

# Contents

Abstract . . . . .	iv
<b>1 Introduction</b>	<b>1</b>
1.1 Performance Regression Detection . . . . .	1
1.2 Understanding the Problem . . . . .	2
1.3 Our Proposal: Horizontal Profiling . . . . .	3
1.4 Contributions . . . . .	3
1.5 Scope and Limitations . . . . .	3
1.6 Thesis Outline . . . . .	4
1.7 Related Publications . . . . .	5
<b>2 Current Practices in Addressing Performance Regressions</b>	<b>6</b>
2.1 Understanding Performance Bugs . . . . .	6
2.1.1 Performance Bugs Categorization . . . . .	7
2.1.2 Performance Bugs and Non-Performance Bugs . . . . .	8
2.1.3 Performance Bug Detection . . . . .	8
2.2 Performance Regression Detection . . . . .	9
2.2.1 Performance Metrics . . . . .	9
2.2.2 Performance Regression Testing . . . . .	11
2.2.3 Performance Testing Overhead . . . . .	13
2.3 Root Cause Isolation of Performance Regressions . . . . .	14
2.3.1 Automatic Analysis . . . . .	14
2.3.2 Assisted Analysis . . . . .	15
2.4 Summary . . . . .	17
<b>3 Tracking and Understanding Performance Variations</b>	<b>19</b>
3.1 Introduction . . . . .	19
3.2 Imperceptible Performance Degradation . . . . .	21
3.3 Methodology . . . . .	21
3.3.1 Challenges . . . . .	21
3.3.2 Workflow . . . . .	22
3.3.3 Projects under Study (Step S1) . . . . .	23
3.3.4 Identifying Benchmarks (Step S3) . . . . .	23
3.3.5 Executing the Benchmarks (Step S4) . . . . .	25
3.3.6 Computing Performance Variation (Step S6) . . . . .	26
3.3.7 Categorizing Performance Variations (Step S9) . . . . .	26
3.4 Overall Results . . . . .	27

3.5	Categorizing Performance Regressions . . . . .	30
3.6	Categorizing Performance Improvements . . . . .	34
3.7	Discussion . . . . .	36
3.8	Related Work . . . . .	36
3.9	Summary . . . . .	37
<b>4</b>	<b>Categorizing Source Code Changes that Cause Performance Variations</b>	<b>38</b>
4.1	Introduction . . . . .	38
4.2	Experimental Setup . . . . .	39
4.2.1	Projects under Study . . . . .	39
4.2.2	Source Code Changes . . . . .	40
4.2.3	Benchmarks . . . . .	40
4.3	Performance Variations of Modified Methods . . . . .	41
4.4	Understanding the Root of Performance Regressions . . . . .	43
4.5	Categorizing Source Code Changes That Affect Method Performance . . . . .	44
4.6	Discussion . . . . .	48
4.7	Related Work . . . . .	49
4.8	Summary . . . . .	50
<b>5</b>	<b>LITO: a Horizontal Profiler</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Implementation . . . . .	52
5.3	Evaluation . . . . .	55
5.4	Comparison . . . . .	59
5.4.1	PerfScope . . . . .	59
5.4.2	PerfScope for Pharo . . . . .	61
5.4.3	Evaluation over new projects and versions . . . . .	68
5.4.4	Discussion . . . . .	68
5.5	Related Work . . . . .	69
5.6	Summary . . . . .	70
<b>6</b>	<b>Performance Evolution Blueprints</b>	<b>71</b>
6.1	Introduction . . . . .	71
6.2	Measuring and Representing Difference of Profiles . . . . .	72
6.3	Identifying Topological Differences . . . . .	75
6.4	Performance Evolution Blueprint . . . . .	76
6.5	Implementation . . . . .	79
6.5.1	Tracking performance failure across software versions . . . . .	80
6.5.2	Comparing two executions . . . . .	81
6.6	Discussion and Limitations . . . . .	83
6.7	Related Work . . . . .	83
6.8	Summary . . . . .	84
<b>7</b>	<b>Conclusions</b>	<b>85</b>
7.1	Contributions of the Dissertation . . . . .	85
7.2	Threats to Validity . . . . .	86
7.3	Impact of Performance Evolution Analysis . . . . .	88

7.4 Future Work . . . . .	88
<b>Bibliography</b>	<b>90</b>

# List of Tables

3.1	Projects, number of analyzed release versions (Vers), number of method modifications (MM), number of lines of code (LOC), number of classes (NOC), number of methods (NOM) . . . . .	24
3.2	Patterns of performance variations . . . . .	30
4.1	M = number of times that a method is modified . . . . .	41
4.2	V= number of times that a method is modified and becomes slower/faster after the modification. (threshold = 5%) . . . . .	43
4.3	Method modifications that affect method performance (R= regression, I= Improvement, R/I = Regression in some benchmarks and Improvement in others)	44
4.4	Source code changes that affect method performance (R= Regression, I= Improvement, R/I = Regression in some benchmarks and Improvement in others)	48
4.5	Comparison of source code changes that cause a variation with the changes that do not cause a variation (R= Regression, I= Improvement, R/I = Regression in some benchmarks and Improvement in others) . . . . .	49
5.1	Detecting performance regressions with LITO using a threshold=5% and a sample rate of 20. . . . .	56
5.2	Risk matrix of a change's expensiveness and frequency [29] . . . . .	60
5.3	Precision, Recall and Reduction Comparison of PerfScope (PS) and LITO (LT) using t=20 . . . . .	67
5.4	Evaluation of PerfScope4Pharo and LITO over two additional projects (prec=precision, red=reduction, Run.=runnable) . . . . .	68

# List of Figures

2.1	Comparing Call Context Trees . . . . .	16
3.1	Methodology structured around a 9 steps workflow. . . . .	22
3.2	Number of versions that i) cause a performance regression in at least one benchmark ii) cause a performance improvement in at least one benchmark iii) do not change the performance of any benchmark and iv) cause a performance regression in one benchmark and a performance improvement in another benchmark. . . . .	28
3.3	Number of Performance Variations $V[v_i, b_{max}]$ of all versions (%), where $b_{max}$ is the benchmark with most variation in that version (Y log scale). . . . .	29
3.4	Performance Variation Distribution of Patterns P1 and P2 . . . . .	31
4.1	Source Code Changes histogram at method level . . . . .	40
4.2	Performance Variations of Modified Methods (threshold = 5%), 111 methods are here reported. . . . .	42
5.1	LITO cost model example . . . . .	52
5.2	The effect of the threshold on the percentage of detected performance regressions and the percentage of selected versions by LITO (> threshold) . . . . .	58
5.3	Evaluating LITO with sample rates of 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. (threshold = 5, 10, 20, 30 and 50) . . . . .	59
5.4	PerfScope - Risk score tuning . . . . .	65
5.5	LITO - sample rate effect on the precision, recall and reduction . . . . .	65
5.6	Comparison between different approaches, t = sample-rate . . . . .	66
5.7	Precision, reduction and recall between different approaches . . . . .	67
6.1	Performance evolution blueprint . . . . .	72
6.2	Performance comparison with JProfiler . . . . .	73
6.3	Performance comparison with Yourkit . . . . .	74
6.4	Common Tree Matching Algorithm - Illustration . . . . .	75
6.5	Equivalence of Node C - according to the flexible node equivalence definition . . . . .	76
6.6	a) CCT of first execution b) CCT of second execution. Nodes E and F are new nodes . . . . .	78
6.7	Using Performance Evolution Blueprint to understand the root cause of performance slow down . . . . .	79
6.8	Illustration of a performance degradation: each line describes the execution time of a benchmark across the versions of Roassal. . . . .	80

6.9	Comparing the benchmark execution in version 1.97 and 1.98 . . . . .	81
6.10	Comparing the benchmark execution in version 1.107 and 1.108 . . . . .	82