UNIVERSITY OF BERGAMO

DOCTORAL THESIS

Using Software Testing to Repair Models

Author: Marco RADAVELLI

Supervisor: Dr. James SMITH

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Computer Science Group School of Engineering

June 11, 2019

Declaration of Authorship

I, Marco RADAVELLI, declare that this thesis titled, "Using Software Testing to Repair Models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:			
Date:			

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITY OF BERGAMO

Abstract

Department of Management, Production and Information Engineering School of Engineering

Doctor of Philosophy

Using Software Testing to Repair Models

by Marco RADAVELLI

Software testing is an important phase in the software development process, aiming at locating faults in artifacts, in order to achieve a degree of confidence that the software behaves according to a specification. While most of the techniques in software testing are applied to debugging, fault-localization, and repair of code, to the best of our knowledge there are fewer works regarding the application of software testing to locating faults in models and to the automated repair of such faults. The goal of this PhD project proposal is to study how testing can be applied to repair models. We describe the research approach and discuss the application cases of combinatorial and feature models. We then discuss future work of applying testing to repair models for other scenarios, such as timed automata.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor. . .

Contents

1.1 Software Testing 1 1.1.1 Black-box software testing 1 1.2 Motivation 1 1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolu	D	eclara	ation of Authorship	iii
1 Introduction 1 1.1 Software Testing 1 1.1.1 Black-box software testing 1 1.2 Motivation 1 1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.2.3 Model Transformations 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 11	A۱	bstra	et	vii
1.1 Software Testing 1 1.1.1 Black-box software testing 1 1.2 Motivation 1 1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3	A	cknov	wledgements	ix
1.1.1 Black-box software testing 1 1.2 Motivation 1 1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 11 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11	1	Intr	oduction	1
1.1.1 Black-box software testing 1 1.2 Motivation 1 1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 11 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11		1.1	Software Testing	1
1.3 Research Questions and Objective 1 2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 1<				1
2 State of the Art 3 2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 1 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 1 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 1		1.2	Motivation	1
2.1 Software Testing 3 2.1.1 Combinatorial Testing 3 2.1.2 The Oracle Problem 3 2.2 Manipulation Techniques 3 2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 11 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 1 <t< td=""><td></td><td>1.3</td><td>Research Questions and Objective</td><td>1</td></t<>		1.3	Research Questions and Objective	1
2.1.1 Combinatorial Testing 2.1.2 The Oracle Problem 3.2.2 Manipulation Techniques 3.2.2.1 SAT and SMT solvers 2.2.2 Search-based Software Engineering 2.2.3 Model Checkers 3.2.3 Automated Program Repair 3.4 Model Transformations 3.4.1 Software Product Lines 3.4.2 Timed Automata 3.4.3 Abstract State Machines 3.5 Systematic Literature Review - Model Repair 3.4 Approach and Uniqueness 3.5 Requiring Timed Automata 5.7 Repair of Configuration Constraints 5.8 Repairing Timed Automata Clock Guards through Abstraction and Testing 5.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 5.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.1 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.3 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.4 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.5 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.5 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.7 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.8 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.9 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.1 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.1 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.3 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.4 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.5 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5.7 MixTgTe: Efficient and Guaranteed	2	Stat	e of the Art	3
2.1.2 The Oracle Problem 2.2 Manipulation Techniques 3.2.2.1 SAT and SMT solvers 2.2.2 Search-based Software Engineering 2.2.3 Model Checkers 3.2.3 Automated Program Repair 2.4 Model Transformations 3.2.4.1 Software Product Lines 3.2.4.2 Timed Automata 3.4.3 Abstract State Machines 3.5 Systematic Literature Review - Model Repair 3.4 Approach and Uniqueness 4 Request-Driven Repair 5.1 Repair of Configuration Constraints 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 6 Tools 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11		2.1	Software Testing	3
2.2 Manipulation Techniques 2.2.1 SAT and SMT solvers 3.2.2.2 Search-based Software Engineering 2.2.3 Model Checkers 3.2.3 Automated Program Repair 3.2.4 Model Transformations 3.2.4.1 Software Product Lines 3.2.4.2 Timed Automata 3.2.4.3 Abstract State Machines 3.2.5 Systematic Literature Review - Model Repair 3.4 Approach and Uniqueness 4 Request-Driven Repair 5.1 Repair of Configuration Constraints 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 5 Tools 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 5.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5 Tools 6 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 5 Tools Times Ti			2.1.1 Combinatorial Testing	3
2.2.1 SAT and SMT solvers 3 2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 11 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11			2.1.2 The Oracle Problem	3
2.2.2 Search-based Software Engineering 3 2.2.3 Model Checkers 3 2.3 Automated Program Repair 3 2.4 Model Transformations 3 2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5 Failure-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 1 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 1 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11		2.2	Manipulation Techniques	3
2.2.3 Model Checkers			2.2.1 SAT and SMT solvers	3
2.2.3 Model Checkers			2.2.2 Search-based Software Engineering	3
2.3 Automated Program Repair 2.4 Model Transformations 2.4.1 Software Product Lines 2.4.2 Timed Automata 3.2.4.3 Abstract State Machines 3.2.5 Systematic Literature Review - Model Repair 3.4 Approach and Uniqueness 4 Request-Driven Repair 5.1 Repair of Configuration Constraints 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11				3
2.4 Model Transformations		2.3		3
2.4.1 Software Product Lines 3 2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5 Failure-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 10 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11		2.4		3
2.4.2 Timed Automata 3 2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5 Failure-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 9 6 Tools 10 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11				3
2.4.3 Abstract State Machines 3 2.5 Systematic Literature Review - Model Repair 3 3 Approach and Uniqueness 5 4 Request-Driven Repair 7 5 Failure-Driven Repair 9 5.1 Repair of Configuration Constraints 9 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9 6 Tools 9 6 Tools 11 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 11 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11				
2.5 Systematic Literature Review - Model Repair				
4 Request-Driven Repair 5 Failure-Driven Repair 5.1 Repair of Configuration Constraints 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 7 7 8 9 6 7 10 11 11		2.5		
 Failure-Driven Repair 5.1 Repair of Configuration Constraints 9.5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9.5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing 9.6 Tools 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations 11 	3	App	proach and Uniqueness	5
 5.1 Repair of Configuration Constraints	4	Req	uest-Driven Repair	7
 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection 9 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing	5	Fail	ure-Driven Repair	9
 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing		5.1	Repair of Configuration Constraints	9
Testing		5.2	Test-Driven Model Evolution in XSS Vulnerability Detection	9
Testing		5.3	Repairing Timed Automata Clock Guards through Abstraction and	
 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web				9
Generation to the Web	6	Too	ls	11
6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations		6.1		
Combinations				
		6.2	•	
7 Conclusions and Future Work 13			Combinations	11
	7	Con	clusions and Future Work	13

A	Frec	quently Asked Questions									15
	A.1	How do I change the colors of links?									15

List of Figures

List of Tables

List of Abbreviations

LAH List Abbreviations HereWSF What (it) Stands For

Introduction

- 1.1 Software Testing
- 1.1.1 Black-box software testing
- 1.2 Motivation
- 1.3 Research Questions and Objective

State of the Art

- 2.1 Software Testing
- 2.1.1 Combinatorial Testing
- 2.1.2 The Oracle Problem
- 2.2 Manipulation Techniques
- 2.2.1 SAT and SMT solvers
- 2.2.2 Search-based Software Engineering
- 2.2.3 Model Checkers
- 2.3 Automated Program Repair
- 2.4 Model Transformations
- 2.4.1 Software Product Lines
- 2.4.2 Timed Automata
- 2.4.3 Abstract State Machines
- 2.5 Systematic Literature Review Model Repair

Approach and Uniqueness

Request-Driven Repair

Failure-Driven Repair

- 5.1 Repair of Configuration Constraints
- 5.2 Test-Driven Model Evolution in XSS Vulnerability Detection
- 5.3 Repairing Timed Automata Clock Guards through Abstraction and Testing

Tools

- 6.1 CTWedge: Migrating Combinatorial Interaction Test Modeling and Generation to the Web
- 6.2 MixTgTe: Efficient and Guaranteed Detection of t-Way Failure-Inducing Combinations

Conclusions and Future Work

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

\hypersetup{urlcolor=red}, or

\hypersetup{citecolor=green}, or

\hypersetup{allcolor=blue}.

If you want to completely hide the links, you can use:

\hypersetup{allcolors=.}, or even better:

\hypersetup{hidelinks}.

If you want to have obvious links in the PDF but not the printed text, use:

\hypersetup{colorlinks=false}.