



Universiteit
Leiden
The Netherlands

Exploring means to facilitate software debugging

SOLTANI, M.S.

Citation

SOLTANI, M. S. (2020, August 25). *Exploring means to facilitate software debugging*. Retrieved from <https://hdl.handle.net/1887/135948>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/135948>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/135948> holds various files of this Leiden University dissertation.

Author: Soltani, M.S.

Title: Exploring means to facilitate software debugging

Issue Date: 2020-08-25

Exploring Means to Facilitate Software Debugging

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 25 Augustus 2020
klokke 16:15 uur

door

Mozhan Soltani

geboren te Tehran, Iran
in 1989

Promotiecommissie

Promotor: Prof. Dr. Thomas Bäck
Copromotors: Dr. Feliene Hermans
Dr. Mike Preuss
Promotiecommissie: Prof. Dr. Tanja Vos (Open University)
Dr. Claire Le Goues (Carnegie Mellon University)
Dr. Hadi Hemmati (University of Calgary)
Prof. Dr. Aske Plaat (voorzitter)
Prof. Dr. Joost Visser (secretaris)

Contents

1	Introduction	7
1.1	Automated Test Generation Techniques	9
1.2	Automated Debugging Techniques	10
1.3	Contributions	11
1.3.1	Overview	12
1.3.2	Motivation for the Contributions and their Outcomes	13
1.4	Publications	15
2	Evolutionary Crash Reproduction	17
2.1	Introduction	17
2.2	Background and Related Work	20
2.2.1	Automated Approaches to Crash Replication	20
2.2.2	Search-based Software Testing	23
2.2.2.1	Genetic Algorithms	24
2.2.3	Unit Test Generation Tools	25
2.2.4	User Studies in Testing and Debugging	26
2.3	The EvoCrash Approach	28
2.3.1	Crash Stack Trace Processing	28
2.3.2	Fitness Function	29
2.3.3	Guided Genetic Algorithm	34
2.3.4	Mocking Strategies	37
2.4	Study I: Effectiveness	38
2.4.1	Research Questions	38

2.4.2	Definition and Context	40
2.4.3	Experimental Procedure	43
2.4.4	Comparison with Coverage-Based Test Generation	44
2.4.5	Crash Reproduction Effectiveness	46
2.4.6	Comparison to State of the Art	50
2.4.7	Threats to Validity	55
2.5	Study II: Usefulness for Debugging	56
2.5.1	Task Selection	56
2.5.2	Experiment Participants	57
2.5.3	Experiment Procedure	58
2.5.4	Data Analysis	60
2.5.5	Statistical Analysis	60
2.5.6	Analysis of the Results	61
2.5.6.1	RQ ₄ : Impact of EvoCrash Tests on Locating Defects	62
2.5.6.2	RQ ₅ : Impact of EvoCrash Tests on Fixing Defects	62
2.5.6.3	RQ ₆ : Impact of EvoCrash Tests on Debugging Time	63
2.5.7	Threats to Validity	64
2.6	Discussion and Lessons Learnt	66
2.7	Conclusions	67
3	Large-scale Evaluation of EvoCrash	73
3.1	Introduction	73
3.2	Background and related work	76
3.2.1	Crash reproduction	76
3.2.2	Search-based crash reproduction with EvoCrash	78
3.2.2.1	Guided genetic algorithm	78
3.2.2.2	Comparison with the state-of-the-art	80
3.3	Benchmark design	80
3.3.1	Projects selection protocol	81
3.3.2	Stack trace collection and preprocessing	83
3.4	The JCrashPack benchmark	84
3.5	Running experiments with ExRunner	85
3.6	Application to EvoCrash: setup	88
3.6.1	Evaluation setup	89
3.7	Application to EvoCrash: results	90
3.7.1	Crash Reproduction Outcomes (RQ1)	91
3.7.1.1	Frames Reproduction Outcomes	92
3.7.1.2	Defects4J applications	93
3.7.1.3	XWiki and Elasticsearch	93

3.7.1.4	Exceptions	95
3.7.2	Impact of Exception Type and Project on Performance (RQ2)	95
3.8	Challenges for crash reproduction (RQ3)	99
3.8.1	Input data generation	99
3.8.2	Complex code	101
3.8.3	Environmental dependencies	102
3.8.4	Static initialization	103
3.8.5	Abstract classes and methods	104
3.8.6	Anonymous classes	104
3.8.7	Private inner classes	105
3.8.8	Interfaces	105
3.8.9	Nested private calls	105
3.8.10	Empty <code>enum</code> type	106
3.8.11	Frames with <code>try/catch</code>	106
3.8.12	Missing line number	107
3.8.13	Incorrect line numbers	108
3.8.14	Unknown	108
3.9	Discussion	109
3.9.1	Empirical evaluation for crash reproduction	109
3.9.2	Usefulness for debugging	110
3.9.3	Benchmark building	111
3.10	Future research directions for search-based crash reproduction	112
3.10.1	Context matters	112
3.10.2	Stack trace preprocessing and target frame selection	113
3.10.3	Guided search	113
3.10.4	Improving testability	114
3.11	Threats to validity	114
3.12	Conclusion	115
4	Fitness Function Evaluation	123
4.1	Introduction	124
4.2	Background and Related Work	125
4.2.1	Related Work	126
4.2.2	EvoCrash	126
4.2.2.1	Weighted Sum (WS) Fitness Function	126
4.2.2.2	Guided Genetic Algorithm (GGA)	127
4.3	Single-Objective and Multi-Objectivization for Crash Reproduction	128
4.3.1	Constraints Relaxation	128
4.3.2	Multi-objectivization	129

4.3.3	Graphical Interpretation	131
4.4	Empirical Evaluation	132
4.4.1	Setup	132
4.4.2	Analysis	133
4.5	Results	134
4.6	Discussion	135
4.7	Conclusion	137
5	The Significance of Bug Report Elements	141
5.1	Introduction	141
5.2	Research Methodology	144
5.2.1	Interviews	145
5.2.1.1	Protocol	146
5.2.1.2	Participants	146
5.2.1.3	Data Analysis	146
5.2.2	Surveying Developers	147
5.2.2.1	protocol	147
5.2.2.2	Participants	148
5.2.2.3	Data Analysis	148
5.2.3	Mining Github Issues	148
5.2.3.1	Analysis of the Mined Issues	149
5.3	The <i>IMaChecker</i> Approach	150
5.4	Results	153
5.4.1	RQ₁ . What types of information do developers perceive as important in bug reports?	154
5.4.2	RQ₂ . Do the important elements in bug reports impact bug resolution times?	155
5.4.3	RQ₃ . How often do bug reports contain the important elements?	163
5.5	Discussion	164
5.5.1	Bug Report Templates and User Support	164
5.5.2	Representative Samples	165
5.5.3	Internal Validity of the Experiments	165
5.5.4	Generalizability of Results	166
5.5.5	Automated Crash Reproduction	167
5.5.6	What Do User Contents Provide?	167
5.6	Related Work	167
5.7	Conclusions	169
6	The Use of Contracts in Open Source Software	185
6.1	Introduction	185

6.2	Related Work	188
6.2.1	Assertion Use and Impact on Quality	188
6.2.2	Empirical Studies on Github Projects	189
6.3	Research Methodology	190
6.3.1	Automated Contract Detection	191
6.3.2	Parsing Commit Logs	196
6.4	Results	199
6.4.1	RQ₁. How often are different types of contracts used?	200
6.4.2	RQ₂. For which use cases do developers use contracts?	201
6.4.3	RQ₃. Does the use of contracts relate to occurrence of defects?	203
6.5	Discussion	203
6.5.1	Preference for Different Contracts	203
6.5.2	Automated Semantic Analysis of Contracts	204
6.5.3	Effect of Using Contracts	205
6.6	Threats to Validity	205
6.6.1	Threats to Internal Validity	205
6.6.2	Generalizability of Findings	206
6.7	Conclusion	206
7	Dutch Summary	209
8	English Summary	211
9	About the Author	213
	Bibliography	215

