Towards Controlling Software Architecture Erosion Through Runtime Conformance Monitoring

Lakshitha de Silva

This thesis is submitted in partial fulfilment for the degree of

Doctor of Philosophy

at the University of St Andrews

May 2014

Abstract

Abstract goes here.

Acknowledgements

Acknowledgements go here.

Declaration

Candidate's Declarations

I, <author>, hereby certify that this thesis, which is approximately <word-count> words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student and as a candidate for the degree of Doctor of Philosophy in September, 2009; the higher study for which this is a record was carried out in the University of St Andrews between 2009 and 2013.

Date:

Signature of candidate:

Supervisor's Declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date:

Signature of supervisor:

Permission for Electronic Publication

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the electronic publication of this thesis:

> *Access to printed copy and electronic publication of thesis through the University of St Andrews.*

Date:

Signature of candidate:

Signature of supervisor:

Dedication goes here.

Contents

Contents i

List of Figures iii

List of Tables iv

Listings v

1 Introduction 1
1.1 Overview 1
1.2 Central claims 1
1.3 Motivation 1
1.4 Main contributions 1
1.5 Organisation of Dissertation 1
1.6 Examples 1
1.6.1 Citing References 1
1.6.2 Tables 2
1.6.3 Lists 3
1.6.4 Figures 3
1.6.5 Code Listings 4
1.6.6 Console Output 5
1.6.7 Defining accronyms 5

2 Context Survey 7
2.1 Introduction 7
2.1.1 Terminology 7
2.1.2 Other surveys 7
2.2 Classification 7
2.3 Discussion 7
2.4 Conclusions 7

3 Chapter 3 Title 9
3.1 Overview 9
3.2 Section-2 9
3.3 Section-3 9
3.4 Conclusions 9

4 Chapter 4 Title **11**
4.1 Overview 11
4.2 Section-2 11
4.3 Section-3 11
4.4 Conclusions 11

Appendix A Appendix-A Title **13**

Appendix B Appendix-B Title **15**

References **17**

LIST OF FIGURES

1.1 The event-driven process used for tapping implementation data 3

LIST OF TABLES

1.1 Controlling architecture erosion with process-oriented strategies 2
1.2 A comparison of performance impact between the use of JDI and instrumentation probes. 2

LISTINGS

1.1 Hello World in Java 4
1.2 Java class transformer 4

CHAPTER ONE

INTRODUCTION

<Chapter Abstract>

1.1 Overview

1.2 Central claims

1.3 Motivation

1.4 Main contributions

1.5 Organisation of Dissertation

A short description of each chapter in this thesis is given below.

Chapter 1 gives an introduction to this dissertation .. bla bla bla

Chapter 2 presents the background survey .. bla bla bla

1.6 Examples

1.6.1 Citing References

References are used in the following paragraph.

Architecture erosion and its effects are widely discussed in literature. Perry and Wolf [Perry and Wolf, 1992] differentiate *architecture erosion* from *architecture drift* as follows: erosion results from violating architectural principles while drift is caused by insensitivity to the architecture. As the underlying causes for both are the same, we will not consider this difference for the purpose of our survey. Additionally, the notion of software architecture erosion is discussed using a number of different terms such as architectural degeneration [Hochstein and Lindvall, 2005], software erosion [Dalgarno, 2009], design erosion [van Gurp and Bosch, 2002], architectural decay [Riaz et al., 2009], design decay [Izurieta and Bieman, 2007], code decay [Eick et al., 2001, Stringfellow et al., 2006] and software entropy [Jacobson, 1992].

1.6.2 Tables

Tables have been configured to use the *booktabs* package which gives professionally typeset tables. Two examples are shown below.

Strategy		Contribution towards controlling erosion
Architecture Design Documentation	►	Records architecture design and rationale with the intent of disseminating architectural knowledge to a wider audience and provides a point of reference for developers throughout system evolution.
Architecture Analysis	►	Uncovers weaknesses in the intended architecture, in particular, sensitive points which can be easily violated in the implementation.
Architecture Conformance Monitoring	►	Establishes the means to verify whether the implementation is faithful to the intended architecture during both the development and subsequent maintenance phases of a system.

Table 1.1: Controlling architecture erosion with process-oriented strategies

Run	Framework unplugged (μs)	Using probes (μs)	Using JDI (μs)
1	108,644	143,002	488,018
2	107,929	141,185	486,951
3	107,319	141,274	482,206
4	106,054	142,333	479,477
Average	107,487	141,949 (+30.9%)	484,163 (+345.6%)

Table 1.2: A comparison of performance impact between the use of JDI and instrumentation probes.

1.6.3 Lists

An example of an itemised list.

- **Naming conventions from architectural to programming constructs.** For example, a component in the architecture is implemented by a class of the same name. Blah blah blah.

- **Combining architecture and implementation in a single artefact.** Conformance checks are minimised or not required in such systems since architecture and implementation are combined in one specification.

An example of an enumerated list.

1. **Initialise the static analyser.** Provide the compiled Java code of the target application to Structure101 to begin its static analysis process.

2. **Analyse the Java bytecode.** Setup Structure101 to perform detailed analysis of the Java bytecode that also includes exploring method invocations among Java types.

1.6.4 Figures

An example showing how to include figures. Figure 1.1 is taken from a pdf file which gives very good quality and therefore should be used for diagrams as much as possible.

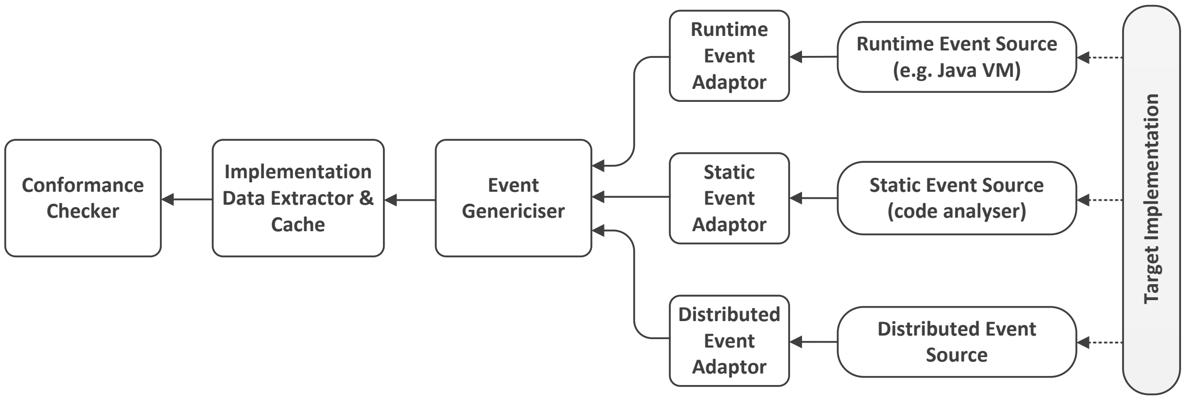

Figure 1.1: The event-driven process used for tapping implementation data

1.6.5 Code Listings

An example of an inline code listing.

```
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!");   // Display messages
  }
}
```

Listing 1.1: Hello World in Java

Program code typeset from as an external file is shown in Listing 1.2. This also shows how to include line numbers.

```
package ness.monitor.runtime.impl.java.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.security.ProtectionDomain;

import org.objectweb.asm.ClassReader;
import org.objectweb.asm.ClassWriter;


class Transformer implements ClassFileTransformer{
  //
  // ClassFileTransformer implementation
  //
  @Override
  public byte[] transform(ClassLoader loader, String name, Class<?> type,
      ProtectionDomain pd, byte[] image) throws IllegalClassFormatException {
    if (Main.isApplicationType(name)) {
      ClassReader reader = new ClassReader(image);
      ClassWriter writer = new ClassWriter(reader, ClassWriter.COMPUTE_MAXS);
      reader.accept(new TypeIntrumentor(writer), ClassReader.SKIP_DEBUG);
      image = writer.toByteArray();
    }
    return image;
  }
}
```

Listing 1.2: Java class transformer

1.6.6 Console Output

```
John:/ $ ls -l
total 16437
drwxrwxr-x+ 52 root  admin     1768 10 Jan 13:21 Applications
drwxr-xr-x   3 root  wheel      102 27 Dec 18:01 Developer
drwxr-xr-x+ 66 root  wheel     2244  9 Jan 18:28 Library
drwxr-xr-x@  2 root  wheel       68 25 Aug 01:15 Network
drwxr-xr-x+  4 root  wheel      136 18 Dec 13:58 System
drwxr-xr-x   6 root  admin      204 18 Dec 14:16 Users
drwxrwxrwt@  4 root  admin      136 13 Jan 17:50 Volumes
drwxr-xr-x@ 39 root  wheel     1326 18 Dec 14:00 bin
drwxrwxr-t@  2 root  admin       68 25 Aug 01:15 cores
dr-xr-xr-x   3 root  wheel     4228 11 Jan 11:31 dev
lrwxr-xr-x@  1 root  wheel       11 18 Dec 13:46 etc -> private/etc
dr-xr-xr-x   2 root  wheel        1 13 Jan 16:10 home
-rwxr-xr-x@  1 root  wheel  8393256 20 Sep 06:22 mach_kernel
dr-xr-xr-x   2 root  wheel        1 13 Jan 16:10 net
drwxr-xr-x@  6 root  wheel      204 18 Dec 14:06 private
drwxr-xr-x@ 62 root  wheel     2108 18 Dec 14:02 sbin
lrwxr-xr-x@  1 root  wheel       11 18 Dec 13:47 tmp -> private/tmp
drwxr-xr-x@ 13 root  wheel      442 19 Dec 11:59 usr
lrwxr-xr-x@  1 root  wheel       11 18 Dec 13:47 var -> private/var
```

1.6.7 Defining accronyms

Blah blah blah virtual machine (VM) blah blah blah blah and Java virtual machine (JVM). However, blah blah blah, JVM and also any other VM blah blah blah. The Java runtime environment (JRE) blah blah blah blah JVM and blah blah blah blah central processing unit (CPU) blah blah blah blah blah. Typically, most blah blah blah integrated development environment (IDE) blah blah blah blah, however most IDEs blah blah blah blah blah blah blah.

Chapter Two

Context Survey

Numerous approaches have been proposed over the years either to prevent architecture erosion or to detect and restore eroded architectures. This chapter presents a survey of those approaches, which include techniques, tools and processes. They are classified primarily into three generic three categories that attempt to minimise, prevent and repair architecture erosion. Within these broad categories, each approach is further broken down to reflect the high-level strategies adopted to tackle erosion. Some of these strategies in turn contain sub-categories under which survey results are presented. Merits and weaknesses of each strategy is discussed, with the argument that no single strategy can address the problem of erosion. The chapter concludes by presenting a case for further work in developing a holistic and practical approach for controlling architecture erosion.

2.1 Introduction

2.1.1 Terminology

2.1.2 Other surveys

2.2 Classification

2.3 Discussion

2.4 Conclusions

CHAPTER THREE

CHAPTER 3 TITLE

<Chapter Abstract>

3.1 Overview

3.2 Section-2

3.3 Section-3

3.4 Conclusions

CHAPTER FOUR

CHAPTER 4 TITLE

<Chapter Abstract>

4.1 Overview

4.2 Section-2

4.3 Section-3

4.4 Conclusions

APPENDIX A

APPENDIX-A TITLE

Appendix B

Appendix-B Title

REFERENCES

[Dalgarno, 2009] Dalgarno, M. (2009). When good architecture goes bad. *Methods and Tools*, 17(1):27–34.

[Eick et al., 2001] Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A. (2001). Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12.

[Hochstein and Lindvall, 2005] Hochstein, L. and Lindvall, M. (2005). Combating architectural degeneration: A survey. *Information and Software Technology*, 47(10):643–656.

[Izurieta and Bieman, 2007] Izurieta, C. and Bieman, J. (2007). How software designs decay: A pilot study of pattern evolution. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement*, pages 449–451. IEEE.

[Jacobson, 1992] Jacobson, I. (1992). *Object-oriented software engineering*. Addison Wesley.

[Perry and Wolf, 1992] Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52.

[Riaz et al., 2009] Riaz, M., Sulayman, M., and Naqvi, H. (2009). Architectural decay during continuous software evolution and impact of 'Design for Change' on software architecture. In *Proceedings of the International Conference on Advanced Software Engineering and Its Applications*, pages 119–126. Springer.

[Stringfellow et al., 2006] Stringfellow, C., Amory, C., Potnuri, D., Andrews, A., and Geor, M. (2006). Comparison of software architecture reverse engineering methods. *Information and Software Technology*, 48(7):484–497.

[van Gurp and Bosch, 2002] van Gurp, J. and Bosch, J. (2002). Design erosion: Problems and causes. *Journal of Systems and Software*, 61(2):105–119.