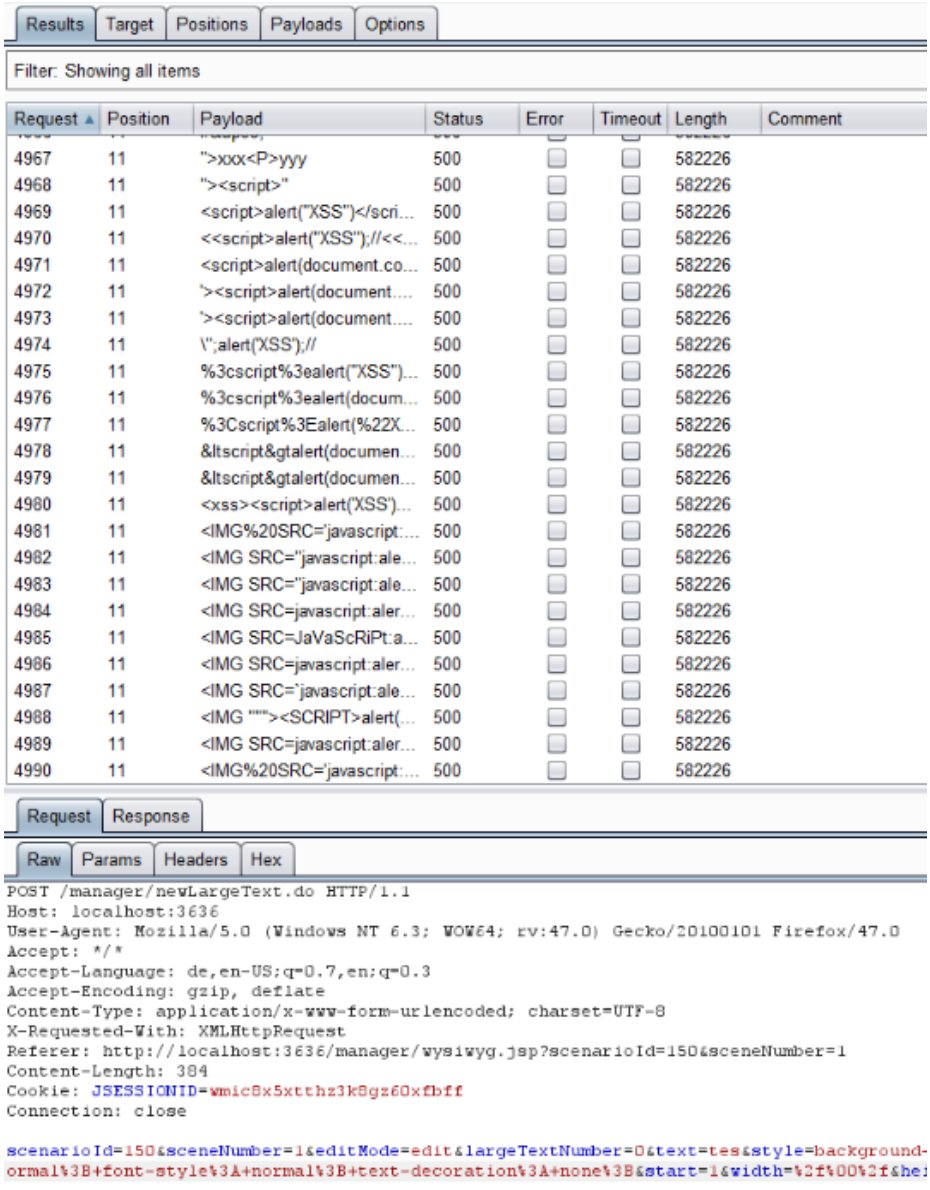
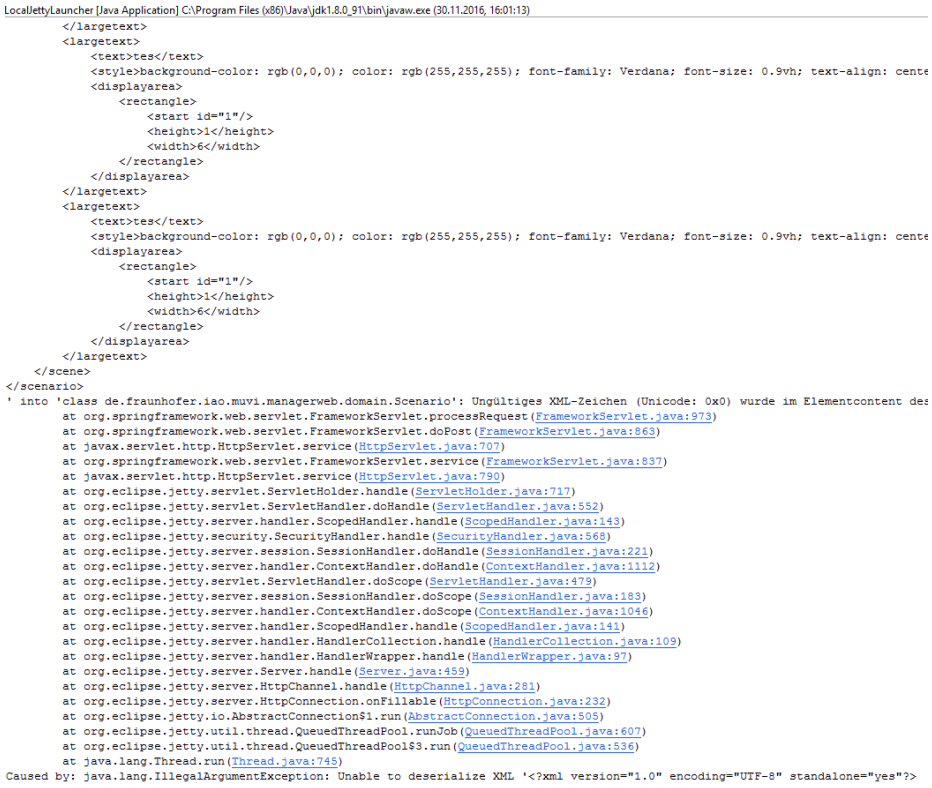


Fig. 3. Burp Suite Professional Interface during fault Injection



(a) Server console log during Injection

Fig. 4. Final XML File

Problem accessing /manager/showScenario.do. Reason:

```
Server Error
```

Caused by:


```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.
<scenario name="Test Burp" description="Burp test seminar" id="150">
    <scene name="testscene" description="" resetcolor="rgb(0,0,0);">
        <largeimage background="rgb(0,0,0);">
            <url>https://upload.wikimedia.org/wikipedia/commons/1/1c/FuBK_testcard_vectorized.svg</url>
            <displayarea>
                <rectangle>
                    <start id="20"/>
                    <height>1</height>
                    <width>1</width>
                </rectangle>
            </displayarea>
        </largeimage>
        <largetext>
            <text>id?|</text>
            <style>background-color: rgb(0,0,0); color: rgb(255,255,255); font-family: Verdana; font-size
            <displayarea>
                <rectangle>
                    <start id="1"/>
                    <height>1</height>
                    <width>6</width>
                </rectangle>
            </displayarea>
        </largetext>
        <largetext>
            <text>tes</text>
            <style>background-color: rgb(0,0,0); color: rgb(255,255,255); font-family: Verdana; font-size
            <displayarea>
                <rectangle>
                    <start id="1"/>
                    <height>1</height>
                    <width>6</width>
                </rectangle>
            </displayarea>
        </largetext>
        <largetext>
            <text>tes</text>
            <style>background-color: rgb(0,0,0); color: rgb(255,255,255); font-family: Verdana; font-size
            <displayarea>
                <rectangle>
                    <start id="1"/>
                    <height>1</height>
                    <width>6</width>
                </rectangle>
            </displayarea>
        </largetext>
        <largetext>
            <text>tes</text>
            <style>background-color: rgb(0,0,0); color: rgb(255,255,255); font-family: Verdana; font-size
            <displayarea>
                <rectangle>
                    <start id="1"/>
                    <height>1</height>
                    <width>6</width>
                </rectangle>
            </displayarea>
        </largetext>
        <largetext>
            <text>tes</text>
            <style>background-color: rgb(0,0,0); color: rgb(255,255,255); font-family: Verdana; font-size
            <displayarea>
```


Fig. 5. Opening scenario after Fault Injection

References

[1] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. Goofi: Generic object-oriented fault injection tool. In *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*, pages 83–88. IEEE, 2001.

[2] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber. Comparison of physical and software-implemented fault injection techniques. *IEEE Transactions on Computers*, 52(9):1115–1133, 2003.

[3] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek. Fault injection experiments using fiat. *IEEE Transactions on Computers*, 39(4):575–582, Apr 1990.

[4] C. Bertram. Injecting failure at netflix. `https://www.youtube.com/watch?v=ioXV28GtXeo&t`, Feb. 2014.

[5] A. T. Cory Bennett. Chaos monkey released into the wild. `http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html`, July 2012. Blog Article.

[6] D. Cotroneo, L. De Simone, A. Iannillo, A. Lanzaro, and R. Natella. Improving usability of fault injection. In *ISSRE Workshops*, pages 530–532, 2014.

[7] J. Evans. Embracing failure: Fault injection and service resilience at netflix. `https://www.youtube.com/watch?v=9R710ry-Cbo`, June 2014. Video.

[8] B. Krebs. Massive internet outage. `https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/`, Oct. 2016. Blog Article.

[9] K. Krishnan. Weathering the unexpected. *Commun. ACM*, 55(11):48–52, 2012.

[10] H. Nakama. Inside azure search: Chaos engineering. `https://azure.microsoft.com/de-de/blog/inside-azure-search-chaos-engineering/`, July 2015.

[11] R. Natella, D. Cotroneo, and H. S. Madeira. Assessing dependability with software fault injection: A survey. *ACM Comput. Surv.*, 48(3):44:1–44:55, Feb. 2016.

[12] PortSwigger. Burp suite overview. `https://portswigger.net/burp/`, Dec. 2016.

[13] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Ysskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin. Fiat-fault injection based automated testing environment. In *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on*, page 394. IEEE, 1995.

[14] C. Sheridan, D. Whigham, and M. Artač. Dice fault injection tool. In *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*, pages 36–37. ACM, 2016.

[15] D. o. C. S. Yury Izrailevsky, Director of Cloud & Systems Infrastructure Ariel Tseitlin. The netflix simian army. `http://techblog.netflix.com/2011/07/netflix-simian-army.html`, July 2011. Blog Article.

[16] H. Ziade, R. A. Ayoubi, R. Velazco, et al. A survey on fault injection techniques. *Int. Arab J. Inf. Technol.*, 1(2):171–186, 2004.

Response to review reports on

“Resilience Assessment of Software Systems via Software Fault Injection”

Johannes Günthör

University of Stuttgart

Institute of Software Technology (ISTE)

70569 Stuttgart, Germany

1. Introduction

This response summarizes the corrections which were applied on the reports, based on the comments given by the two reviewers. Both review reports were thoroughly analyzed. In the following, the reports are identified as R1 (by 'name of reviewer 1') and R2 (by 'name of reviewer 2').

The response is presented in a tabular form whereby the columns are:

- ID: An identifier for the reviewer's comment, CA for Reviewer 1 and CB for Reviewer 2, which is used in some cases in the “Description” column to refer to other comments.
- Comment: The literal transcription of the comment given by the reviewer.
- Action: The action taken by the author of the paper:
  - Fix - the comment identified an issue and is corrected in the final paper
  - No fix - the comment identified an issue and is not corrected in the final paper. The motivation is given in the column “Description”
  - Not valid - the comment is proven to be wrong.

- Description: Elaborates the action taken by the author and gives motivations for not correcting an issue identified by the reviewer.

Note that references in the “Description” column refer to those in the seminar paper while the references prefixed with “R” are given in Section 4 of this document.

2. Response to R1

Table 1: General issues

ID	Comment	Action	Description
CA1	There are some little words missing such as "it" or "is" throughout the paper	Fix	After the review submission the content of the paper was grammatically reworked.
CA2	Also in some places the prepositions are wrong, e.g. "part in" instead of "part of". It would be better to use paragraphs.	Fix	I went through the entire paper looking for “part”. The reviewer was right. In two places it was better to use “section”
CA3	I suggest to append the Figures 4, 5 and 6 onto the conclusion as they are currently tearing it appart.	Not valid	Positioning of pictures is made by latex template which contains conventions like where to place a picture.

Table 2: Specific issues

ID	Comment	Action	Description
CA4	Where did you get the information in the introduction from? It would be nice to enumerate mention your sources	Fix	Addition of one reference. Part of the introduction is a short overview about the upcoming section which has no citation. The other part of the introduction is my

			own opinion which also has no quotes.
CA5	Also prefer to keep the explanation of the paper's structure short and uniform. Use a sentence such as e.g. "Section ?? presents...".	No fix	A short introduction was given in the first submission and I got the comment it was to short because the introduction should not just explain what is in which section but also what the paper is about.
	The example in Section 2.2 with the Add functionality (I prefer Add operation) is technically wrong: It is not a fault of the addition itself but rather of the representation of the result. So if a and b were parameters for a function then yes, the function would be faulty if someone were to add the given numbers and expects a result that fits into an int.	Fix	Reviewer was correct the example I brought up was wrong and I deleted it.

Table 3: Other issues

ID	Comment	Action	Description
CA6	In Section 3 I personally don't understand why the age of the tools is import to be mentioned.	No fix	It was mentioned because the conclusion needs this data and therefore it was important.
CA7	Avoid the use of "screw up" in Section 5.1 "Injection Technique". It's slang and shouldn't appear in a paper.	Fix	Changed “screw up” into “use it wrong” like suggested
CA8	What is a "definer" Section 2.3 "Injection of Code Changes"?	Not valid	To define something … or defining a variable. The word seems fine to me.

3. Response to R2

Table 4: General issues

ID	Comment	Action	Description
CB1	You repeat yourself a few time. For example, section 3.4 Line 1 and 2 you write 2 times older and outdated tools	No fix	I don't see this as an error. Fixing this problem would take the action to merge both sentences into one which is harder to read.
CB2	The layout looks quite good and is applied consistently, but I would do longer blocks and I would avoid such short paragraphs like in 4.1	No fix	The length of a block is decided by subject and not the visual representation. Blocks can be spited if it is useful to cut a long block one into 2 short ones IF two different subjects were discussed in the one long block. Merging short blocks only makes it more complicated to read.
CB3	Additionally, i would explain Simian Army in the Tools section and not in the Netflix section	No fix	I was thinking about this but I thought it was better there because the simian army is really used and also fits in this section. Additionally if a paper is read form top to bottom it would be confusing seeing the simian army pop up in the tool section and get the explanation later (In the Netflix part of "Software Fault Injection Today" Section

Table 5: Specific issues

ID	Comment	Action	Description
CB4	The goal to give an overview of tools and state of the art is reached, but you just should not only sum up the paper in the first section.	No fix	Already stated in a comment above. The first submitted paper had corrections saying the introduction should be longer and an overview of the upcoming content.
CB5	The paper is quite a bit struggling because you explain the tools and i did not expect in the netflix subsection that a tool is explained. I would do that in an extra section.	Fix	Personally I think this was clear but I added an explanation in the paper that the simian army is a set of tools. Splitting it into two sections is on the other hand a bit of overkill.
	You could expand usage in companies and expand the interesting part of the experiment with some more data.	No fix	First of all the paper is already quite long and just adding data to make it longer while the subject is already understood is just stupid. The experiment did not really gather data. It was more an injection and see what is going on and describe this. It is also questionable what the data would be worth since the student developed project changes rapidly.
CB6	Your result presentation is quite good, but you could just add some screenshots of other tools or some figures to explain something.	No fix	Adding more screenshots would exceed the maximum number of 15 pages. Also just adding pictures if there are no useful ones is kind od stupid.
CB7	The screenshots of your experiment only show the	No fix	Again maximum number of pages would be exceeded. Also it would

	Interface of Burp suite and Server logs. I think one page is too much for showing server log. Maybe you could provide screenshots of MuVi?		not help the understanding of the experiment to see 36 HD displays.
CB8	The references are cited and most facts are good referenced, but I think you should add "accessed at..." references to your web links.	No fix	It would not help anything. The referenced internet pages / documents are accessible when the paper is published. Everyone reading this in the future can get the accessed time by the time the paper was published.

Table 6: Other issues

ID	Comment	Action	Description
CV9	I would put more pictures in your paper and avoid screenshots of code no one can read because it is too small.	No fix	Adding pictures just to have more pictures is not really useful. Also more pictures would exceed the maximum number of pages(15). Picture size was already increased and is already readable making the even bigger would not gain any purpose.