MSc in Informatics Engineering

Entermediate Report

Evaluate the robustness of Cloud

Gonçalo Silva Pereira

Supervisor	Raul Barbosa
Co-Supervisor	Henrique Madeira

Department of Informatics Engineering
University of Coimbra

June 11, 2015

Dedication

Acknowledgements

I would like to thank to ——— and to professors Raul Barbosa and Henrique Madeira, who are role models, by their support and help to take good decisions.

Thank my girlfriend for her support, understanding and the fellowship along this path. At my friends and colleagues of Department of Informatics Engineering for the patience and for all times they have given me support.

Last but certainly not least, I would like to thank to my family for encouragement, love and all the unconditional and constant support that let me to fulfill this dream. Obrigado!

Gonçalo Silva Pereira

> “Bridges are normally built on-time, on-budget, and do not fall down. On the other hand, software never comes in on-time or on-budget. In addition, it always breaks down.”
>
> *Alfred Z. Spector, Google Research*

> “I have no special talents. I am only passionately curious.”
>
> *Albert Einstein*

Contents

Contents i

List of Figures ii

List of Tables ii

Abbreviations iii

Abstract 1

1 Introduction 2
- 1.1 Contextualization 2
- 1.2 The project 2
- 1.3 Objectives 2
- 1.4 Document Structure 3
- 1.5 Management 3
  - 1.5.1 Meetings 3
  - 1.5.2 Risks 3
  - 1.5.3 Planning and Tracking 4

2 State of the Art 4
- 2.1 Software Implemented Fault Injection of Software Faults 5
- 2.2 ODC Model 5

3 Research objectives and approach method 7
- 3.1 Cloud Computing 7
- 3.2 Tools - GCC Parser, Bison and Eclipse CDT 8

4 Fault Injector Development 9
- 4.1 Generate derivations 9
  - 4.1.1 Fault Types - Missing 10
  - 4.1.2 Fault Types - Wrong 11
  - 4.1.3 Fault Types - Extraneous 11
- 4.2 Constraints 11
- 4.3 Applications to inject faults 13

5 Work plan and implications 14
- 5.1 Analyze the effects 14
- 5.2 Compile programs 14

6 Conclusion 15
- 6.1 Global Vision 15
- 6.2 Future Work 15

A Appendix 17
- A.1 Appendix A - Gantt diagrams 18
- A.2 Appendix B - Risks table 19
- A.3 Appendix C - Decision Tree 20

References 21

List of Figures

1 *Overview of the injection tool.* 9
2 *First semester gantt.* 18
3 *Second semester gantt.* 18
4 *Risks.* 19
5 *Decision Tree.* 20

List of Tables

1 *Faults.* 9
2 *Representativeness faults.* 10
3 *Operators Status and related constraints.* 15
4 *Constraints Status.* 16

Abbreviations

API Application Programming Interface

DDOS Distributed Denial of Service

IaaS Infrastructure-as-a-Service

ODC Orthogonal Defect Classification

PaaS Platform-as-a-Service

SaaS Software-as-a-Service

Abstract

Nowadays, the Information and Communications Technologies is responsible for 2-4% of CO^2 emissions, but in the next five or ten years this will increase to 10% [1]. Because of this, the next challenge is reduce the costs of ICT and its impact in the environment while maintaining growth of IC services.

Cloud computing is a novel paradigm that provides on-demand self-service resources (computing, network and storage) and promises reduce the costs of ICT, but isn't free of external disturbance like security attacks, power surges, workload faults and others.

So, the theme of my dissertation is "Evaluate the robustness of Cloud". I will design and implement a fault injector for software coded in C to evaluate the capacity of cloud to recover from faults.

Keywords: Faults, Errors, Failures, Vulnerabilities, Fault Injection, Fault Tolerance, Security, Robustness.

1 Introduction

In the next subsections will be introduced the context and the scope of this project.

1.1 Contextualization

The present dissertation describes the work developed in scope of MSc in Informatics Engineering. It is focused in "Evaluate the robusteness of Cloud" and this is one issue very important nowadays, because of the increasing usage of these. They are characterized by the placement of data and software on remote infrastructure. Despite of the numerous benefits, the reliability of these platforms hasn't keep the needs, and users trust on their applications to systems outside of personal control.

In this context, naturally arises the problem of confidence in the entity that manages the platform where applications have been executed. Any organization that put an application in the cloud (for example, Microsoft Azure or Amazon EC2) will should to accept the assurances given by the service provider.

This internship deals with the challenge of assessing the robustness of cloud platforms. The computing service provider uses virtualization to manage and allocate computing power to meet actual needs of the application. Although, there are solid virtualization platforms, fault tolerance is still a research problem.

resilience

1.2 The project

This project is based mainly in inject software faults. It was decided since there already are other people involved in the part of hardware faults.

1.3 Objectives

The main objective of this work is to build a tool to inject software faults in code of some programs before the compilation.

But this objective is divided in some other goals:

- Generate derivations of main code of selected programs;
- Verify and analyze the effect of produced faults;
- Compile the programs with injected faults, by using make file.

1.4 Document Structure

In this document are specified all the related subjects with the project.

The second section be present the state-of-the-art in the related areas with particular emphasis to Cloud Computing and Fault Injection.

Third section is an important section of this report, because of the research involved in the execution of this work. It was necessary to take some important decisions based in research results, knowledge and my own experience.

Fourth section describes the work that have been done in Fault Injector, and the work that should be done in the next semester.

Fifth section explain the other modules that need to be executed in this project to can view and evaluate the results of the fault injector.

Finally, in the last section I will do an overview analyses to my work, in general the operators and the constraints developed. I will also talk in the work to be done in the next semester.

1.5 Management

In this section is described the planning of work developed in dissertation.

1.5.1 Meetings

In relation a meetings, the supervisor Raul Barbosa and me agreed that meet weekly was the best option. And the meetings were going on, with one or another change of schedule to reconcile with the other activities from both. In addition, I attended to several general meeting of the project. In that meeting, we could discuss concepts and the direction of the project with colleagues and teachers, among them: Raul Barbosa (supervisor), Henrique Madeira (co-supervisor), João Durães and João André Ferro.

1.5.2 Risks

The main-risks of execution of this project are:

- Equipment Failure
- Data lost
- Publication of similar research
- Personal issues interfere with progress
- Student loses interest

- Dispute between student and supervisor
- Supervisor takes excessive time to check final drafts
- Student wants to submit thesis without supervisor approval

The preventative measures and recovery measures can be seen at Appendix A.2.

1.5.3 Planning and Tracking

In Appendix A.1, is showed the Gantt diagram with the tasks that have been done during the first semester. I'm not showing here the planned Gantt because of I postponed this dissertation six months and the scope and context have changed and now the two Gantt diagrams would be incomparable.

About the development of this project, I have used an *Agile Life Cycle* based in a *Incremental Model*.

porque? ajuda? com que objectivo? foi uma boa opção? quais eram as alternativas? em que falhavam? porque nao foram escolhidas?

What is the requirements of this project???

2 State of the Art

Nowadays, people use a lot of services based in cloud and many of companies choose to use them too. Using it, companies reduce the costs of IT infrastructure and don't need to buy "physical storage" and don't care where are the data. The cloud service provides that the data is secure. But, like as any system, the cloud have problems such as another computer systems, software and hardware faults. Very important is the resilience of the cloud too.

The increased use of cloud is related to a low usage of many dedicated servers, lower voltage levels, reduction of noise margins and increasing clock rates[1]. The cloud providers offers resources ready to deliver[1].

With this work, I pretend to inject software faults and analyze how the system react to them.

There are many studies showing that the software faults[2] it's the main cause of computer failures. About 44% of the software faults cannot be emulated[3].

I have the opportunity to access to the application (executable only, not the source) of Robert Natella, called by SAFE, that inject software faults, as I also want to do (I will describe it in next section).

2.1 Software Implemented Fault Injection of Software Faults

In the next subsections I will describe some fault injectors that have been previously done.

SAFE by Robert Natella

Safe is an application to inject realistic software faults in programs codded in C and C++. This tool uses MCPP as parser, to get the tree of code. The decision of use MCPP instead of GCC parser was a workaround for some of the shortcomings of the GCC's C preprocessor.

After that, write some files, variations of original files (code with simple mutations) with operators applied. Robert Natella implemented thirteen operators in SAFE, same as João Durães[4], but with the difference that Robert sinonimo: implemented at source code level, and João at binary level.

JACA Tool

JACA[5] is a tool taht have been made to validate Java applications. It injects high-level software faults and is based on computational reflection to inject interface faults in Java applications[6]

J-SWFIT

Java Software Fault Injection Tool[7] is a tool that don't need the source code to perform the injection, the mutation of the code is performed directly at bytecode level.

2.2 ODC Model

Orthogonal Defect Classification (ODC)[8] Model is a framework developed by IBM[9], created to improve the level of technology available to assist the decisions of a software engineer, via measurement and analysis. ODC can be used to classifying and analyzing defects during software development.

For that, this model have eight categories:

- **Function** - This defect affects significant capability, end-user features, product Application Programming Interface (API), interface with hardware architecture, or global structure(s). It would require a formal design change.

- **Assignment** - Typically an assignment defect indicates an initialization of control blocks or an data structure.
- **Interface** - Problems in the interaction with other components, modules, device drivers, call statements, control blocks, or parameter lists.
- **Checking** - Based in the program logic that is checked and failed to validate data and values before the usage, loop conditions, etc.
- **Timing/serialization** - Errors that happen in shared and real-time resources.
- **Build/package/merge** - Errors that occur in the integration of library systems, management of changes, or in version control.
- **Documentation** - Errors in the documentation, that con be propagated to publications and maintenance notes.
- **Algorithm** - Problems that can be fixed by reimplementing an algorithm or local data structure, include efficiency or correctness that affect the task.

3 Research objectives and approach method

In this section are discussed the main aspects in study.

3.1 Cloud Computing

Three levels of Cloud Computing Service Models:

- **Infrastructure-as-a-Service (IaaS)** - as the name suggests, provides an computing infrastructure, such as virtual machines, firewalls, load balancers, IP addresses, virtual local area networks and others. Examples: Amazon EC2, Windows Azure.
- **Platform-as-a-Service (PaaS)** - provides an computing platform, normally includes operating system, programming language execution environment, database, web server and others. Examples: AWS Elastic Beanstalk, Windows Azure, Heroku.
- **Software-as-a-Service (SaaS)** - provides access to application softwares ofter referred as *on-demand self-service* sofwares. Use it without install, setup and run the application. Service provider do all things for you. Google Apps, Microsoft Office 365.

The cloud computing isn't free of external disturbances[1], the most important are:

- Security attacks;
- Accidents;
- Power surges;
- Workload faults;
- Malfunction;
- Worms;
- Distributed Denial of Service (DDOS) attacks.

3.2 Tools - GCC Parser, Bison and Eclipse CDT

In the beginning of planning the basic software without any user interface, it was necessary to research the best applications, as the best way for using them to obtain panned results (fault injector). For that, I thought that I could use the same tools that I have used in Compilers course, Lex and Yacc.

For parsing the code, analyze and modify it,

In the end, I selected Eclipse CDT Plugin as standalone (only import libraries to project), because of my habilities in programming in Java Language, the maintainability of software, the low learning level than the developers need to modify it.

GCC Parser

Nowadays, GCC use a hand-written parser to improve syntactic error diagnostics, giving human meaningful messages on syntax errors.

Eclipse CDT

Eclipse CDT, as the name suggests, is an plugin for Eclipse that provides a fully functional C and C++ Integrated Development Environment. Some of the features included in this plugin that are interesting for this project are:

- Source navigation;
- Code editor with syntax highlighting;
- Source code refactoring and code generation.

Is possible to use this plugin in standalone mode, importing .jar files to the project. Using it I can code Fault Injector in Java, making the software more maintainable and easy to use, write, compile and debug.

Problems with the rewriting of tree

Reflection

But I was forced to take decisions after that, for example, after create the tree of code, I can go through the tree in the recursive way or using *Visitor Pattern.*

Performance analyses

4 Fault Injector Development

The Fault Injector currently in development is coded in Java and

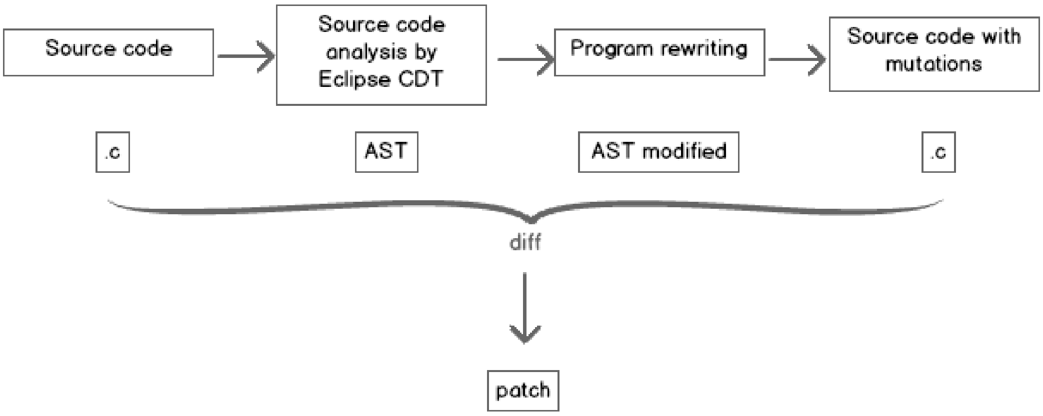

Figure 1: *Overview of the injection tool.*

Fault Type	Description
MFC	Missing function call
MVIV	Missing variable initialization using a value
MVAV	Missing variable assignment using a value
MVAE	Missing variable assignment with an expression
MIA	Missing IF construct around statements
MIFS	Missing IF construct + statements
MIEB	Missing IF construct + statements + ELSE construct
MLAC	Missing AND in expression used as branch condition
MLOC	Missing OR in expression used as branch condition
MLPA	Missing small and localized part of the algorithm
WVAV	Wrong value assigned to variable
WPFV	Wrong variable used in parameter of function call
WAEP	Wrong arithmetic expression in function call parameter
	add another faults

Table 1: *Faults.*

4.1 Generate derivations

I chose to use the most representative faults[4], divided into missing, wrong and extraneous, specified individually further down:

Fault nature	Fault specific types	# Faults	ODC types ASG	CHK	INT	ALG	FUN
Missing	*if* construct plus statements (MIFS)	71				✓	
	AND sub-expr in expression used as branch condition (MLAC)	47		✓			
	function call (MFC)	46				✓	
	if construct around statements (MIA)	34		✓			
	OR sub-expr in expression used as branch condition (MLOC)	32		✓			
	small and localized part of the algorithm (MLPA)	23				✓	
	variable assignment using an expression (MVAE)	21	✓				
	functionality (MFCT)	21					✓
	variable assignment using a value (MVAV)	20	✓				
	if construct plus statements plus *else* before statements (MIEB)	18				✓	
	variable initialization (MVIV)	15	✓				
Wrong	logical expression used as branch condition (WLEC)	22		✓			
	algorithm - large modifications (WALL)	20					✓
	value assigned to variable (WVAV)	16	✓				
	arithmetic expression in parameter of function call (WAEP)	14			✓		
	data types or conversion used (WSUT)	12	✓				
	variable used in parameter of function call (WPFV)	11			✓		
Extraneous	variable assignment using another variable (EVAV)	9	✓				
Total faults for these types in each ODC type		452	93	135	25	192	41
Coverage relative to each ODC type (%)		68	65	81	51	72	100

Table 2: *Representativeness faults.*

Table of more representative faults of Durães

4.1.1 Fault Types - Missing

- **MIFS** - if construct plus statements

  This operator is based in the remotion of one conditional if. To do that, I need to verify the constraints c02, c08 and c09.

- **MLAC** - AND sub-expression in expression used as branch condition
- **MFC** - function call
- **MIA** - if construct around statements
- **MLOC** - OR sub-expression in expression used as branch condition
- **MLPA** - small and localized part of the algorithm

- **MVAE** - variable assignment using an expression
- **MFCT** - functionality
- **MVAV** - variable assignment using an value
- **MIEB** - if construct plus statements plus else before statements
- **MVIV** - variable initialization

4.1.2 Fault Types - Wrong

- **WLEC** - logical expression used as branch condition
- **WALL** - algorithm - large modifications
- **WVAV** - value assigned to variable
- **WAEP** - arithmetic expression in parameter of function call
- **WSUT** - data types or conversion used
- **WPFV** - variable used in parameter of function call

4.1.3 Fault Types - Extraneous

- **EVAV** - variable assignment using another variable

4.2 Constraints

The constraints defined below was specified by João Durães in ... .

Constraints	**Description**
C01	Return value of the function **must not** being used
C02	Call **must not be** the only statement in the block
C03	Variable **must be** inside stack frame
C04	**Must be** the first assignment for that variable in the module
C05	Assignment **must not be** inside a loop
C06	Assignment **must not be** part of a for construct
C07	**Must not be** the first assignment for that variable in the module
C08	The if construct **must not be** associated to an else construct
C09	Statements **must not include more than** five statements and not include loops
C10	Statements are in the same block, **do not include more than** 5 stats. not loops
C11	There **must be** at least two variables in this module

4.3 Applications to inject faults

The same applications that João Durães have collect information?

- MinGW, Last Update: 2015-06-08
- ScummVM, Last Update: 2015-05-17
- CDEX, Last Update: 2015-04-24
- FireBird, Last Update: 2015-04-15
- Joe, Last Update: 2015-03-22
- FreeCiv, Last Update: 2015-03-14
- GAIM or Pidgin, Last Update: 2015-01-07
- BASH, Last Update: 2013-12-10
- ZSNES, Last Update: 2013-05-07
- VIM, Last Update: 2013-04-25
- pdftohtml, Last Update: 2013-04-24

5 Work plan and implications

Built three separated modules:

- Generate the derivations of main code of selected programs;
- Verify and analyze the effect of produced faults;
- Compile the programs with injected faults, by using make file.

5.1 Analyze the effects

The fault injected results is equal to the real software faults?

5.2 Compile programs

Select five to ten programs to be tested.
Justificar a utilização de patchs

After the compilation and execution of the programs, the results need to be evaluate. For measure that, I will use the *CRASH Scale*[10]:

- **C**atastrophic - Operating System crashed or multiple tasks affected;
- **R**estart - Task or process hangs, requiring restart;
- **A**bort - Task or process aborts abnormally (i.e. "code dump" or "segmentation violation");
- **S**ilent - Test Process exits without error code returned when one should be;
- **H**indering - Test Process exits with an error code not relevant to the situation or incorrect error code returned;
- **P**ass - The module exits properly, possibly with an appropriate error core.

This *CRASH Scale* is one way to show results of the effect of faults on an end-use system.

6 Conclusion

6.1 Global Vision

In table 3, it's possible to understand the operators that was implemented in the first semestre of this dissertation. As can be seen, I have implemented five of thirteen operators that João Durães was especified.

In table 4, is also possible to check that I have implemented three of eleven constraints related to the thirteen operators.

MISSING	MIFS	Missing IF construct and surrounded Statements	C02	C08	C09		
	MLAC	Missing "and sub-expression" in logical expression used in branch condition	C12				
	MFC	**Missing function call**	C01	C02			
	MIA	Missing IF Around statements	C08	C09			
	MLOC	Missing "or sub-expression" in logical expression used in branch condition	C12				
	MLPA	Missing Localized Part of the Algorithm	C02	C10			
	MVAE	Missing Variable Assignment with an Expression	C02	C03	C07	C06	
	~~MFCT~~						
	MVAV	Missing Variable Assignment with a Value	C02	C03	C07	C06	
	MIEB	Missing IF construct plus statements plus else before statements	C08n				
	MVIV	Missing Variable initialization with a value	C02	C03	C04	C05	C06
WRONG	~~WLEC~~						
	~~WALL~~						
	WVAV	Wrong Value Assigned to a Variable	C03	C04	C06		
	WAEP	Wrong Arithmetic Expression in a function Parameter					
	~~WSUT~~						
	WPFV	Wrong Variable in parameter of function Call	C03	C11			
Extraneous	~~EVAV~~						

Implementado
Em vista
Em falta

Table 3: *Operators Status and related constraints.*

6.2 Future Work

In the future, I have planned to implement the other operators and constraints. And apply this software in testing of open source softwares that I will select.

I will use regression testing to verify if when I coded one new operator or constraint I don't screwed the operators and constraints previous implemented. From version to version I use a regression testing to test the fault injector to guaranty that application don't regraded.

"The purpose of regression testing is to ensure that changes made to software, such as adding new features or modifying existing features, have not adversely affected features of the software that should not change. Regres- sion testing is usually performed by running some, or all, of the test cases created to test modifications in previous versions of the software."

Current	C01	Return value of the function **must not** being used
	C02	Call **must not be** the only statement in the block
	C03	Variable **must be** inside stack frame
	C04	**Must be** the first assignment for that variable in the module
	C05	Assignment **must not be** inside a loop
	C06	Assignment **must not be** part of a for construct
	C07	**Must not be** the first assignment for that variable in the module
	C08	The if construct **must not be** associated to an else construct
	C09	Statements **must not include** more than five statemens and not include loops
	C10	Statements are in the same block, **do not include** more than 5 stats. or loops
	C11	There **must be** at leat two variables in this module

			Operators		Versions
Extra	C08n	The if construct **must be** associated to an else construct	MIEB		a
	C12	Must have **at least two** branch conditions	MLAC	MLOC	b
					c
					d
					e
					f
					g
					h

Implementado
Em vista

Table 4: *Constraints Status.*

Regression Testing
System testing
Unit tests

A Appendix

A.1 Appendix A - Gantt diagrams

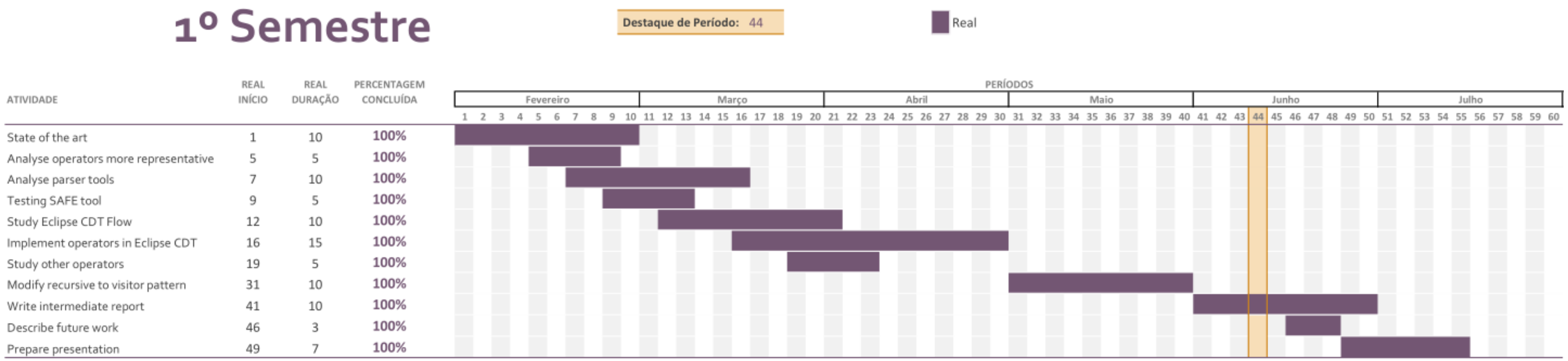

Figure 2: *First semester gantt.*

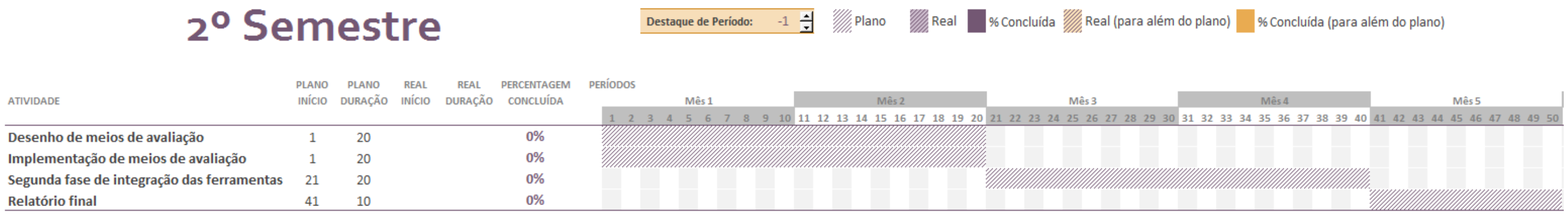

Figure 3: *Second semester gantt.*

A.2 Appendix B - Risks table

Risc Area	Preventative Measures	Recovery Measures
Equipment Failure	Ensure regular maintenance is undertaken	Use alternative sources/type of equipment as appropriate
	Allow for sufficient funding for repairs	
	Indentify alternative sources/type of equipment	
Data lost	Back-up data regularly	
Publication of similar research	Regularly search electronic publications databases	Modify project
	Continue literature review throughout candidature	
	Ensure timely submission	
Personal issues interfere with progress	Take leave of absence (unless for sickness or bereavement)	Re-apply for admission when able to commit
	Take annual leave	
	Take sick leave	
	Communicate with supervisor	
Student loses interest	Select motivating topic at the start	
	Enrolling area ensures a dynamic research culture	
	Improve communication between student and supervisor	
	Look for warning signs	
	Register for support programs/seminars	
	Talk to fellow students in research area	
Dispute between student and supervisor	Understand each other's roles and expectations	
	Agree on dispute resolution process when initiating relationship	
Supervisor takes excessive time to check final drafts	Supervisor to plan out workload	
	Student plan ahead to ensure supervisor will be available	
	Student/Supervisor to review chapters/sections at regular intervals	
Student wants to submit thesis without supervisor approval	Student to be counselled regarding implications - a recomendation of fail or major revision from examiners likely if thesis below standard	Review of thesis by alternative person within University recommended

Figure 4: *Risks.*

A.3 Appendix C - Decision Tree

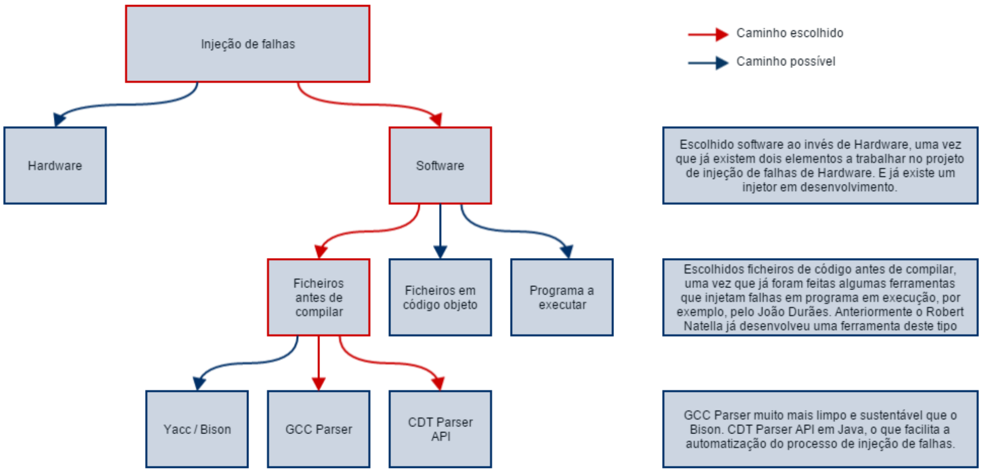

Figure 5: *Decision Tree.*

References

[1] K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, *Resilience assessment and evaluation of computing systems.* Springer, 2012.

[2] A. Avizzienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing."

[3] H. Madeira, D. Costa, and M. Vieira, "On the emulation of software faults by software fault injection," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on.* IEEE, 2000, pp. 417–426.

[4] J. A. Duraes and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *Software Engineering, IEEE Transactions on*, vol. 32, no. 11, pp. 849–867, 2006.

[5] L. Regina, E. Martins *et al.*, "Jaca—a software fault injection tool," in *null.* IEEE, 2003, p. 667.

[6] E. Martins, C. M. Rubira, and N. G. Leme, "Jaca: A reflective fault injection tool based on patterns," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on.* IEEE, 2002, pp. 483–487.

[7] B. P. Sanches, T. Basso, and R. Moraes, "J-swfit: A java software fault injection tool," in *Dependable Computing (LADC), 2011 5th Latin-American Symposium on.* IEEE, 2011, pp. 106–115.

[8] N. Bridge and C. Miller, "Orthogonal defect classification using defect data to improve software development," *Software Quality*, vol. 3, no. 1, pp. 1–8, 1998.

[9] R. Chillarege, *Orthogonal Defect Classification.* Handbook of Software Reliability Engineering, ed. Michael R. Lyu (Los Alamitos, CA: IEEE Computer Science Press, 2004.

[10] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, "Comparing operating systems using robustness benchmarks," in *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on.* IEEE, 1997, pp. 72–79.