dqualizer: Domain-Centric Runtime Quality Analysis of Business-Critical Application Systems

Sebastian Frank
University of Hamburg
Hamburg, Germany
sebastian.frank@uni-hamburg.de

Julian Brott
University of Hamburg
Hamburg, Germany
julian.brott@uni-hamburg.de

Dominik Kesim
University of Stuttgart
Stuttgart, Germany

Heiko Holz
Novatec Consulting GmbH
Leinfelden-Echterdingen, Germany
University of Education Ludwigsburg
Ludwigsburg, Germany
heiko.holz@novatec-gmbh.de

Matthias Eschhold
Novatec Consulting GmbH
Leinfelden-Echterdingen, Germany
matthias.eschhold@novatec-gmbh.de

André van Hoorn
University of Hamburg
Hamburg, Germany
andre.van.hoorn@uni-hamburg.de

ABSTRACT

The runtime quality of application systems — e.g., performance, reliability, and resilience — directly influences companies' business success. Over the last few years, corresponding analysis measures such as load tests or monitoring have become widespread in practice, and mature commercial and open-source tools have been developed. However, these measures are all at the technical level and not interpreted at the (business) domain level. At the same time, software architecture and software development approaches such as Domain-Driven Design (DDD), which are becoming increasingly widespread, essentially do not consider runtime quality concerns despite their criticality.

Our envisioned *dqualizer* approach aims to close the gap between the domain-specificity of application systems and the (technical) measures and findings of quality assurance utilizing a domain-centric approach. For this purpose, we integrate means to model and monitor runtime quality metrics into DDD-based techniques, e.g., Domain Story Telling, that enable domain experts to describe domain-centric runtime quality concerns. Our preliminary results comprise the prototypical extension of a domain story editor for specifying load and resilience tests and reporting test results. Using the editor, we gathered feedback from domain experts in a qualitative user study. Despite the editor's limitations regarding functionality and usability, the feedback indicated that domain experts are able to model runtime quality analyses.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; **Software fault tolerance**; • **Applied computing** → *Service-oriented architectures*; **Business process monitoring**.

KEYWORDS

domain-driven design, domain story telling, runtime quality

ACM Reference Format:
Sebastian Frank, Julian Brott, Dominik Kesim, Heiko Holz, Matthias Eschhold, and André van Hoorn. 2023. dqualizer: Domain-Centric Runtime Quality Analysis of Business-Critical Application Systems. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578245.3584853

1 INTRODUCTION

Domain-driven Design (DDD) techniques [25] foster communication between domain and technical experts and focus on developing systems according to the needs of the business domain. Since today's application landscapes become larger and more complex as the number of users increases, more flexible architectural styles, such as microservices, and suitable landscapes, such as DDD, are needed. Conclusively, the runtime quality of application systems — e.g., performance and resilience — becomes more critical as the impact of these attributes increases with use. Over the last few years, corresponding analysis measures such as load [16] and resilience tests [2] have become widespread in practice, and mature commercial and open-source tools have been developed.

When using these analysis measures, developers, for example, are interested in specific methods' response times to optimize execution times. In contrast, domain experts are interested in domain-related metrics, such as the number of products sold in a specific domain. However, DDD techniques only focus on functionality and do not consider runtime qualities, despite their criticality. Thus, analysis measures are still located and interpreted at the technical

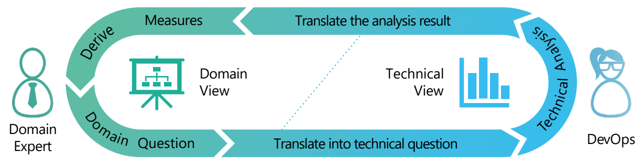


Figure 1: The dqualizer vision: Domain experts can state questions and interpret analysis results on the domain level.

level. Hence, technical data must be translated between the technical and the business metric levels, which poses a major challenge.

With the dqualizer approach presented in this paper, we envision closing the gap between the domain-specificity of application systems and the (technical) measures and findings of quality assurance utilizing a domain-centric approach shown in Figure 1. Domain experts' questions are mapped to technical, executable tests. Vice versa, dqualizer translates the analysis results into a representation understandable for domain experts. The approach integrates means to model and monitor runtime quality metrics into DDD-based techniques that enable domain experts to describe domain-centric runtime quality concerns. Subsequently, the dqualizer approach intends to gather the relevant metrics with open-source tools and aggregate the analysis results on the domain level.

As a first step, we extend an existing editor for domain stories [27] to support the annotation of domain elements to specify Runtime Quality Analysis (RQA) tests. We base the format of the required data on a well-known quality scenario format [3] and use terminology and explanations suitable for non-technical stakeholders. We derive the RQA test inputs from the established load testing tool JMeter [24] and the resilience engineering tool Chaos Toolkit [6]. The editor prototype presents RQA results in a text-based report similar to the work by Okanović et al. [22].

We conducted a qualitative user study in which four domain experts from industry had to solve four tasks on a modified cinema application example [13]. The participants' performance in solving the tasks and feedback provided in a questionnaire indicate that domain experts are able to specify RQA tests using our approach. However, the usability of the editor still needs to be improved.

To summarize, this work makes the following contributions:

- A presentation of dqualizer's vision, concept, and components for domain-driven specification and comprehension of RQA tests
- The prototypical editor and the underlying concept for annotating domain stories
- The results and lessons learned from the user study that evaluates the capabilities of the editor and dqualizer approach

2 BACKGROUND

2.1 Domain-Driven Design

Domain-driven Design (DDD) [25] is a holistic architecture and development approach for software systems in technically complex business domains. The domain determines the architectural decisions of a software system and is, therefore, always the top priority for developers, architects, and business experts. The collaboratively developed ubiquitous language supports their communication.

Domain Storytelling (DST) [13] is a method for analyzing and exploring the domain. It is conducted interactively in workshops and relies on communication between all stakeholders, who identify important business events, business processes, necessary neighboring systems, and technical terms. DST is a collaborative modeling technique that emphasizes transforming domain knowledge into software by telling and visualizing stories about the problem domain [13]. The goal is a mutual domain model that is maintained throughout the life cycle of the system [7].

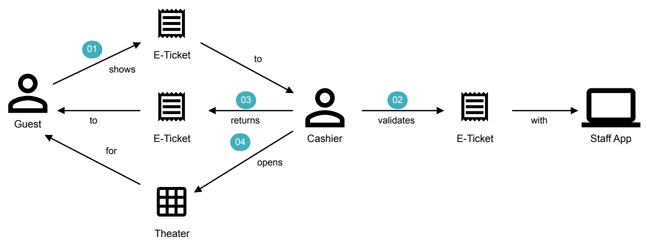


Figure 2: Modified *Entrance Control* story of the Arthouse cinema example [13]

Hofer and Schwentner [13] suggest using a simple pictographic language to avoid complex notations or interactions inherent to formal modeling languages. This notation allows a simple representation of people, their activities, and business processes. The DST language consists of three element types, i.e., *actors*, *work objects*, and *activities*. Actors are a story's protagonists and can be persons, groups, or software systems. Actors always play an active role in a domain story and have a label describing their role. Actors create, use, or exchange work objects, e.g., documents, things, and digital objects. Actors and work objects are connected by arrows called activities. Activities are always verbs in the domain language, whereas actors and work objects are nouns. Domain stories can be modeled using a tool, e.g., the WPS Domain Story Modeler [27].

Figure 2 shows a domain story based on the Arthouse cinema example [13]. We modified the example in our work to represent a digital workflow that involves an application. In this example, the *guest*, the *cashier*, and the *Staff App* are actors. The elements *e-ticket* and *theater* are work objects. The arrows labeled with *shows*, *returns*, *opens*, and *validates* are the activities. Thus, we can read the domain story as follows: *"The guest shows the e-ticket to the cashier. Next, the cashier validates the e-ticket with the cinema's Staff App. After the e-ticket has been successfully validated, the cashier returns the e-ticket to the guest and opens the theater."*

2.2 Runtime Quality Analysis

Dqualizer focuses on quantifiable runtime quality properties — particularly performance and resilience — which must be considered over the entire software life cycle, i.e., from requirements, through architecture and implementation to testing, and operation [4]. Requirements can be expressed, e.g., using quality scenarios [3].

Application Performance Monitoring (APM) assists developers and stakeholders in continuously monitoring a system's runtime behavior and identifying deviations in quality characteristics [12]. APM involves interdependent activities that may be executed concurrently, namely (1) data collection, (2) data storage and processing, (3) data presentation, and (4) data interpretation and use.

Load testing [16] is a process to gain information on a system's capability to fulfill its functional and non-functional properties under specific conditions to uncover problems. It emerged as an addition to functional testing approaches to ensure that systems meet their quality requirements by observing their behavior in a given environment. In general, load testing requires three steps, i.e., (1) load design, (2) execution, and (3) analysis. JMeter [24] is an example of a state-of-the-art load testing tool.

Resilience describes a system's capability to recover from conditions that cause deviations from the system's normal state [26]. As such, resilience is a measurable metric related to other dependability metrics, e.g., reliability and fault tolerance. Testing resilience requires preparation and a well-designed testing strategy. Common practices for resilience testing are fault-injections [14] and chaos experiments [2]. Both strategies aim to test different dependability metrics, such as resilience, but differ in methodology. In traditional fault-injection approaches, the injection engine injects the fault into the hardware or software layer to investigate a system's dependability measures in a test environment. Chaos experiments can be, for example, automated using the Chaos Toolkit [6].

3 RELATED WORK

DDD, as introduced by Evans [7], and DST [13] do not include practices for domain-centric modeling of quality requirements. Evans [7] uses annotations to describe quality properties as business rules. Hofer and Schwentner [13] use similar annotations to describe additional properties of domain story elements, but not related to quality. Both annotation types lack a well-defined formalism for modeling domain-centric quality requirements.

Perillo et al. [23] provide a solution that uses annotations to create more detailed domain model descriptions for Java-based applications. However, their solution is entirely technical and uses Unified Modeling Language (UML) diagrams to describe domain models, even in DDD contexts. In contrast to dqualizer, this approach disregards quality requirements in its annotations and domain model.

Quality annotations are also common to other modeling approaches, such as UML [21]. For instance, the MARTE profile [10] extends UML with performance and availability annotations. Although MARTE uses a formal language focusing on quality requirements, it does not describe its properties from a domain perspective. While MARTE allows separating the domain model from its implementation, formal specifications do not fit into the DDD landscape as they lack the expressiveness of a ubiquitous language.

Le et al. [19] propose a domain-specific language (DSL) that extends UML classes with additional domain properties. However, no solution is available to transform the annotated model into RQA tests. This would require more specific details like the workload type, or experiment setting that domain experts typically can not provide. Therefore, extensive communication between technical and domain experts is still necessary to create RQA tests.

In summary, related works either lack a focus on domain models or testable quality specifications. Thus, the concerns of domain experts regarding runtime quality can not be sufficiently considered.

4 DQUALIZER APPROACH

4.1 Objective and Overview

The dqualizer approach aims to close the gap between the business domain and technology through an innovative approach for domain-centric RQA of business-critical application systems. We hypothesize that DDD is suitable for performing RQA for domain-specific questions. In summary, dqualizer's primary objectives are i) definition and interpretation of questions regarding runtime quality at the domain level and ii) translation of runtime quality issues

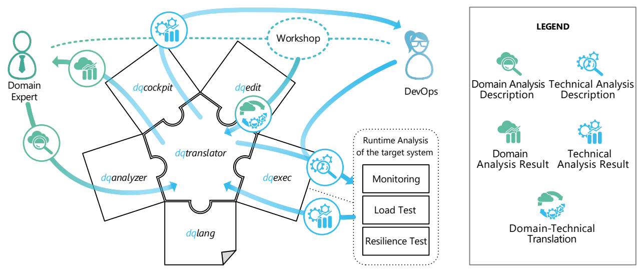

Domain Expert
dqcockpit
Workshop
dqedit
DevOps
dqtranslator
dqanalyzer
dqexec
dqlang
Runtime Analysis of the target system
Monitoring
Load Test
Resilience Test
LEGEND
Domain Analysis Description
Technical Analysis Description
Domain Analysis Result
Technical Analysis Result
Domain-Technical Translation

Figure 3: Overview of dqualizer's envisioned architecture

and results between business and technical domains using a mapping — as illustrated in Figure 1. While we focus on the domain experts' perspective, we emphasize that technical experts (e.g., DevOps engineers) can also state questions regarding the domain, e.g., whether a service is involved in critical business processes.

Besides its primary goals, dqualizer must also be able to deal with several challenges. Due to their flexibility, a microservice-based system's architecture will likely experience frequent changes, which can (partially) invalidate the initial mapping. Thus, dqualizer aims to detect and report mismatches between the mapping and the architecture. Furthermore, techniques for a (semi-)automatic mapping extraction from monitoring data should reduce the effort in handling such situations and providing an initial mapping. In addition, dqualizer also aims to provide assistance or at least limited functionality in not ideal-typical situations, i.e., where industry applications have not been designed using DDD with APM in place.

Figure 3 depicts the dqualizer architecture with the components and the connections to existing RQA tools. DDD-based modeling languages and workshop-based techniques, such as event storming and DST, are to be extended in dqualizer to support domain questions regarding runtime qualities. At the same time, the results of the technical quality analysis — i.e., monitoring as well as load and resilience tests — are to be adequately represented at the domain level so that domain experts can evaluate the results. In the dqualizer architecture (see Figure 3), this is implemented by a language (*dqlang*), a mapping editor (*dqedit*), an analysis configurator (*dqanalyzer*), and a dashboard for result interpretation (*dqcockpit*).

To link DDD and technical quality analyses, dqualizer requires a mapping model and transformations that enable the translation between the business and technical levels as well as the associated quality scenarios and analyses. This is realized by the central translator (*dqtranslator*) and the analysis execution (*dqexec*). We aim to publish the dqualizer tooling under an open-source license.

4.2 dqualizer Components

4.2.1 dqanalyzer. The dqanalyzer component enables domain experts to specify and perform RQA without profound knowledge of the underlying technical infrastructure and analysis tools such as JMeter [24]. For example, the question "Can a contract be processed within 3 s even under high load" could be answered with load tests. The dqanalyzer user input concept is based on the ubiquitous language domain experts are familiar with. Thus, this component subsumes the extensions to existing DDD techniques, e.g., the extended domain story editor presented in Section 5.1. Our extensions aim to hide technical details that domain experts can not provide.

However, due to the technical complexity of the stated domain questions, RQA must be performed at the technical level. Thus, the domain-specific question entered by the domain experts must be translated by the dqtranslator component.

4.2.2 dqedit. The dqedit component is the editor for the mapping necessary to translate domain questions to actual RQA. The editor can be used during DDD-based workshops, e.g., for event storming or DST, where the domain elements are identified and defined. Technical experts must then provide the necessary implementation details, e.g., which services represent certain actors. Furthermore, the technical experts must provide default values for inputs kept abstract on the domain level, e.g., the number of requests for a "normal" workload. Thus, dqedit enables DevOps engineers to translate domain questions to RQA tests and technical metrics.

4.2.3 dqlang. Processing inputs and specifications from various domains requires a generic language model. Domain-driven architecture and development approaches such as DDD already provide the domain-motivated and technology-neutral ubiquitous language, which is reflected in the communication between domain experts and DevOps as well as in the software architecture. DDD-based modeling languages and workshop-based techniques, such as event storming and DST, will be extended in dqualizer to support questions regarding RQA. The dqlang is a collection of languages utilized by dqualizer e.g., the DST extension for domain-level RQA specification or the language for defining mappings in dqedit.

4.2.4 dqtranslator. Dqtranslator assures consistent information processing despite differing language models. From the input perspective, domain-specific analysis descriptions from dqanalyzer and technical technology requirements from dqedit are processed and translated to the generic dqlang model. This allows dqualizer to use a consistent language autonomous from domain-specific characteristics internally. To output data to users, the analysis results are translated from the dqlang model back to the domain-specific input language initially provided via dqanalyzer. This enables dqualizer users to perform input specification and output interpretation in a uniform language model they are familiar with.

4.2.5 dqexec. The dqexec component executes the analysis requests specified in the dqanalyzer component. The specification is received in the generic format of dqlang and then mapped to the input model of the external analysis tooling. For APM analysis, we plan to support open-source tools[1], e.g., inspectIT and Kieker. Load tests can, for instance, be executed with JMeter and resilience tests, e.g., with Chaos Toolkit. Further third-party analysis and test tools can be integrated by providing an appropriate mapping from dqlang to the tool's input and output models. Besides delegating the RQA execution, dqexec is also responsible for choosing the most appropriate analysis tool, repeating tests to reach a certain accuracy, and enriching the tests with tool-specific default values. We envision dqexec to deliver preliminary results for long-running RQA, e.g., by utilizing (fast) simulations before executing experiments.

[1]https://openapm.io/

4.2.6 dqcockpit. Dqcockpit is a dashboard visualizing the results of executed RQAs. The results are translated from dqlang to the appropriate domain-specific language to enable the user to interpret the data without prior knowledge of the toolset used by dqexec. Finally, dqcockpit offers a management overview dashboard that displays the current system state based on automatically extracted analysis data which is translated to the domain level.

5 PRELIMINARY RESULTS

In this section, we present the extended domain story modeler and outline a user study conducted to evaluate our initial dqualizer concept. Further details are available as supplementary material [17].

5.1 Extended Domain Story Modeler

We extended the existing Domain Story Modeler [27] to act as the front-end of the dqualizer approach for domain experts, i.e., they can use it to express domain questions, trigger RQAs, and investigate the results. Thus, the extended modeler represents the component dqanalyzer and partially dqcockpit. Since other components of the dqualizer approach still need to be implemented, the approach is not fully executable. However, the extended modeler allows the early evaluation of concepts for dqualizer in user studies.

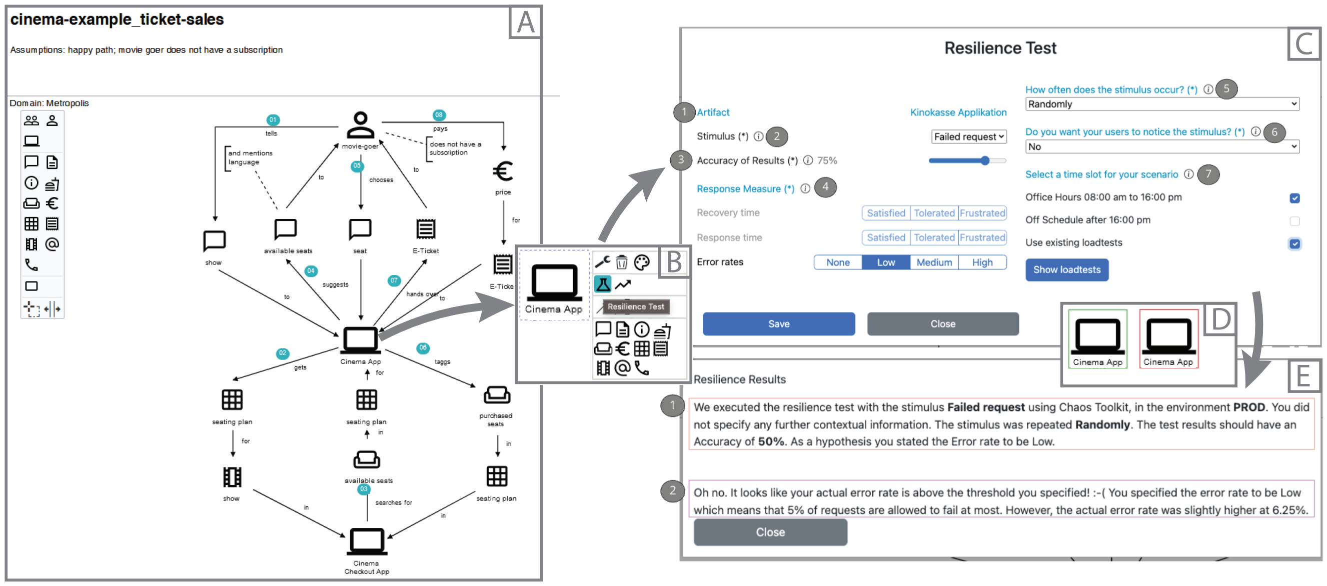
Figure 4 shows the extended modeler's workflow comprising (i) annotating domain story elements, (ii) specifying RQA tests, and (iii) investigating results. These steps are detailed in the following.

5.1.1 Annotation-based Modeling. We use annotations to specify RQA tests in domain stories. An annotation creates a relationship between the annotated element and the information declared in the annotation, e.g., as in other modeling languages such as UML profiles. We separate the annotation into two components: (i) the view where domain experts define the RQA test and determine the content of the annotation, and (ii) the visual reference that distinguishes annotated elements from other elements.

First, the domain expert selects an element in a domain story (see Figure 4 (A)) for adding a RQA test annotation. Then, a context menu (see Figure 4 (B)) shows RQA test types available for the selected element. The developed prototype currently supports load and resilience tests. In the future, the dqualizer mapping rules could restrict the availability of RQA tests on elements. However, the prototype does not support this function as the dqtranslator has not been implemented yet. Next, the Graphical User Interface (GUI) opens a separate view (see Figure 4 (C)) for specifying the RQA test.

When the domain expert saves the RQA test, the modeler checks the annotation for completeness. As a visual reference, a red border line surrounds incomplete annotations if the annotation misses mandatory information. Otherwise, a green border line indicates a completed annotation (see Figure 4 (D)).

5.1.2 RQA Test Specification. We need additional information to execute RQA tests with tools like JMeter and ChaosToolkit. Our prototype outputs a data model using the Javascript Object Notation (JSON) format for further processing in dqualizer. As a general structure of our data model, we use the quality scenario description template [3]. This allows us to describe the content of RQA tests in a well-established structure using four predetermined keywords, i.e., *artifact*, *stimulus*, *environment*, and *response measure*.



Resilience Test

1 Artifact: Kinokasse Applikation
2 Stimulus (*): Failed request
3 Accuracy of Results (*) 75%
4 Response Measure (*)
Recovery time: Satisfied | Tolerated | Frustrated
Response time: Satisfied | Tolerated | Frustrated
Error rates: None [✓ Low] Medium High
5 How often does the stimulus occur? (*): Randomly
6 Do you want your users to notice the stimulus? (*): No
7 Select a time slot for your scenario
Office Hours 08:00 am to 16:00 pm ✓
Off Schedule after 16:00 pm ☐
Use existing loadtests ✓

Resilience Results

1 We executed the resilience test with the stimulus **Failed request** using Chaos Toolkit, in the environment **PROD**. You did not specify any further contextual information. The stimulus was repeated **Randomly**. The test results should have an Accuracy of **50%**. As a hypothesis you stated the Error rate to be Low.

2 Oh no. It looks like your actual error rate is above the threshold you specified! :-(You specified the error rate to be Low which means that 5% of requests are allowed to fail at most. However, the actual error rate was slightly higher at 6.25%.

Figure 4: Workflow of our prototype showing (A) the *Ticket Sales* story in the editor, (B) the context menu to annotate elements, (C) the view for specifying resilience tests, (D) the test status, and (E) the view displaying the analysis result.

Throughout our concept and prototype implementation, we aim to hide technical complexity from domain experts to reduce the cognitive load and increase usability. Thus, for technical terms, we use alternate terminology more suitable for non-technical stakeholders. As indicated by the question marks in Figure 4 (C), we also provide help texts to assist domain experts in understanding the requested information. We do not request exact threshold values for quality metrics since domain experts usually have difficulties providing them. Instead, we use Apdex levels [1] that describe the meaning of the values, e.g., that the user is "satisfied" (see Figure 4 (C4)). The values associated with the levels in the prototype are examples and must be chosen according to the domain. In future versions of dqualizer, technical experts can provide the fitting values.

We have gathered evidence that domain experts do not know how long a RQA should take to achieve reliable results. Therefore, we use the term *confidence* (*accuracy* in earlier versions) with a value between 0% a 100% for all tests. Other dqualizer services must determine a suitable duration based on the confidence value.

Our prototype currently implements data models and views to describe resilience and load tests in a uniform manner.

Resilience Test Specifications. We derive the required inputs for a resilience RQA test in dqualizer from the required inputs of chaos experiments [2] in ChaosToolkit as shown in Table 1. Figure 4 (C) shows how the inputs are displayed in the GUI. The annotated element is automatically assigned as the artifact (C1).

The experiment type in *stimulus* (C2) can either be *Failed request*, *Late response*, or *Unavailable*. These three options are the most used in ChaosToolkit for application-level fault injections [18]. Currently, we do not consider infrastructure or network injections, as they require technical information not available in domain stories.

The *response measure* (C4) comprises *Response time*, *Recovery time*, and *Error rate*. We use the Apdex levels *Satisfied*, *Tolerated*,

Table 1: Mapping the inputs of Chaos-Toolkit-based resilience tests to our terminology and the scenario keywords

ChaosToolkit	Alt. Terminology	Scenario Keyword
fault type fault repetition injection type duration	type occurrence injection type confidence	stimulus
target service	artifact	artifact
execution context	environment	environment
steady-state-hypothesis	response measure	response measure

and *Frustrated* from the example provided by the Apdex Users Group [1] for response times and recovery rates. The *Error rates* object is separated into the three Apdex levels *Low*, *Medium*, and *High*. We defined *Medium* as below 5%. The *occurrence* (C5) determines how often the *stimulus* triggers against the *artifact*. At this point, we opted for two choices, i.e., *once* and *more than once*. The *environment* (C7) differentiates between *Production* and *Testing*. In testing environments, we apply the standard approach in resilience testing to use data from the production environment that reflects user behaviors between regular and post office hours [2].

Load Test Specifications. We map the terminology of JMeter [24] to scenario keywords and provide an alternative terminology as shown in Table 2. We assign two JMeter concepts to the *stimulus*, i.e., the *sampler*, and *loop count*. The *sampler* requires more configuration as there are many different samplers, e.g., Hypertext Transport Protocol (HTTP). Currently, we consider HTTP samplers only, but future versions of dqualizer may add more types.

Table 2: Mapping the inputs of JMeter-based resilience tests to our terminology and scenario keywords

JMeter	Alt. Terminology	Scenario Keyword
sampler loop count	sampling type confidence	stimulus
endpoint	artifact	artifact
number of threads ramp-up	base load / highest load growth rate	environment
assertion	response measure	response measure

The *environment* describes the load profile. Currently, the prototype differentiates between the load types *load peak* and *continuous load*. Both types are common workload patterns [8] occurring in many other contexts, such as Cloud Computing. Further, we assign the concepts *number of threads* and the *ramp-up* time to the keyword *environment*. The term *highest load* applies for changes in the workload after a dedicated time frame derived from the *growth rate*. The term *base load* only applies when the workload does not change during the load test. The terms *confidence* and *artifact* are equivalent to the ones in the resilience data model. The *baseLoad* is divided into three Apdex Levels, i.e., *Low*, *Medium*, and *High*. Similarly, the *highest load* is expressed using the three Apdex levels *Very High*, *Extremely High*, *Dangerously High*. As of now, the *Growth rate* can take three values, i.e., *Linear*, *Cubic*, and *Quadratic*.

We assigned the concept of an *endpoint* to the *artifact* and assigned the *assertion* rules to the *response measure*. The *response measure* comprises the response time and the optional result metric. The latter describes a range of metrics that may be included in the analysis results report. We have included the metrics response times and two percentiles, i.e., the 90th and the 95th percentile.

5.1.3 RQA Test Result Presentation. For displaying the results of the RQA tests, we need to ensure that the report is understandable and contains enough information to make the result comprehensible to domain experts. Tools such as Grafana[2] display data in graph-like visualizations. However, domain experts may not have the necessary knowledge to derive conclusions from these reports.

We base our concept for reporting analysis results on the report generator by Okanović et al. [22]. Thus, our result report (see Figure 4 (E)) consists of two components. The first component is a summary statement (E1) which summarizes the details of the RQA. If we consider that the execution of one or more RQA tests can take several hours or days, domain experts may have forgotten details about the tests. Therefore, the summary allows domain experts to get an overview of the previously modeled RQA test at one glance. The second component is the conclusion statement (E2), which includes details of the executed RQA test in natural language. Thus, the report enables domain experts to understand the relationship between the *response measure* and the result. The conclusion statement also summarizes a test run as either successful or failed.

[2]https://grafana.com/

Since our concept allows the execution of multiple RQAs, multiple analysis results can be displayed simultaneously. For this purpose, the report view is separated by the headlines *Resilience Test Results* and *Load Test Results*. Currently, there are no backend services that can provide analysis data. Therefore, we prepared a set of possible result descriptions for the expert evaluation to illustrate the concept.

5.2 Qualitative User Study

We conducted a qualitative user study to evaluate the modeler's capabilities in supporting domain experts in specifying RQA tests and to gather feedback regarding the further development of dqualizer tooling. The research questions are formulated as follows:

RQ1: How well does the proposed modeling concept improve the currently implemented processes of domain experts to model RQA tests on the domain level?

RQ2: Are domain experts capable of giving the information we need for modeling RQA tests on the domain level within our proposed modeling concept?

RQ3: How well can domain experts model their business questions within our proposed modeling concept?

RQ4: How understandable is the representation of specification elements for modeling resilience and load tests within our proposed modeling concept?

RQ5: How understandable is the representation of RQA results within our proposed modeling concept?

5.2.1 Method. We are unaware of other established solutions focusing on modeling non-functional requirements in the context of DDD and DST. It is thus infeasible to compare our approach to existing solutions in a quantitative evaluation design. Greenberg and Buxton [9] argue that quantitative study designs may even be harmful to evaluating novel ideas, especially during the prototype design, by creating circumstances that constrain the experts' subjective feedback. Therefore, the most feasible approach to validate our concept and answer the stated research questions is to ask domain experts for their judgment in a qualitative user study.

Isenberg et al. [15] also state that in evaluation contexts that require domain experts, there is no need for generality but rather for transferring the needs of these domain experts to people with similar knowledge and needs. In such circumstances, Isenberg et al. [15] argue that four participants are enough to yield valuable results. Therefore, we invited four experts from two companies to participate in the evaluation. Two participants work in the insurance domain, and two in the taxing domain.

During the evaluation, the participants received a set of tasks that they needed to solve with the assistance of our prototype. We defined the tasks for the fictional domain *Arthouse cinema* provided by Hofer and Schwentner [13] so that participants did not need to provide their own. After the participants completed the tasks, we asked them to fill out a questionnaire to gather feedback and ask for details about the participants' observations. The questionnaire consists of 39 questions [17] and encourages the participants to report missing features, problems, and missing information in single-choice questions using a 5-point Likert scale [20] and free text. Some questions are based on the System Usability Scale [5].

5.2.2 Tasks. We briefly explained the *Arthouse cinema* domain which was the tasks' basis. Hofer and Schwentner [13] do not provide any architectural documentation that complements the domain stories. Therefore, we created a simplified component diagram and a fictional scenario to give participants more context [17].

We created four tasks that each had to be solved by the participants within 10 minutes. We designed the tasks to be independent of each other. The tasks focus on different features of the prototype. Task 1 requires creating a resilience test to verify that an error rate is below 5% in the *Ticket Sales* domain story. Task 2 requires creating a load test in which the response time stays tolerable. Task 3 asks the participants to create a resilience test with simulated load tests. Task 4 is an open task where the participants have to phrase a business question of their choice for the domain story *Entrance Control*. Task 4 intends to find pitfalls and misleading design choices in modeling a business question for an observed domain. To provide a more realistic use case, we derived the tasks' business questions from an example business question collected in a previous workshop. This example includes a concrete metric threshold for error rates of 5%, which we also used in the tasks. The tasks 1–4 help to answer the research questions RQ1, RQ4, and RQ5. Task 4 is also mandatory to find an answer to RQ3.

5.2.3 Procedure. We conducted the interviews remotely on Microsoft Teams. The participants received a document that contained additional information and instructions for preparing the prototype locally on their machines. The instructions were sent to the participants more than three days before their scheduled interview.

The interviews consisted of three parts. In part one, we explained the procedure and invited the participants to ask questions. In part two, we provided the task description, waited five minutes for the participants to read it, and answered their questions. Then, we proceeded to the solving of the four tasks. After each task, we asked questions about the participants' solutions before presenting the correct solution and discussing any deviations. After each task, we asked the participants if the test was successful, the quality threshold was satisfied, and why the test did (not) succeed. Finally, we asked the participants to answer the questionnaire.

5.2.4 Results.

RQ1. In the questionnaire, the participants rated with a median value of 3.5 out of 5 that they feel confident to use the prototype in their work. The participants also agreed that the prototype is easy to use, with a median of four. However, there are still improvements necessary to increase usability. The participants stated the need for (i) more extensive documentation regarding the test parameters, (ii) extensive tool demonstration examples, (iii) the possibility to change the tool's language, (iv) a way to provide actual values instead of the predefined values.

RQ2. During the evaluation, we did not observe that domain experts were unable to provide the information required for modeling resilience and load tests. The participants' answers indicated that they did not miss any features. However, some reported problems understanding some of the information, particularly for modeling resilience tests. Most of the feedback from the participants included the lack of precision in providing response measures. We assumed that domain experts should work solely with labeled levels, e.g., "high", using the Apdex measurement index. However, even a small indication of the exact values for the response measure would increase the understandability of modeled RQA tests.

RQ3. After working on task 4, the participants agreed with a median of 4.0 out of 5 that they did not miss any features. Nevertheless, the participants mentioned several times that they would like more control over specifying the response measures, e.g., the load design, by providing exact percentage values. Furthermore, one participant wanted to include a value for availability in a resilience test. We observed a lack of interaction between resilience tests and load tests, i.e., the prototype does not connect them to simulate load behavior for resilience tests. In task 4, we noticed that almost all participants modeled the business questions similarly to each other. One participant stated that it is difficult to think of a business question in such a short time.

In the open feedback, two participants explicitly stated that the approach would be a valuable progression toward a solution that allows meaningful modeling of runtime analysis tests. One of them appreciated that it would allow having everything in one place, i.e., the modeled test, the domain story, and the analysis results.

RQ4. We noticed that almost every participant had read through the information texts several times before deciding on the modeling specification. The participants voiced critical feedback regarding the understandability of the fields *Response Measure*, *How often should the stimulus occur*, and *Should the users be affected by the stimulus*. However, all participants agreed (4 or 5) that they understood the requested information for modeling resilience and load tests, except for one participant on load testing. While modeling the load tests, the participants had difficulties understanding the load peak design and the checkboxes for including the *Result metrics*. In these cases, the information texts helped the participants to decide. Moreover, we have noticed that most participants mistakenly interpreted the *accuracy* as another term for availability or p-value expressing statistical significance.

RQ5. The participants agreed with a median of 3.5 out of 5 that the analysis results were rather understandable. Moreover, we also received positive feedback regarding the summary of their modeled test in the analysis results. The participants could identify whether the test was successful but could not explain why. Only in some cases did the analysis results include example values regarding the measured response times. According to the participants' feedback, a detailed summary of key facts for a test run would be better suited.

The participants also stated that the textual summary is irritating and difficult to read. Another issue is that the summary suggests an interpretation made by the application, tempting the domain experts not to make interpretations themselves. Most participants stated that this could cause false beliefs about the application.

6 DISCUSSION

In the following, we summarize the lessons learned from the user study and discuss the threats to the study's validity.

6.1 Lessons Learned

- Domain experts can model their business questions easily and are able to provide the required information within our modeling concept. However, the understandability of the information and information fields needs to be improved.

- The representation of the RQA results within our proposed concept is understandable from the point of view of a domain expert, although the results are missing detailed information with regard to the test runs. In addition, a more compact and structured representation compared to prose text is needed.
- Another issue is that the prototype currently allows modeling resilience and load tests on each element of the domain story. This may not be meaningful in the context of DST as it would not make sense to model any type of test on a particular domain story element.
- It turned out that the Apdex ratings alone are insufficient to make decisions towards specifying the response measures. Therefore, we need to complement it with custom values and actual reference values to help domain experts to get a better understanding of the individual information fields.
- We are limited by design decisions in the existing Domain Story Modeler, e.g., technically, we can not identify elements displayed in a domain story. Therefore, revising the existing modeler or developing a new modeler is necessary to overcome some of the aforementioned limitations.

6.2 Validity Concerns

In this first evaluation, we have identified three validity concerns. The selection of participants may have influenced the results, as three of them participated in a previous dqualizer requirements workshop. The participants' experience with our modeling concept and idea may result in a bias toward our prototype. However, the interviewed experts can also evaluate the satisfaction of their requirements best, which is vital in a first evaluation. Nonetheless, we have to ensure the validity of future evaluations.

Further, our questionnaire may not provide enough value for our research questions. We have compensated for this using the NASA-TLX [11] as a guideline for creating our questionnaire and based it on questions from similar published qualitative studies.

Lastly, the tasks may not reflect the use cases of domain experts. However, we designed the closed tasks based on real business questions and metrics from previous workshops. Furthermore, we reduced this risk by creating an open modeling task, where domain experts must specify their own RQA test.

7 CONCLUSION

This work introduced the dqualizer approach and its envisioned components for enabling domain experts to specify and interpret RQA tests using Domain-Driven Design techniques. Our editor prototype allows domain experts to specify RQA tests by annotating elements in domain stories and presents the results as text-based reports. As indicated by the results of a qualitative user study, domain experts are able to successfully specify RQA tests using the editor. However, feedback regarding usability and functionality needs to be considered in further development.

In future work, we plan to develop and evaluate further dqualizer concepts and components, e.g., mapping and test generation. We also aim to extend and reevaluate the editor's usability and functionality. Particularly, we plan to supplement the Apdex levels with actual values, revise the result presentation, add support for specifying more complex tests spanning multiple domain elements, and utilize mapping information to limit actions on domain elements.

ACKNOWLEDGMENTS

This research was supported by the Federal Ministry of Education and Research (BMBF), grant number 01IS22007A/B.

REFERENCES

[1] Apdex Users Group 2023. Apdex Performance Index. https://www.apdex.org.
[2] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41.
[3] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice* (4 ed.). Addison-Wesley Longman Publishing Co., Inc., USA.
[4] André B Bondi. 2014. *Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice.* Pearson Education.
[5] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.
[6] Chaos Toolkit Team. 2023. ChaosToolkit. https://chaostoolkit.org
[7] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley.
[8] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. 2014. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications.* Springer.
[9] Saul Greenberg and Bill Buxton. 2008. Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM, 111–120.
[10] Object Management Group. 2019. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. https://omg.org/spec/MARTE/1.2/PDF
[11] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology.* Vol. 52. Elsevier, 139–183.
[12] Christoph Heger, André van Hoorn, Mario Mann, and Dušan Okanović. 2017. Application performance management: State of the art and challenges for the future. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering.* ACM, 429–432.
[13] Stefan Hofer and Henning Schwentner. 2021. *Domain Storytelling: A Collaborative, Visual, and Agile Way to Build Domain-Driven Software.* Addison-Wesley.
[14] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. 1997. Fault injection techniques and tools. *Computer* 30, 4 (1997), 75–82.
[15] Tobias Isenberg, Petra Isenberg, Jian Chen, Michael Sedlmair, and Torsten Möller. 2013. A systematic review on the practice of evaluating visualization. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2818–2827.
[16] Zhen Ming Jiang and Ahmed E Hassan. 2015. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering* 41, 11 (2015), 1091–1118.
[17] Dominik Kesim. 2023. Supplementary Material. https://doi.org/10.5281/zenodo.7651402
[18] Dominik Kesim, André van Hoorn, Sebastian Frank, and Matthias Häussler. 2020. Identifying and prioritizing chaos experiments by using established risk analysis techniques. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).* IEEE, IEEE, 229–240.
[19] Duc Minh Le, Duc-Hanh Dang, and Viet-Ha Nguyen. 2018. On domain driven design using annotation-based domain specific language. *Computer Languages, Systems & Structures* 54 (2018), 199–235.
[20] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).
[21] Object Management Group. 2017. Unified Modelling Language Specification. (2017). https://www.omg.org/spec/UML/
[22] Dušan Okanović, André van Hoorn, Christoph Zorn, Fabian Beck, Vincenzo Ferme, and Jürgen Walter. 2019. Concern-driven reporting of software performance analysis results. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering.* ACM, 1–4.
[23] José Roberto C Perillo, Eduardo M Guerra, and Clovis T Fernandes. 2009. Daileon: a tool for enabling domain annotations. In *Proceedings of the Workshop on AOP and Meta-Data for Software Evolution.* ACM, 1–4.
[24] The Apache Software Foundation. 2019. JMeter. https://jmeter.apache.org
[25] Vaughn Vernon. 2016. *Domain-driven design distilled.* Addison-Wesley.
[26] Marco Vieira, Katinka Wolter, Alberto Avritzer, and Aad van Moorsel. 2012. *Resilience Assessment and Evaluation of Computing Systems.* Springer Science & Business Media, 2012.
[27] WPS 2023. WPS Domain Story Modeler. https://egon.io/.