UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Realtime data collection in IDEs

by

David Chenaux

Thesis for the Master of Science in Computer Science
Supervised by Prof. Dr. Philippe Cudré-Mauroux

eXascale Infolab
Department of Informatics - Faculty of Science - University of Fribourg

January 12, 2017

UNIVERSITY OF FRIBOURG
Faculty of Science
Department of Informatics
eXascale Infolab

Thesis for the Master of Science in Computer Science
Supervised by Prof. Dr. Philippe Cudré-Mauroux

by David Chenaux

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

Abstract 2

Acknowledgements 3

List of Figures 6

List of Tables 7

1 Introduction 8
- 1.1 Problem definition . . . 8
- 1.2 Goals and objectives . . . 9
- 1.3 Organization . . . 9

2 Related work 10
- 2.1 What is Program Analysis ? . . . 10
- 2.2 Program Analysis approaches . . . 12
  - 2.2.1 Static methods . . . 12
  - 2.2.2 Dynamic methods . . . 13
- 2.3 Program Analysis tools . . . 14
  - 2.3.1 Static Analysis tools . . . 14
  - 2.3.2 Dynamic Analysis tools . . . 15
- 2.4 Dynamic Analysis limitations . . . 16

3 Development 17
- 3.1 Proposed solution . . . 17
- 3.2 Environement . . . 17
- 3.3 Data capture model . . . 17
- 3.4 Data model . . . 18
- 3.5 User interface . . . 18

4 Installation guide 19
- 4.1 Setup the environment . . . 19
- 4.2 Use the packaged version . . . 19
- 4.3 From source code . . . 19

5 Experiments **20**
5.1 Test script and machine 20
5.2 Data extraction times 20
5.3 Writing times to database 20
5.4 Reading times from database 21

6 Conclusion **22**
6.1 Conclusion 22
6.2 Future work 22

A Glossary **23**

B License of the software **24**

Bibliography **25**

List of Figures

List of Tables

2.1 Comparison of Dynamic analysis with Static Analysis 11
2.2 Dynamic Analysis Tools 15
2.3 Dynamic Analysis Techniques comparaison 16

Chapter 1

Introduction

Integrated development environments have been around for a few decades already, yet none of the modern IDEs was able to successfully integrate their source code editors with the actual data stream flowing though the code. Ability to display the actual data running through the system promises many potential benefits, including easier debugging and code recall, which results in significantly lower code maintenance costs.

1.1 Problem definition

Every developer is more or less feared about the debugging and code reviewing phase of their software. Obviously, this process can sometimes take several painfully hours and each programmer knows how frustrating it can be to search for a hidden bug in thousands lines of codes. In order to support the programmers in this hated task, debuggers are the most useful existing tools which are part of the so called *static program analysis.*

With the apparition of object-oriented programming language, searching for syntactic errors in the code is not anymore sufficient. Therefore, a new research field was pushed forward which is called the *dynamic program analysis* and consists in analyzing the software during it's execution. This procedure allows to take in account some possible inputs which weren't probed with the SPA. Yet none of the modern IDEs was able to successfully integrate their source code editors with the actual data stream flowing though the code. This is why the present project, which goals and objects are defined in the next section, is aiming to contribute to the subject.

1.2 Goals and objectives

The goal of this project is to design a proof-of-concept system in one programming language that allows full code instrumentation. This system should be able to seamlessly capture all values for all variables in source code and store them somewhere, with further possibility to easily retrieve saved values. The system should also provide an API to the storage in order to make the data accessible for navigation and display in third-party applications. Also, a basic visualizing interface will also be included in order to allow an easy review of the results. Finally, an evaluation of system's performances will be established through different experiments.

1.3 Organization

The thesis is divided in four main sections :

1. **Related work**: In this first chapter of the thesis, an insight of the existing work on the field *program analysis* will be presented and in particular the DPA. This is including a definition of the field and its particularities, an overview of some available solutions side by side with the current restrictions.

2. **Development**: This part is focusing on the development of the proof-to-concept system with a presentation of the proposed solution and detailed information about its structure.

3. **Installation guide**: Simply an installation guide of the software which describes the needed environment, the package installation and the compilation of the system.

4. **Experiments**: Finally in this section a few experiments will be conducted in order to test and check the performance and results of the software.

The thesis concludes with some outputs and is proposing some future improvements which seem to be important.

Chapter 2

Related work

"Sharing is good, and with digital technology, sharing is easy."
Richard Stallman

As stated in the previous chapter, the goal of this thesis is to implement a dynamical program analysis system. In order to build a theoretical background, program analysis will be defined in this chapter along with the presentation of some available solutions and their restrictions.

2.1 What is Program Analysis ?

Programming environments are an essential key for the acceptance and success of a programming language. After Ducassé and Noyé [1994], without the appropriate developments and maintenance tools, programmers are likely to have a bad software understanding and therefore produce low-quality code. They will be therefore reluctant to use a language without appropriate programming environments, however powerful the programming language is.

As already introduced in the previous chapter, program analysis is an automated process which aims to analyze the behavior of a software regarding a property such as correctness, robustness, safety and liveness. Program analysis can be separated in two methods : the SPA which is performed without running the software and the DPA which is obviously fulfilled during runtime. [Wikipedia, 2016]

The SPA is a really simple solution because it does not require running the program for analyzing the dynamic behavior of a program. It consists in going through the source code and highlight coding errors or ensure conformance to coding guidelines. A classic example of static analysis would be a compiler which is capable of finding lexical, syntactic and even semantic mistakes. The main advantage of this method is that it allows to reason about all possible executions of a program and gives assurance about any execution, prior to deployment.

Nevertheless, according to Gosain and Sharma [2015], since the widespread use of object oriented languages, SPA is found to be ineffective. This can be explained because of the usage of run-time features like dynamic binding, polymorphism, threads etc. To remedy this situation, developers call on DPA which can, after Marek et al. [2015], gain insight into the dynamics and runtime behavior of those systems during execution. Because the runtime behavior depends on many other factors, such as program inputs, concurrency, scheduling decisions, and availability of resources, static analysis is not capable of retrieving those values. The following table, proposed by Gosain and Sharma [2015], is resuming the main differences between static and dynamic analysis.

Dynamic Analysis	Static Analysis
Requires program to be executed	Does not require program to be executed
More precise	Less precise
Holds for a particular execution	Holds for all the executions
Best suited to handle run-time programming lan guage features like polymorphism, dynamic bind ing, threads etc.	Lacks in handling run-time programming lan- guage features.
Incurs large run-time overheads	Incurs less overheads

Table 2.1: Comparison of Dynamic analysis with Static Analysis

A relevant point which comes out of this comparison, is that Dynamic Program Analysis is not replacing the Static Analysis, but on the contrary it is a complementary tool. Indeed, even if Static Program Analysis is not sufficient anymore, it gives nevertheless important information about the code for the programmer. The DPA is coming in a second phase when the SPA has been processed and the errors corrected. As it can be deduced, the main advantage of DPA is that it can examine the actual and exact run-time behavior of the program, whereas SPA main advantage is that it does not depend on input stimuli and can be generalized for all executions. To illustrate these differences, some program analysis solution will be presented further in this chapter.

2.2 Program Analysis approaches

Now that a definition of Program Analysis has been established, some different approaches have to be exposed in order to fully understand the subject. Since the field is really vast, it is not the aim to cover the entire subject, but the reading of this section should give a good overview to the reader. First, the main static analysis methods will be steered following logically with the dynamic analysis methods.

2.2.1 Static methods

The static methods are regrouped in four different categories proposed by Nielson et al. [2004] and briefly presented here, some information was also gathered from the Wikipedia [2016] page which is proposing a grouping based on the same criteria.

Data Flow Analysis: is a technique which consist in gathering information about the values and their evolution at each point of the program. In the Data Flow Analysis the program is considered as a graph in which the nodes are the elementary blocks and the edges describe how control might pass from one elementary block to another.

Constrained Based Analysis: or Control Flow Analysis, aims to know which functions can be called at various points during the execution ; what "elementary blocks" may lead to what other "elementary blocks".

Abstract Interpretation: consists in proving that the program semantics satisfies its specification according to Cousot [2008]. What the program executions actually do should satisfying ,what the program executions are supposed to do. It can be explained as a partial execution of a program which gather information about its semantics without performing all the calculations.

Type and Effect Systems: are two similar techniques. The first one is using types, which are a concise, formal description of the behavior of a program fragment. Rémy [2017] explains that programs must behave as prescribed by their types. Hence, types must be checked and ill-typed programs must be rejected. Effect systems can be described, after Nielson and Nielson [1999] as an extension of annotated type system where the typing judgments take the form of a combination of a type and an effect. This combination is associated with a program relative to a type environment.

2.2.2 Dynamic methods

Now that the main static analysis methods have been defined in the preceding section, the dynamic methods will be exposed here. As it was already stated, dynamic analysis is a quite recent research field which status could be still defined as academical. Therefore the different techniques are not as well established as for the static analysis and can vary a lot in accordance with the author of the different papers. For this work, the following different method were selected which are proposed by Gosain and Sharma [2015] in their survey of Dynamic Program Analysis Techniques and Tools.

Instrumentation based approach: needs a code instrumenter used as a pre-processor in order to inject instrumentation code into the target program. This can be done at three different stages : source code, binary code and bytecode. The first stage adds instrumentation code before the program is compiled, the second one adds it by modifying or re-writing compiled code and the last one performs tracing within the compiled code.

VM Profiling based technique: uses the profiling and debugging mechanism provided by the particular virtual machine, for example the JPDA for Java SDK or the PDB for Python. These profilers give an insight into the inner operations of a program, especially the memory and heap usage. To capture these profiling information plug-ins are available and can access the profiling services of the VM. Benchmarks are then used for actual run-time analysis which acts like a block-box test for a program. This process involves executing or simulating the behavior of the program while collecting data which is reflecting the performance. Unfortunately this technique has the drawback of generating high run-time overheads.

Aspect Oriented Programming: aims to increase modularity by allowing the separation of cross-cutting concerns. Because there is no need to add instrumentation code as the instrumentation facility is integrated within the programming language, the additional behavior is added to existing code without modifying the code itself. AOP adds the following constructs to a program : aspects, join-point, point-cuts and advices. These constructs can be considered like classes. Most popular languages have their aspect oriented extensions like AspectC++ and AspectJ. In python, there are some libraries who aims to reproduce AOP behavior but there isn't any canonical one. Actually there is a debate to what extent aspect oriented practices are useful or applicable to Python's dynamic nature.

2.3 Program Analysis tools

This section is dedicated to the available solutions in terms of program analysis. As it will be explained in the next chapter, the proof-to-concept system will be coded in *Python* and therefore an additional information will be given for solutions available in this language. As already exposed in this order, first, some Static Analysis solutions will be presented following with the dynamic method ones.

2.3.1 Static Analysis tools

Following, some of the most popular tools (commercial or free) for SPA are described, selected in widespread languages : Java, C/C++ and Python. The description are based on the official website of the tools and also on the Gomes et al. [2009] paper.

Starting with C/C++, **Splint** is a very well known tool, allowing to check for security vulnerabilities and coding mystakes. Splint is based on Lint and tries to minimize the efforts needed for its deployment. Additionally, with some annotation, Splint can extend its performances over Lint. Splint can among others detect : Dereferencing a possibly null pointer, Memory management errors including uses of dangling references and memory leaks, Problematic control flow such as likely infinite loops. **Astrée**, where as it is based on abstract interpretation, is analyzing safety-critical applications written or generated in C. It proves the absence of run-time errors and invalid concurrent behavior for embedded applications as found in aeronautics, earth transportation, medical instrumentation, nuclear energy, and space flight. Another worth mentioning tool is the **PolySpace Verifer** tool developped by MathWorks who also created the famour Matlab software.

Concerning Java, one recognized tool is Findbugs. With the advantage of being a Libre software, the tools uses a series of ad-hoc techniques designed to balance precision, efficiency and usability. FindBugs operates on Java bytecode, rather than source code. Another Libre software is **Checkstyle** which, as his names indicates it, allows to report any breach of standards in the source code. Finally a commercial tool, **Jtest** which is an integrated Development Testing solution, can perform Data-flow analysis Unit test-case generation and execution, static analysis, regression testing, runtime error detection, code review, and design by contract.

In the Python world, **Pylint** is a coding standard checker which follows the style recommended by the PEP 8 specification. It is also capable of detecting coding errors and is integrable in IDEs. Speaking of IDEs, **PyCharm** includes also static analysis functions like PEP8 checks, testing assistance, smart refactorings, and a host of inspections.

2.3.2 Dynamic Analysis tools

As for the static tools, the most popular DPA tools are presented here. Following, a table proposed by Gosain and Sharma [2015] with an summary of some available DPA tools regrouped by technique. The table indicates the concerned language and also which type of dynamic Analysis is done by the tool.

Technique	Tool	Language	Type of Dynamic Analysis done								
			Cache Modelling	Heap Allocation	Buffer Overflow	Memory Leak	Deadlock Detection	Race Detection	Object LifeTime	Metric Computation	Invariant Detection
Instr.Based	Daikon	C,C++									✓
	Valgrind	C,C++				✓		✓			
	Rational Purify	C, C++, Java				✓					
	Parasoft Insure++	C,C++		✓		✓					
	Pin	C	✓								
	Javana	Java	✓						✓		
	DIDUCE	Java									✓
AOP Based	DJProf	Java		✓					✓		
	Racer	Java						✓			
VM Profiling Based	Caffeine	Java							✓		
	DynaMetrics	Java								✓	
	*J	Java								✓	
	JInsight	Java				✓	✓		✓		

TABLE 2.2: Dynamic Analysis Tools

Valgrind, Purify and Insure++ are instrumentation based and can automatically detect memory management and threading bugs among with profiling a program in details. While Valgrind is a instrumentation framework for building dynamic analysis tools, the two others are fully-fledged analysis software. **Javana** comes with an easy-to-use instrumentation framework so that only a few lines of instrumentation code need to be programmed for building powerful profiling tools. **Daikon** and **Diduce** are the most known tools for invariant detection and are respectively an offline and online tool. Last but not least, **Pin** is a dynamic binary instrumentation framework developed by Intel. It enables the creation of dynamic program analysis tools and can be used to observe low level events like memory references, instruction execution, and control flow as well as higher level abstractions such as procedure invocations, shared library loading, thread creation and system call execution.

For AOP based tool, the two selected programs are **DjProf** and **Racer**. The first one is a profiler used for the analysis of heap usage and object life-time analysis and the second one is a data race detector tool for concurrent programs.

***J** and **DynaMetrics** are two academical research projects about Virtual Machine profiling and are proposing solution for computing dynamic metrics for Java. The first one, proposed by Dufour et al. [2003], relies on JVMPI, while the second solution, from Singh [2013], relies on the new JVMTI. **JInsight** is for exploring run-time behaviour of Java programs visually and **Caffeine** helps to check conjectures about Java programs.

In addition to this table, some Python tools are also available even if the field seems to not to be really well developed for this programming language. That could be explainable because of the dynamic nature of the language. This might be why the following tools are developed *in* Python but not *for* it. The first tool is **Angr** which is a python framework for analyzing binaries. It focuses on both static and dynamic instrumentation analysis, making it applicable to a variety of tasks. **Triton** is another binaries analyzer framework and proposes python bindings. Its main components are Dynamic Symbolic Execution engine, a Taint Engine, AST representations of the x86 and the x86-64 instructions set semantics, SMT simplification passes, an SMT Solver Interface

2.4 Dynamic Analysis limitations

As the DPA is a quite new research field, it induce ineluctably some drawbacks and limitations. The following table created by Gosain and Sharma [2015] gives a good overview of the different techniques and some drawbacks.

	Instrumentation		VM Profiling	AOP
	Static	Dynamic		
Level of Abstraction	Instruction/Bytecode	Instruction/Bytecode	Bytecode	Programming Language
Overhead	Runtime	Runtime	Runtime	Design and deployment
Implementation Complexity	Comparatively low	High	High	Low
User Expertise	Low	High	Low	High
Re-compilation	Required	Not Required	Not Required	Required

TABLE 2.3: Dynamic Analysis Techniques comparaison

Chapter 3

Development

"For me, open source is a moral thing."
Matt Mullenweg

In this chapter, the proposed solution will be explained in details including the Setup, Data capture model, Data model and it's User interface.

3.1 Proposed solution

start with the detailed description of what we actually propose

3.2 Environement

The project is articulated around 4 main technologies. First the data is captured in *Python* with the help of the integrated Debugger Framework. Then the extrated data is stored in a *MongoDB* Database. Finally they are processed and showed with the help of *Python*, *Html/CSS* and *Javascript*. Each module of the solution is presentend in details in the following sections.

Speak about the server, jenkins, linux..

3.3 Data capture model

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.4 Data model

how you store data in a database

3.5 User interface

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 4

Installation guide

"If Microsoft ever does applications for Linux it means I've won."
Linus Torvalds

In this chapter, the complete installation process of the developed script will be presented.

4.1 Setup the environment

First format windows and install linux because you are not a looser

4.2 Use the packaged version

In order to simplify the installation process, a packaged version has been built and is ready to download on the project's GitHub page.

The installation process is really straightforward and since it's a pip package.

4.3 From source code

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 5

Experiments

In this section different type of experients will be conducted in order to test and check the performance of the developped software. In order to test the following variables the same script was used for all experiments.

5.1 Test script and machine

In order to conduct the different experiments, a test script has been chosen.

Lenovo Thinkpad T460p CPU : Intel Core i7-6700HQ @ 2.60GHz x 8 OS : Fedora 25 64bits GPU : Intel HD Graphics 530 RAM : 15.1Gio

5.2 Data extraction times

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5.3 Writing times to database

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5.4 Reading times from database

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 6

Conclusion

6.1 Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

6.2 Future work

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Appendix A

Glossary

AOP Aspect Oriented Programming

AST Abstract Syntax Tree

DPA Dynamic Program Analysis

IDE Integrated development environments

JPDA Java Platform Debugger Architecture

JVMPI Java Virtual Machine Profiling Interface

JVMTI Java Virtual Machine Tools Interface

Libre or Free software, is distributed under terms that allow users to run the software for any purpose as well as to study, change, and distribute the software and any adapted versions.

PDB The Python Debugger

pip Pip Installs Packages is a package management system used to install and manage software packages written in Python

SDK Software Development Kit

SMT Satisfiability Modulo Theories

SPA Static Program Analysis

VM Virtual Machine

Appendix B

License of the software

Copyright (c) 2016 DAVID CHENAUX

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Bibliography

Patrick Cousot. Abstract interpretation, August 2008. URL https://www.di.ens.fr/~cousot/AI/. [Online; accessed 9-January-2017].

Mireille Ducassé and Jacques Noyé. Logic programming environments: Dynamic program analysis and debugging. *The Journal of Logic Programming*, 19:351–384, 1994.

Bruno Dufour, Laurie Hendren, and Clark Verbrugge. * j: a tool for dynamic analysis of java programs. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 306–307. ACM, 2003.

Ivo Gomes, Pedro Morgado, Tiago Gomes, and Rodrigo Moreira. An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*, 2009.

Anjana Gosain and Ganga Sharma. *A Survey of Dynamic Program Analysis Techniques and Tools*, pages 113–122. Springer International Publishing, Cham, 2015. ISBN 978-3-319-11933-5. doi: 10.1007/978-3-319-11933-5_13. URL http://dx.doi.org/10.1007/978-3-319-11933-5_13.

Lukáš Marek, Yudi Zheng, Danilo Ansaloni, Lubomír Bulej, Aibek Sarimbekov, Walter Binder, and Petr Tůma. Introduction to dynamic program analysis with disl. *Science of Computer Programming*, 98, Part 1:100 – 115, 2015. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2014.01.003. URL http://www.sciencedirect.com/science/article/pii/S0167642314000070. Fifth issue of Experimental Software and Toolkits (EST): A special issue on Academics Modelling with Eclipse (ACME2012).

F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer Berlin Heidelberg, 2004. ISBN 9783540654100.

Flemming Nielson and Hanne Riis Nielson. *Correct System Design: Recent Insight and Advances*, chapter Type and Effect Systems, page 114–136. Springer International Publishing, 1999.

Didier Rémy. Type systems for programming languages, January 2017.

Paramvir Singh. Design and validation of dynamic metrics for object-oriented software systems. 2013.

Wikipedia. Program analysis — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Program_analysis&oldid=732080552. [Online; accessed 7-January-2017].