Implementing a testbed tool for search-based stress tests

Nauber Gois, Pedro Porfírio, André Coelho
Universidade de Fortaleza, UNIFOR, Av. Washington Soares, 1321 , Fortaleza - CE, Brazil
E-mail: naubergois@gmail.com, porfirio@unifor.br, acoelho@unifor.br

Abstract

Metaheuristic search techniques have been extensively used to provide solutions for a more cost-effective testing process. The use of metaheuristic search techniques for the automatic generation of test has been a burgeoning interest for many researchers in recent years. Search Based Software Testing refers to the use of metaheuristics for the optimization of a task in the context of software testing. Experimentation is important to realistically and accurately test and evaluate search based tests. Experiments involving stress search based tests are inherently complex and typically time-consuming to set up and execute. Such experiments are also extremely difficult to repeat. In this paper, We propose a testbed tool named IAdapter TestBed to evaluate metaheuristic research approaches in search-based software testing. Two experiments were conducted to validate the proposed tool. In the first experiment the metaheuristics converged to scenarios with no antipatterns. In the second experiment, the metaheuristics exclude the scenarios with Unbalanced Processing antipattern.

Keywords— Testbed, Search-based stress test, Hybrid metaheuristics

1 Introduction

Performance problems such as high response times in software applications have a significant effect on the customer's satisfaction. The use of stress testing is an increasingly common practice owing to the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneous access due to several users can result in highly critical failures and negatively affect the customers perception of the company [1] [2] [3] [4].

Stress software testing is an expensive and difficult activity. The exponential growth in the complexity of software only makes the cost of testing continue to rise. Test case generation can be seen as a search problem. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to the fitness function using a metaheuristic search technique. Search-based software testing is the application of metaheuristic search techniques to generate software tests cases or perform test execution. Stress Search-based testing is seen as a promising approach to verifying timing constraints. A common objective of a stress search-based test is to find scenarios that produce execution times that violate the specified timing constraints [5].

Experiments involving stress search-based tests are inherently complex and typically time-consuming to set up and execute. Such experiments are also extremely difficult to repeat. People who might want to duplicate published results, for example, must devote substantial resources for setting up and the environmental conditions are likely to be substantially different. Comparing a new metaheuristic to existing ones, it is advantageous to test on the problem instances already tested by previous papers. Then, results will be comparable on a instance-by-instance basis, allowing relative gap calculations between the two heuristics. A Testbed make possible follow a formalized methodology and reproduce tests for further analysis and comparison. It seems natural that one of the most important parts of a comparison among heuristics is a testbed [6].

This paper addresses the problem of comparing the use of several metaheuristics in search-based tests. In this paper, We propose a flexible testbed tool to evaluate several metaheuristics in a search-based software testing. A tool named IAdapter (github.com/naubergois/newiadapter), a JMeter plugin for performing search-based load tests, was extended. The IAdapter Testbed is an open-source tool that provides tools for search-based test research. This tool emulates test scenarios in a controled environment using mock objects and implementing performance antipatterns. Two experiments were conducted to validate the proposed tool. The experiments use genetic, algorithms, tabu search, simulated annealing and an hybrid approach proposed by Gois et al. [7].

The outline of this paper is as follows. In Section 2, the background information on stress tests, workload model, stress search-based tests and mock objects. Section 3 presents detailed features about common performance antipatterns. In Section 4, the research approach is further defined. Section 5 shows the results of two experiments performed. Conclusions and further work are presented in Section 6.

DOI reference number:

2 Background

Stress testing projects should start with the development of a model for user workload that an application receives. This should take into consideration various performance aspects of the application and the infrastructure that a given workload will impact. A workload is a key component of such a model. The term workload represents the size of the demand that will be imposed on the application under test in an execution. The metric used for measure a workload is dependent on the application domain, such as the length of the video in a transcoding application for multimedia files or the size of the input files in a file compression application [3].

Search-Based Testing is the process of automatically generating test according to a test adequacy criterion,encoded as a fitness function, using search-based optimization algorithms, which are guided by a fitness function. The role of the fitness function is to capture a test objective that, when achieved, makes a contribution to the desired test adequacy criterion . Search–Based Testing uses metaheuristic algorithms to automate the generation of test inputs. Metaheuristics are strategies that guide the search process to efficiently explore the search space in order to find optimal solutions [5].

A common goal of stress search-based testing is to find workloads that produce execution times that exceed the timing constraints specified. If a temporal error is found, the test was successful. The application of evolutionary algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled [5].

IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin uses genetic algorithms, tabu search and simulated annealing in a collaborative mode (hybrid metaheuristic) [7]. JMeter is a desktop application designed to test and measure the performance and functional behavior of applications. The JMeter tool can test an emulated class using Mock objects. A mock object is a dummy implementation for an interface or a class. A Mock Object is a substitute implementation for emulating other domain code. Basic mock object allows testing a unit faking the communication with collaborating objects. It should be simpler than the real code, not duplicate its implementation [8].

3 Common performance application problems and performance antipatterns

Performance is critical to the success of today's software systems. Many software products fail to meet their performance objectives when they are initially constructed. There are several antipatterns that detailed features about common performance problems. Antipatterns are conceptually similar to patterns in that they document recurring solutions to common design problems. They are known as antipatterns because their use produces negative consequences. Performance antipatterns document common performance mistakes made in software architectures or designs [9]. Table 1 present some of the most common performance antipatterns.

Table 1: Performance antipatterns

Antipattern
Blob or The God Class
Unbalanced-Processing
Circuitous Treasure Hunt
Empty Semi Trucks
Tower of Babel
One-Lane Bridge
Excessive Dynamic Allocation
Traffic Jam
The Ramp
More is Less

Four antipatterns were chosen to this research by the simplicity of implementation: Unbalanced-Processing, Circuitous Treasure Hunt, Tower of Babel and The Ramp.

A characteristic of Unbalanced Processing is the tendency to overload a particular resource. The overloaded resource will be executing a certain type of job very often, thus in practice, damaging other classes of jobs that will experience very long waiting times. Figure 1 shows a sample of the Unbalanced Processing. In The Fig. 1, four tasks are performed. The task D it is waiting for the task C conclusion that are submmited to a heavy processing situation.

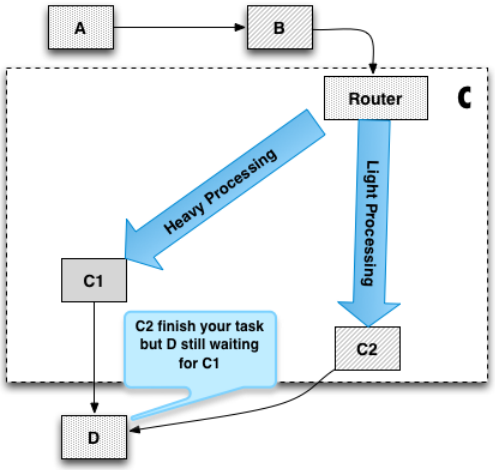

Figure 1: Unbalanced Processing sample [10].

Circuitous Treasure Hunt antipattern occurs when software retrieves data from a first componet, uses those results

in a second component, retrieves data from the second component, and so on, until the last results are obtained. Circuitous Treasure Hunt are typical performance antipatterns that causes unnecessarily high amount of frequent database requests [4].

Tower of Babel antipattern most often occurs when information is translated into an exchange format, such as XML, where the sending process is then parsed and translated into an internal format by the receiving process. When the translation and parsing is excessive, the system spends most of its time doing this [4].

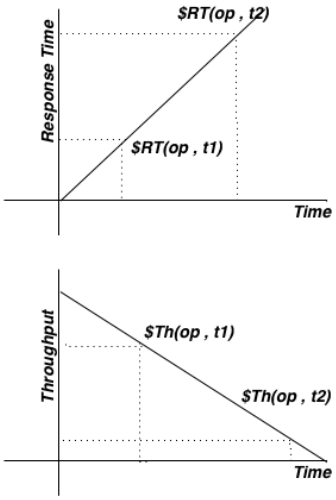

Figure 2: The Ramp sample [11].

Ramp is an antipattern where the processing time increases as the system is used. Fig. 2 shows a system with the Ramp problem: (i) the monitored response time of the operation opx at time t1, i.e. $RT(opx, t1), is much lower than at time t2, i.e. $RT(opx, t2), with t1 < t2; (ii) the monitored throughput of the operation opx at time t1, i.e. $Th(opx, t1), is much larger than at time t2, i.e. $Th(opx, t2), with t1 < t2 [4].

4 The IAdapter Testbed system

In this section, We devise a new testbed that has the ability to reproduce different types of web workloads. The proposed solution extends a tool named IAdapter to create a testbed tool to validade load, performance and stress search based test approaches [7]. This new testbed must accomplish three main goals. First, it must reproduce a workload by using an antipattern implementation. Second, it must be able to provide metrics with the aim of being used for research evaluation studies. Finally, it should be extensible, allowing the creation of new metaheuristic approaches.

The testbed tool proposed consists of four main modules. Figure 3 presents the main architecture of the Testbed solution proposed. The emulator module provides workloads to the Test module. The Test module uses a class loader to find all classes that extends AbstractAlgorithm in the classpath and run all workloads with each metaheuristic found. The Test Scenario library provides the scenario representation used by the metaheuristics and store the testbed results in a database. The Operation services are responsible for finding neighbors of some workload provided as a parameter and perform crossover operations.

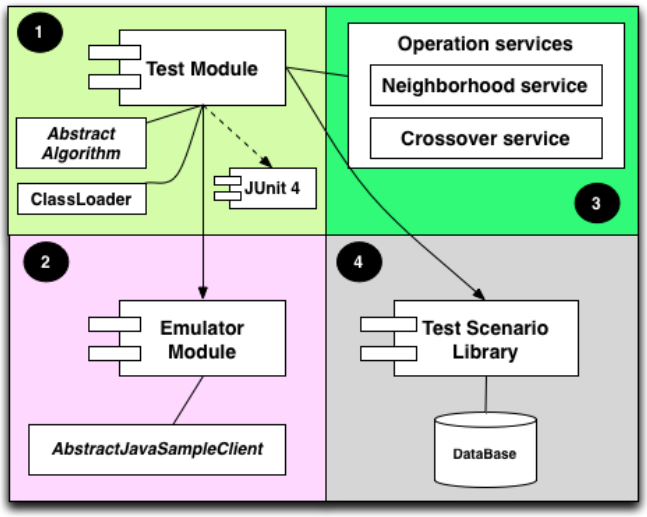

Figure 3: Testbed main architecture.

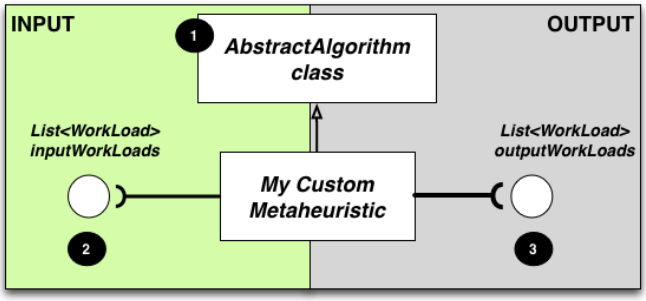

Figure 4: Test Module class diagram.

4.1 Test Module

The Test Module (Figure 3 -❶) is responsible for for the loading of all classes that extend AbstractAlgorithm in the classpath and perform the tests under the application. Figure 4 shows the class diagram for custom and provided heuristics. All heuristic classes extends the class AbstractAlgorithm. The heuristics receives as input a list of workloads (Figure 4 -❷) and must return a list of output workloads (the individuals selected for the next generation) (Figure 4 -❸). Each workload represent an individual in the search space. Figure. 5 presents the Test Module life cycle. Given an initial population (Figure 5 -❶), a metaheurist

select a new set of workloads based on an objective function (Figure 5 -❷). The choosen metaheurist generate a new set of individuals based on crossover or neighborhood operators (Figure 5 -❸). JMeterEngine run each workload (Figure 5 -❺) and the choosen metaheuristic obtain a fitness value for each workload based on some objective function (Figure 5 -❻). Each Metaheuristic could define your own objective function. After all these steps the cycle begins until the maximum number of generations it is reached (Figure 5 -❼).

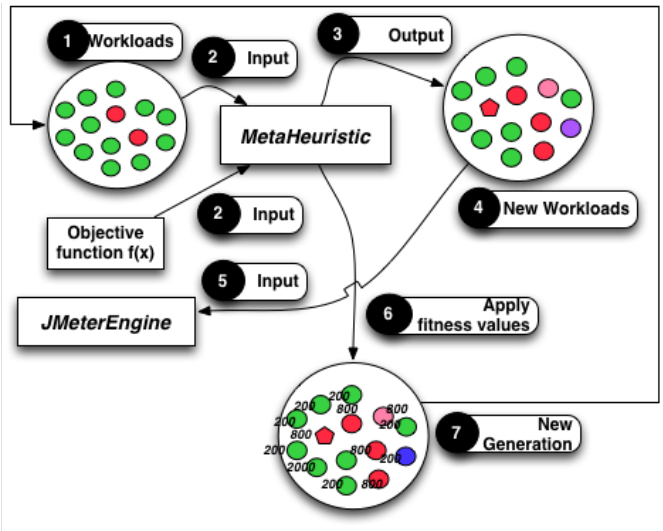

Figure 5: Test module life cycle.

4.2 Emulator Module

The Emulator Module is responsible for implementing and providing successful scenarios and the most common performance antipatterns (Figure 3 -❷). All classes must extend the AbstractJavaSamplerClient class or use JUnit 4. The AbstractJavaSamplerClient class allows the creation of a JMeter Java Request. Figure 6 presents the main features of the emulator module. The module implements 2 happy scenarios (Figure 6 -❷) and 4 antipatterns test scenarios (Figure 6 -❶), in its first version. The Mock Layer provides emulated databases and components for the test scenarios. The Mock Layer use the Mockito and PowerMocks frameworks (Figure 6 -❸).

4.3 Test Scenario Library

This modules provides a common representation for all workloads (Figure 3 -❹). Each workload is composed by a linear vector with 21 positions (Figure 7 -❶). The first position represents an metadata with the name of an individual. The next positions represent 10 scenarios and their numbers of users (Figure 7 -❷). Each scenario is an atomic operation: the scenario must log into the application, run the task goal,

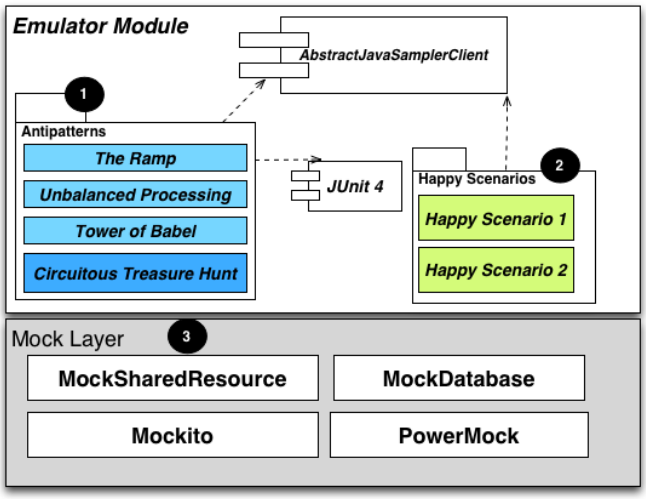

Figure 6: Emulator module

and undo any changes performed, returning the application to its original state.

4.4 Operation services

The services are responsible for performing some operations performed by metaheuristics. Figure. 7 presents the solution representation and an example using the crossover operation. In the example, solution 1 (Figure 7 -❸) has the Login scenario with 2 users, the Search scenario with 4 users, Include scenario with 1 user and the Delete scenario with 2 users. After the crossover operation with solution 2 (Figure 7 -❹), We obtain a solution with the Login scenario with 2 users, the Search scenario with 4 users, the Update scenario with 3 users and the Include scenario with 5 users (Figure 7 -❺). Figure. 7 -❻ shows the strategy used by the proposed solution to obtain the neighbors for the Tabu search and simulated annealing algorithms. The neighbors are obtained by the modification of a single position (scenario or number of users) in the vector.

5 Experiments

In this section, We present the results of experiments which we carried out to verify the antipatterns implementation and the metaheuristics used by the testbed tool. We conducted two experiments in order to verify the effectiveness of the testbed tool. Each experiment use two different antipatterns and the happy scenarios. The experiments ran for 17 generations. The experiments used an initial population of 4 individuals by metaheuristic. The genetic algorithm used the top 10 individuals from each generation in the crossover operation. The Tabu list was configured with the size of 10 individuals and expired every 2 generations. The mutation operation was applied to 10% of the population on each generation. The experiments uses tabu search,

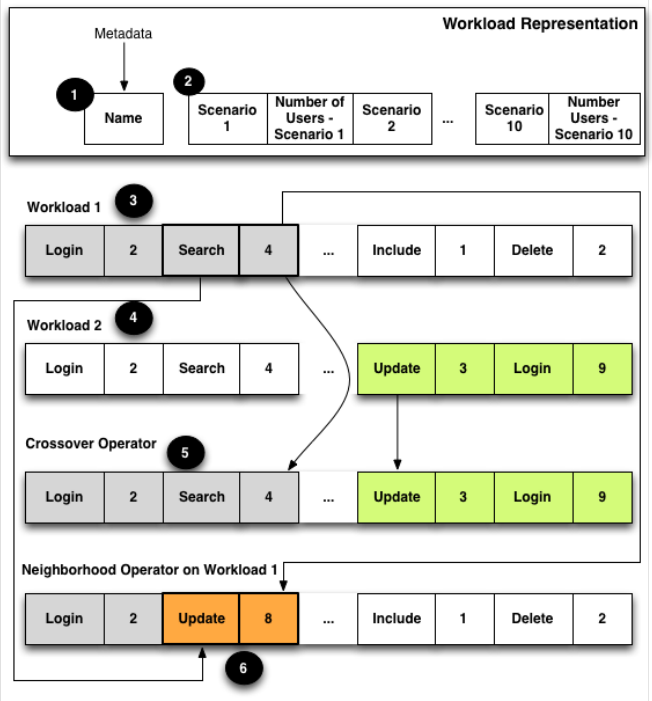

Figure 7: Solution representation, crossover and neighborhood operators

genetic algorithms and the hybrid metaheuristic approach proposed by Gois et al. [7]. The objective function applied is intended to maximize the number of users and minimize the response time of the scenarios being tested. In this experiments, better fitness values means to find scenarios with more users and lower values of a response time. A penalty is applied when the response time is greater than the maximum response time expected.

5.1 Experiment Research Questions

The following research question is addressed: • Does the IAdapter testbed correctly emulates an antipattern? • Does the IAdapter testbed be able to provide metrics with the aim of being used for stress search-based evaluation studies? • Is the IAdapter testbed extensible, allowing the use of several metaheuristic approaches?

5.2 Variables

The independent variables are the test scenarios (antipatterns and happy scenarios). The dependent variable are: the number of antipatterns found in best workloads and the metaheuristic with the best fitness value.

5.3 Hypotheses

- With regard to antipatterns implementation or emulation: – H_0 (A null hypothesis) :the best workloads found in the experiments contain antipatterns – H_1 : the best workloads found in the experiments do not contain antipatterns.

- With regard to metrics provided by the IAdapter: – H_0 (A null hypothesis) : The IAdapter testbed does not be able to provide metrics with the aim of being used for stress search-based evaluation studies. – H_1 : The IAdapter testbed is able to provide metrics with the aim of being used for stress search-based evaluation studies.

- With regard to the IAdapter testbed customization: – H_0 (A null hypothesis) :the IAdapter testbed is not extensible, disallowing the use of several metaheuristic approaches. – H_1 : the IAdapter testbed is extensible, allowing the use of several metaheuristic approaches.

5.4 The Ramp and Circuitous Treasure Hunt experiment

The experiment was carried out for 8 continuous hours. All tests in the experiment were conducted without the need of a tester, automating the process of executing and designing performance test scenarios.In this experiment, Scenarios were generated with the Ramp and Circuitous Treasure antipattern as well as scenarios with Happy Scenario 1, Happy Scenario 2 and mixed scenarios. Figure 8 present the fitness value obtained by each metaheuristic.

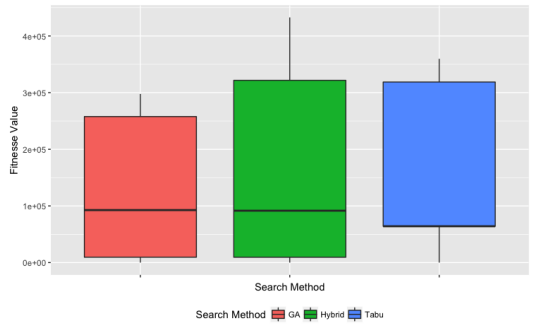

Figure 8: Average, median, maximum and minimal fitness value by Search Method

Table 2 shows 4 best individuals found in the experiment. None of the best individuals has one of the antipatterns used in the experiment, excluding the scenarios with antipatterns.

Table 2: Best individuals found in the first experiment

Metaheur.	Gen.	Users	Fit	Scenarios
Hybrid	17	145	432760	Happy 1 & 2
Hybrid	17	145	432740	Happy 1 & 2
Hybrid	17	146	431760	Happy 1 & 2
Hybrid	16	143	426740	Happy 1 & 2

5.5 The Tower Babel and Unbalanced Processing experiment

The experiment was carried out for 6 continuous hours. In this experiment, Scenarios were generated with Tower Babel and Unbalanced Processing antipattern as well as scenarios with Happy Scenario 1, Happy Scenario 2 and mixed scenarios.

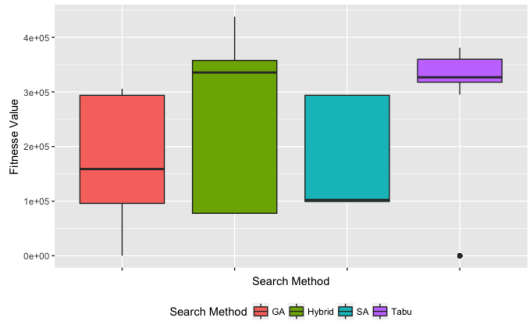

Figure 9: Finesse value by generation in all tests

Table 3 shows the 4 best workloads found in the second experiment. Despite the fact of doing 300 conversions of the JSON format to XML. The antipattern implementation does not return a higher response time than happy paths. While happy paths returns from 10 to 15 seconds from a single user, Tower Babel antipattern has a response time of 10 to 29 seconds. None of the best individuals found implements the Unbalanced Processing antipattern.

Table 3: Best individuals found in the second experiment

Metaheur.	Gen.	Users	Fit	Scenarios
Hybrid	17	148	437780	Happy 1,2 & Tower
Hybrid	17	145	432740	Happy 1,2 & Tower
Hybrid	16	146	431800	Happy 1,2 & Tower
Hybrid	17	145	428780	Happy 1,2 & Tower

In the second experiment, the metaheuristics converged to scenarios with a happy path and Tower Babel antipattern, excluding the scenarios with Unbalanced Processing antipattern. The hybrid metaheuristic returned individuals with higher fitness scores. The SA algorithm obtained the worst fitness values.

5.6 Threats to validity

• Construct Validity: In this work, we just evaluate the use of four antipatterns. However, several antipatterns could be applied. The testbed's common representation and the strategies used for crossover and neighborhood operators need of a better design, using an abstraction pattern to contemplate a major number of possible solutions. • Conclusion Validity: The Tower Babel antipattern was not excluded by the metaheuristics used in the experiment, requiring new studies with new approaches and experiments.

6 Conclusion

IAdapter Testbed is an open-source facility that provides software tools for search based test research. The testbed tool emulates test scenarios in a controled environment using mock objects and implementing performance antipatterns. Two experiments were conducted to validate the proposed approach. The experiments use genetic, algorithms, tabu search, simulated annealing and a new hybrid approach proposed by Gois et al. [7]. In the first experiment, none of the best workloads has one of the antipattern scenarios. In the second experiment, the metaheuristics converged to scenarios with an happy path and Tower Babel antipattern, excluding the scenarios with Unbalanced Processing antipattern. In both experiments the hybrid metaheuristic returned individuals with higher fitness scores. Future works include the use of new antipatterns and more experiments with the use of the Tower Babel antipattern.

References

[1] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber, "Realistic load testing of Web applications," in *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[2] Z. Jiang, "Automated analysis of load testing results," Ph.D. dissertation, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1831726

[3] I. Molyneaux, *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*, 1st ed. "O'Reilly Media, Inc.", Jan. 2009.

[4] A. Wert, M. Oehler, C. Heger, and R. Farahbod, "Automatic detection of performance anti-patterns in inter-component communications," *QoSA 2014 - Proceedings of the 10th International ACM SIGSOFT Conference on Quality of Software Architectures (Part of CompArch 2014)*, pp. 3–12, 2014.

[5] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, vol. 51, no. 6, pp. 957–976, 2009.

[6] J.-Y. Gendreau, Michel and Potvin, *Handbook of Metaheuristics*, 2010, vol. 157.

[7] N. Gois, P. Porfirio, A. Coelho, and T. Barbosa, "Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach," in *Proceedings of the 2016 Latin American Computing Conference (CLEI)*, 2016, pp. 718–728.

[8] M. a. Brown and E. Tapolcsanyi, "Mock Object Patterns," *Matrix*, pp. 1–17, 2003.

[9] W. H. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray, *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[10] A. Wert, J. Happe, and L. Happe, "Supporting swift reaction: Automatically uncovering performance problems by systematic experiments," *Proceedings - International Conference on Software Engineering*, no. May, pp. 552–561, 2013.

[11] V. Vetoio, "PhD Thesis in Computer Science Automated generation of architectural feedback from software performance analysis results Catia Trubiani," *Language*, 2011.