Behavior-driven Load Testing Using Contextual Knowledge — Approach and Experiences

Henning Schulz
Novatec Consulting GmbH, Karlsruhe, Germany

Dušan Okanović, André van Hoorn
University of Stuttgart, Germany

Vincenzo Ferme
Kiratech S.p.A., Paradiso (Lugano), Switzerland*

Cesare Pautasso
Software Institute, USI Lugano, Switzerland

ABSTRACT

Load testing is widely considered a meaningful technique for performance quality assurance. However, empirical studies reveal that in practice, load testing is not applied systematically, due to the sound expert knowledge required to specify, implement, and execute load tests.

Our Behavior-driven Load Testing (BDLT) approach eases load test specification and execution for users with no or little expert knowledge. It allows a user to describe a load test in a template-based natural language and to rely on an automated framework to execute the test. Utilizing the system's contextual knowledge such as workload-influencing events, the framework automatically determines the workload and test configuration. We investigated the applicability of our approach in an industrial case study, where we were able to express four load test concerns using BDLT and received positive feedback from our industrial partner. They understood the BDLT definitions well and proposed further applications, such as the usage for software quality acceptance criteria.

ACM Reference Format:
Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. 2019. Behavior-driven Load Testing Using Contextual Knowledge — Approach and Experiences. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297663.3309674

1 INTRODUCTION

Load testing is a well-known measurement-based technique to assess the performance and related quality attributes of a system under synthetic workload [11]. However, an extensive amount of expertise and effort needed to create and conduct meaningful load tests hinders a systematic application in practice [5]. Particularly, defining test objectives and representative workload specifications requires a sound understanding of the production workload and the influences of events such as marketing campaigns.

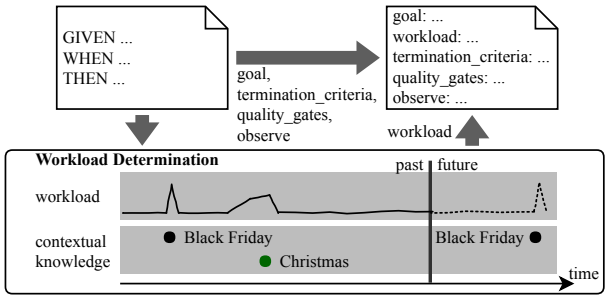


Figure 1: **Overview of the transformation of a BDLT definition to a declarative load test.**

Several approaches have been proposed to ease and automate certain aspects of load testing. Approaches to automatically extracting workload specifications from run-time data [17, 21] and declarative load testing [9], as well as automation of test execution [2], ease the load test specification and reduce the effort of running them. However, such approaches still require expertise, because they either do not provide means for automatically executing the extracted workload specifications, or require specifying the workload manually. Furthermore, selecting suitable run-time data requires the knowledge about influencing events.

In this paper, we adopt Behavior-driven Development [15], a functional software quality assurance technique, for load testing practices. In our approach, load test concerns, e.g., exploring CPU configurations for an expected workload, for instance, during Black Friday, are defined using the template-based natural Behavior-driven Load Testing language:

> ***Given*** *the next Black Friday,* ***when*** *varying the CPU cores between 1 and 4,* ***then*** *run the experiment for 1h and ensure the maximum CPU utilization is less than 60%.*

As illustrated in Figure 1, the definition is then transformed into a declarative load test using our existing approaches [9, 17]. Test parameters such as the test goal, termination criteria, and quality gates can be directly transformed. The test workload is determined using collected run-time data and by relying on the contextual knowledge — in this case the Black Friday — for forecasting to the described scenario. Hence, the workload can be defined without explicitly dealing with run-time data or workload specifications.

We evaluate our approach in an industrial case study assessing the ability of the Behavior-driven Load Testing (BDLT) language to express relevant load testing concerns as well as the usability,

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3309674

*This work was done while Vincenzo Ferme was at the University of Stuttgart.

benefits, and limitations of the overall approach. For that, we obtained feedback from the case study system's experts. We were able to express all load test concerns our industrial partner provided in BDLT. The feedback was consistently positive, especially regarding the usage of natural language. The main usage of our approach in this context would be the replacement of manually-defined load test scripts and as acceptance criteria of Scrum user stories [19]. Limitations of our approach are certain events requiring individual implementations and the natural language, which hinders complex expressions such as non-trivial subsets of parameter combinations.

The remainder of this paper is structured as follows. Sect. 2 provides the background of our approach. Sect. 3 details our approach. Sect. 4 presents the evaluation using the two case studies. Sect. 5 discusses related work. Sect. 6 concludes the paper and outlines future work. The supplementary material is available online [18].

2 BACKGROUND

In this section, we provide the background of our work including Behavior-driven Development (BDD), the ContinuITy [17] approach, and the BenchFlow [9] approach.

Behavior-driven Development (BDD) [15] builds on Test-driven Development (TDD), where the goal is not to specify the implementation of a software, but rather its expected results. It provides a natural language description of the expected behavior, rather than using some programming language. Usual starting points are Scrum user stories, and each generated test comes from a sentence in a story. This way also non-technical stakeholders can be more involved in the development, providing faster and better feedback, resulting in a clear set of acceptance criteria. The focus of the software development shifts to requirements and a business-oriented point of view, rather than implementation.

ContinuITy [17] uses recorded run-time data, including the users' requests, to automatically generate and evolve representative load tests. In this paper, we show how contextual knowledge, such as events that influence the user behavior, can be added to a workload specification. For instance, for a future marketing event such as the Black Friday, a workload specification that represents the expected workload during Black Friday is extracted. The WESS-BAS approach [21] is utilized to extract the workload specifications from recorded requests and ContinuITy adds information required for test execution, respecting API changes.

BenchFlow [8] is a framework for declarative performance test specification and automated execution on the API level. Using the declarative language, users can state the performance test concern and configure the process to answer the stated concern. The framework generates executable test artifacts and automates the execution of performance tests to reach the stated goal, e.g., load tests, exhaustive exploration tests, and tests with termination criteria. The declarative definition of performance tests allows for executing them without necessarily knowing the specific underlying technologies and tools required for this execution [9].

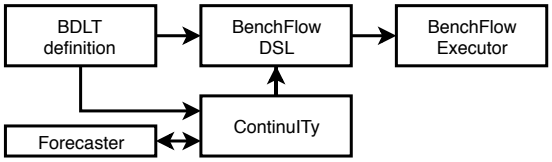


Figure 2: **An overview of our approach**

3 OUR APPROACH

The main idea of our approach is to let users state their performance concerns using a BDLT definition, which are then automatically transformed to load tests. However, there are two main challenges.

Common BDD practices are used to specify and test software functionality, and usually generate test code that is executed as part of the test. In our case, we target the performance testing domain, so we have to extend existing BDD languages to accommodate concepts for this specific domain, i.e., to generate load tests ensuring consistent results. This includes both the experimental setup and the workload data.

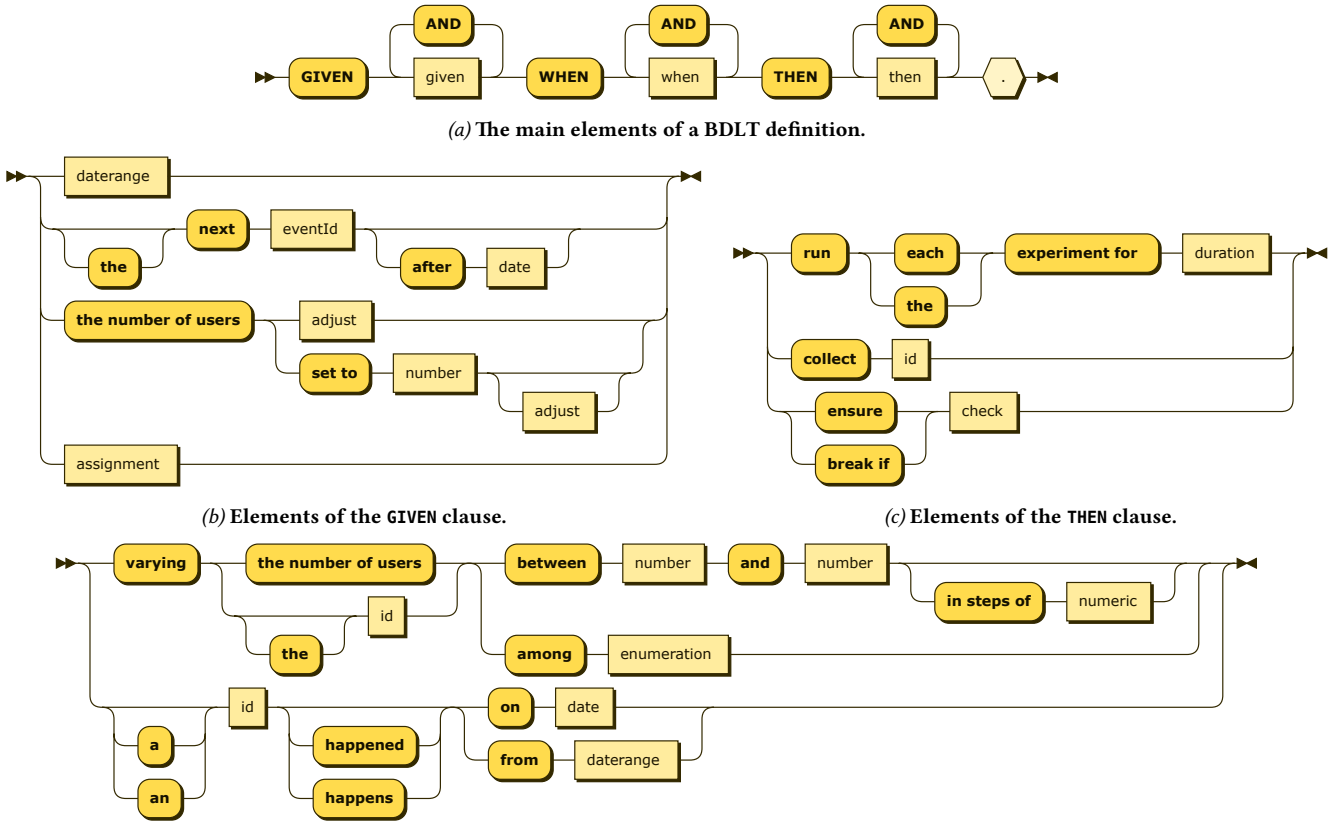
To simulate the behavior of real users, we want to use existing run-time data obtained from the operational use in production. From these data, we extract the workload information, i.e., the workload intensity and the workload mix [11, 21]. We also allow the users of our approach to test with an expected number of users, and possibly include certain events that can cause changes in the workload, e.g., holidays or outages. Based on the existing workload data and knowledge on how these events affect the workload, we use well-known techniques to predict the future workload and include the impact of the specified events.

In the following sections, we provide an overview of our approach (Sect. 3.1), the language that is used for the specification of behavior-driven load tests (Sect. 3.2), and how these definitions are transformed into BenchFlow test specifications (Sect. 3.3).

3.1 Overview of the Approach

A simplified overview of our approach is shown in Figure 2. The load test is specified using the new *BDLT* language, which contains the conditions that are present before starting the load test, the changes that occur when the test starts, and the stop and acceptance criteria for the test (see Section 3.2). Based on this definition, an instance of the *BenchFlow DSL* is generated. The executor interprets the provided BenchFlow DSL specification, generates executable test artifacts, manages the deployment of the application under test, runs the test, and collects test results. Details about the test execution with BenchFlow can be found in [8].

The workload for the load test is generated by *ContinuITy* based on the request logs and the pre-configured contextual knowledge. The test specification can use some historical workload, or a workload that can exist in the future. Using a historical workload includes running tests with an original intensity or with a constant intensity calculated from some specified time period, e.g., the maximum number of users for that time period. If a user wants to test the behavior of the system under some future workload, the *Forecaster* is used.



(a) **The main elements of a BDLT definition.**

(b) **Elements of the `GIVEN` clause.**

(c) **Elements of the `THEN` clause.**

(d) **Elements of the `WHEN` clause.**

Figure 3: **Elements of a BDLT definition.**

The Forecaster uses well-known techniques to forecast the intensity time series to future dates, respecting the possible influencing events, e.g., holidays, marketing or sport events, stored in the contextual information. These events are manually annotated in the recorded workload, but we plan to obtain them from external calendars and incident reporting tools. We also allow for the user to specify custom events and how they influence the workload, e.g., increased workload intensity due to a number of messages buffered during an outage. These events are pre-processed by ContinuITy, before being passed to the Forecaster. In the current implementation of our approach, we rely on the Prophet tool[1], but other forecasting tools can also be used.

3.2 Behavior-driven Load Test Language

In this section, we present the main features of the BDLT language for load test specifications.[2] Note that the target service specification is not a part of BDLT definitions, but rather pre-configured. Every BDD [15] is composed of three main elements: `GIVEN`, `WHEN`, and `THEN`. We adopt this description as follows (Figure 3a):

`GIVEN` (Figure 3b) is used to specify the starting conditions of a load test, namely: a workload from/for a specific period of time (*daterange*), a workload for some future (*next*) event, and specific test parameters (*assignment*). *The number of users* for the test can also be altered, i.e., *adjust*ed by a specified percentage from the original number, or *set to* a specific (constant) value. The *number* in the clause can be a numerical value or calculated based on the specified time period and context information.

`WHEN` is used to specify optional changes in the workload or the test configuration that occur during the experiment execution (Figure 3d). We can specify events, which will occur (*happen*) during the test execution, and allow the number of users and configuration parameters to be changed (*varying*). Event specification is the extension point for the custom event processing (see Section 3.1). If an extension for a specific event is registered — e.g., an outage — ContinuITy processes it before passing it to the Forecaster. In the outage example, the extension could calculate the number of requests that would be sent during the outage and use this number for forecasting the recovery spike to get a more accurate forecast. If no extension is registered, the event is directly forwarded to the Forecaster. Varying the test configurations and the number of users for a test can result in actually running several experiments, one for each combination of these parameters. This is particularly useful for exploratory testing, where using one test definition, multiple configurations can be tested [8].

In `THEN` (Figure 3c), the user specifies how long the test should *run*, which metrics to *collect*, when it should stop (*break if*), and what the acceptance criteria is (*ensure*).

[1]Prophet, https://github.com/facebook/prophet

[2]The full specification using the Extended Backus-Naur Form (EBNF) notation [1] is available in the supplementary material.

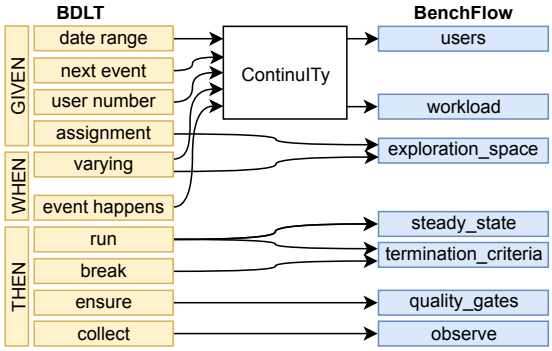


Figure 4: **Transformation of BDLT clauses to BenchFlow DSL elements. The number of users and the workload mix are pre-processed by ContinuITy.**

3.3 Transformation to Declarative Load Tests

The mapping of BDLT concepts to BenchFlow [9] is presented in Figure 4. Listing 1 presents an excerpt of the resulting BenchFlow test generated from the example in Sect. 1.

All the information related to the number of users, i.e., *daterange*, *next event*, *the number of users*, *event*, and *varying* the number of users, are first processed by ContinuITy to define the workload and stored into the *users* section of the BenchFlow test specification. In our example, ContinuITy determines the date of the next Black Friday and forecasts the intensity for that date. WESSBAS [21] is used to generate a Markov-based workload model which is transformed to *workload*.

Values for the test parameters and how they should be varied (the number of CPU cores to be explored in our example) are mapped to the *exploration_space* (*cpu* property).

The information from *run* is mapped to *steady_state* for individual experiments (1 h in our example), and to *termination_criteria* for the overall test duration. For four different CPU configurations in our example, the *max_time* is 4 h. Furthermore, *break* is also mapped to *termination_criteria* and *ensure* is mapped to the *quality_gates* (the maximum CPU utilization of 60 % in *max_cpu* property). *collect* is mapped to *observe*. *quality_gates* and *observe* are both based on the configuration of data collectors in the *data_collection* section, which is pre-configured.

4 INDUSTRIAL CASE STUDY

In this section, we present the results of our industrial case study. The goal of this case study is to investigate the applicability and the benefits of applying our BDLT approach in an industrial context. Hence, we address the following three research questions.

RQ1: How expressive is the BDLT language in regards to load test concerns of industrial use cases?

Because we aim to replace other load test definitions in industrial contexts with BDLT, RQ1 addresses its expressiveness. It is fundamental that the BDLT language is able to express industrial use cases.

RQ2: How would BDLT be used in industrial contexts?

```
configuration:
  users: ...
  load_function:
    steady_state: 1h
  goal:
    type: exhaustive_exploration
    exploration_space:
      shop-service:
        resources:
          cpu: [ '1', '2', '3', '4' ]
  quality_gates:
    services:
      shop-service:
        max_cpu: <= 60%
  termination_criteria:
    test:
      max_time: 4h
    experiment:
      type: fixed
      number_of_trials: 1
workload: # generated Markov chains
```

Listing 1: **BenchFlow test derived from the exemplary BDLT definition in Sect. 1.**

With this question, we investigate how practitioners would use BDLT. The usage both evaluates the applicability and leads to future improvements and extensions of the language.

RQ3: What are the benefits and limitations of using BDLT in comparison to defining load test scripts?

Here we are interested in general feedback of practitioners regarding benefits and limitations of our approach, which we express with RQ3.

In the following, we provide our methodology (Sect. 4.1), the data used in the case study (Sect. 4.2), the results of the case stuy (Sect. 4.3), a discussion of the results (Sect. 4.4), and the lessons we learned (Sect. 4.5).

4.1 Methodology

We apply our approach at an industrial partner from the logistics sector. Following DevOps practices, the company develops and operates an IoT system running in a Cloud environment, using Docker and Kubernetes. Devices are sending messages to an IoT endpoint which forwards the messages to the backend application via messaging queues.

In order to answer the research questions, we developed BDLT definitions that express the load test concerns the industrial partner has and collected feedback regarding benefits of these definitions. In doing so, we proceeded as follows. In two meetings, we presented our general research plan and discussed high-level architectural and organizational aspects of their IoT system. We then focused on one DevOps team that was working on load testing, and with them we defined the scope of the collaboration and received production data to use in our case study. In two iterations, we (1) defined the BDLT definitions according to our current understanding of the system and (2) refined with them the definitions and collected their feedback. In each of these meetings, we presented the

Table 1: **Overview of the BDLT definitions and the generated BenchFlow load tests.**

Name	`goal`	`load_function`	`steady_state`	`quality_gates`
configuration exploration	`exhaustive_exploration`	constant	1h	CPU load, message latency
continuous quality assurance	`load`	constant	1h	number of instances, cost
recovery spike	`load`	step function	2h	queue length
more devices	`load`	constant	5h	CPU load

BDLT definitions as well as generated load tests, i.e., BenchFlow DSL instances and graphs visualizing the workload specifications.

4.2 Input Data

We used the following data[3]: (1) the message logs of a small subset of devices from one week, consisting of a time stamp, a device ID, and a message type; (2) the load intensity over time, i.e., the number of messages per hour for one year; (3) contexts we identified, which influence the load intensities. The WESSBAS approach [21] was used to transform the message logs into a Markov-chain-based workload model representing the device behaviors and the relative frequencies (mix), with applied intensities adjusted with the influence of the identified contexts.

We identified two specific contexts: public holidays and recoveries from outages of the Cloud infrastructure that can happen irregularly. Public holidays turned out to decrease the intensity. Recoveries significantly increase the intensity, because the devices buffer all messages locally and send them during the recovery, which makes them particularly interesting for load testing. A recovery is detected by observing message rates, i.e., when the rate is below 10% of the average rate for that hour and weekday and then it spikes to two times the average rate. We label the spike as a recovery with the number of *buffered messages* calculated as the difference between the sent messages during the outage and the normally sent messages. For forecasting the workload intensity during expected future outages, we register an event extension for the event *outage* (see Sect. 3.2). Based on the clause ***When*** *an outage happened from <start> to <end>*, the extension calculates the expected number of buffered messages. The forecasting of the test workload is done using the determined number of buffered messages.

4.3 Experiments and Results

We developed four BDLT definitions describing our industrial partner's concerns. An overview is provided in Table 1. The *configuration exploration* test aims at finding the optimal system configuration. After that configuration has been established, the *continuous quality assurance* test is executed continuously, e.g., every night, to detect performance regressions. The *recovery spike* test prepares for load spikes that might occur in the future. Finally, the *more devices* test covers the foreseen scenario of adding more devices.

Configuration Exploration The BDLT definition for the *configuration exploration* test is provided in Listing 2. It uses the maximum expected intensity of the next three months to assess the performance of the SUT under different configurations of the CPU, number of instances, and RAM. Each experiment is to be executed for one hour, and the CPU load and message latency are to be compared against thresholds as quality gate. The transformation to a BenchFlow test is as follows. The `GIVEN` clause implies using a constant value as a load function. Because the *varying* keyword is used in the `WHEN` clause, the BenchFlow goal *exhaustive_exploration* is used, testing all configuration combinations. The `THEN` clause defines an execution time of at most one hour per experiment and *qualiy_gates* on CPU and message latency metrics. The constant load value is determined by the forecasting approach, as illustrated in Figure 5. The figure shows the observed intensity points as well as the learned model for the forecast. The load value is then extracted as the maximum of the forecasted intensities.

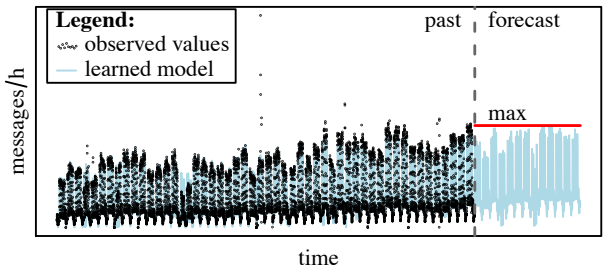


Figure 5: **Observed intensities and forecast for the configuration exploration test.**

```
GIVEN the next three months
 AND the number of users set to the maximum
WHEN varying the CPU cores between 0.5 and 4 in steps of 0.5
 AND varying the number of instances between 1 and 5
 AND varying the RAM among (1GB, 2GB, 4GB)
THEN run each experiment for 1 hour
 AND ensure the average CPU load is less than 15%
 AND ensure the message latency is less than 2 seconds
```

Listing 2: **BDLT definition: configuration exploration.**

```
GIVEN 2018
 AND the number of users set to the 95th percentile
THEN run the experiment for 1h
 AND ensure the number of instances is less than 3
 AND ensure the summarized cost is less than X
```

Listing 3: **BDLT definition: continuous quality assurance.**

Continuous Quality Assurance The *continuous quality assurance* test is a simple load test and is easily expressible as BDLT, as illustrated in Listing 3. Instead of relying on forecasted workload, this definition calculates the 95^{th} percentile of the number of users for

[3]For the sake of confidentiality, we do not provide the exact dates or values in the following and add randomly chosen obfuscation factors per hour, day, and week as well as for the global trend to all plots.

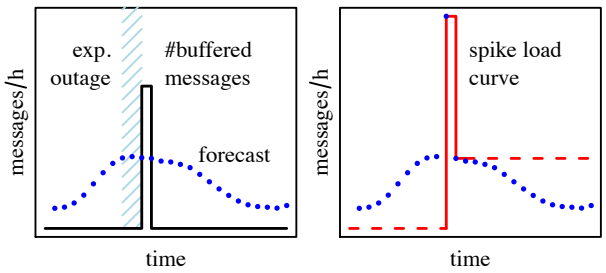


Figure 6: **Calculation of the number of buffered messages (left) and forecast of the recovery spike load curve (right).**

the year 2018, as stated by the `GIVEN` clause. Hence, the number of users is the same each time the test is executed, allowing for a comparison of test executions. The `THEN` clause defines to execute the test for one hour and to compare against the number of instances and cost thresholds. These metrics are of interest in this test because it is assumed to be executed in an environment with auto scaling in place. The BenchFlow test is generated similarly to the configuration exploration test, with the difference that the test goal is *load*, denoting a single load test. Furthermore, different metrics are used as *quality_gates* (number of instances and cost, which is retrieved from the Cloud services) and the intensity is calculated from the past data instead of a forecast.

```
GIVEN 2018/10/15 9:00
WHEN an outage happened from 2018/10/15 7:00
     to 2018/10/15 9:00
THEN run the experiment for 2 hours
 AND ensure the final queue length is less than 100
```

Listing 4: **BDLT definition: recovery spike.**

Recovery Spike The *recovery spike* test aims at preparing for load intensity spikes that might happen in the future because of outages. The BDLT definition is provided in Listing 4. Because the current intensity highly influences the number of messages that get buffered and thus, the spike height, the test focuses on a specific date in the `GIVEN` clause. The `WHEN` clause defines the expected outage by utilizing the custom *event* statement. The `THEN` clause defines to execute the test for two hours, which is one hour for the spike and one hour for normal load, and the queue length at the end of the test as the pass criterion. The BenchFlow test is generated as a simple load test, because there is no configuration exploration. The load function is a step function replaying the expected spike curve. The experiment duration is two hours, as defined, and the queue length is used as quality gate. Because of the custom event statement, the load function is forecasted in two steps. This is illustrated in Figure 6. In the first step, a forecast to the time range during which the outage is expected to happen is done. Then, the number of *buffered messages* is calculated as the sum of messages that would be sent during the outage time range. In the second step, that number is used as context for the forecasting, resulting in a spike curve, which is used as the load function.

More Devices The last relevant BDLT definition is for the *more devices* test and is provided in Listing 5. It covers the scenario when

```
GIVEN calendar week 5 in 2019
 AND the number of users set to the maximum increased by 30%
 AND the number of instances is 4
THEN run the experiment for 5 hours
 AND ensure the average CPU load is less than 20%
```

Listing 5: **BDLT definition: more devices.**

more devices will be added to the system at a known point in time in the future. This knowledge is unknown to the forecaster and has to be added as a user-defined input. For that, the `GIVEN` clause defines the date in the future when the devices will be added. The number of users to be used is defined as the maximum forecasted intensity increased by a given percentage, e.g., 30%. In addition, a custom configuration of the system is used, which is also to be used at that date. Here we use the number of instances as an example. Because there is only a fixed set of test parameters, there is no `WHEN` clause. The `THEN` clause defines running the experiment for five hours and using the CPU load as pass criterion. The generated BenchFlow test again has the *load* goal and a constant load function. The experiment is executed for five hours and there is a quality gate on the CPU load metric. The load intensity is determined by a forecast to the specified date, similar to the *configuration exploration* test. However, the forecasted intensity is increased by 30%, according to the statement in the `GIVEN` clause.

4.4 Discussion

We discuss the research questions based on the BDLT definitions and our industrial partner's feedback. Regarding RQ1, we were able to express all use cases named by our industrial partner using BDLT. However, we had to make use of the extension mechanism for the custom outage event. Hence, additional case-specific implementations had to be added. Because there is no silver bullet for custom events, such custom implementations are inevitable.

Regarding RQ2, the DevOps team members would use it instead of defining load tests manually, as they currently do. In addition, they noticed that the usage of the natural language makes the test definitions easily understandable for non-experts such as product owners. Hence, BDLTs could also be defined by non-experts. Furthermore, a BDLT could be used as an acceptance criterion of a Scrum user story.

RQ3 targets the benefits and limitations of BDLT. In general, our industrial partner found that our BDLT approach "has potential" and they are interested in further development. Especially, the use of natural language was rated positively, as mentioned before. The identified limitations of BDLT are the need for extensions for custom events, such as the outage event, and the current focus of our approach on HTTP APIs. For this reason, executing the generated tests in the context of this case study requires additional implementations, i.e., extend BenchFlow to support the used messaging protocol. Another limitation arised from applying BDLT to a more complex load test concern from our previous work [3], which is an exploration of a non-trivial subset of configuration possibilities. The BDTL language lacks in a concept of precisely describing such subsets. As a workaround, we can specify all configuration possibilities and accept more executed tests. Finally, our industrial partner

mentioned the requirement to compare test executions, similar to the continuous quality assurance test.

4.5 Lessons Learned

In our research, we learned several lessons, which we present in the following.

BDLT is easy to understand. Particularly, in the meetings with our industrial partner, they understood the BDLT definitions we presented well. Additionally, they were able to extend the definitions and express load test concerns on their own. Furthermore, they rated the language to be understandable for non-experts such as product owners, allowing utilization as acceptance criteria in Scrum user stories.

BDLT helps finding load test concerns, i.e., we noticed that discussing about load tests defined in BDLT reveals new concerns. In our meetings, the precise but well understandable load test definitions in natural language have formed a good basis for discussing the test's concern and shaping it. In addition, our industrial partner came up with new concerns that arose based on BDLT definitions we already had.

Some specific load test parameters require individual extensions , if they cannot be expressed through standardized language templates. For instance, the outage, which requires determination of the number of buffered messages first, is an example where standard processing is not sufficient. Therefore, we conclude that BDLT or related languages cannot be universal but need to be extensible.

Natural language entails limitations. Precisely, there can be constructs such as load test parameter combinations where natural language lacks in concise descriptions. As an example, it is hard to concisely describe non-trivial subsets of configuration possibilities. Hence, future works should focus on expressing such complex constructs and assessing the limitations of the natural language for load test definition.

5 RELATED WORK

In this section, we present related work in the research areas more closely related to the context of the paper.

Behavior-driven Development. BDD is a functional testing technique proposed to enable developers to specify the behaviour of the application under test, abstracting away from the details, and to use the provided specification to test the software. One of the most widely accepted and used representative of such techniques is Cucumber [24], a language and a tool for BDD of functional requirements. BDD techniques have been recognized as a valid and reliable source of information about how the system works [24]. Some BDD techniques have been proposed in different domains, as for example a safety verification behaviour-driven language and execution tool by Wang and Wagner [23]. To the best of our knowledge, there are no BDD techniques in the context of performance testing, although some behaviour-driven languages have been proposed as discussed in the next paragraph. In this paper, we propose BDLT to overcome the mentioned limitation.

Non-functional Quality Concern Specification Languages. Different languages for non-functional quality concern specification have been proposed in the literature [13]. The category of languages more closely related to the one presented in this paper is called Controlled Natural Languages (CNL) [12]. The CNL more closely related to our work is Canopus [4]. Canopus consists of a behaviour-driven language for performance test specification. Compared to Canopus, the language proposed in this paper is more rich, as for example it also enables the users to specify context information useful to automatically derive workload specifications.

Declarative Performance Engineering aims at providing methods and tools abstracting away the complexity of specifying and executing high quality performance evaluations, by providing abstractions and automation enabling performance engineering activity specification using declarative languages. The most prominent works in the area are by Walter et al. [22] and Ferme et al. [9]. Walter et al. introduced the term *Declarative Performance Engineering*, as an approach that "envisions to reduce the current abstraction gap between the level on which performance-relevant concerns are formulated and the level on which performance evaluations are actually executed". Ferme et al. proposed a tool named BenchFlow, able to automatically execute performance tests. Although the execution is automated, the definition of the test specification is still deemed to the users, and in particular the workload has to be manually specified. The approach we propose in this paper shifts away this need from the users and enables the users to automatically obtain workload specifications given a context of interest.

Workload Definition and Extraction. Different approaches for workload characterization have been proposed [7, 14]. These approaches extract different workload models from recorded requests, e.g., based on Markov chains [14, 21], extended finite state machines (EFSM) [20], or stochastic form-oriented models [6]. The main limitation of these works is the need for directly dealing with recorded requests and complex workload models. Additionally, not always re-executing past workload is sufficient. Different approaches, such as the one by Herbst et al. [10], as well as the mentioned Prophet tool, can be used for workload intensity forecasting. However, the proposed approaches are mainly used for capacity planning and have not been integrated into load testing approaches, yet. In this work, we propose an approach that integrates workload characterization and forecasting as a part of the test execution process, encapsulated by natural-language-based test definitions.

6 CONCLUSION AND FUTURE WORK

Despite its recognition, load testing is rarely used in practice, due to the high amount of expertise required to specify, implement, and execute load tests. In this paper, we address this issue by proposing an approach to Behavior-driven Load Testing (BDLT), allowing load test specification in natural language. By relying on collected contextual knowledge such as workload-influencing events (e.g., a marketing campaign), workload details to be used in the load test are abstracted away. Furthermore, BDLT allows to easily define complex concerns such as configuration parameter explorations.

We utilize our existing approaches [9, 17] to generate load tests based on BDLT definitions and collected run-time data.

In an industrial case study, we show the general applicability of BDLT. Despite minor limitations of natural language hindering complex statements, we were able to precisely express four different load test concerns in BDLT. The BDLT definitions were easily understood, also by non-experts, and foster collaboration. However, we identified the need for custom extensions of the language because of certain events that cannot be handled generally.

For future work, we propose focusing on extending the expressiveness of the BDLT language, because we identified limitations regarding natural language descriptions of non-trivial parameter combinations. Furthermore, more studies regarding its applicability in different domains are required. Also, as we identified Scrum acceptance criteria as a use case of BDLT, further use cases are of interest. Finally, we are planning to integrate more approaches to automated load test extraction and execution into BLDT, such as more extensive context-based test generation and microservice-based test modularization, and also natural language reporting of the test results tailored to stated performance concerns [16].

ACKNOWLEDGEMENTS

This work has been supported by the German Federal Ministry of Education and Research (grant no. 01IS17010, ContinuITy), the German Research Foundation (HO 5721/1-1, DECLARE), and by the Swiss National Science Foundation (project no. 178653). The authors would like to thank the industrial partner for participating in the case study.

REFERENCES

[1] 1996. ISO/IEC Information technology - Syntactic metalanguage - Extended BNF. *ISO/IEC 14977:1996(E)* (1996).

[2] Varsha Apte, T V S Viswanath, Devidas Gawali, Akhilesh Kommireddy, and Anshul Gupta. 2017. AutoPerf: Automated load testing and resource usage profiling of multi-tier internet applications. In *Proc. ICPE 2017*. 115–126.

[3] Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, Henning Schulz, and André van Hoorn. 2018. A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing. In *Proc. ECSA 2018*. 159–174.

[4] Maicon Bernardino, Avelino F Zorzo, and Elder M Rodrigues. 2014. Canopus: A Domain-Specific Language for Modeling Performance Testing. In *Proc. ICSEA 2014*. 157–167.

[5] Cor-Paul Bezemer, Simon Eismann, Vincenzo Ferme, Johannes Grohmann, Robert Heinrich, Pooyan Jamshidi, Weiyi Shang, André van Hoorn, Monica Villavicencio, Jürgen Walter, and Felix Willnecker. 2018. How is Performance Addressed in DevOps? A Survey on Industrial Practices. In *Proc. ICPE 2019*.

[6] Yuhong Cai, John C. Grundy, and John G. Hosking. 2007. Synthesizing Client Load Models for Performance Engineering via Web Crawling. In *Proc. ASE 2007*. 353–362.

[7] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. 2016. Workload Characterization: A Survey Revisited. *Comput. Surveys* 48, 3 (2016), 48:1–48:43.

[8] Vincenzo Ferme and Cesare Pautasso. 2017. Towards Holistic Continuous Software Performance Assessment. In *Proc. QUDOS@ICPE 2017*. 159–164.

[9] Vincenzo Ferme and Cesare Pautasso. 2018. A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments. In *Proc. ICPE 2018*. 261–272.

[10] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. 2013. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In *Proc. ICPE 2013*. 187–198.

[11] Zhen Ming Jiang and Ahmed E Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Trans. Soft. Eng.* 41, 11 (2015), 1091–1118.

[12] Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics* 40, 1 (2014), 121–170.

[13] Abderrahman Matoussi and Régine Laleau. 2008. *A Survey of Non-Functional Requirements in Software Development Process.* Research Report TR-LACL-2008-7. LACL.

[14] Daniel A. Menascé and Virgilio A. F. Almeida. 2002. *Capacity Planning for Web Services: Metrics, Models and Methods* (1st ed.). Prentice Hall, Upper Saddle River, NJ, USA.

[15] Dan North. 2006. Introducing BDD. https://dannorth.net/introducing-bdd/. (2006).

[16] Dušan Okanović, André van Hoorn, Christoph Zorn, Fabian Beck, Vincenzo Ferme, and Jürgen Walter. 2019. Concern-driven Reporting of Software Performance Analysis Results. In *Proc. ICPE 2019*.

[17] Henning Schulz, Tobias Angerstein, and André van Hoorn. 2018. Towards Automating Representative Load Testing in Continuous Software Engineering. In *Proc. ICPE 2018*. 123–126.

[18] Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. 2019. Behavior-driven Load Testing Using Contextual Knowledge—Approach and Experiences. (Feb. 2019). https://doi.org/10.5281/zenodo.2558279

[19] Ken Schwaber and Mike Beedle. 2001. *Agile Software Development with Scrum.* Prentice Hall PTR.

[20] Mahnaz Shams, Diwakar Krishnamurthy, and Behrouz Homayoun Far. 2006. A Model-Based Approach for Testing the Performance of Web Applications. In *Proc. SOQUA 2006*. 54–61.

[21] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. 2018. WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction – a Model-Driven Approach for Session-Based Application Systems. *Software and System Modeling* 17, 2 (2018), 443–477.

[22] Jürgen Walter, André van Hoorn, Heiko Koziolek, Dusan Okanovic, and Samuel Kounev. 2016. Asking "What"?, Automating the "How"? - The Vision of Declarative Performance Engineering. In *Proc. ICPE 2016*. 91–94.

[23] Yang Wang and Stefan Wagner. 2018. Combining STPA and BDD for Safety Analysis and Verification in Agile Development: A Controlled Experiment. In *Proc. XP 2018*. 37–53.

[24] Matt Wynne, Aslak Hellesoy, and Steve Tooke. 2017. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers.* O'Reilly UK Ltd.