Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 00 (2017) 1–19
Information and Software

A Survey on Stress Testing design approaches

Nauber Gois, Pedro Porfírio, André Coelho

UNIFOR, Av. Washington Soares, 1321 - Edson Queiroz, Fortaleza - CE, 60811-905, Brazil

Abstract

The objective of this paper is surveys stress test design approaches. We performed a systematic review of studies that use stress tests based on a comprehensive set of 97 articles obtained after a multi-stage selection process and have been published in the time span 1994–2016. The results of the review show that are two types of workload models applied on stress tests generative models and descriptible models. Descriptible model just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload to fault-inducing. The Workload design phase of stress testing uses two main aproaches: model-based tests and search-based tests. In model-based stress testing, tests use a model to describe the behavior of a system. Search-Based Software Testing is the use of a meta-heuristic optimizing search technique to automate or partially automate a testing task.

Keywords:
Systematic Review, Stress Testing, Model-based Testing, Search-based Testing,

1. Introduction

Load, performance, and stress testing are typically done to locate bottlenecks in a system, to support a performance-tuning effort, and to collect other performance-related indicators to help stakeholders get informed about the quality of the application being tested [1] [2].

Typically, the most common kind of performance testing for Internet applications is load testing. Application load can be assessed in a variety of ways [3]:

- Concurrency. Concurrency testing seeks to validate the performance of an application with a given number of concurrent interactive users [3].
- Stress. Stress testing seeks to validate the performance of an application when certain aspects of the application are stretched to their maximum limits. This can include maximum number of users, and can also include maximizing table values and data values [3].
- Throughput. Throughput testing seeks to validate the number of transactions to be processed by an application during a given period of time. For example, one type of throughput test might be to attempt to process 100,000 transactions in one hour [3].

The performance testing aims at verifying a specified system performance. This kind of test is executed by simulating hundreds of simultaneous users or more over a defined time interval [4]. The purpose of this assessment is to demonstrate that the system reaches its performance objectives [1]. Term often used interchangeably with “stress” and “load” testing. Ideally “performance” testing is defined in requirements documentation or QA or Test Plans [5].

In a load testing, the system is evaluated at predefined load levels [4]. The aim of this test is to determine whether the system can reach its performance targets for availability, concurrency, throughput, and response time. Load testing is the closest to real application use [6]. A typical load test can last from several hours to a few days, during which system behavior data like execution logs and various metrics are collected [7].

Stress testing investigates the behavior of the system under conditions that overload its resources. The stress testing verifies the system behavior against heavy workloads [1] [5], which are executed to evaluate a system beyond its limits, validate system response in activity peaks, and verify whether the system is able to recover from these conditions. It differs from other kinds of testing in that the system is executed on or beyond its breakpoints, forcing the application or the supporting infrastructure to fail [4] [6].

2. Method

This paper surveys the state of the art literature in stress testing design research. The thesis extends the survey presented by Jiang et al. [8] and Afzal et al. [7] to the Stress Testing context. This survey will be useful for stress testing practitioners and software engineering researchers with interests in testing and analyzing software systems. The paper use the systematic review method proposed by Kitchenham [9].

The aim of a systematic review is to find as many primary studies relating to the research question as possible using an unbiased search strategy. The rigour of the search process is one factor that distinguishes systematic reviews from traditional reviews [9].

Figure 1 presents the summary result of systematic review process. The systematic review is based on a comprehensive set of 97 articles obtained after a multi-stage selection process and have been published in the time span 1994–2016. Of these 97 files, 31 files on model based tests, 17 files on search-based tests and 2 on the FOREPOST technique were selected.

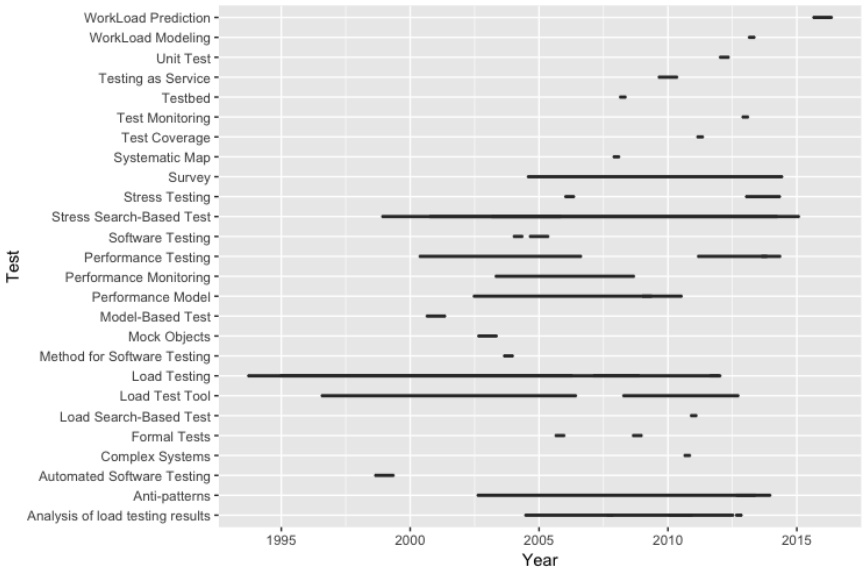

Figure 1: Bubble chart for results of search with the keyword 'Load Testing'

2.1. Planning a Systematic Review

A systematic review of the literature details a protocol describing the process and the methods to be applied. The most important activity during the planning phase is the formulation of research questions. To Kitchenham, before undertaking a systematic review researchers must ensure that it is necessary and the protocol should be able to answer some questions [10]:

- What are the objectives of this review?
- What sources were searched to identify primary studies? Were there any restrictions?
- What were the criteria for inclusion / exclusion and how they are applied?
- What criteria were used to evaluate the quality of the primary studies?
- How were the quality criteria applied?
- How was the data extracted from primary studies?
- What were the differences between studies investigated?
- Because the data were combined?

2.2. Research Questions

In order to examine the evidence of stress testing properties, we proposed the following four research questions:

- How is a stress test workload are designed?
- How is a stress test model-based test are designed?
- How is a stress test search-based test are designed?
- What other stress testing design techniques are used?

2.3. Generation of search strategy

The population in this study is the domain of software testing. Intervention includes application of stress test techniques to test different types of non-functional properties. The primary studies used in this review were obtained from searching databases of peer-reviewed software engineering research that met the following criteria:

- Contains peer-reviewed software engineering journals articles, conference proceedings, and book chapters.
- Contains multiple journals and conference proceedings, which include volumes that range from 1996 to 2017.
- Used in other software engineering systematic reviews.

The resulting list of databases was:

- ACM Digital Library
- Google Scholar
- IEEE Electronic Library
- Inspec
- Scirus (Elsevier)
- SpringerLink

The search strategy was based on the following steps:

- Identification of alternate words and synonyms for terms used in the research questions. This is done to minimize the effect of differences in terminologies.
- Identify common stress testing properties for searching.
- Use of Boolean OR to join alternate words and synonyms.

- Use of Boolean AND to join major terms

We used the following search terms:

- Load Testing: load test, Load Testing
- Stress Testing: stress test, stress testing
- Performance Testing: performance tests
- Test tools: jmeter, load runner, performance tester

All papers found are stored in `https://www.mendeley.com/community/pesquisatesteperformance`. Figures 2 and 3 show the bubble chart for 'Load Testing' and 'Stress Testing' keywords. Figures **??** and **??** present the word cloud extracted for title and abstract of the papers found with keywords 'Load testing' and'Stress testing'.

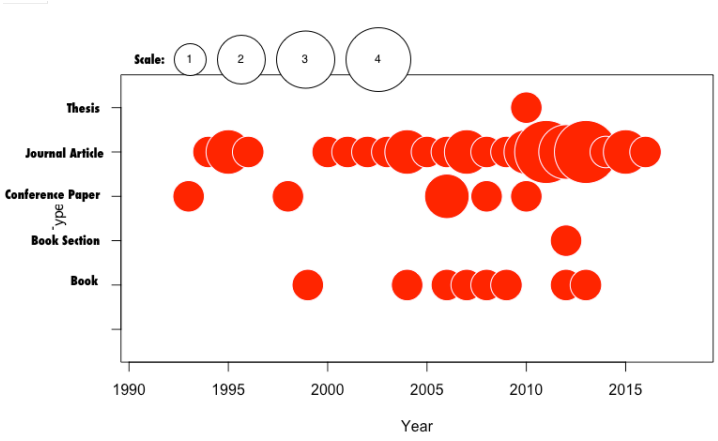

Figure 2: Bubble chart for results of search with the keyword 'Load Testing'

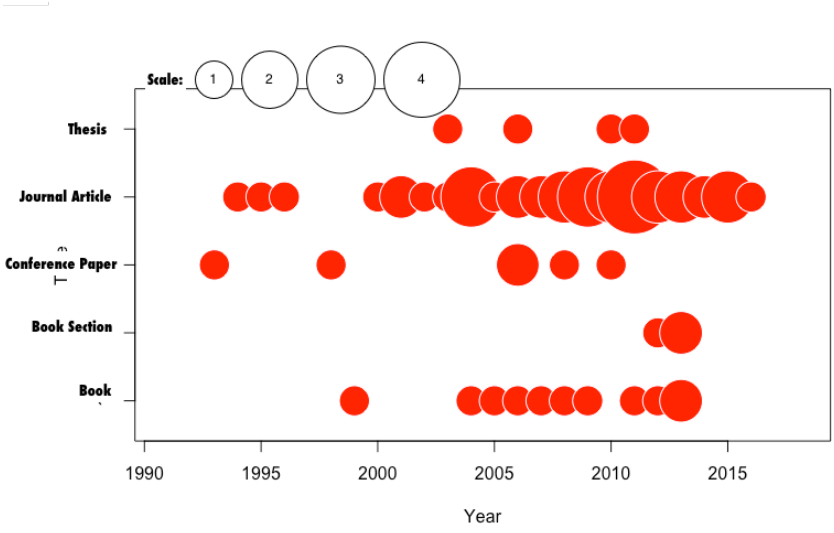

Figure 3: Bubble chart for results of search with the keyword 'Stress Testing'

2.4. Study selection criteria and procedures for including and excluding primary studies

The idealized selection process was done in two parts: an initial document selection of the results that could reasonably satisfy the selection criteria based on a title and the articles abstract reading, followed by a final selection of the initially selected papers based on the introduction and conclusion reading of the papers. The following exclusion criteria is applicable in this review, i.e. exclude studies that:

- Do not relate to stress testing.
- Do not relate to load testing tool.
- Do not relate to load/stress testing model.

From 366 initial papers, 97 papers was selected.

2.5. Data Synthesis

Data synthesis involves collating and summarising the results of the included primary studies. Synthesis can be descriptive (non-quantitative). The studies was categorized by:

- Type of stress test properties;
- Type of research paper (Thesis, Journal Article, Conference Paper, Book Section or Book)
- Methodology used by the test (Model based Test, FOREPOST, Search-based Tests)

Figure 4 presents the type of research paper by year.

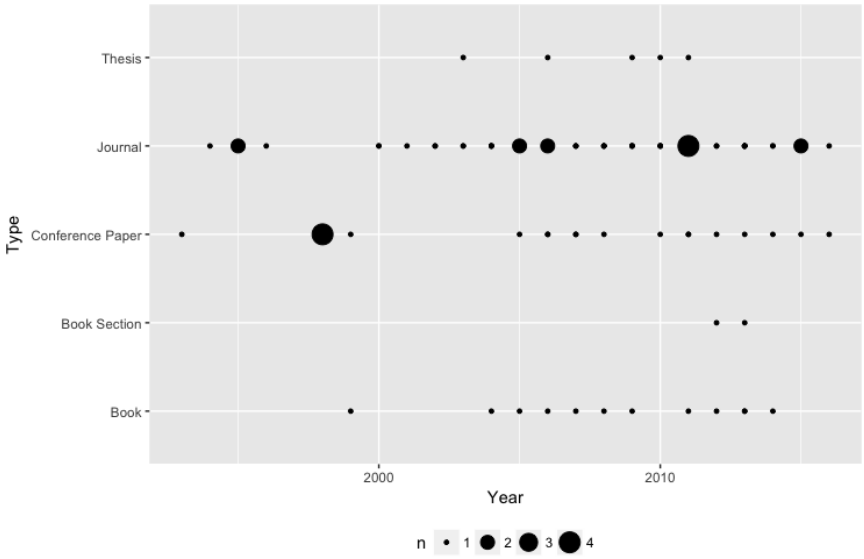

Figure 4: Summary of type of research paper by year

2.6. Stress Test Process

Contrary to functional testing, which has clear testing objectives, Stress testing objectives are not clear in the early development stages and are often defined later on a case-by-case basis. The Fig. 5 shows a common Load, Performance and Stress test process [8].

The goal of the load design phase is to devise a load, which can uncover non-functional problems. Once the load is defined, the system under test executes the load and the system behavior under load is recorded. Load testing practitioners then analyze the system behavior to detect problems [8].

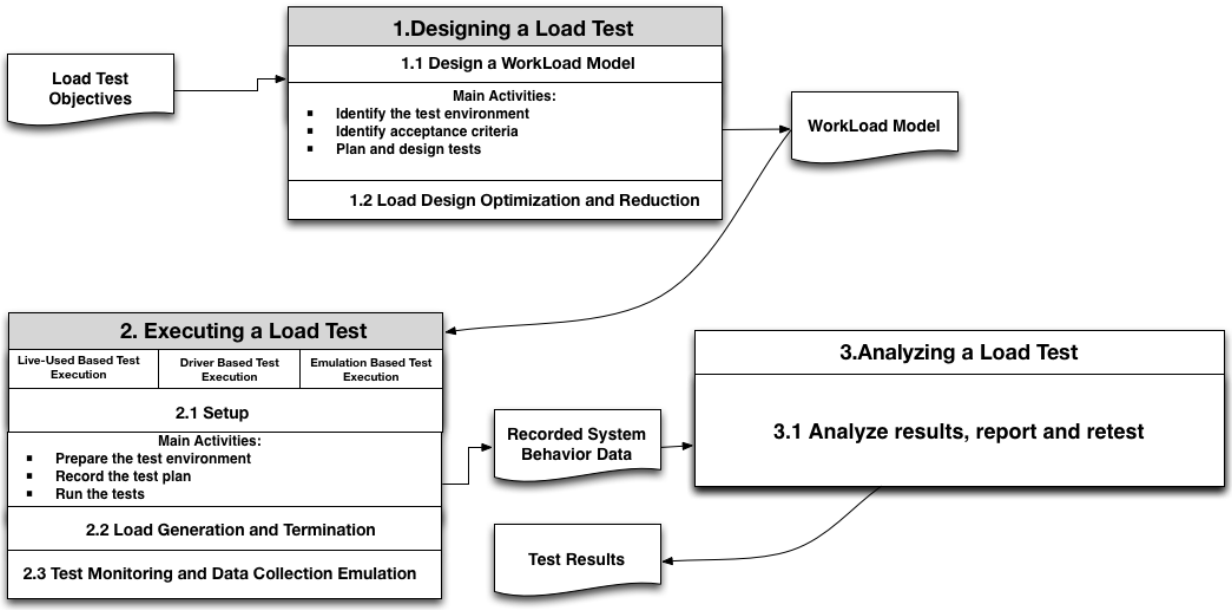

Figure 5: Load, Performance and Stress Test Process [8][11]

Once a proper load is designed, a load test is executed. The load test execution phase consists of the following three main aspects: (1) Setup, which includes system deployment and test execution setup; (2) Load Generation and Termination, which consists of generating the load; and (3) Test Monitoring and Data Collection, which includes recording the system behavior during execution[8].

The core activities in conducting an usual Load, Performance and Stress tests are [11]:

- Identify the test environment: identify test and production environments and knowing the hardware, software, and network configurations helps derive an effective test plan and identify testing challenges from the outset.
- Identify acceptance criteria: identify the response time, throughput, and resource utilization goals and constraints.
- Plan and design tests:identify the test scenarios.In the context of testing, a scenario is a sequence of steps in an application. It can represent a use case or a business function such as searching a product catalog, adding an item to a shopping cart, or placing an order [2]. This task includes a description of the speed, availability, data volume throughput rate, response time, and recovery time of various functions, stress, and so on. This serves as a basis for understanding the level of performance and stress testing that may be required to each test scenario [5].
- Prepare the test environment: configure the test environment, tools, and resources necessary to conduct the planned test scenarios.
- Record the test plan: record the planned test scenarios using a testing tool.
- Run the tests: Once recorded, execute the test plans under light load and verify the correctness of the test scripts and output results.
- Analyze results, report, and retest: examine the results of each successive run and identify areas of bottleneck that need addressing.

3. Research Question 1:How is a stress test workload are designed?

The design of a stress test depends intrinsically on the load model applied to the software under test. Based on the objectives, there are two general schools of thought for designing a proper load to achieve such objectives [7]:

- Designing Realistic Loads (Workload Descriptive).
- Designing Fault-Inducing Loads (Workload Generative).

In Designing Realistic Loads, the main goal of testing is to ensure that the system can function correctly once. Designing Fault-Inducing Loads aims to design loads, which are likely to cause functional or non-functional problems [7].

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 [6].

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 [12] [6] [13].

Workload is also defined by the load distribution between the identified transactions at a given time. Workload helps researchers study the system behavior identified in several load models. A workload model can be designed to verify the predictability, repeatability, and scalability of a system [12] [6].

Workload modeling is the attempt to create a simple and generic model that can then be used to generate synthetic workloads. The goal is typically to be able to create workloads that can be used in performance evaluation studies. Sometimes, the synthetic workload is supposed to be similar to those that occur in practice in real systems [12] [6].

There are two kinds of workload models: descriptive and generative. The main difference between the two is that descriptive models just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload in the first place [4].

In descriptive models, one finds different levels of abstraction on the one hand and different levels of fidelity to the original data on the other hand. The most strictly faithful models try to mimic the data directly using the statistical distribution of the data. The most common strategy used in descriptive modeling is to create a statistical model of an observed workload (Fig. 6). This model is applied to all the workload attributes, e.g., computation, memory usage, I/O behavior, communication, etc. [4]. Fig. 6 shows a simplified workflow of a descriptive model. The workflow has six phases. In the first phase, the user uses the system in the production environment. In the second phase, the tester collects the user's data, such as logs, clicks, and preferences, from the system. The third phase consists in developing a model designed to emulate the user's behavior. The fourth phase is made up of the execution of the test, emulation of the user's behavior, and log gathering.

Generative models are indirect in the sense that they do not model the statistical distributions. Instead, they describe how users will behave when they generate the workload. An important benefit of the generative approach is that it facilitates manipulations of the workload. It is often desirable to be able to change the workload conditions as part of the evaluation. Descriptive models do not offer any option regarding how to do so. With the generative models, however, we can modify the workload-generation process to fit the desired conditions [4]. The difference between the workflows of the descriptive and the generative models is that user behavior is not collected from logs, but simulated from a model that can receive feedback from the test execution (Fig. 7).

Both load model have their advantages and disadvantages. In general, loads resulting from realistic-load based design techniques (Descriptive models) can be used to detect both functional and non-functional problems. However, the test durations are usually longer and the test analysis is more difficult. Loads resulting from fault-inducing load design techniques (Generative models) take less time to uncover potential functional and non-functional problems, the resulting loads usually only cover a small portion of the testing objectives [8]. The presented research work uses a generative model.

There are several approaches to design generative or descriptive workloads:

- Model-based Stress testing: a usage model is proposed to simulate users' behaviors.

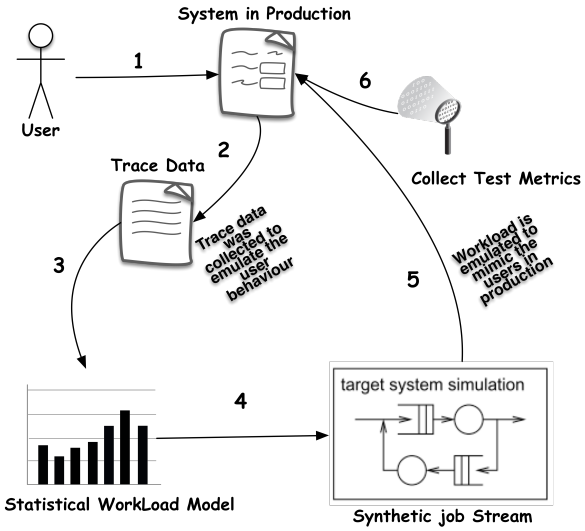

Figure 6: Workload modeling based on statistical data [4]

- Feedback-ORiEnted PerfOrmance Software Testing: is an adaptive, feedback-directed learning testing system that learns rules from system execution [14] [15].
- Search-based Stress testing.

4. Research Question 2:How is a stress test workload are designed?

Model-based testing is an application of models to represent the desired behavior of a System Under Test or to represent testing strategies in a test. Some research aproaches proposes models to simulate or generate realistic loads. Model-based testing (MBT) is a variant of testing that relies on explicit behaviour models that encode the intended behaviours of a system under test. Test cases are generated from one of these models or their combination [16] [17].

The model paradgim is what paradigm and notation are used to describe the model. There are many different modelling notations that have been used for modelling the behaviour of systems for test generation purposes [16] [18].

- State-Based (or Pre/Post) Notations. These model a system as a collection of variables, which represent a snapshot of the internal state of the system, plus some operations that modify those variables. Each operation is usually defined by a precondition and a postcondition, or the postcondition may be written as explicit code that updates the state [16].
- Transition-based Notations. These focus on describing the transitions between different states of the system. Typically, they are graphical node-and-arc notations, like finite state machines (FSMs). Examples of transition-based notations used for MBT include FSMs themselves, statecharts, labelled transition systems and I/O automata [16].
- History-based Notations. These notations model a system by describing the allowable traces of its behaviour over time. Message-sequence charts and related formalisms are also included in this group. These are graphical and textual notations for specifying sequences of interactions between components [16].
- Functional Notations. These describe a system as a collection of mathematical functions. The functions may be first-order only, as in the case of algebraic specifications, or higher-order, as in notations like HOL [16].

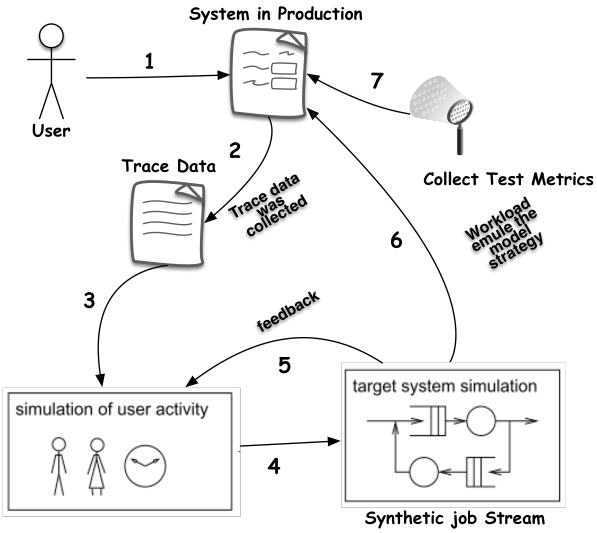

Figure 7: Workload modeling based on the generative model [4]

- Operational Notations. These describe a system as a collection of executable processes, executing in parallel. They are particularly suited to describing distributed systems and communications protocols. Examples include process algebras such as CSP or CCS as well as Petri net notations. Slightly stretching this category, hardware description languages like VHDL or Verilog are also included in this category [16].
- Stochastic Notations. These describe a system by a probabilistic model of the events and input values and tend to be used to model environments rather than SUTs. For example, Markov chains are used to model expected usage profiles, so that the generated tests scenarios [16].
- Data-Flow Notations. These notations concentrate on the data rather than the control flow. Prominent examples are Lustre, and the block diagrams of Matlab Simulink, which are often used to model continuous systems [16].

A User Community Modeling Language (UCML) is a set of symbols that can be used to create visual system usage models and depict associated parameters [19]. The Fig. 8 shows a sample where all users realize a login into the application unde test. Once logged in, 40% of the users navigates on the aplication, 30% of the users realizes downloads. 20% of users realizes uploads and 10% of users performs deletions.

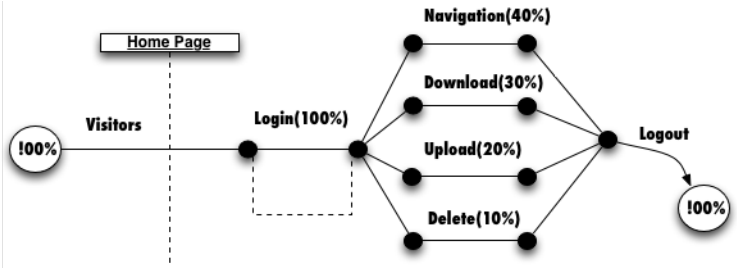

Figure 8: User community modeling language [19]

Another technique to create workload models it is Stochastic Formcharts. The work of Draheim and Weber's Formoriented analysis is a methodology for the specification of ultra-thin client based systems. Form-oriented models describe a web application as a bipartite state machine which consists of pages, actions, and transitions between them. Stochastic Formcharts are the combination of formoriented model and probability features. The Fig. 9 shows a sample

where all users have a probability of 100% of realize a login into the application under test. Once logged in, users have a probability of 40% of navigate on the application and so on [20].

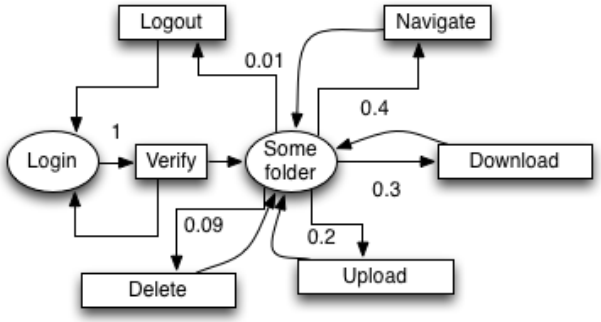

Figure 9: Stochastic Formcharts Example [20] [19]

One way to capture the navigational pattern within a session is through the Customer Behavior Model Graph (CBMG). Figure 10 depicts an example of a CBMG showing that customers may be in several different states—Home, Browse, Search, Select, Add, and Pay—and they may transition between these states as indicated by the arcs connecting them. The numbers on the arcs represent transition probabilities. A state not explicitly represented in the figure is the Exit state [21] [8] [22].

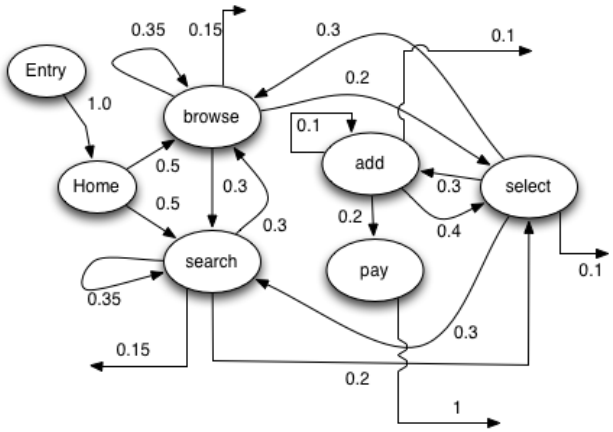

Figure 10: Example of a Customer Behavior Model Graph (CBMG) [21] [8] [22]

Garousi et al. proposes derivate Stress Test Requirements from an UML model. The input model consists of a number of UML diagrams. Some of them are standard in mainstream development methodologies and others are needed to describe the distributed architecture of the system under test (Fig. 11).

Vogele et al. presents an approach that aims to automate the extraction and transformation of workload specifications for an model-based performance prediction of session-based application systems. The research also presents transformations to the common load testing tool Apache JMeter and to the Palladio Component Model [23]. The workload specification formalism (Workload Model) consists of the following components, which are detailed below and illustrated in Fig. 12:

- An Application Model, specifying allowed sequences of service invocations and SUT-specific details for generating valid requests.
- A set of Behavior Models,each providing a probabilistic representation of user sessions in terms of invoked services.

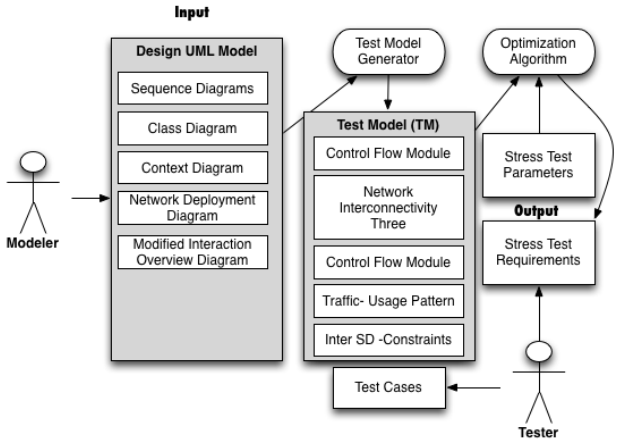

Figure 11: Model-based stress test methodology

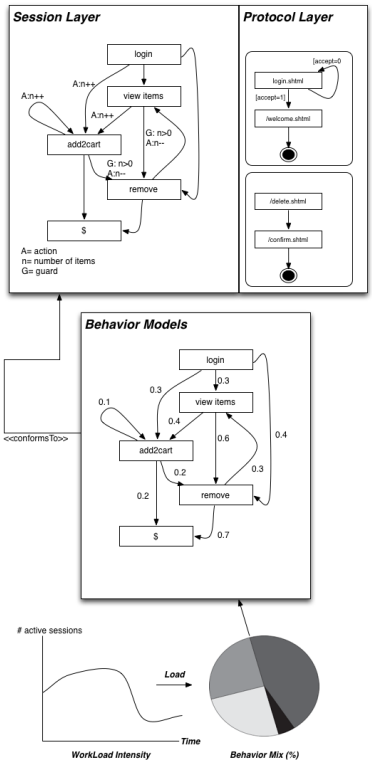

Figure 12: Exemplary workload model

- A Behavior Mix, specified as probabilities for the individual Behavior Models to occur during workload generation.
- A Workload Intensity that includes a function which specifies the number of concurrent users during the workload generation execution.

5. Research Question 3:How is a stress test workload are designed?

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 [24].

Search–Based Testing uses metaheuristic algorithms to automate the generation of test inputs that meet a test adequacy criterion. Many algorithms have been considered in the past, including Simulated Annealing, Parallel Evolutionary Algorithms [25],Evolution Strategies, Estimation of Distribution Algorithms , Scatter Search , Particle Swarm Optimization , Tabu Search and the Alternating Variable Method. An advantage of meta-heuristic algorithms is that they are widely applicable to problems that are infeasible for analytic approaches. All one has to do is come up with a representation for candidate solutions and an objective function to evaluate those solution [26].

The application of metaheuristic search techniques to test case generation is a possibility which offers much benefits. Metaheuristic search techniques are high-level frameworks which utilise heuristics in order to find solutions to combinatorial problems at a reasonable computational cost. Such a problem may have been classified as NP-complete or NP-hard, or be a problem for which a polynomial time algorithm is known to exist but is not practical [27].

One of the most popular search techniques used in SBST belong to the family of Evolutionary Algorithms in what is known as Evolutionary Testing. Evolutionary Algorithms represent a class of adaptive search techniques based on

natural genetics and Darwin's theory of evolution. They are characterized by an iterative procedure that works in parallel on a number of potential solutions to a problem. Figure 13 shows the cycle of an Evolutionary Algorithm when used in the context of Evolutionary Testing [26].

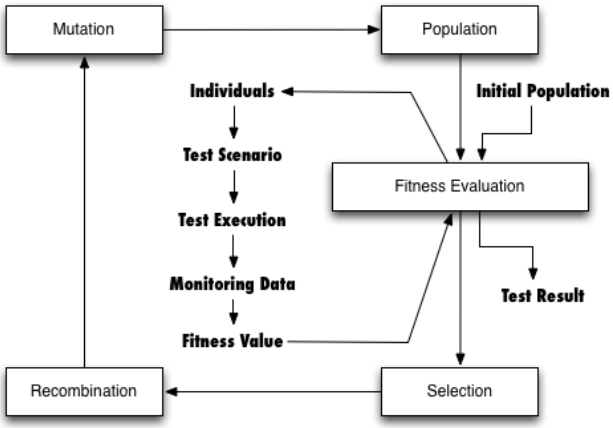

Figure 13: Evolutionary Algorithm Search Based Test Cycle[26].

First, a population of possible solutions to a problem is created, usually at random. Starting with randomly generated individuals results in a spread of solutions ranging in fitness because they are scattered around the search–space. Next, each individual in the population is evaluated by calculating its fitness via a fitness function. The principle idea of an Evolutionary Algorithm is that fit individuals survive over time and form even fitter individuals in future generations. Selected individuals are then recombined via a crossover operator. After crossover, the resulting offspring individuals may be subjected to a mutation operator. The algorithm iterates until a global optimum is reached or another stopping condition is fulfilled [26].

The fitness evaluation is the most time consuming task of SBST. However, for time consuming functional testing of complex industrial systems, minimizing the number of generated individuals may also be highly desirable. This might be done using an assumption about the "potential" of individuals in order to predict which individuals are likely to contribute to any future improvement. This prediction could be achieved by using information about similar individuals that have been executed in earlier generations.

5.0.1. Non-functional Search-Based Testing

SBST has made many achievements, and demonstrated its wide applicability and increasing uptake. Nevertheless, there are pressing open problems and challenges that need more attention like to extend SBST to test non-functional properties, a topic that remains relatively under-explored, compared to structural testing. There are many kinds of non-functional search based tests [7]:

- Execution time: The application of evolutionary algorithms to find the best and worst case execution times (BCET, WCET).
- Quality of service: uses metaheuristic search techniques to search violations of service level agreements (SLAs).
- Security: apply a variety of metaheuristic search techniques to detect security vulnerabilities like detecting buffer overflows.
- Usability: concerned with construction of covering array which is a combinatorial object.
- Safety: Safety testing is an important component of the testing strategy of safety critical systems where the systems are required to meet safety constraints.

A variety of metaheuristic search techniques are found to be applicable for non-functional testing including simulated annealing, tabu search, genetic algorithms, ant colony methods, grammatical evolution, genetic programming and swarm intelligence methods. The Fig. 14 shows a comparison between the range of metaheuristcs and the type of non-functional search based test.The Data comes from Afzal et al. [7]. Afzal's work was added with some of the latest research in this area ([28] [29] [30] [31] [32] [33]).

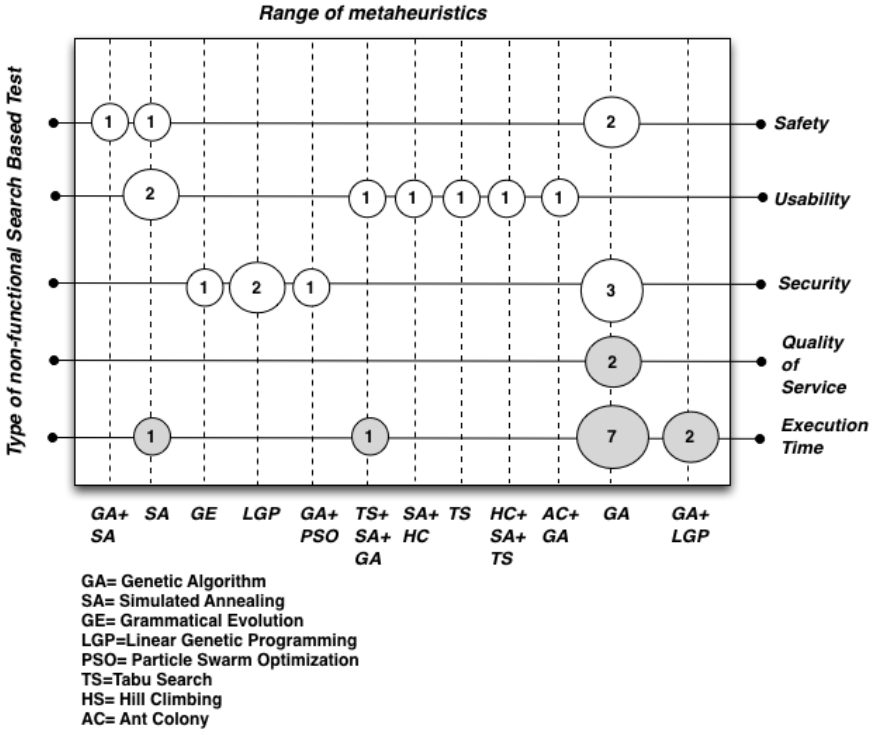

Figure 14: Range of metaheuristics by Type of non-functional Search Based Test[7].

The search for the longest execution time is regarded as a discontinuous, nonlinear, optimization problem, with the input domain of the system under test as a search space [34]. The application of SBST algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled [7].

There are two measurement units normally associated with the fitness function in stress test: processor cycles and execution time. The processor cycle approach describes a fitness function in terms of processor cycles. The execution time approach involves executing the application under test and measuring the execution time [7] [35].

Processor cycles measurement is deterministic in the sense that it is independent of system load and results in the same execution times for the same set of input parameters. However, such a measurement is dependent on the compiler and optimizer used, therefore, the processor cycles differ for each platform. Execution time measurement is a non deterministic approach, there is no guarantee to get the same results for the same test inputs [7]. However, stress testing where testers have no access to the production environment should be measured by the execution time measurement [6] [7].

Table 1 shows a comparison between the research studies on load, performance, and stress tests presented by Afzal et al. [7]. Afzal's work was added to some of the latest research in this area ([28] [29] [30] [31] [32] [33]). The columns represent the type of tool used (prototype or functional tool), and the rows represent the metaheuristic approach used by each research study (genetic algorithm, Tabu search, simulated annealing, or a customized algorithm). The table also sorts the research studies by the type of fitness function used (execution time or processor cycles).

The studies can be grouped into two main groups:

- Search-Based Stress Tesing on Safety-critical systems.

Table 1: Distribution of the research studies over the range of applied metaheuristics

	Prototypes		**Functional Tool**
	Execution Time	Processor Cycles	Execution Time
GA + SA + Tabu Search			Gois et al. 2016 [33]
GA	Alander et al.,1998 [36] Wegener et al., 1996 and 1997 [37][38] Sullivan et al., 1998 [34] Briand et al., 2005 [39] Canfora et al., 2005 [40]	Wegener and Grochtmann, 1998 [41] Mueller et al., 1998 [42] Puschner et al. [43] Wegener et al., 2000 [44] Gro et al., 2000 [45]	Di Penta, 2007 [46] Garoussi, 2006 [28] Garousi, 2008 [47] Garousi, 2010 [29]
Simulated Annealing (SA)			Tracey, 1998 [48]
Constraint Programming			Alesio, 2014 [31] Alesio, 2013 [30]
GA + Constraint Programming			Alesio, 2015 [32]
Customized Algorithm		Pohlheim, 1999 [49]	

- Search-Based Stress Testing on Non Safety-critical systems.

5.0.2. Search-Based Stress Tesing on Safety-critical systems

Domains such as avionics, automotive and aerospace feature safety-critical systems, whose failure could result in catastrophic consequences. The importance of software in such systems is permanently increasing due to the need of a higher system flexibility. For this reason, software components of these systems are usually subject to safety certification. In this context, software safety certification has to take into account performance requirements specifying constraints on how the system should react to its environment, and how it should execute on its hardware platform [30].

Usually, embedded computer systems have to fulfil real-time requirements. A faultless function of the systems does not depend only on their logical correctness but also on their temporal correctness. Dynamic aspects like the duration of computations, the memory actually needed during program execution, orthe synchronisation of parallel processes are of major importance for the correct function of real-time systems [38] .

The concurrent nature of embedded software makes the order of external events triggering the systems tasks is often unpredictable. Such increasing software complexity renders performance analysis and testing increasingly challenging. This aspect is reflected by the fact that most existing testing approaches target system functionality rather than performance [30].

Reactive real-time systems must react to external events within time constraints. Triggered tasks must execute within deadlines. Shousha develops a methodology for the derivation of test cases that aims at maximizing the chance of critical deadline misses [50].

The main goal of Search-Based Stress testing of Safety-critical systems it is finding a combination of inputs that causes the system to delay task completion to the greast extent possible [50]. The followed approaches uses metaheuristics to discover the worst-case execution times.

Wegener et al. [37] used genetic algorithms(GA) to search for input situations that produce very long or very short execution times. The fitness function used was the execution time of an individual measured in micro seconds [37]. Alander et al. [36] performed experiments in a simulator environment to measure response time extremes of protection relay software using genetic algorithms. The fitness function used was the response time of the tested software. The results showed that GA generated more input cases with longer response times [36].

Wegener and Grochtmann performed a experimentation to compare GA with random testing. The fitness function used was duration of execution measured in processor cycles. The results showed that, with a large number of input parameters, GA obtained more extreme execution times with less or equal testing effort than random testing [38] [41].

Gro et. al. [45] presented a prediction model which can be used to predict evolutionary testability. The research confirmed that there is a relationship between the complexity of a test object and the ability of a search algorithm to produce input parameters according to B/WCET [45].

Briand et al. [39] used GA to find the sequence of arrival times of events for aperiodic tasks, which will cause the greatest delays in the execution of the target task. A prototype tool named real-time test tool (RTTT) was developed to facilitate the execution of runs of genetic algorithm. Two case studies were conducted and results illustrated that RTTT was a useful tool to stress a system under test [39].

Pohlheim and Wegener used an extension of genetic algorithms with multiple sub-populations, each using a different search strategy. The duration of execution measured in processor cycles was taken as the fitness function. The GA found longer execution times for all the given modules in comparison with systematic testing[49].

Garousi presented a stress test methodology aimed at increasing chances of discovering faults related to distributed traffic in distributed systems. The technique uses as input a specified UML 2.0 model of a system, augmented with timing information.The results indicate that the technique is significantly more effective at detecting distributed traffic-related faults when compared to standard test cases based on an operational profile [28].

Alesio, Nejati and Briand describe a approach based on Constraint Programming (CP) to automate the generation of test cases that reveal, or are likely to, task deadline misses. They evaluate it through a comparison with a state-of-the-art approach based on Genetic Algorithms (GA). In particular, wthe study compares CP and GA in five case studies for efficiency, effectiveness, and scalability. The experimental results show that, on the largest and more complex case studies, CP performs significantly better than GA. The research proposes a tool-supported, efficient and effective approach based on CP to generate stress test cases that maximize the likelihood of task deadline misses [30].

Alesio describe stress test case generation as a search problem over the space of task arrival times. The research search for worst case scenarios maximizing deadline misses where each scenario characterizes a test case. The paper combine two strategies, GA and Constraint Programming (CP). The results show that, in comparison with GA and CP in isolation, GA+CP achieves nearly the same effectiveness as CP and the same efficiency and solution diversity as GA, thus combining the advantages of the two strategies. Alesio concludes that a combined GA+CP approach to stress testing is more likely to scale to large and complex systems [32].

5.0.3. Search-Based Stress Testing on Non Safety-critical systems

Usually, the application of Search-Based Stress Testing on non safety-critical systems deals with the generation of test cases that causes Service Level Agreements violations.

Tracey et al. [48] used simulated annealing (SA) to test four simple programs. The results of the research presented that the use of SA was more effective with larger parameter space. The authors highlighted the need of a detailed comparison of various optimization techniques to explore WCET and BCET of the of the system under test [48].

Di Penta et al. [46] used GA to create test data that violated QoS constraints causing SLA violations. The generated test data included combinations of inputs. The approach was applied to two case studies. The first case study was an audio processing workflow. The second case study, a service producing charts, applied the black-box approach with fitness calculated only on the basis of how close solutions violate QoS constraint. The genome representation is presented in Fig 15. The representation models a wsdl request to a webservice.

In case of audio workflow, the GA outperformed random search. For the second case study, use of black-box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet [46].

Gois et al. proposes an hybrid metaheuristic approach using genetic algorithms, simulated annealing, and tabu search algorithms to perform stress testing. A tool named IAdapter, a JMeter plugin used for performing search-based stress tests, was developed. Two experiments were performed to validate the solution. In the first experiment, the signed-rank Wilcoxon non- parametrical procedure was used for comparing the results. The significant level adopted was 0.05. The procedure showed that there was a significant improvement in the results with the Hybrid Metaheuristic approach. In the second experiment, the whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established [33].

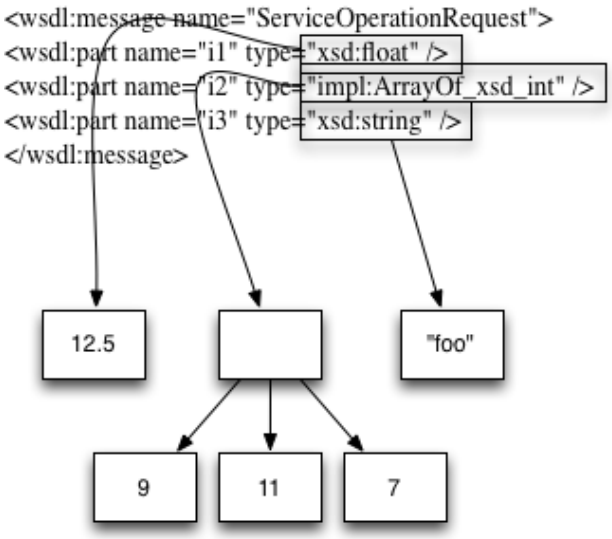

Figure 15: Genome representation [46].

6. Research Question 4:How is a stress test workload are designed?

Feedback-ORiEnted PerfOrmance Software Testing (FOREPOST) is an adaptive, feedback-directed learning testing system that learns rules from system execution traces and uses these learned rules to select test input data automatically to find more performance problems in applications when compared to exploratory random performance testing [51].

FOREPOST uses runtime monitoring for a short duration of testing together with machine learning techniques and automated test scripts to reduce large amounts of performance-related information collected during AUT runs to a small number of descriptive rules that provide insights into properties of test input data that lead to increased computational loads of applications.

The Fig. 16 presents the main workflow of FOREPOST solution. The first step, The Test Script is written by the test engineer(1). Once the test script starts, its execution traces are collected (2) by the Profiler, and these traces are forwarded to the Execution Trace Analyzer, which produces (3) the Trace Statistics. The trace statistics is supplied (4) to Trace Clustering, which uses an ML algorithm, JRip to perform unsupervised clustering of these traces into two groups that correspond to (6) Good and (5) Bad test traces.

The user can review the results of clustering (7). These clustered traces are supplied (8) to the Learner that uses them to learn the classification model and (9) output rules. The user can review (10) these rules and mark some of them as erroneous if the user has sufficient evidence to do so. Then the rules are supplied (11) to the Test Script. Finally, the input space is partitioned into clusters that lead to good and bad test cases, to find methods that are specific to good performance test cases. This task is accomplished in parallel to computing rules, and it starts when the Trace Analyzer produces (12) the method and data statistics that is used to construct (13) two matrices (14). Once these matrices are constructed, ICA decomposes them (15) into the matrices for bad and good test cases correspondingly. Finally, the Advisor (16) determines top methods that performance testers should look at (17) to debug possible performance problems.

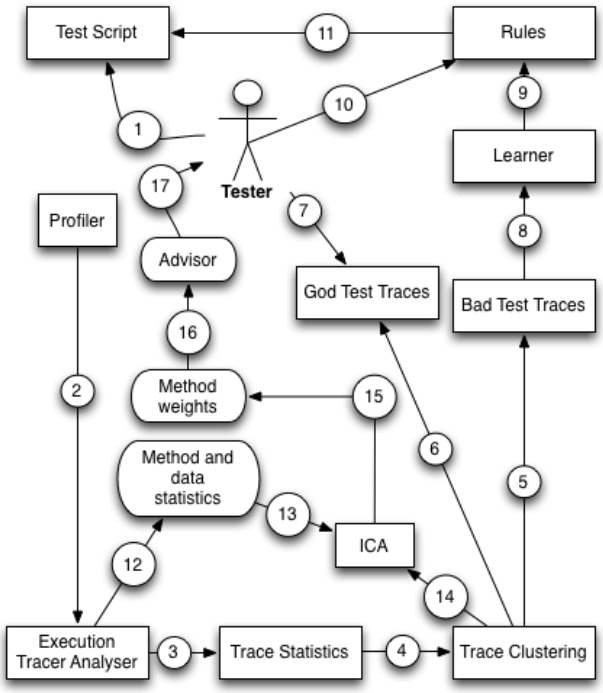

Figure 16: The architecture and workflow of FOREPOST

7. Conclusion

This systematic review investigated the use of stress test techniques. Figure **??** present the results summary of the systematic review. The Test Design phase could use Realistic Load and Fault-Inducing Load. Realistic Load just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload in the first place. There are several approaches to design generative or descriptive workloads.

In Model-based Stress testing, usage model is proposed to simulate users' behaviours. The model paradigm is what paradigm and notation are used to describe the model. There are many different modelling notations that have been used for modelling the behaviour of systems for test generation purposes: State-Based (or Pre/Post) Notations; Transition-based Notations; History-based Notations; Functional Notations; Operational Notations; Stochastic Notations and Data-Flow Notations. There are many kinds of non-functional search based tests: Execution time, Quality of service, Security, Usability and Safety.

Feedback-ORiEnted PerfOrmance Software Testing (FOREPOST) is an adaptive, feedback-directed learning testing system that learns rules from system execution traces and uses these learned rules to select test input data automatically to find more performance problems in applications when compared to exploratory random performance testing.

Search-Based Testing is the process of automatically generating test according to a test adequacy criterion. The application of SBST algorithms to stress tests involves finding the best- and worst-case execution times (B/WCET) to determine whether timing constraints are fulfilled. There are two measurement units normally associated with the fitness function in stress test: processor cycles and execution time. The processor cycle approach describes a fitness function in terms of processor cycles. The execution time approach involves executing the application under test and measuring the execution time.

The stress test execution consists of deploy the system and setup test execution ; generating the workloads according to the configurations and terminating the load when the load test is completed and recording the system behaviour. There are three general approaches of load test executions: Live-User Based, Driver-Based and Emulation-Based. In Live-User Based Executions, The test examines a system's behavior when the system is simultaneously used by

many users or execute a load test by employing a group of human testers. The driver based execution approach automatically generate thousands or millions of concurrent requests for a long period of time using a software tool. The emulation based load test execution approach performs the load testing on special platforms and doesn't require a fully functional system and conduct load testing.

There are several antipatterns that details features about common performance problems. Blob is an antipattern whose problem is on the excessive message traffic generated by a single class or component. Unbalanced Processing it's characterises for one scenario where a specific class of requests generates a pattern of execution within the system that tends to overload a particular resource.

References

[1] C. Sandler, T. Badgett, T. Thomas, The Art of Software Testing (2004) 200.

[2] M. Corporation, Performance Testing Guidance for Web Applications (Nov. 2007).
URL http://www.amazon.com/Performance-Testing-Guidance-Web-Applications/dp/0735625700http://msdn.microsoft.com/en-us/library/bb924375.aspx

[3] W. E. Perry, Effective methods for software testing, 2004. doi:10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C.
URL http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract$\backslash$nhttp://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Effective+Methods+for+Software+Testing{\#}2$\backslash$backslash$nhttp://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Effective+methods+for

[4] G. a. Di Lucca, A. R. Fasolino, Testing Web-based applications: The state of the art and future trends, Information and Software Technology 48 (2006) 1172–1186. doi:10.1016/j.infsof.2006.06.006.

[5] W. E. Lewis, D. Dobbs, G. Veerapillai, Software testing and continuous quality improvement, 2005.
URL http://books.google.com/books?id=fgaBDd0TfT8C{\&}pgis=1

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

[7] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (6) (2009) 957–976. doi:10.1016/j.infsof.2008.12.005.

[8] Z. Jiang, Automated analysis of load testing results, Ph.D. thesis (2010).
URL http://dl.acm.org/citation.cfm?id=1831726

[9] B. Kitchenham, S. Charters, Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3, Engineering 45 (4ve) (2007) 1051. arXiv:1304.1186, doi:10.1145/1134285.1134500.
URL http://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Guidelines+for+performing+Systematic+Literature+Reviews+in+Software+Engineering{\#}0{\%}5Cnhttp://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf

[10] M. Marinho, S. Sampaio, T. Lima, H. de Moura, A Systematic Review of Uncertainties in Software Project Management, International Journal of Software Engineering & Applications 5 (6) (2014) 1–21. arXiv:1412.3690, doi:10.5121/ijsea.2014.5601.
URL http://arxiv.org/abs/1412.3690

[11] B. Erinle, Performance Testing With JMeter 2.9, 2013.

[12] D. G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, Cambridge University Press, 2013.

[13] M. C. Gonçalves, Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem.

[14] Q. Luo, A. Nair, M. Grechanik, D. Poshyvanyk, FOREPOST: finding performance problems automatically with feedback-directed learning software testing, Empirical Software Engineering (2015) 1–51doi:10.1007/s10664-015-9413-5.

[15] A. Wert, M. Oehler, C. Heger, 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) (2014) 3–12doi:10.1145/2602576.2602579.
URL http://dx.doi.org/10.1145/2602576.2602579

[16] A. P. Mark Utting, B. Legeard, A taxonomy of model-based testing approaches, Software Testing Verification and Reliability 24 (8) (2012) 297–312. doi:10.1002/stvr.

[17] Model-based generation of testbeds for web services, Testing of Software and ... (2008) 266–282.

[18] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, H. Zedan, Using formal specifications to support testing, ACM Comput. Surv. 41 (2) (2009) 1–76. doi:http://doi.acm.org/10.1145/1459352.1459354.

[19] X. Wang, B. Zhou, W. Li, Model-based load testing of web applications, Journal of the Chinese Institute of Engineers 36 (1) (2013) 74–86. doi:10.1080/02533839.2012.726028.
URL http://www.tandfonline.com/doi/abs/10.1080/02533839.2012.726028

[20] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, G. Weber, Realistic load testing of Web applications, in: Conference on Software Maintenance and Reengineering (CSMR'06), 2006. doi:10.1109/CSMR.2006.43.

[21] D. A. Menascé, G. Mason, TPC-W : A Benchmark for E-commerce (June) (2002) 1–6.

[22] P. N. Mohammad S. Obaidat, F. Zarai, Modeling and Simulation of Computer Networks and Systems Methodologies and Applications.

[23] C. Vogele, A. van Hoorn, E. Schulz, W. Hasselbring, H. Krcmar, WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction???a model-driven approach for session-based application systems, Software and Systems Modeling (Oc-

tober) (2016) 1–35. doi:10.1007/s10270-016-0566-5.
URL "http://dx.doi.org/10.1007/s10270-016-0566-5

[24] M. Harman, P. McMinn, A theoretical and empirical study of search-based testing: Local, global, and hybrid search, IEEE Transactions on Software Engineering 36 (2) (2010) 226–247. doi:10.1109/TSE.2009.71.

[25] E. Alba, F. Chicano, Observations in using parallel and sequential evolutionary algorithms for automatic software testing, Computers and Operations Research 35 (10) (2008) 3161–3183. doi:10.1016/j.cor.2007.01.016.

[26] A. I. Baars, K. Lakhotia, T. E. J. Vos, J. Wegener, Search-based testing, the underlying engine of Future Internet testing, Federated Conference on Computer Science and Information Systems (FedCSIS 2011) (2011) 917–923.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={\&}arnumber=6078178

[27] P. McMinn, R. Court, S. Testing, P. Street, Search-based software test data generation: a survey, Software testing, Verification and reliability 14 (2004) 1–58. doi:10.1002/stvr.294.

[28] V. Garousi, Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms, Ph.D. thesis (2006).

[29] V. Garousi, A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation, IEEE Transactions on Software Engineering 36 (6) (2010) 778–797. doi:10.1109/TSE.2010.5.

[30] S. Di Alesio, S. Nejati, L. Briand, A. Gotlieb, Stress testing of task deadlines: A constraint programming approach, IEEE Xplore (2013) 158–167doi:10.1109/ISSRE.2013.6698915.

[31] S. Di Alesio, S. Nejati, L. Briand, A. Gotlieb, Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing, Principles and Practice of Constraint Programming 813–830doi:10.1007/978-3-319-10428-7_58.

[32] S. D. I. Alesio, L. C. Briand, S. Nejati, A. Gotlieb, Combining Genetic Algorithms and Constraint Programming, ACM Transactions on Software Engineering and Methodology 25 (1).

[33] N. Gois, P. Porfirio, A. Coelho, 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.

[34] M. O. Sullivan, S. Vössner, J. Wegener, D.-b. Ag, Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis — 1–20.

[35] N. J. Tracey, A search-based automated test-data generation framework for safety-critical software, Ph.D. thesis, Citeseer (2000).

[36] J. T. J. Alander, T. Mantere, P. Turunen, Genetic Algorithm Based Software Testing, in: Neural Nets and Genetic Algorithms, 1998.

[37] J. Wegener, H. Sthamer, B. F. Jones, D. E. Eyres, Testing real-time systems using genetic algorithms, Software Quality Journal 6 (2) (1997) 127–135. doi:10.1023/A:1018551716639.
URL http://www.springerlink.com/index/uh26067rt3516765.pdf

[38] B. J. J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, Systematic testing of real-time systems, EuroSTAR'96: Proceedings of the Fourth International Conference on Software Testing Analysis and Review.

[39] L. C. Briand, Y. Labiche, M. Shousha, Stress testing real-time systems with genetic algorithms, Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05 (2005) 1021doi:10.1145/1068009.1068183.

[40] G. Canfora, M. D. Penta, R. Esposito, M. L. Villani, 2005., Canfora, G., An approach for QoS-aware service composition based on genetic algorithms.

[41] J. Wegener, M. Grochtmann, Verifying timing constraints of real-time systems by means of evolutionary testing, Real-Time Systems 15 (3) (1998) 275–298. doi:Doi 10.1023/A:1008096431840.

[42] F. Mueller, J. Wegener, A comparison of static analysis and evolutionary testing for the verification of timing constraints, Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No.98TB100245)doi:10.1109/RTTAS.1998.683198.

[43] P. Puschner, R. Nossal, Testing the results of static worst-case execution-time analysis, Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)doi:10.1109/REAL.1998.739738.

[44] H. Wegener, Joachim and Pitschinetz, Roman and Sthamer, Automated Testing of Real-Time Tasks, Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00).

[45] H. Gross, B. F. Jones, D. E. Eyres, Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems, Software, IEE Proceedings- 147 (2) (2000) 25–30. doi:10.1049/ip-sen.

[46] M. D. Penta, G. Canfora, G. Esposito, Search-based testing of service level agreements, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, pp. 1090–1097.

[47] V. Garousi, Empirical analysis of a genetic algorithm-based stress test technique, Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08 (2008) 1743doi:10.1145/1389095.1389433.

[48] N. J. Tracey, J. a. Clark, K. C. Mander, Automated Programme Flaw Finding using Simulated Annealing.

[49] H. Pohlheim, M. Conrad, A. Griep, Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences, Analysis (724) (2005) 804—-814. doi:10.4271/2005-01-0750.

[50] M. Shousha, Performance stress testing of real-time systems using genetic algorithms, Ph.D. thesis, Carleton University Ottawa (2003).

[51] M. Grechanik, C. Fu, Q. Xie, Automatically finding performance problems with feedback-directed learning software testing, 2012 34th International Conference on Software Engineering (ICSE) (2012) 156–166doi:10.1109/ICSE.2012.6227197.