The Daily Crash: A Reflection on Continuous Performance Testing

Gururaj Maddodi*, Slinger Jansen*, Jan Pieter Guelen† and Rolf de Jong†
* Utrecht University, Princetonplein 5, 3584 CC Utrecht, Netherlands
Email: g.maddodi@uu.nl, slinger.jansen@uu.nl
†AFAS Software, Philipsstraat 9, 3833 LC Leusden, Netherlands
Email: j.guelen@afas.nl, r.dejong@afas.nl

***Abstract*—Software architects base their design tasks on experience mostly, when developing new architectures. The requirements that are placed on these architectures, such as high-availability and minimum performance requirements, are becoming more demanding continuously. In this paper, we reflect on a method for continuous performance testing, to provide architects with feedback on their design/implementation and prevent problems before they affect the users or other systems. If architects employ the method, they are no longer flying blind, and can continuously evaluate and improve their application. We illustrate the use of the method at a large software company and report on the outcomes of using continuous performance testing in the development of their upcoming enterprise resource planning application release that is going to be used by over a million users.**

Keywords–Workload Generation; Performance Testing; Software Architecture.

I. Introduction

Performance demands placed upon modern software systems are constantly increasing. Systems ranging from simple websites to large business solutions need to support concurrent access by large numbers of users [1][2]. Both conventional wisdom and several studies [3][4] indicate that many software projects have performance problems. Catching these performance problems before they affect users of a system in production is becoming a focal point in the development of software systems.

To gain insight into performance variations of a software system, it has to be monitored before and after changes are made to the system. This can be done through testing the system with a test workload. Workload generation is a process of simulating usage load that the software system is expected to handle in a production environment. Performance testing of a software application involves measuring the resource usage or throughput, by giving a set of inputs to the system by means of generated workload. In performance testing, a realistic workload is simulated/generated and thrown at the working system, while different monitoring mechanisms are being used to evaluate system behavior. One of the most common problems in performance testing is that an unrepresentative workload is used, which gives misleading results [5]. The selected workload used for testing has a major influence on the eventual performance results [6]. Hence, an outline of workload generation techniques and tests is a helpful tool in testing software performance.

Also as most software producing organizations continuously upgrade software products, either with new features or bug fixes, these systems must be continuously tested for their performance. Continuous Performance Testing (CPT) is a methodology of testing the performance of a software system every time a change is made to it. This term stems from the continuous software improvement movement, which prefers guided incremental improvement over discontinuous burst of unreliable major software releases and updates [7].

The contributions of this paper are: (1) an outline of available workload generation techniques is provided, and (2) a reflection on a method of CPT in practice at a large software organization. We structure the work as follows. First, the research method is described in Section II. Secondly, an overview is provided of the topic of workload generation with a structured literature review in Section III. Thirdly, a literature review on continuous performance testing is described in Section IV. In Section V, the CPT method is presented, based on the literature study. In Section VI, a case study is presented at a software company describing a practical implementation and results of the method. The case is evaluated with practitioners, who share their experience using CPT and insights it has given to their development process. We describe analysis and discussions in Section VII. Conclusions are described in Section VIII, where we illustrate that CPT, when based on realistic workloads, is an essential tool for any software architect.

II. Research Method

The research was conducted in two phases. First, a literature study was performed to establish the available methods for CPT. Secondly, through design research, a new method for CPT has been created. Thirdly, a case study is conducted to evaluate the practical aspects of CPT.

A. Literature Study

This section details the literature study protocol created to find papers discussing workload generation methods and performance testing. We used the following keywords to find publications on workload generation and continuous performance testing:

"Testing workload generation" OR "Workload generator" OR "Deriving workload" OR "Test user generation" OR "Performance profiling" OR "software performance testing" OR "Workload characterization" OR "Continuous integration testing"

A literature protocol is established based on the recommendations of Webster and Watson [8], and Kitchenham and Charters [9]. Our data collection strategy consisted of using three scientific search engines: Google Scholar, CiteSeerX, and ieeexplore. Besides the mentioned keywords, references

This is an AMUSE paper. See amuse-project.org for more information.

to papers were also looked at, a technique Webster and Watson [8] calls "going backward". This method was also taken in the opposite direction ("going forward") as the references of important papers were also considered.

B. The Case-study Method

The case-study was conducted at AFAS Software. AFAS is a Dutch vendor of Enterprise Resource Planning (ERP) software. The privately held company currently employs over 350 people and annually generates €100 million in revenue. AFAS currently delivers a fully integrated ERP suite, which is used daily by more than 1.000.000 professional users from more than 10.000 customers. The NEXT version of AFAS' ERP software is completely generated, cloud-based, and tailored for a particular enterprise, based on an ontological model of that enterprise. The ontological enterprise model will be expressive enough to fully describe the real-world enterprise of virtually any customer, and as well form the main foundation for generating an entire software suite on a cloud infrastructure platform of choice: AFAS NEXT is entirely platform and database-independent. AFAS NEXT will enable rapid model-driven application development and will drastically increase customization flexibility for AFAS' partners and customers, based on a software generation platform that is future proof for any upcoming technologies. AFAS is continuously evaluating NEXT for its performance, hence they are an attractive choice for the case-study. The case company was also chosen pragmatically, as a long term relationship exists between the company and Utrecht University.

The case study comprised of interviewing four experts from AFAS Software, including software architects and project managers with years of experience in software architecture design. The interviews conducted as part of the case-study were semi-structured. An interview protocol was defined with questions pertaining to: benefits of CPT that AFAS are seeing, tools and frameworks used for testing, workload generation, alert mechanisms, and organizational aspects of decision making. The interviews were recorded and then later transcribed to extract findings regarding the CPT process at AFAS. One of the co-authors of this paper is also the person who implemented some of the basic principles of CPT. He is now working at AFAS and his previous work is being continued and worked on by his colleagues.

Threats to validity can be internal or external. To counter internal validity threats, one of the co-authors, who has in-depth knowledge of the process of CPT at the company, was asked to validate the findings. Also, a company representative was asked to review the findings. External threats we foresee is how the findings can be applicable to other organizations. Further validation can be done by involving more organizations, which we see as future work.

III. Workload Generation

Workload generators have been around since the early 90s, both as academic and commercial tooling. In order to generate for specific scenarios, the generators generally allow for a number of input parameters to be set. Workload generation techniques can have a number of different characteristics associated with them, and can generally can be classified into two types: static (do not depend on earlier of future actions) and dynamic (based on temporal correlations or other outside sources). Early tools were designed for specific circumstances, mostly generating HTTP requests for static environments, such as Webjamma [10] and S-clients [11]. Other tools generate dynamic behavior, such as TPCW [12], SPEC WEB99 [13], Webload [14], and JMeter [15].

Some workload generation methods target non-existent systems and in these cases the generated load is tested against a performance model of the system. In those cases, both the workload and the system are simulated. This does require detailed performance measurements from the original workload. This helps to know the effects of a task on the system (or similar measurements), so the basis of the model has corresponding link to a performance measurement. Burstiness is a commonly overlooked factor when generating workload [16]. The usage of a software system is not evenly distributed across time, but has bursts and lulls. Applying a smaller degree of burstiness to a workload can give the workload more realistic behavior.

Finally, workload generators vary in the workload they output. In the empirical approach, sampling is done of existing data (data recorded from live sessions). In contrast, analytic approaches use mathematical modeling to generate synthetic workload. The empirical approach is easiest to implement, however it lacks the flexibility as the recorded traces are only representative of one specific configuration and software version. A workable analytic model has the shortcoming that it does not accurately exhibit the large number of unique possible characteristics.

A. Workload Characterization and Generation Techniques

In this section, we describe some of the commonly used workload generation and characterization techniques available in literature.

Descriptive Generation [6][17][18][19] is a technique frequently associated with simple statistical methods: averaging, variances, standard deviation, correlations, distributions, and their corresponding visualization such as histograms or scatter plots. These statistic generation methods are meant to generate workload characteristics such as active users, user think time, actions per sec, etc.

Clustering [19][20] is a workload generation technique that groups similar actions as a unit, making generalizations possible. The selected clusters can be made based on other workload generation methods such as correlations, or by making a selection manually based on functionality within the chosen system. Antonatos [21] describes a method, in which the available network traffic is clustered based on protocol and message type, dividing the traffic into 5 clusters: HTTP request, HTTP Images, HTTP Text, HTTPapp data, and so on.

Markov chains [18][22][23][24] use the temporal characteristics of a workload. A Markov chain consists of series of different states a target system can exist in and the transitions between those states. Transitions are given probabilities of occurrence with several states being active at once, for e.g., many users can use a system at once and their actions are not affected by each-other but only by their own previous actions.

Stochastic form charts [19][20][25] are similar to Markov chains, in that, in addition to states and transitions between those states, actions can also be present. A state can only have transitions towards actions with a corresponding probability,

while an action can have one transition towards a state. An example could be a series of web-pages as states and user actions such as logging in or saving as the actions to transition to different or same web pages.

Reactivity [26] is a technique of workload generation which takes output of the target system into account, such as response time of the system to earlier tasks. Response time is a critical factor to the user in an interactive system. Also findings from [27] show that, user think time i.e., the time between subsequent user tasks, is affected by the time it takes for the system to respond. Short response times (0.1 sec) keep the user in the flow improving think time, while longer (10 secs) response times disrupt the user's attention and results in longer think times and different tasks.

Layering is a technique used to model workload from multiple layers instead of only a single layer responsible for the actual workload. This follows the reasoning that, not only the current and preceding task but also the complete state of its parent service/applications affects the total workload.

Regression based modeling [28] is an analytical approach of predicting performance of a target system. By collecting performance test results at different workloads and performing a regression analysis on the results, predictions can be made about the performance for any workload. Hence, only a subset can be used to predict their performance data by extrapolation.

It is important to note that these techniques can be combined to have required characteristics in the generated workloads. With all these techniques, it is important that one should already have the most important characteristics of the workload in mind to generate an appropriate workload. For e.g., if the ideal workload contains a large variability over time, then considering burstiness is necessary. Also the target system and its limitations are important. For instance, should the user only perform certain actions from a given state, then using state diagrams is necessary.

B. Workload Generation Techniques in Practice

There have been many research works to improve realism of the generated workload as well as variability. Many approaches from code analysis to Domain-specific Languages (DSLs) have been adopted. In this section, we describe some of the recent works on workload generation that use combinations of the techniques described above.

Ittershagen et al. [29], proposes a simulation based approach to estimate application's observable shared resource usage. It uses a combined approach of extracting embedded software's processor usage and memory access patterns to get an abstract workload model. This model can then be used in host-based simulation or on a target specific architecture. A compiler infrastructure based system is used, through which the application is processed, which gives the CPU usage and memory access patterns that are used to create an abstracted workload model.

Busch et al. [30] describes an automated workload characterization approach to estimate the performance behavior of I/O intensive software applications in cloud a platform. The proposed algorithm extracts workload characterization models by a non-invasive and lightweight monitoring to get performance metrics such as, request size, read/write ratio, etc. The approach works by dividing high-level operations into several low-level ones corresponding to a workload metric.

In Casaliccio et al. [31], a workload characterization for Desktop-as-a-service (DaaS) is presented. The study was conducted over three months on actual user traffic by measuring resource usage on a DaaS provider's servers. The aim was to study the resource usage and develop a statistical performance model. Several observations were reported from this study: the session length could be modeled with an exponential distribution, and the CPU Load as well as the Disk load (r/w rate) had a long-tail distribution. It was also observed that the peaks in CPU Load or disk read/write rate exceeds the average value by two (or more) orders of magnitude.

Zakay et al. [32] proposes a workload generation method by combining realism of workload tracing (through sampling) and flexibility of workload modeling. The workload traces were divided into sub-traces, which represent activity of real users. New workloads with varying characteristics can then be created by combining the sub-traces in various ways. In order to keep the reconstructed workload close to a real one, re-sampling is done at the user level also taking into account daily and weekly user activity for added realism.

In Van Hoorn et al. [33], a framework for performance testing of session-based applications is proposed using analytical modeling and automatic workload generation. A DSL based approach is used to generate workload specifications, which are then used by workload generation tools to generate workloads. The proposed DSL, WESSBAS-DSL, uses the Markov4JMeter workload modeling formalism for behavior modeling of user session (session duration and 'think' times). The generated WESSBAS-DSL instances are then transformed into corresponding JMeter test plans.

Vogele et al. [34] extend the WESSBAS-DSL approach of [33] by transforming of WESSBAS instances into workload specifications of Palladio Component Model [35], representing architecture-level performance models. The approach enables layered modeling and automatic extraction of workloads. Using WESSBAS-DSL several aspects of session-based system's workload can be modeled such as, Workload Intensity, Application Model, Behavior Models, and Behavior Mix.

Table I shows the workload generation techniques described in III-A used in the literature described above. From Table I, it can be seen that several workload generation techniques can be used depending on the requirement of the test and workload specification.

IV. Performance Testing Methods

In Brunnert et al. [36], a continuous performance evaluation scheme is proposed for new releases or updates for enterprise applications. The main contribution is that architectural information is considered, while defining a resource profile for an application. A model is defined for content and structure to describe performance modeling for an application. A continuous delivery mechanism is also described with steps: create resource usage profile for current version and put it into a repository, predict performance for current version and compare with previous, and if change is detected send notification to development team.

Bezemer et al. [37] describes performance maintenance and improvement of software application in cloud platform

TABLE I. COMPARISON OF WORKLOAD GENERATION METHODS.

Literature	Proposed Method	Description	Workload Generation Techniques Used
Ittershagen et al. [29]	Code analysis	Generate abstract workload model by analysis code and observing resource usage pattern	Simulation of system and workload
Busch at al. [30]	Usage tracing	I/O Workload characterization by using lightweight monitoring cloud usage	Descriptive (statistics means for file size, workload intensity etc.)
Casaliccio et al. [31]	Usage tracing	Mathematical performance model generation for workload characterization by tracing usage of Daas server	Layering, Descriptive (statistical properties of CPU, read/write loads), Burstiness
Zakay et al. [32]	Usage tracing and modeling	Generating workload by tracing real usage load and resampling parts of it to generate workloads with variability and realism	clustering (users), Burstiness, Regression modeling (resampling workload trace parts)
van Hoorn et al. [33]	Domain-specific Language	Generation of performance test plan with workload by using DSL to model workload specifications	Layering (session and protocol), Clustering (behavior mix), Markov chains
Vogele et al. [34]	Domain-specific Language	Using DSL model and transforming it into a PCM model to include architecture-level performance metrics	Markov chains and Layering (similar to [33])

after their deployment. The authors argue that continuously detecting Performance Improvement Opportunities (PIOs), which are defined as situations during which performance can be improved, can assist in performing perfective maintenance (the goal of which is improving performance and therefore perfecting a software system after delivery) of deployed software. An algorithm is proposed to detect PIOs based on response times. From analyzing the PIOs an association rule set can be defined which can be used for detecting bottlenecks.

Rasal et al. [38] proposed a reactive-based framework for performance testing of web applications. In the work, reactivity is described as the way users behave for a provided quality of service. In the proposed scheme web log data is captured from the server-side (from a website set up by the experimenters for a company). A usage pattern is derived in terms of execution time (time taken to process a request) and think time (time user takes to send the next request) from both the server-side and the client-side. A usage pattern model is derived and then it is used to generate automated test cases.

Arcelli et al. [39] proposes a framework for automated generation of software models subject to continuous performance analysis and refactoring. The aim is to detect potential performance anti-patterns (PA), which are mistakes induced during design phase, using the principles of model-driven engineering. The algorithm consists of identifying specific metrics which could degrade performance and associate threshold to them. PA are evaluated by comparing metrics to their threshold. Then, the PA and its solution are identified and refactoring is applied to get the new software model. With the PA knowledge base built, framework can process software models automatically.

In Horkỳ et al. [40], performance unit tests are used to generate performance documentation to help developers make informed decisions (i.e. keeping performance in mind) during software development. In the setup phase, workload for the system is determined, the system is subjected to the workload, and performance is measured. The observed results are evaluated against test criteria using statistical hypothesis testing. Performance documentation is presented for methods in the software framework which would help make choices between different software libraries.

Lou et al. [41] proposes an approach (FOREPOST - Feedback-ORiEnted PerfOrmance Software Testing) to automatically identify input combinations or specific workload to find performance bottlenecks. It uses an adaptive feedback-directed learning system using application execution traces. If-then rules are used to automatically select input values for testing. Initially, a small number of random input values are selected, and then using clustering, execution profiles are grouped into performance categories. The categorized data are then classified using machine learning to obtain rules. The learned rules are used to generate test-cases.

Danciu et al. [42] proposes an approach of analyzing performance of Java 2 Platform, Enterprise Edition applications and providing feedback related to performance in the Integrated Development Environment (IDE) itself. The source code is parsed and converted into a Palladio component model. This is then used to predict the response time of the application. The predicted performance results is presented within the IDE, making developers performance aware during development.

The literature study presented above, encompasses different phases of performance testing of software applications; from testing performance during development/modeling application [39], application upgrading [36], to maintaining deployed software [37]. It also shows reactive approaches [38], and proactive approaches where application development choices are influenced by performance [40][42]. The literature described takes focus on resource usage, throughput, response time, or a combination of them. Various workload generation method were used: real production loads for approaches that use reactive mechanism or deployed application [37][38], predicted workloads [41][42], workloads generated from the software model [36][39], or just stress test specific parts (as only limited functionality is available) of the application by iteratively calling them [40].

In the next section, we extract steps and process phases from the methods and frameworks above, to create an abstract method that can be deployed by practitioners.

V. THE CONTINUOUS PERFORMANCE TESTING METHOD

In Table II, an overview of the steps of CPT that are found in literature is described. It is important to choose the purpose of performance tests and hence the criteria being measured. Table II shows that each work has a specific criterion, such as resource usage [36][37][41] or response time [37][38].

The processes and frameworks found in the literature, all have a workload generation phase as part of the test. For example, Brunnert et al. [36] use usage models that help in predicting resource usage while Lou et al. [41] and Danciu et al. [42] use predicted workloads. Furthermore, Bezemer et al. [37] and Rasal et al. [38] use real production workload, and Horky et al. [40] iteratively call parts of an application to stress test the system. Monitoring points are defined in the test setup, be it logs or the software model itself. Following up, there is a test that involves either performance prediction

TABLE II. COMPARISON OF (CONTINUOUS) PERFORMANCE TESTING METHODS.

Paper	Perf. criteria	Workload	Monitoring	Testing	Alert mechanism
Brunnert et al. [36]	Resource usage	Synthetic workload (usage modeling, think times)	Resource prediction from performance model	Performance prediction	Comparison b/w current and previous version
Bezemer et al. [37]	Throughput, response time and resource usage	Live production workload	Logs	Build ruleset and compare	Provides performance improvement opportunities through bottleneck identification
Rasal et al. [38]	Execution time	Live production workload	Logs	Build usage pattern	Implicit, not stated
Arcelli et al. [39]	Over-used S/W components and resource usage	None, deduced from software model	S/W model annotated with performace indices	Model transformation and threshold comparison	Automated model refactorisation
Horky et al. [40]	Resource usage	Iteratively call a piece of code	Measure resource usage on local system	Run the piece of code and measure response	Performance documentation
Lou et al. [41]	Resource usage	Workload predicted	Measure resource usage on local system	Tracing execution of methods	Identified bottlenecks as methods
Danciu et al. [42]	Response time	Synthetic Workload using modeling	Resource Prediction using performance model	Tracing execution of methods	Identified bottlenecks as methods presented in IDE

for analytical approaches or measuring parameters, such as response time. The obtained results are compared either to previous values or to a threshold. At the last stage, an alert mechanism is used to notify the results to the stakeholders of performance tests. Several ways of alert mechanisms can be observed in the literature, such as, through PIOs [37], performance documentation, or presenting results on the IDE during development.

Based on these observations, an abstract method for CPT is presented with the following steps:

1) **Assess critical performance criteria -** In this step performance metrics to be measured are selected. This is important as this determines the type of test to be performed and the workload parameters.
2) **Decide on workload generation technique to use -** This step is very important as a representative workload is necessary to get reasonable results for the purpose of the test. As stated earlier, this step depends on the type of performance to be performed. Section III gives more information on selecting an appropriate technique.
3) **Identify monitoring points and build in monitoring -** In this step, the points where the application performance in relation to identified metrics is to be located.
4) **Decide on performance testing method and tools to use -** This depends on the performance criteria and stage of development cycle of application. It can be simulation based approach for software under development to monitoring application under use. Section IV describes some of the methods.
5) **Set up alert mechanisms and describe actions -** Setup system to send performance results to concerned personnel for the next course of action.

Using the steps described above, the proposed method for CPT is shown in Fig. 1. By introducing a feedback loop from the testing phase, comprising of alert mechanism and decision making step back to testing, performance evaluation can be done on a continuous basis. In the next section, we validate this method by means of a case-study at AFAS, where a similar CPT method is used. We compare the proposed method with the tools and mechanism used at AFAS, and present the usefulness of CPT being performed in development of NEXT.

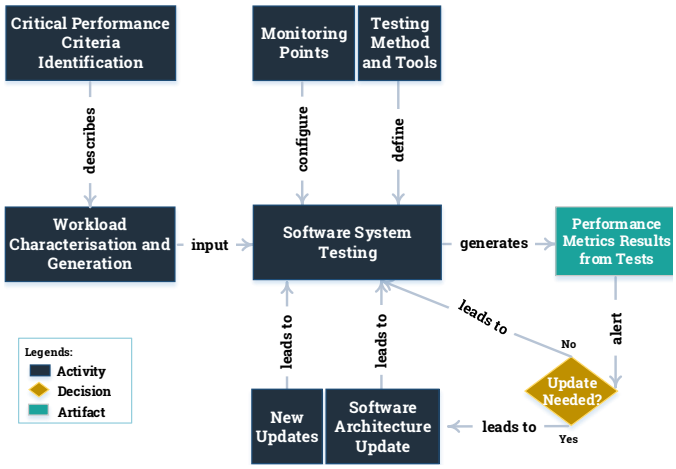

FIGURE 1. CONTINUOUS PERFORMANCE TESTING WORKFLOW.

VI. CASE STUDY AND EXPERT EVALUATION

Currently, AFAS Software is using CPT to evaluate performance changes for NEXT on a regular basis. One of the key design drivers of this redesign is the focus on Command Query Responsibility Segregation (CQRS) [43][44] distributed architecture. CQRS is an architectural pattern developed to support scalability, and provides a practical solution to Brewer's theorem [45] in a distributed system architecture. This pattern prescribes separation in client-server communication between commands and queries, commands being actions which can modify the data while queries are the request to access the said data. This strict separation allows architects to make different architectural choices on command and query side of the system. CPT has become a major tool in the design and validation of architectural choices on both the command and the query side of the CQRS architecture for NEXT.

A. Implementation at the Case Company

The case study was performed with an intention to find out how, in an organization and in real software production setup, performing CPT has lead to improvement in software quality and development process. To this end, the case-study included interviewing experts from AFAS on the process of performance testing of NEXT. The interview protocol was designed with

several sections each discussing different aspects of CPT as described in Section V. The questions were designed to extract the process and tools used at AFAS in each aspect of the CPT. In the Table III, we describe the AFAS context at each aspect of CPT and the tools used by AFAS in their CPT implementation.

B. Case Findings

From transcribing the interviews of the case-study and comparing it with the CPT method described, the following observations were made:

First, the performance criteria that are important for the selected performance test and the purpose of the test are identified. At AFAS, the performance testing is more stress testing the system. The metrics that are measured are: the number of user operations which correspond to commands/queries and events projected per second on their designed CQRS architecture choices. The aim of the tests is to see the trends on the selected data stores and eventually make a choice of selection for stores on command and query side.

A workload generation mechanism is used to suit the test, for instance to do stress tests, concurrent commands and queries are generated, as the tests are mainly focused on measuring throughput of the data stores. The workload is generated by a custom-built application which uses a template for commands and queries to generate workload. The workload generator generates the required number of commands or queries. It is known beforehand the number of events which are to be projected. This helps comprehensively test the designed architecture for its performance.

Monitoring points are identified in the architecture for the selected test. At AFAS the monitoring points are introduced at the command, query and event projection. Plugins are introduced at monitoring points which have performance counters that measure the number of operations (commands, queries, and events) per second. This selection of monitoring points helps to measure the number of commands/queries handled per second as well as the time it takes for the whole process chain to execute.

Currently performance testing is done mainly for the selection of data stores. Testing is done on many different database options (MongoDB, MySQL, MSSQL and PostgreSQL) and are repeated for every database choice. The application is hosted on a server and input with the generated workload. The tests are run with different stores and the said performance metrics are registered at the monitoring points. Since CQRS architecture allows separate choices to be made on command and query side, parts of the system can be optimized independently. one software architect stated :*"It (CPT) helps with the way we want to choose stores or changes we made were right or wrong in terms of performance"*. Another software architect stated: *"From the testing it is noted that PostgreSQL performs better on the query side for transactions"*.

Alert mechanism works by sending emails to the stakeholders after the tests each day. Also, the recorded results are published on the company's intranet and can be accessed by anyone within the organization. The plotted results show metrics from last few weeks, and can be useful to see trends in performance of the application as it evolves. The concerned team monitors the results and makes decisions when it is necessary to investigate the cause. A software architect stated: *"The performance test results become very important when changes are implemented and some variation in performance is expected"*. If the results obtained after changes differ very much than expected, then it is inspected as one of the architects mentioned: *"If the change can be explained, after some new feature was implemented and some drop was expected then it might be fine. But if the change cannot be explained then it will be investigated"*.

In the case-study, a performance testing scenario was created with the goals of comparing database options and operating systems for NEXT. The configurations consisted of MSSQL and MongoDB on Windows and Linux operating systems. As described earlier, the critical performance criteria were defined to be measured, i.e., throughput, response time, CPU, and disk I/O. The workload characterization was done with the metrics in mind to test individual components of CQRS architecture. Hence, the workload consisted of database operations (in large numbers) on a work item, such as Insert (create commands), InsertQMB (create events), Update (update commands), UpdateQMB (update events), Query, Query40 (40 simultaneous queries at a time). Several interesting observations were noted: SQL offered most throughput in inserts, MongoDB showed higher average throughput but gaps were observed (as it processes writes in large batches), MongoDB on Linux showed lower throughput for inserts and queries, etc. This testing configuration gave a basis for performance testing NEXT on a daily basis as new options (in database, event bus, etc.) are tested and the new features are added to the software model NEXT.

The NEXT platform currently exists in a pre-release phase, but parts of it are already being used within the company. The platform is evolving rigorously (*"We make massive architecture changes almost daily (as model-driven development is used and model evolves everyday and thus leads to many changes in implementation)"*) and CPT is helping to evaluate changes from a performance point of view. The company benefits from continuous insight into the platform's performance evolution, especially because the platform is in such an early phase of development. That said, employees in the company states that CPT is useful in any maturity phase of the platform, because at a later stage they expect that CPT can help them identify small changes with large impact on the performance.

C. Expert Evaluation

In the case study, the interviewees indicate that CPT has been essential in the development process of NEXT in the areas of architecture, implementation, and software quality.

With regards to **architecture**, one senior software architect at AFAS, when asked about how CPT has influenced architectural choices for NEXT, stated: *"Performance testing has given cause to rethink [...] things, what you think helps to improve performance but doesn't actually*". This flexibility is seen by others as well: *"Doing performance testing during design/implementation phase is important because the impact of changes can be seen can and work on it rather than during production"*. He also stated that the selection of test is important based on state of the application: *"But it is important to select the type of test as it is not known about production load during development"*. One instance was cited where performance test had helped to identify a bottleneck and prompted a change in architecture - *"A bottleneck was detected*

TABLE III. PERFORMANCE TESTING TOOLS.

Process	Case-study Method	Tools
Assess critical performance criteria	Requests handled per second and time taken to process each request, it can be just the commands or queries or the whole process measuring commands and no. of events it generates or queries and execution of all the projected evens, future: resource usage CPU, memory, IO	This is a choice made based on current state of the product
Workload generation	Commands and/or queries are generated containing user information and in order to stress the system either on command side or query side or both and measure how many requests can be handled	Workload generator which uses a CSV file containing random user information and a template specifying no. of commands and/or queries
Identify monitoring points	monitoring points are inserted on command, query side, and event handler to measure no. of commands	Custom-built plugins at command, query, and event bus side
Performance testing method	Stress test system with concurrent user load (commands or queries or both) and on different data stores and compare the identified performance criteria	RabbitMQ, Event Tracing for Windows, ELK stack, and datastores such as MongoDB, MySQL, MSSQL, and PostgreSQL
Setup alert mechanism and describe action	Email, intranet, and coffee room dashboard, the results can be monitored for trends	Custom tools for transforming events into graphs and publishing it through email and web

in initial testing of ServiceFabric[1] *in event dispatching. One component of the ServiceFabric was detected as producing the bottleneck. As a result a new component was planned to replace the existing one to alleviate the problem"*. Also, in terms of selections of stores, one of the trend that was reported was that PostGreSQL was seen to perform better on query side.

CPT has helped in making **design decisions** as well as validating the designed changes. Project manager at AFAS gave an example: *"it (CPT) has helped us make decision whether to use of dynamic commands vs typed commands in the backend architecture. We expected some performance issues, but through performance testing we observed that no significant degradation in performance occured. Hence it (CPT) helped us validate changes"*.

CPT has furthermore helped in making decisions about where to deploy the application and what kinds of **resources will be used**. Currently, the most important reason to use CPT at AFAS is to create an overview of resource usage on each database platform and find the most performant database for the development of NEXT.

The representatives at the company indicate that **quality** problems that would normally be caught in production are now caught earlier. *"CPT has helped in finding bugs we don't find in normal usage. It also help with deciding whether changes were right or wrong in terms of performance. It has helped now and will help in future if the application monitored properly (after changes) and we get more informative results"*.

VII. ANALYSIS AND DISCUSSION

Continuous performance testing is the best way to provide architects with continuous insight into the implementation of their designs. There are other ways of reporting on architecture patterns, such as empirical qualitative evaluations, but these report on a particular implementation of a pattern, and are thus less usable by other architects. One of the most powerful features of CPT is that it requires architects to provide an upfront expectation of what the performance will be, providing them benchmarks and targets to constantly adjust their design by. The case study is but one observation, but its results can be generalized to other large and long-lived systems.

Frequent exchange of knowledge is required by architects in different working sessions. Their design decisions have long lasting effects on developer productivity, flexibility, variability, and performance of a system. Architects need experience and frequent interactions with other architects in cross-organization expert groups. In the future, we suggest that other authors too report case studies, such that synthesis becomes possible and the CPT framework can be perfected. In particular, academia still need to develop and evaluate tools for CPT.

[1] https://azure.microsoft.com/en-us/services/service-fabric/

VIII. CONCLUSION

This paper functions as a call to action for practitioners and researchers to employ CPT in practice and report on their results. The case study illustrates how an organization has benefited from CPT in an early-stage development project. The main advantages are found in software architecture, design decisions, resource usage, and software quality.

We present directions for future research. Firstly, workload generation techniques can be combined, to create more realistic workloads. In the case-study, random data was used to generate template-driven workload, that was based on the model underlying the application. Secondly, it would be interesting to create realistic workloads based on the devices that are used to access the application. Once a change in end-user behavior is detected, switching from mainly web application to mainly mobile, for instance, could change the workloads that are used for testing accordingly. Thirdly, in case of an ERP application, the user category can dictate the kind of functionality used more often in each category and time of day the application is used most often. This, however, we leave to future research.

ACKNOWLEDGMENT

This research was supported by the NWO AMUSE project (628.006.001): a collaboration between Vrije Universiteit Amsterdam, Utrecht University, and AFAS Software in the Netherlands. The NEXT Platform is developed and maintained by AFAS Software.

REFERENCES

[1] K. Wiegers and J. Beatty, Software requirements. Pearson Education, 2013.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 1, 2002, pp. 80–96.

[3] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," IEEE transactions on software engineering, vol. 26, no. 12, 2000, pp. 1147–1156.

[4] Compuware, "Applied Performance Management Survey," 2006.

[5] R. Jain, The art of computer systems performance analysis. John Wiley & Sons, 2008.

[6] S. Elnaffar and P. Martin, "Characterizing computer systems workloads," Tr. 2002-461, School of Computing, Queen University. Ontario, Canada, 2002.

[7] H. H. Olsson and J. Bosch, "Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation," in Proc. of 6^{th} International Conference of Software Business (ICSOB) June 10-12, 2015, Braga, Portugal. Springer International Publishing, Jun. 2015, pp. 154–166.

[8] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," MIS quarterly, vol. 26, no. 2, 2002, pp. 13–23.

[9] B. A. Kitchenham, "Systematic review in software engineering: Where we are and where we should be going," in Proc. of the 2^{nd} International Workshop on Evidential Assessment of Software Technologies Sep 22, 2012, Lund, Sweden. ACM, 2012, pp. 1–2.

[10] T. Johnson, "Webjamma," 1998.

[11] G. Banga and P. Druschel, "Measuring the capacity of a web server." in USENIX Symposium on Internet Technologies and Systems, Dec. 1997, pp. 61–72.

[12] D. A. Menascé, "Tpc-w: A benchmark for e-commerce," IEEE Internet Computing, vol. 6, no. 3, 2002, pp. 83–87.

[13] A. G. Saidi, N. L. Binkert, L. R. Hsu, and S. K. Reinhardt, "Performance validation of network-intensive workloads on a full-system simulator," Ann Arbor, vol. 1001, 2005, pp. 48 109–2122.

[14] Y. Sherman, U. Hare, and I. Kinreich, "Method of load testing web applications based on performance goal," Aug. 13 2002, uS Patent 6,434,513.

[15] E. H. Halili, Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites, Jun. 2008.

[16] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Injecting realistic burstiness to a traditional client-server benchmark," in Proc. of the 6^{th} international conference on Autonomic computing, Jun. 2009, pp. 149–158.

[17] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with cart models," in Proc. The IEEE Computer Society's 12^{th} Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS), Oct. 2004, pp. 588–595.

[18] A. Van Hoorn, M. Rohr, and W. Hasselbring, "Generating probabilistic and intensity-varying workload for web-based software systems," in SPEC International Performance Evaluation Workshop, Jun. 2008, pp. 124–143.

[19] M. Calzarossa, L. Massari, and D. Tessera, "Workload characterization - issues and methodologies," in Performance Evaluation: Origins and Directions. Springer, 2000, pp. 459–482.

[20] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," in IEEE 7^{th} International Symposium on Service Oriented System Engineering (SOSE), Mar. 2013, pp. 49–60.

[21] M. Yuksel, B. Sikdar, K. Vastola, and B. Szymanski, "Workload generation for ns simulations of wide area networks and the internet," in Proc. of Communication Networks and Distributed Systems Modeling and Simulation Conference, 2000, pp. 93–98.

[22] A. Bahga and V. K. Madisetti, "Synthetic workload generation for cloud computing applications," Journal of Software Engineering and Applications, vol. 4, no. 07, 2011, pp. 396–410.

[23] H. Hlavacs, E. Hotop, and G. Kotsis, "Workload generation by modeling user behavior," Proc. OPNETWORKS, 2000.

[24] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in Proc. of CCA, vol. 8, 2008.

[25] C. Lutteroth and G. Weber, "Modeling a realistic workload for performance testing," in Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE. IEEE, 2008, pp. 149–158.

[26] A. Pereira, L. Silva, W. Meira, and W. Santos, "Assessing the impact of reactive workloads on the performance of web applications," in IEEE International Symposium on Performance Analysis of Systems and Software, Mar. 2006, pp. 211–220.

[27] D. A. Menasce, V. A. Almeida, L. W. Dowdy, and L. Dowdy, Performance by design: computer capacity planning by example. Prentice Hall Professional, 2004.

[28] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in 4^{th} International Conference on Autonomic Computing (ICAC'07), Jun. 2007, pp. 27–27.

[29] P. Ittershagen, P. A. Hartmann, K. Grüttner, and W. Nebel, "A workload extraction framework for software performance model generation," in Proc. of the Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO), Jan. 2015, pp. 3:1–3:6.

[30] A. Busch, Q. Noorshams, S. Kounev, A. Koziolek, R. Reussner, and E. Amrehn, "Automated workload characterization for i/o performance analysis in virtualized environments," in Proc. of the 6^{th} ACM/SPEC International Conference on Performance Engineering, Jan. 2015, pp. 265–276.

[31] E. Casalicchio, S. Iannucci, and L. Silvestri, "Cloud desktop workload: A characterization study," in IEEE International Conference on Cloud Engineering (IC2E). IEEE, Mar. 2015, pp. 66–75.

[32] N. Zakay and D. G. Feitelson, "Workload resampling for performance evaluation of parallel job schedulers," Concurrency and Computation: Practice and Experience, vol. 26, no. 12, 2014, pp. 2079–2105.

[33] A. van Hoorn, C. Vögele, E. Schulz, W. Hasselbring, and H. Krcmar, "Automatic extraction of probabilistic workload specifications for load testing session-based application systems," in Proc. of the 8^{th} International Conference on Performance Evaluation Methodologies and Tools, Dec. 2014, pp. 139–146.

[34] C. Vögele, A. van Hoorn, and H. Krcmar, "Automatic extraction of session-based workload specifications for architecture-level performance models," in Proc. of the 4^{th} International Workshop on Large-Scale Testing, Feb. 2015, pp. 5–8.

[35] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," Journal of Systems and Software, vol. 82, no. 1, 2009, pp. 3–22.

[36] A. Brunnert and H. Krcmar, "Continuous performance evaluation and capacity planning using resource profiles for enterprise applications," Journal of Systems and Software, 2015.

[37] C.-P. Bezemer and A. Zaidman, "Performance optimization of deployed software-as-a-service applications," Journal of Systems and Software, vol. 87, 2014, pp. 87–103.

[38] Y. M. Rasal and S. Nagpure, "Web application: Performance testing using reactive based framework," IJRCCT, vol. 4, no. 2, 2015, pp. 114–118.

[39] D. Arcelli and V. Cortellessa, "Assisting software designers to identify and solve performance problems," in Proc. of the 1^{st} International Workshop on Future of Software Architecture Design Assistants, May 2015, pp. 1–6.

[40] V. Horkỳ, P. Libič, L. Marek, A. Steinhauser, and P. Tuuma, "Utilizing performance unit tests to increase performance awareness," in Proc. of the 6^{th} ACM/SPEC International Conference on Performance Engineering, Jan. 2015, pp. 289–300.

[41] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk, "Forepost: finding performance problems automatically with feedback-directed learning software testing," Empirical Software Engineering, 2016, pp. 1–51.

[42] A. Danciu, A. Brunnert, and H. Krcmar, "Towards performance awareness in java ee development environments," in Proc. of the Symposium on Software Performance: Descartes/Kieker/Palladio Days, Nov. 2014, pp. 152–159.

[43] G. Young, "Cqrs and event sourcing. feb. 2010," URl: http://codebetter.com/gregy oung/2010/02/13/cqrs-and-event-sourcing.

[44] J. Kabbedijk, S. Jansen, and S. Brinkkemper, "A case study of the variability consequences of the cqrs pattern in online business software," in Proc. of the 17^{th} European Conference on Pattern Languages of Programs, Jul. 2012, pp. 2:1–2:10.

[45] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," ACM SIGACT News, vol. 33, no. 2, 2002, pp. 51–59.