Interactive Elicitation of Resilience Scenarios Based on Hazard Analysis Techniques

Sebastian Frank[1,2], Alireza Hakamian[2], Lion Wagner[2], Dominik Kesim[2], Christoph Zorn[2], Jóakim von Kistowski[3], and André van Hoorn[1]

[1] Department of Informatics, University of Hamburg, Germany
{firstname.lastname}@uni-hamburg.de
[2] Institute of Software Engineering, University of Stuttgart, Germany
[3] DATEV eG, Nürnberg, Germany

Abstract. *Context.* Microservice-based architectures are expected to be resilient. *Problem.* In practice, the elicitation of resilience requirements and the quantitative evaluation of whether the system meets these requirements is not systematic or not even conducted. *Objective.* We explore (1) the usage of the scenario-based Architecture Trade-Off Analysis Method (ATAM) and established hazard analysis techniques, i.e., Fault Trees and Control Hazard and Operability Study (CHAZOP), for interactive resilience requirement elicitation and (2) resilience testing through chaos experiments for architecture assessment and improvement. *Method.* In an industrial setting, we design a structured ATAM-based workshop, including the system's stakeholders, to elicit resilience requirements. To complement the workshop, we develop RESIRIO — a semi-automated, chatbot-assisted, and CHAZOP-based approach — for elicitation. We evaluate RESIRIO through a user study. The requirements from both sources are specified using the ATAM scenario template. We use and extend Chaos Toolkit to transform and automate two scenarios. We quantitatively evaluate these scenarios and suggest resilience improvements based on resilience patterns. *Result.* We identify 12 resilience scenarios in the workshop. We share lessons learned from the study. In particular, our work provides evidence that an ATAM-based workshop is intuitive to stakeholders in an industrial setting and that stakeholders can quickly learn to use RESIRIO in order to successfully obtain new scenarios. *Conclusion.* Our approach helps requirements and quality engineers in interactive resilience requirements elicitation.

Keywords: Interactive Elicitation · Requirements Engineering · Resilience · Hazard Analysis

1 Introduction

Context and Problem. An intrinsic quality property of the microservices architectural style is resilience, i.e., the system meets performance and other Quality of Service (QoS) requirements despite different failure modes or workload variations [21]. However, real-world postmortems [13] often show that systems suffer either unacceptable QoS degradation or recovery time. It is necessary to assure system resilience in the context of microservice-based architectures. The first step is to elicit resilience requirements, which is the focus of this paper.

Practitioners who use Chaos Engineering [2,18], including tools such as Chaos Toolkit (CTK) [8] for resilience testing, require to (1) think about hazards [17] as causes of QoS degradation, (2) set up chaos experiments by specifying failure mode types and hypotheses of expected quality behavior, and (3) execute each experiment to detect deviations from the hypotheses. This approach lacks the systematic identification of causes of a hazard through hazard analysis methods.

Objective. We contributed to this problem in our previous work [15], which serves as a foundation for this paper, but lacks a systematic process for elicitation and specification. In the context of an industrial system, we now integrate hazard analysis techniques [17], i.e., Fault Tree and Control Hazard and Operability Study [10] into more systematic, interactive requirement elicitation processes and use scenarios as a more formal description of requirements. Scenarios enable resilience testing through resilience experiments (aka chaos experiments) for architecture assessment and improvement. Therefore, our research question is: *How to leverage hazard analysis techniques to interactively elicit resilience scenarios, which can be utilized to evaluate resilience through resilience experiments and suggest architectural improvements quantitatively?*

Research Overview. To answer our research question, we devise a method consisting of three main activities (as illustrated in Figure 1). Our main contributions lie in interactive resilience requirements elicitation, where we propose an ATAM-based workshop and chatbot-assisted tooling for elicitation purposes. The outcome of the elicitation activity is a set of structured scenarios that are the basis for requirements assessment and architecture design improvement.

Main Contribution 1*: Workshop-based Elicitation.* The Architecture Trade-Off Analysis Method scenario template [3] is a mechanism for eliciting quality requirements and consists of the following elements: (1) source, (2) stimuli, (3) artifact(s), (4) system's environment, (5) its response, and (6) response measure.

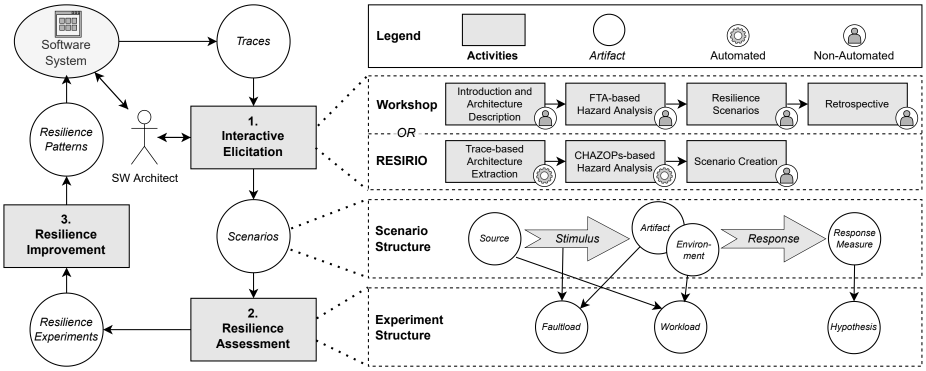


Fig. 1: Overview of this work and its contributions

ATAM has already been used in practice to elicit and specify quality requirements other than resilience, e.g., availability, performance, and maintainability. Thus, we hypothesize that ATAM can be adopted for effectively eliciting resilience requirements and evaluating them through chaos experiments. Therefore, we use the ATAM scenario template to describe resilience requirements in semi-structured textual language. The aim of the ATAM method is to assess architectural design decisions to find trade-offs between quality requirements and identify risks early. Therefore, (1) eliciting precise statements over quality requirements, (2) eliciting precise statements over architectural design decisions, and (3) evaluating whether the architectural design decisions satisfy the quality requirements are major goals in ATAM. In this paper, we do not perform a trade-off analysis, but use an ATAM-based workshop to elicit and specify resilience requirements by involving system stakeholders. Our structured workshop also comprises a hazard analysis based on the Fault Tree Analysis (FTA) [17].

Main Contribution 2: Chatbot-based Elicitation. Traditional elicitation techniques, such as interviews or our workshop, impose a high amount of time on participants, require locations to meet, and the presence of expert analysts. Therefore, as also outlined by various researchers, e.g., Rietz [26], there is a strong need for novel elicitation approaches. We designed an interactive elicitation technique called RESIRIO that utilizes a combination of a chatbot and an architecture visualization. First, RESIRIO extracts the system's architecture and applies a CHAZOP-like method to identify hazards from Jaeger and Zipkin traces. Then, the chatbot assists the engineer in the scenario specification. To evaluate RESIRIO's usability and effectiveness, we conduct an expert user study.

Results. The workshop's result is a set of 12 resilience scenarios. We use CTK to transform two scenarios into experiments, and conduct a measurement-based resilience evaluation. The evaluation results suggest that the architecture does not take temporary failures when services communicate with each other into account. Resilience patterns [1] describe architectural changes to enable applications to handle failures gracefully and recover from them. We improve system resilience by applying the *retry* pattern [22,21]. We validate the improvement by re-executing the respective scenario. Furthermore, our user study gives evidence that RESIRIO helps novice requirements engineers in fast requirements elicitation, but has limitations in specifying various and precise scenarios. Regarding the chatbot, engineers preferred the fast input of Quick Replies over writing text.

Summary of Further Contributions. In addition to the interactive elicitation techniques, the paper makes the following contributions:

- Automating scenario execution using CTK for measurement-based evaluation.
- We share lessons learned that benefit both practitioners and researchers regarding resilience requirement elicitation, evaluation, and improvement.
- Artifacts — including scenarios, resilience experiments, and results of the experimentation — are available online [27].

This paper is a revised and extended version of our workshop paper [11]. Our new contribution is the RESIRIO approach and its evaluation. Due to space limits, we consolidated parts of the workshop-based elicitation and the experiments.

2 Industrial Setting: Domain Context and System

The system's purpose is to calculate payments. An accounting department's wage clerks use the payment accounting system to calculate each registered employee's income taxes. The payment accounting system has to gather data from health insurance providers and send its results to the corresponding tax office to execute the calculations. This process presumes that a company that wants to use the payment accounting system provides its employee and tax information to the health insurance provider and tax offices.

All of the payment accounting system's tasks are currently taken care of by a monolithic legacy system. In peak times, up to 13 million calculation requests have to be handled in a day or single night. Under normal circumstances, this number is significantly lower. In order to handle such varying loads more efficiently, stakeholders desire a better scaling system. Therefore, the old system is being replaced by a more scalable microservice-based Spring application. The investigated part of the system under study, which is still under development, consists of seven services as detailed later. The system is deployed to a Platform as a Service (PaaS), i.e., Cloud Foundry (CF). Together with the industrial partner, we decided on a scenario-based approach, as our industrial partner already employed ATAM for other quality attributes.

3 Workshop-Based Elicitation and Specification

This section elaborates on the planning, execution, and results of the workshop.

3.1 Structured Workshop Approach

Before the workshop, we received documentation regarding the architecture of the system. This allowed us to specify an architecture model of the system, including a component diagram and an explanation of the implemented components. Using ATAM, we required to know key architectural design decisions. Therefore, knowing the architecture description in advance allowed us to focus more on the hazard analysis and developing resilience scenarios.

The full-day workshop consisted of four sessions leveraging different methods. The moderators explained each technique and method at the beginning of each session. The participants were stakeholders of the system and comprised two software architects, one product owner, and one quality assurance engineer.

Session 1: Introduction and Architecture Description for achieving a common understanding of the workshop process and the system's architecture. (1) We resolved misunderstandings regarding the elicited architecture description through asking questions, and (2) refined the prepared architectural models.

Session 2: Hazard Analysis to identify potential causes for degradation in QoS. Index cards were used as a means to collect hazards. Afterward, the participants arranged the hazards and their causes in a fault-tree-like fashion. To not break the participants' creative flow, we relaxed the strict construction rules of fault trees, e.g., we allowed events having multiple parents, which resulted in a graph. For this reason, we refer to this session's result as a *fault graph*. Note that the (directed acyclic) fault graph can be transformed into an equivalent fault tree by creating duplicate sub-trees for nodes having more than one parent.

Session 3: Resilience Scenarios for collecting and prioritizing resilience scenarios based on the previously identified hazards. We provided a table based on the ATAM scenario template. Then, the stakeholders jointly created scenarios by informally analyzing the fault graph in a sequence driven by the associated severity (in descending order) of the hazards.

Session 4: Retrospective to collect feedback about the workshop from the participants and to inform them about the next steps, which comprise (1) refinement resilience requirements, and (2) execution of resilience experiments.

3.2 Workshop Results

Elicited Architecture Description: Figure 2 shows the component diagram of the system as specified in the first session of the workshop. It describes a snapshot of the system as used in the workshop and the subsequent activities. It represents a typical microservice-based architecture. As such, the system is deployed to a CF and contains several services. Each service has its own PostgreSQL database. The only exception is the *Calculations* service, which employs a Mongo database. The *API-Gateway* service handles all incoming connections and routes all communications. A *Eureka* service is employed to provide service discovery for all internal components. The *Frontend* service is the only external component that a user can access directly. The *Calculations* service is the central hub of the system since the calculation of payments is the system's main feature. Once this service receives a calculation request from the gateway, it collects all necessary data asynchronously from the other services. The *Companies* service is used to handle data the *Frontend* displays, but is not relevant for the calculation.

Hazard Analysis: Figure 3 shows the fault graph created in the second workshop session. The stakeholders agreed on unavailability or long response of settlement calculations as the main system hazard. Therefore, *user's settlement*

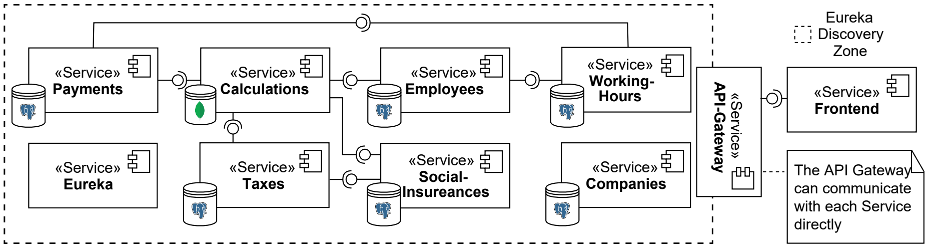


Fig. 2: Component Diagram of the Payment Accounting System

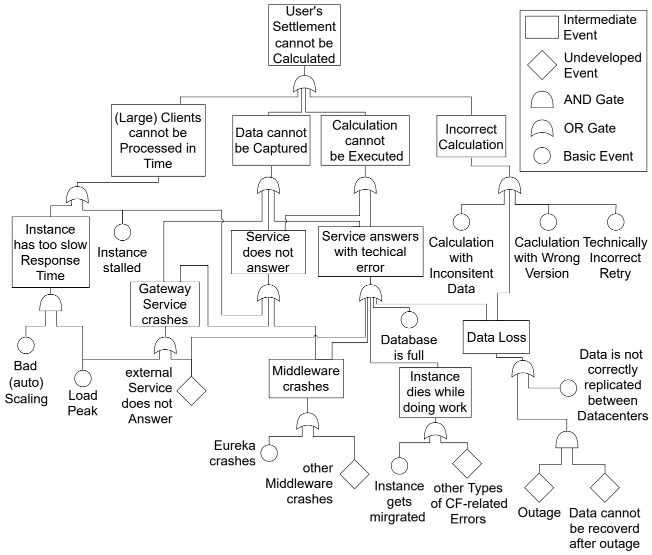

User's Settlement cannot be Calculated
Intermediate Event
Undeveloped Event
AND Gate
OR Gate
Basic Event
(Large) Clients cannot be Processed in Time
Data cannot be Captured
Calculation cannot be Executed
Incorrect Calculation
Instance has too slow Response Time
Instance stalled
Service does not answer
Service answers with techical error
Calculation with Inconsitent Data
Caclulation with Wrong Version
Technically Incorrect Retry
Gateway Service crashes
Database is full
Data Loss
Bad (auto) Scaling
Load Peak
external Service does not Answer
Middleware crashes
Instance dies while doing work
Data is not correctly replicated between Datacenters
Eureka crashes
other Middleware crashes
Instance gets mirgrated
other Types of CF-related Errors
Outage
Data cannot be recoverd after outage

Fig. 3: Cleaned Fault Graph

can not be calculated is the top event in the fault graph. The stakeholders analyzed possible causes from the top event until we reached basic events that we could not further decompose. We connected different causes by logical operators, i.e., *AND* and *OR*. For example, users can not calculate their settlement if it is not processed in time. This can occur when the assigned *instance stalls* OR *responds to slow*. We argue that the latter can be experienced if the system receives a sudden *(work)load peak* AND its *(auto) scaling does not work correctly*. The hazards at the leaf nodes are potential candidates for fault/failure injection during resilience experiments and can be initiated by tools such as CTK. The stakeholders selected and prioritized the set of resilience experiments.

Resilience Scenarios: We gave the participants an empty table according to the ATAM scenario template with the columns (1) source, (2) stimulus, (3) artifacts, (4) environment, (5) response, and (6) response measure. Further, we explained the meaning of each column to the participants, who then added index cards to the table. We began by identifying possible *sources*. The *stimuli* and *artifacts* were then derived from the previously created fault graph. The *environment* represents different time periods when the identified stimuli occur. The *responses* are the stakeholders' assumptions about how the system should respond to the particular stimulus. The *response measures* are based on their internal Service Level Objectives (SLOs). For example, a workload peak resulting in a system failure was transposed into multiple scenarios. Users of the system are the source since they cause the load peak. The respective stimulus is the workload peak itself. A service was chosen as the artifact to represent that a load peak can influence all service instances. As the environment, the payslip calculation period was chosen to imply an existing base workload. At last, the stakeholders chose the responses and response measures based on their SLOs.

ID	Short Name	Source	Stimulus	Artifact	Environment	Response	Response Measure
01	Peak(LinCo)/ Ser/Abr	User	Linear increasing load peak (cold start)	Service	Payslip calculation period	All requests are handled correctly and in time	Wage calculation $\leq$ 1 s, in 99 % of the cases, payslip calculation $\leq$ 20 s (300 Employees)
02	Peak(ExCo)/ Ser/Abr		Exponentially increasing load peak (cold start)				
03	Peak(LinCo)/ Ser/NoAbr		Linear increasing load peak (cold start)		Not during payslip calculation period		
04	Peak(ExCo)/ Ser/NoAbr		Exponentially increasing load peak (cold start)				
05	Failure(CF)/ Ins/Ber	Cloud Foundry	Instance terminates	Instance	During wage calculation	User unaware, calc. correct & in time, $\geq$1 instance running	
06	Bug/Ins/Ber	Bug				Developer gets notified	Developer gets notified within 5 min
07	Failure(MW)/ Bac/Ber	Middleware Operator	Middleware terminates	Backend	During wage calculation	Abort calculation, caller will be notified	Notification arrives within 1 s in 99 % of the cases
08	Failure(MW)/ Bac/Abr		Middleware terminates but recovers		Async. payslip calculation	Process is aborted but can be picked up	Restarts and SLOs are satisfied
09	Depl(GW)/ FroBac/NoIdle	Deployment	Gateway terminates	Front- and Backend	Not Idle	Frontend shows error, gateway restarts	Downtime of gateway instance is below 1 min
10	Failure(GW)/ FroBac/NoIdle	Technical Issue					
11	Failure(SerE)/ Ser/Ber	Receiving Service	Service terminates	Sending Service S, Receiving Service R	During wage calculation, no instance available	Error message and services R restarts	Downtime is below 1 min
12	Failure(SerE)/ Ins/Ber				During wage calculation, one instance not available	Calculation correct and in time	Wage calculation response time is below 2 s

Table 1: Scenarios created during the workshop

The stakeholders elaborated 12 resilience scenarios, summarized in Table 1. Scenarios 01 to 04 are different variations of an unexpected load peak, including linear and exponentially increasing loads. Scenarios 05 and 06 describe the failure of a single service instance. Scenarios 07 and 08 are about middleware failures. Scenarios 09 and 10 revolve around gateway failures. Lastly, Scenarios 11 and 12 describe the failure of multiple instances. Actors such as end-users, elements of the CF platform, different bugs, and technical issues caused by the middleware or deployment artifacts and issues intrinsic to individual services of the system comprise the established sources. In total, all scenarios can affect all services. The environments cover different states of the system according to the identified system domain context, e.g., payslip calculation periods or simply services being non-idle independent of the different calculations. The response and response measures were specified by the stakeholders based on their internal SLOs.

Retrospection: The brief retrospective at the end of the workshop showed that the participants were satisfied with the agenda, content, and outcomes. However, comments were made concerning time management.

3.3 Key Lessons Learned

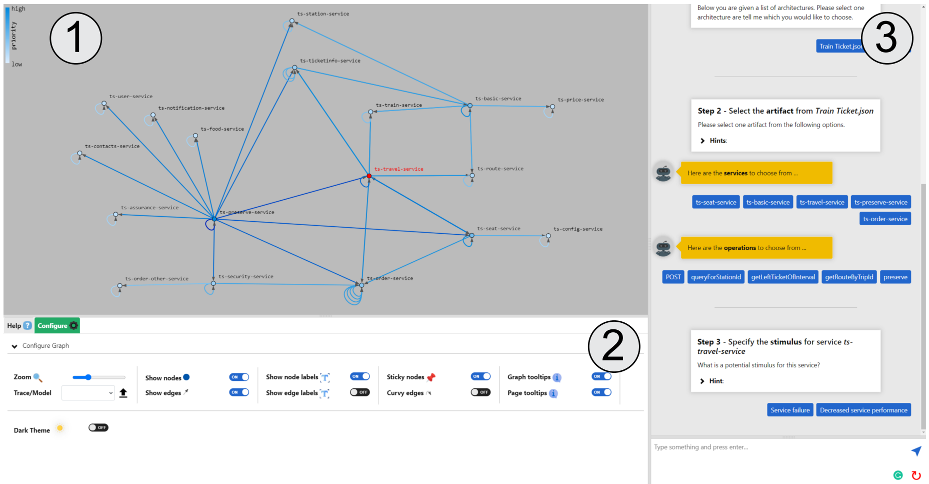
After the workshop and the retrospective session, this section presents the key lessons learned. For more details on each lesson, please refer to [11].

- Elicitation of resilience requirements involves hazard analysis.
- ATAM is a useful method to adopt resilience elicitation.
- Loose adoption of formalisms is already good enough.
- The workshop requires considerable refinement that can be done “offline”.
- A tightly planned one-day workshop is sufficient.
- The resilience elicitation helps to refine “classical” QoS requirements.

4 Chatbot-Based Elicitation and Specification

Although we tried to design the workshop (see Section 3) to be lightweight, its execution still requires significant time and effort. We also noticed that some of the elicited scenarios, particularly their triggering conditions, are rather generic, e.g., Scenarios 11 and 12 describe that a service terminates due to a service failure. Therefore, we designed an approach called RESIRIO to complement the workshop. RESIRIO aims to accelerate the elicitation of simple resilience scenarios through automation and interaction with visualizations and chatbots.

Similar to the workshop sessions, the RESIRIO process consists of three steps: architecture analysis, hazard analysis, and scenario creation. However, the first two steps are fully automated. In the *architecture analysis*, RESIRIO extracts the system's architecture from Jaeger or Zipkin execution traces [23]. This analysis substitutes the workshop session for the architecture elicitation, specification, and refinement. Since the fault tree analysis is difficult to automate, RESIRIO applies a CHAZOP-like technique in the *hazard analysis* to identify



Configure Graph

Zoom
Trace/Model
Show nodes ON
Show edges ON
Show node labels ON
Show edge labels OFF
Sticky nodes ON
Curvy edges OFF
Graph tooltips ON
Page tooltips ON
Dark Theme OFF

Below you are given a list of architectures. Please select one architecture are tell me which you would like to choose.

Step 2 - Select the **artifact** from *Train Ticket.json*

Please select one artifact from the following options.

Hints:

Here are the **services** to choose from ...

Here are the **operations** to choose from ...

Step 3 - Specify the **stimulus** for service *ts-travel-service*

What is a potential stimulus for this service?

Hint:

Type something and press enter...

Fig. 4: The graphical interface of the RESIRIO prototype

potential hazards and prioritize suggestions. Finally, in the *scenario creation*, the architecture is visualized, and a chatbot assists the user in the specification of concrete scenarios. The following sections detail these three steps.

We implemented the RESIRIO approach as a prototype. Figure 4 shows the graphical interface of the prototype, which consists of the (1) architecture graph, the (2) configuration view, and the (3) chatbot. Regarding the architectural documentation, including architectural views, design decisions, and use of technologies, please refer to the work by Zorn [30].

4.1 Architecture Analysis

Traces from Zipkin and Jaeger are the source of the architecture description for two reasons: (1) there is an existing basis of literature and academic work and (2) both frameworks are widely used in the industry and open-source projects.

To unify both representations of Zipkin and Jaeger into one format, we developed a generic meta-model for the system's architecture. This model captures the system's services, operations, and dependencies. Individual parsers for Zipkin and Jaeger transform the JSON-formatted traces into a representation described by our meta-model. This method allows for an extension regarding other tracing tools, given that the necessary parsers are implemented.

Since our data is relational, a graph is suitable for the representation. The spans from a trace describe dependencies between services and are represented by directed edges. The graph is visualized as node-link-diagram, as displayed in Figure 4. It is extended with tooltips and a context menu for further inspection of trace details. Once the graph is opened, it creates a list of available meta-information. Nodes display names of processes and the endpoint information (hostname and port). Edges display the spans' execution details, such as the duration of a call, logs, and tags. In addition, we apply a force-directed layout and extended it with a zooming function and the option to drag the graph.

Guide Word	Description	Operation Level Effect	Service Level Cause
NO	No Information	Response Time Spike*, No Response	Service Failure*
MORE	More data passed than expected	Response Time Deviation*	Decreased Service Performance*, High Utilization
LESS	Less data passed than expected	Response Time Deviation (harmless)	-
PART OF	Information incomplete (for group flows)	Failed Operation / Exception	Software Bug, Service Failure, API Change
REVERSE	Flow of information in wrong direction	Response Send To Wrong Service	Software Bug
OTHER THAN	Information complete, but incorrect	Exception After Operation Response	Software Bug, API Change
EARLY	Data arrives earlier in a sequence	One Operation Call With Quick Response (harmless)	-
LATE	Data arrives later in a sequence	One (Async) Operation Call With Slow Response	Decreased Service Performance, High Utilization

Table 2: Interpretations for the CHAZOPs guide words in the context of traces. Entries marked with * are implemented in the prototype.

4.2 Hazard Analysis

We base our hazard analysis on CHAZOP [10] to automatically identify hazards, generate suggestions, and prioritize services in the specification. CHAZOP is a risk assessment technique. It requires iterating over all interconnections, data-flows, and attributes in a system. Then, predefined guide words are interpreted, and causes, consequences, and protection mechanisms are derived.

In our approach, we iterate over the dependencies in the graph. In a brainstorming session, we interpreted the CHAZOP guide words. We did not explicitly select particular data-flows or attributes but treated a dependency like a request. Furthermore, we thought about how a deviation could be detected in traces on the operation level. We considered the calling and called operation, the response time, and the request status as known information. Next, we tried to identify the deviations' causes on the service level. For practical reasons, we did also consider interpretations that did not fully match the guide words, e.g., we see a *Response Time Spike* as interpretation for the guide word *NO*, since the complete absence of a request cannot be detected from a trace alone. The (non-exhaustive) list of interpretations from our brainstorming session can be found in Table 2.

For the prototypical implementation of RESIRIO, we selected two different hazard types on the operation level (*Response Time Deviation*, *Spike Response Times*) and two derived hazard types on the service level (*Service Failure*, *Degraded Service Performance*). These hazard types can easily be detected by inspecting response times in spans. We classify response times that deviate by 50% filtered by three times the standard deviation as *Response Time Deviation*. Furthermore, a *Response Time Spike* denotes outliers in the response times. If a hazard is detected in a service's operation, the algorithm suggests *Service Failure* as service hazard in case of *Response Time Spike*. For *Response Time Deviation*, the algorithm suggests *Decreased Service Performance* on the service level.

The identified hazards are involved in the computation of a priority value. The priority value depends on the number of incoming and outgoing edges and the number of hazards found in the architectural element. A service with many connections to other services has a higher priority than services with fewer connections. The priority of an edge is defined by the two services it connects. If hazards are found in a service or operation during the analysis, the service's priority is multiplied by the number of found hazards. The resulting formula is loosely based on the formula of the Risk Priority Number in Failure Modes and Effects Analysis (FMEA) hazard analysis method [17].

In RESIRIO, the priority value is used in two different ways in the scenario specification process: On the one hand, the chatbot suggests the five services and five operations with the highest priority to the engineer. On the other hand, each service and dependency in the graph representation is assigned a color from white (low priority) to blue (high priority). The color palette is interpolated from the lowest priority to the highest priority in the graph.

Instead of triggering deviations to the component's parameter based on expert knowledge, we examine response times from previous executions of the system to extract hazards. This kind of hazard analysis requires two prerequisites. First, it is necessary that the system ran before and went into a hazardous state. For some systems, this is not tolerable. Second, it requires that the hazard type is defined in advance. Therefore, very system-specific hazards are usually not detectable. We emphasize that the inspection of trace metrics and resulting hazards provides resilience engineers only with suggestions. Resilience engineers may still use their expertise to define custom stimuli types.

4.3 Scenario Creation

In order to enable the engineer to specify resilience scenarios, we present her an architecture graph and a chatbot, as displayed in Figure 4. The chat follows the design of a typical messenger interface and has three kinds of content types. The user content (blue) is on the right side of the chat component. The left side of the chat component contains the chatbot responses (yellow). Rich content guides the interaction and covers the middle of the chat content, rendering *Card Responses*, *Accordions*, and *Quick Replies*. The architecture and chatbot components are linked with each other, for example, selecting a service in the architecture graph, will be interpreted by the chatbot as choice of an artifact in the specification. Furthermore, the chatbot highlights the corresponding graph artifact when choosing an artifact for the scenario specification.

The goal of the specification process is to output fully specified scenarios. Our work extends the original resilience scenario with the parameters *component*, *normal availability*, *normal response times*, *normal response cases*, and *recovery time*. The *artifact* and the *component* are always used in combination with each other. While the *component* describes a concrete service or operation name, the *artifact* contains the type, e.g., whether a service or operation is selected. We limit the use of the extended scenario structure to trace metrics in favor of a more detailed scenario description. Thus, we divide the *response measure* further into sub-parameters. A *response measure* for services has *normal availability* and *recovery time*. The *response measure* for operations has *normal response time*,

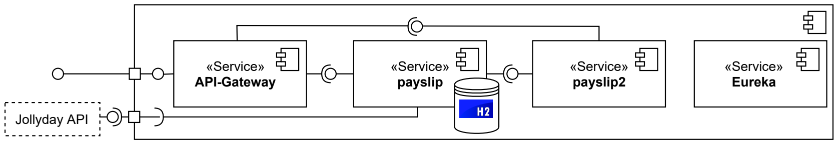


Fig. 5: Mocked Payroll Accounting System

normal response case, and *recovery time*. The parameters *normal availability*, *normal response time*, and *normal response cases* describe a system's normal operation state. Quantitative measures are used to specify these parameters (e.g., 100 ms or 99%) that can easily be checked against runtime metrics of the inspected system. If system metrics deviate from the given metrics of the normal operation state, a return to the normal state is necessary. The *recovery time* parameter describes the maximum tolerable time to return to the normal state, also using a quantitative measure (e.g., 5 minutes, or 4 seconds).

The interaction concept is rather simple. The chatbot aims to elicit one element of the extended scenario format in one step. For each step, the engineer selects default responses, which are predefined or generated from the previous analysis phases, or enters custom values. In a regular interaction, the chatbot first encourages the engineer to select services or operations to determine the *artifact* and *component*. Next, it proceeds with the description of the incident, followed by the description of the desired system response. In the end, the scenario can be edited or saved. In the latter case, the scenario appears in the configuration view in a separate tab. This tab summarizes the responses and is equipped with an export button that downloads the scenario in the JSON format.

5 Resilience Evaluation

This section aims to evaluate the resilience of the system under study. Therefore, we implemented a subset of the previously elicited resilience scenarios into resilience experiments using CTK. We compare the system's behavior against the expected behavior described in the scenarios' response part.

5.1 Experiment Setup

Examined Software System Due to legal constraints and to maintain anonymity, our industrial partner provided us with a mocked version as a proxy for the real payroll accounting system. This version, shown in Figure 5, is used throughout this paper as the system under test. It implements a similar business logic but with less computational overhead. The system uses typical patterns of the microservice architectural style, i.e., *API-Gateway-service* as a central gateway that manages all incoming requests and *Eureka* [20] to provide service discovery. The *payslip-service* utilizes an H2 in-memory database and the third-party API *Jollyday*. It can forward requests to the *payslip-service2*. Requests can also be sent directly to *payslip-service2* using a different endpoint.

The following six endpoints are used during the experiments:

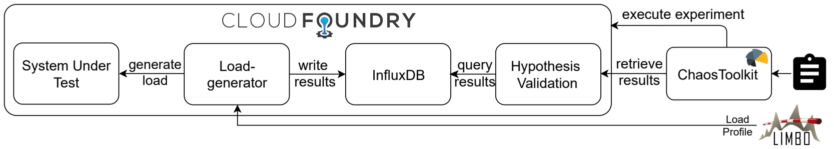


Fig. 6: Used structure of the experiment framework

INTERNAL_DEP — Calls the *payslip-service2* via *payslip-service*.
DB_READ — Reads an entry from the database of the *payslip-service*.
EXTERNAL_DEP — Calls the third-party API *Jollydays* via *payslip-service*.
DB_WRITE — Writes an entry into the database of the *payslip-service*.
GATEWAY_PING — Checks whether the *API-Gateway-service* responds.
UNAFF_SERVICE — Sends a request directly to *payslip-service2*.

The actual payment accounting system is deployed to a paid CF. Due to financial constraints and legal issues, the mock system is deployed to a local CF environment [9], which has similar properties as a paid CF. As CF is a constraint given by the stakeholders, we did not consider other cloud providers.

Experiment Tools Figure 6 shows our experiment framework comprising three tools, i.e., CTK, load generator, and hypothesis validator. During an experiment, these tools interact with the system to monitor the experiments and provide detailed insights, e.g., response times of calls to individual endpoints.

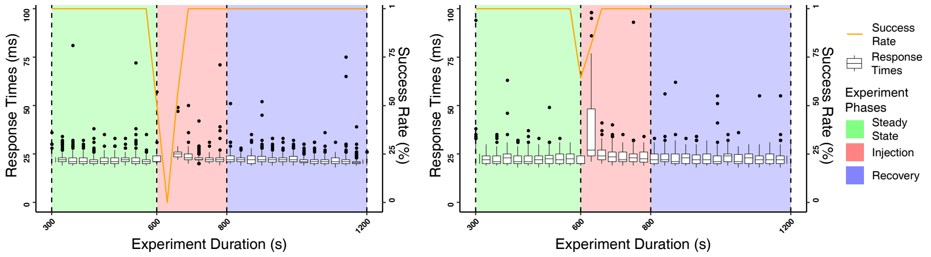
To execute the experiments, we used CTK [8], which can execute and monitor chaos tests and has drivers for various PaaS solutions. We leveraged the CF driver to terminate a service instance at a specific point in time and validate the steady-state hypothesis. The load that the system receives is controlled by an adapted version of the load generator from the TeaStore microservices benchmark [16] that monitors response times, number of successful, dropped, and failed requests. The collected data is written into an InfluxDB [14]. During the evaluation, a Spring service collects the data from the InfluxDB and calculates whether a hypothesis holds. We also created a dashboard application that provides convenient features, like synchronized starting of CTK and the load generator, live monitoring, and automated CTK setup. Since the dashboard does not add functionalities in executing experiments, it is not part of Figure 6.

5.2 Experiment Execution

Based on Scenarios 04 and 05, we implemented three resilience experiments. The first experiment investigates a load peak with an exponential increase (Scenario 04), while the remaining two investigate instance termination due to an internal CF error for random instances (Scenario 05) and specifically the *payslip-service* (Scenario 05'). The selection of experiments is based on the industrial partner's preferences. In all experiments, the effect on all endpoints is examined. In the following, we will only discuss the results of a subset of endpoints for Scenario 05'. The residual results can be found in the supplementary material [27].

Target Service	*payslip-service*
Experiment Type	Terminate *payslip-service* application instance
Hypothesis	Response measure of Scenario 05 holds
Blast Radius	*payslip-service*

Table 3: Resilience experiment design for Scenario 05'



(a) Without retry pattern

(b) With retry pattern

Fig. 7: Comparison of experiment results at endpoint INTERNAL_DEP

The design of the experiment related to Scenario 05' is given in Table 3. The target service of this experiment is the *payslip-service*, which holds the core business logic of the mock system. We use CTK to terminate running CF application instances to simulate the scenario's stimulus. The stimulus refers to an error that occurs in CF, which leads to a loss of an application instance. We assume that the blast radius only affects the *payslip-service* and that CF registers the loss of the *payslip-service* instance and starts a new instance. Our hypothesis is that the response measure of Scenario 05 still holds.

During the experiments, the system is exposed to an almost constant, synthetic load. We generated a load profile with a target load of 20 requests per second and some noise. The requests are evenly distributed over all six endpoints. To assess whether the system still responds correctly and in time, we measure response times of the requests and compute their success rate.

5.3 Experiment Results

The measurements for endpoint INTERNAL_DEP are visualized in Figure 7a. In the steady-state phase, we assume that the system is working as expected, i.e., the response times satisfy the SLOs. In the injection phase, CTK terminates the *payslip-service* instance. In the recovery phase, we assume that the system returns to a steady state, i.e., the response times satisfy the SLOs. We omitted the load generator's warmup and cooldown phase due to readability and analysis reasons, which refers to the overall first and last 300 s. Further, a 30 s binning was applied, and extreme outliers (>100 ms) are not shown.

The success rates at the endpoint INTERNAL_DEP (Figure 7a), DB_READ, EXTERNAL_DEP, and DB_WRITE drop to 0 % as the *payslip-service* is terminated after 600 s and rises back to 100 % as it recovers in about 1.5 min. During this downtime, no response times are recorded since no requests arrive at the *payslip-service*. During the steady-state and recovery phase, the response times are stable at around 20 ms and 15 ms, respectively. During the injection phase, there is a slight increase as the *payslip-service* has restarted.

5.4 Discussion of Results

As visible in Figure 7a, the response time and success rate values are almost identical in the steady state phase and the recovery phase. Furthermore, the increase in the success rate indicates that the *payslip-service* becomes available after 30 s to 60 s. Thus, the CF platform can re-instantiate the *payslip-service* quickly, leading to a quick recovery of the system.

Response times are slightly higher while the *payslip-service* is re-instantiated, which was expected as normal cold-start behavior. Endpoints GATEWAY_PING and UNAFF_SERVICE should remain unaffected during the injection because the *payslip-service* is not required to answer the requests. Nevertheless, response times at endpoint GATEWAY_PING are affected, which indicates a propagation of the failure effects from the *payslip-service* to the *API-Gateway-service*.

After the injection started, the success rate drops to 0 % at the endpoints INTERNAL_DEP, DB_READ, EXTERNAL_DEP, and DB_WRITE. The CTK terminates the single *payslip-service* instance. The load generator flags all requests as failed, leading to a success rate of 0 %.

We hypothesized that the response measure of Scenario 05 holds, i.e., requests are answered in time (99 % in less than 1 s) and correctly. As the response times are far below 1 s, our hypothesis regarding the response times is technically fulfilled. However, several requests are not answered at all, which is indicated by the dropped success rate. We consider these as incorrect response. Therefore, we assume that the hypothesis regarding correctness is not fulfilled.

6 Resilience Improvement

The previous section's experiments showed that the system does not respond as described in Scenario 05 to a failure of an instance of the *payslip-service*. While the response times are technically below 1 s in 99 % of all cases, requests are temporarily not answered at all. We consider that an incorrect response. Therefore, we aim to improve the system's success rate concerning Scenario 05 by applying resilience pattern(s). Resilience patterns [1] describe architectural changes to enable applications to handle failures gracefully and recover from them. A request may fail due to temporary failures in network components. Retrying requests is one well-known resilience pattern, which provides a solution to the problem of temporary failures within the application. Hence, we hypothesize that adding retry improves overall system resilience. We determine the efficacy of improvements to the system's resilience by re-executing the experiments.

6.1 Architectural Modifications

The system under test was fortified with a retry pattern [22], i.e., the *API-Gateway-service* sends another request to the *payslip-service* if a request fails or

	Steady State								Injection								Recovery							
	w/o Pattern				w Pattern				w/o Pattern				w Pattern				w/o Pattern				w Pattern			
Endpoint	p_5	$\tilde{x}$	$\overline{x}$	p_{99}	p_5	$\tilde{x}$	$\overline{x}$	p_{99}	p_5	$\tilde{x}$	$\overline{x}$	p_{99}	p_5	$\tilde{x}$	$\overline{x}$	p_{99}	p_5	$\tilde{x}$	$\overline{x}$	p_{99}	p_5	$\tilde{x}$	$\overline{x}$	p_{99}
INTERNAL_DEP	19	22	22.5	33	19	22	24.0	51	19	21	22.4	32	19	22	24.6	90	19	21	22.1	31	19	22	23.0	34
DB_READ	11	12	13.3	21	11	12	13.0	24	11	12	13.1	20	11	12	13.2	30	11	12	12.9	20	11	12	12.8	19
EXTERNAL_DEP	11	12	12.6	21	10	12	13.3	23	11	12	12.5	21	10	12	14.1	31	11	12	12.3	19	10	12	12.2	19
DB_WRITE	11	13	13.4	21	11	12	13.1	22	11	12	13.1	20	11	12	13.3	27	11	12	13.0	20	11	12	12.7	19
GATEWAY_PING	11	12	13.3	21	11	12	13.3	24	11	12	13.1	20	11	12	13.6	32	11	12	12.9	19	11	12	12.9	21
UNAFF_SERVICE	10	11	11.9	19	10	11	11.6	19	10	11	11.7	18	10	11	11.6	19	10	11	11.8	18	10	11	11.5	19

Table 4: Statistical summaries of the three experiment phases. p_α: α-th percentile; $\tilde{x}$: median; and $\overline{x}$: mean. Values are given in ms.

remains unanswered. The retry pattern seems to be a reasonable choice since response times are far below the threshold of 1 s, as indicated by the previous experiment. Due to its specific purpose, the system has to accept requests near real-time and always answer correctly. Thus, resilience patterns that rely on backup or restricting behavior, like circuit breakers or flow limiters, are unsuited. To avoid bad retry behavior, we configured the Spring-Retry as follows. We set the maximum number of retries of each *payslip-service* request to be 4, the initial delay to 10 ms, the factor for the exponential increase to 3, and the maximum delay to 150 ms — resulting in retries after 10 ms, 30 ms, 90 ms, and 150 ms.

6.2 Experiment Results and Discussion

Figure 7b visualizes the system's response times and success rates with the retry pattern for endpoint INTERNAL_DEP. Table 4 shows the associated statistical values. In general, similar behavior can be observed at all the endpoints. Comparing the plots of Figure 7 shows that the mean response times in the steady state phase do not vary significantly when the retry pattern is added. Although, at the beginning of the injection phase, far more high response times can be observed. In addition, the boxplots show a slightly higher interquartile range in the plot where the retry pattern is integrated.

The plots also show that the success rate does not drop to zero anymore with the active pattern. For endpoint INTERNAL_DEP, the success rate drops to approximately 70 %. For the endpoints GATEWAY_PING and UNAFF_SERVICE, the success rate remains at 100 %. The two endpoints do not depend on the *payslip-service*, which explains the high success rate at these endpoints.

The application of the retry pattern can explain the response time spikes during the injection (see the Figure 7). Requests sent shortly before the restart of the *payslip-service* fail, but are retried by the *API-Gateway-service* until the *payslip-service* recovered after approximately 10 s. However, as several retries have been aggregated, the *payslip-service* has to handle a high amount of requests upon recovery, resulting in a visible spike in response times.

In contrast to the system without the retry pattern, the success rate drops less. Thus, the retry pattern improves the scenario satisfaction as it increases the percentage of correct responses while the response times stays below 1 s.

7 RESIRIO User Study

To evaluate the developed prototype on usability and effectiveness, we designed an expert user study. In the following, we give an overview of the procedure and results. Artifact and more results can be found in the supplementary material. The user study aimed to get feedback from inexperienced engineers and experts. Separate research question and hypotheses are investigated. The goal is to accept a hypothesis H_{x1} by rejecting the corresponding null-hypothesis H_{x0}. The research questions and derived hypotheses are:

- **RQ1:** Are users of RESIRIO able to create resilience scenarios successfully?
  - H_{11}**:** Study participants can complete the study tasks successfully.
- **RQ2:** How effective is RESIRIO compared to traditional elicitation processes?
  - H_{21}**:** Study participants create ATAM-based scenarios faster with RESIRIO than with a traditional approach, e.g., the workshop.
- **RQ3:** How supportive is RESIRIO during the elicitation process?
  - H_{3a1}**:** RESIRIO supports inexperienced engineers in the elicitation process.
  - H_{3b1}**:** RESIRIO supports experienced engineers in the elicitation process.
- **RQ4:** How usable are ATAM-based scenarios created with RESIRIO for resilience assessment?
  - H_{41}**:** RESIRIO's scenarios can be used for system resilience assessment.

7.1 Design and Methodology

We perform a qualitative study, since there is a limited number of domain experts for scenario-based resilience engineering. The study has three different strategies to collect feedback. First, subjective feedback is collected through interview-style and open-ended questions. Secondly, the prototype measures metrics like the duration of the study, the number and type of interactions, interactions with features besides the chatbot, and the number of elicited scenarios. To support the measurement of effectiveness and the quality of the prototype, we include close-ended questions. Finally, usability is measured based on a Likert scale from questions as suggested by Brooke [4] in the System Usability Study. If participants have a particular interest in a feature or make suggestions for improvements, the study conductor asked additional questions to get more insights.

We created two tasks. At the beginning of each task, the participants are given a general description of the inspected system, which introduces a failure to the system. In each task, an ATAM-based scenario to fix the failure needs to be created. In the first task, which was considered beginner-friendly, we present a predefined solution. The only challenge in this task is to create the scenario through the use of the chatbot. Additionally, it is possible to create the resilience scenario only with the use of *Quick Replies*. The second is designed to be more challenging. On the one hand, the correct solution was not shown to the participants. On the other hand, the task description contains names of services and operations that are not available through *Quick Replies*. The absence of *Quick Replies* requires participants to use the architecture graph and the text input.

7.2 Study Execution

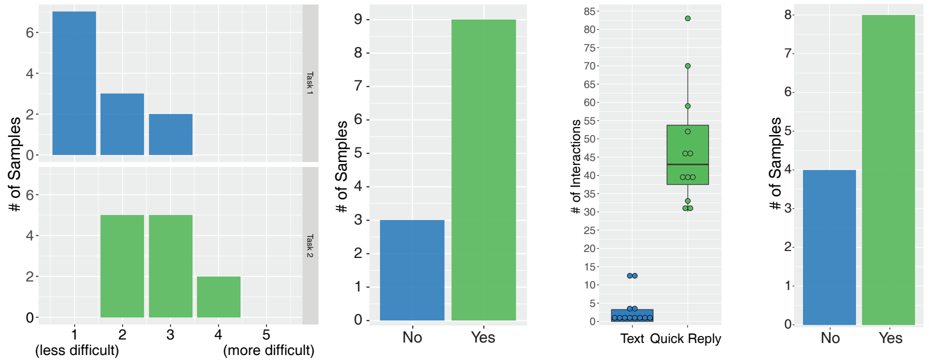
We invited participants from the industry and academia for the evaluation. Five employees from the industry company did take part in the study: two software architects, one DevOps Engineer, one quality assurance engineer, and one doctoral researcher. Four participants already participated in the workshop. Three participants are considered experts of the system since they were involved in its development. The system is the industrial system as described in Section 3.2. We also requested the participation of researchers familiar with the Train Ticket system [29]. Seven researchers agreed to participate. Four participants are considered experts of the system since they were involved in its development.

We informed the participants beforehand that the study takes place virtually and requires approximately 45-60 minutes to complete. After their verbal agreement to the consent form, they were sent the task description. In the first part, the goal and purpose of the study are given. In the second part, the prototype is introduced. The third part contains the actual tasks fitted to each system. After the participants read the first two parts of the task description, they were given a link to the prototype. The participants were encouraged to describe their actions with the prototype, which was done in most cases. The study conductor stayed passive and just reminded the participants to carefully reread the task description, which resolved the problems in all cases. After completing the practical part, the study conductor started a verbal conversation with the participants and asked them about noticeable events during the practical part. Examples of noticeable events were not using the chatbot text interactions or creating a new scenario from the beginning. These questions usually led to a discussion with open-ended questions. After all noticeable events were covered, the study conductor asked the participants to fill out the questionnaire.

7.3 Results and Discussion

All participants claimed that they completed the given tasks successfully. The questionnaire confirms that the first task was perceived easier than the second task (see Figure 8d), as intended. We created a set of solutions for the more complex second task before the study was executed. In the evaluation, we compared the elicited scenarios of the participants with the prepared solutions. Overall, six participants had to reconfigure the parameters of a scenario because they were not satisfied with their initially chosen parameters, or the chatbot did not assign the specified parameters as intended. Three of the 24 final scenarios contained invalid parameters. Therefore, H_{10} is rejected, and H_{11} can be accepted.

The fastest participant used 7 minutes to complete both tasks, while the slowest used more than 24 minutes. The median lies at approx. 16 minutes. Although the second task is the difficult task, recordings indicate an approximated median of only five minutes to complete the second task. This can be ascribed to the learning effect that users had for solving the first task. The workshop required several participants to elicit 12 scenarios over a working day. In contrast to that, a single person can elicit a scenario within 5 minutes after little training.



(a) How challenging was it to solve the tasks?

(b) Enough assistance during the elicitation?

(c) Type of interactions during the study?

(d) Enough information to test resilience?

Fig. 8: Results from questions and measurements of the user study.

Therefore, H_{20} is rejected, and H_{21} can be accepted. Three participants did not agree with this conclusion. However, all participants who had previously taken part in the workshop confirmed our conclusion.

Six participants got stuck during the elicitation process, but three of them could return to the conversation without the help of the Chatbot Reset functionality. Nevertheless, most of the participants agreed that they were provided with enough assistance (see Figure 8c). Additionally, the results of the System Usability Study confirmed that RESIRIO (i) was perceived as easy to use, (ii) does not require too much knowledge to use, and (iii) has well-integrated features.

A significant finding is an overall preference of using Quick Replies over the use of text interactions (see Figure 8b), which is an explanation for the quick elicitation. While six participants did not use text interactions, the other six used only three to thirteen text interactions. Quick Replies were the prototype's most used and liked feature, followed by the Architecture Graph. Two participants requested general documentation of the prototype and wanted to see all the available functionalities of the prototype. Other suggestions for improvements were more Quick Replies, role assignment (i.e., software architect, testing engineer), and displaying the scenario at all times. In general, experts responded with more critical feedback. While they were happy with the prototype's usability, they stated that features were missing to highlight details of a scenario and give more insights into the technical background of the analysis process. Therefore, H_{3a1} can be accepted and H_{3a0} is rejected. However, we can not reject H_{3b0} for the current state of the approach based on the experts' feedback.

Overall, most participants agreed that the elicited scenarios are sufficient to test a system's resilience (see Figure 8a). For this reason, we accept H_{41} and reject H_{40}. However, especially the experts noted that the available options for stimuli and responses as too limited to create usable scenarios in practice. Half of the participants agreed that more quick replies are necessary.

7.4 Key Lessons Learned

This section presents the key lessons learned from the user study.

- Engineering teams in traditional elicitation processes such as workshops are concerned about the documentation process. Automated interactive tooling such as RESIRIO address that concern.
- The tendency to interact through the architecture graph and quick replies is higher than having time-consuming conversations with a chatbot.
- Chatbot-assisted tooling such as RESIRIO, which involves interaction between one person and a chatbot, does not replace traditional elicitation processes because traditional methods involve discussions between people, leading to more coverage of stimuli and response measures.

8 Related Work

We explored (1) the usage of the scenario-based ATAM and established hazard analysis techniques for interactive resilience requirement elicitation and (2) testing through resilience experiments for architecture assessment and improvement. This section classifies research areas and existing works close to us.

8.1 Measurement-Based Resilience Evaluation

A workshop is an effective technique for requirement elicitation [24]. In our case, the workshop's preparation and conduction are based on the scenario template of Bass et al. [3]. Our difference to existing works on measurement-based resilience evaluation is that we have an explicit step on eliciting resilience requirements. In the next paragraphs, we elaborate on this in more detail.

Cámara et al. [5,6,7] propose an approach for resilience analysis of self-adaptive systems. The core idea consists of three parts: (1) specification of resilience properties using Probabilistic Computation Tree Logic, (2) modeling causes of a hazard, e.g., high-load using experimentation and collecting traces of system behavior, and (3) verification of resilience properties using model checking. In contrast to model checking-based verification, we evaluate a scenario's response measure by analyzing measurements. Furthermore, Cámara et al. do not focus on the elicitation of resilience requirements.

In Chaos Engineering [2,18], system resilience is evaluated through failure injection [19]. There are works on both (1) using engineering methods to identify failure modes [15], i.e., causes of a hazard systematically before failure injection, and (2) ad-hoc failure injection with no systematic failure mode identification [13]. However, they do not explicitly specify resilience requirements and lack a methodical way for requirement elicitation. Our work is a step toward closing this gap.

In the context of resilience requirement elicitation, Yin et al. [28] propose a goal-oriented technique for representing resilience requirements. The high-level idea is to represent a resilience goal — e.g., all requests are processed correctly — and identify possible causes of hazards that act as obstacles for achieving a resilience goal — e.g., node failure. However, they do not discuss how to identify hazards and their causes. Goal orientation and developing scenarios are two activities in requirements engineering [24] that benefit the elicitation process. According to Pohl [24], scenario development benefits elicitation by making goals understandable for stakeholders and may refine or identify new goals. Our work uses scenario development without goal-oriented modeling as all stakeholders know the system's high-level quality goal.

To our knowledge, this is the first work using ATAM for eliciting and specifying resilience requirements before evaluating the resilience through experiments.

8.2 Chatbot-based Requirements Elicitation

Rietz [26] proposes a conversational interface called chatbot for requirements elicitation, which is applicable for novice users in requirements engineering. The core idea is to guide users in the elicitation process by asking questions from abstract to more precisely. The user interaction, including questions and answers, is stored in a knowledge base for later re-use and interaction improvements.

Friesen et al. [12] proposed the CORDULA approach. Users of the approach need to use special text format to interact with the chatbot, which identifies domain concepts and imprecise statements such as *a big file*. Afterward, in a separate debugging window, users can edit the specification. In contrast to previous works, we use the structured template of ATAM for requirements specification. In addition, we use architecture visualization and hazard analysis techniques to guide the users in the elicitation process.

The work by Surana et al. [25] allows users to elicit and classify requirements as functional or non-functional requirements. The authors benefit from machine learning algorithms for the elicitation and classification tasks. However, the chatbot expects users' familiarity with the system's requirements and is more usable for classification tasks than the elicitation process.

9 Threats to Validity

9.1 Workshop

Conclusion validity One threat is the reliability of measures, which means repeating the workshop yields the same resilience requirements list. Elicitation of resilience requirements involves human judgment. Hence, it is a subjective measure. Therefore, we can not entirely rule out this threat.

Internal validity One threat is instrumentation, which means our tools and techniques were not suitable. We conducted a one-day structured workshop and used the scenario template of Bass et al. [3] for eliciting resilience requirements. We refined all the resilience requirements through several iterations after the workshop and validated them against the workshop participants.

Construct validity For us, the main threat in this category is mono-method bias, which means we did not use other elicitation methods. Therefore, there is a threat that elicited resilience requirements are biased. We can not entirely rule out this threat as we did not apply other methods and cross-check the results.

External validity The heterogeneity poses a threat, i.e., different roles and expertise of participants. Workshops with less heterogeneity in the stakeholders could lead to no resilience requirements. We can not entirely rule out this threat. Although our study is based on a microservice architectural style, we argue that our approach of resilience requirements elicitation is independent of the architecture style. The important is eliciting resilience requirements and understanding design decisions that satisfy the requirements.

9.2 RESIRIO User Study

Conclusion validity As most questions asked during the study were open-ended, and of qualitative nature, it is hard to prove whether our conclusions are correct. The number of participants was limited, resulting in ambiguous conclusions from measurements and general uncertainty in the evaluation.

Internal validity The target group consisted of participants with different levels of experience. The knowledge of the participants is hard to compare. Four participants had previously seen the prototype in a demonstration. Although they did not use the prototype, some transfer of knowledge might have happened.

Construct validity The comparison of RESIRIO's efficiency to traditional elicitation processes is limited for the lack of existing measurements. We do not think that values extracted from experience reports and other literature reviews provide a good enough basis for a thorough comparison. Feedback from users is always subjective. They could, for example, not be able to judge whether created scenarios can be used for a specification in practice.

External validity Participants came from the same institutions and companies, complicating the generalization of results in surroundings that are different from those found in the participants' environments.

9.3 Experiment Design

We used the mock system for quantitative evaluation of resilience requirements that are based on the actual system. There is a threat that evaluation results are inaccurate. However, the purpose of the experiments is to exemplary show how elicited requirements and derived experiments can help to improve the system — we do not claim the accuracy of the quantitative results. Furthermore, due to legal issues, we used CF Dev [9]. We faced instability, e.g., resource drainage of Dev nodes, in the environment during experimentation. There is a threat of a negative impact on results due to this instability. To counteract this threat, we re-executed experiments to gain insight into approximate measurements, ensuring reliable data with no unintended node or service crash.

10 Conclusion

The successful development of resilience scenarios depends on the outcome of the hazard analysis. Our workshop approach to scenario-based resilience evaluation assumes business domain experts to derive an initial list of hazards. FTA can then be a means to analyze the hazards and derive resilience scenarios. Our semi-automated approach RESIRIO is a means for quicker and easier elicitation of such scenarios, but it can not fully replace the workshop. We plan to (1) extend our process with an explicit formalization step after the elicitation for refinement of the scenarios, (2) formally verify response measures of resilience scenarios, and (3) create processes for continuous hazard analysis when a system faces changes, e.g., updates and refinement/development of resilience scenarios.

Acknowledgment This work has been supported by the Baden-Württemberg Stiftung (ORCAS — Efficient Resilience Benchmarking of Microservice Architectures) and the German Federal Ministry of Education and Research (dqualizer and Software Campus 2.0 — Microproject: DiSpel).

Data Availability Our artifacts [27] comprise (i) the resilience scenarios, (ii) the RESIRIO project and user study documents, and (iii) the data and R scripts as a CodeOcean capsule. We are working on making parts of the created/modified experiment tools available as open-source software. For confidentiality reasons, the system under test cannot be published.

References

1. Microsoft Azure (2021), https://docs.microsoft.com/en-us/azure/architecture/framework/resiliency/reliability-patterns
2. Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., Rosenthal, C.: Chaos Engineering. IEEE Softw. **33**(3), 35–41 (2016)
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Longman Publishing Co., Inc., USA, 4 edn. (2021)
4. Brooke, J.: SUS: a 'quick and dirty' usability scale. Usability evaluation in industry **189** (1996)
5. Cámara, J., de Lemos, R.: Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In: Proc. 7th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 53–62 (2012)
6. Cámara, J., de Lemos, R., Laranjeiro, N., Ventura, R., Vieira, M.: Robustness-driven resilience evaluation of self-adaptive software systems. IEEE Transactions on Dependable and Secure Computing **14**(1), 50–64 (2017)
7. Cámara, J., de Lemos, R., Vieira, M., Almeida, R., Ventura, R.: Architecture-based resilience evaluation for self-adaptive systems. Computing **95**(8), 689–722 (2013)
8. Chaos Toolkit: Chaos Toolkit (2020), https://chaostoolkit.org
9. Cloud Foundry Foundation: Cloud Foundry dev documentation (2020), https://github.com/cloudfoundry-incubator/cfdev
10. Dunjó, J., Fthenakis, V., Vílchez, J.A., Arnaldos, J.: Hazard and operability (HAZOP) analysis. A literature review. Journal of hazardous materials **173**(1-3), 19–32 (2010)

11. Frank, S., Hakamian, M.A., Wagner, L., Kesim, D., von Kistowski, J., van Hoorn, A.: Scenario-based resilience evaluation and improvement of microservice architectures: An experience report. In: ECSA 2021 Companion Volume. vol. 2978 (2021)
12. Friesen, E., Bäumer, F.S., Geierhos, M.: CORDULA: software requirements extraction utilizing chatbot as communication interface. In: Joint Proceedings of REFSQ-2018 Workshops (2018)
13. Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V.: Gremlin: Systematic resilience testing of microservices. In: Proc. 36th IEEE Int. Conf. on Distributed Computing Systems (ICDCS). pp. 57–66 (2016)
14. InfluxData Inc.: InfluxDB website (2020), https://www.influxdata.com/
15. Kesim, D., van Hoorn, A., Frank, S., Häussler, M.: Identifying and prioritizing chaos experiments by using established risk analysis techniques. In: Proc. 31st Int. Symposium on Software Reliability Engineering (ISSRE) (2020)
16. von Kistowski, J., Eismann, S., et al.: Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In: Proc. MASCOTS. pp. 223–236 (2018)
17. Leveson, N.G.: Safeware — System Safety and Computers: A Guide to Preventing Accidents and Losses Caused by Technology. Addison-Wesley (1995)
18. Miles, R.: Learning Chaos Engineering – Discovering and Overcoming System Weaknesses through Experimentation. O'Reilly Media, Inc. (2019)
19. Natella, R., Cotroneo, D., Madeira, H.: Assessing dependability with software fault injection: A survey. ACM Computing Surveys (CSUR) **48**(3), 44:1–44:55 (2016)
20. Netflix Inc.: Eureka (2020), https://github.com/Netflix/eureka
21. Newman, S.: Building Microservices. O'Reilly (2015)
22. Nygard, M.T.: Release It!: Design and Deploy Production-ready Software. Pragmatic Bookshelf (2018)
23. Okanovic, D., et al.: Towards performance tooling interoperability: An open format for representing execution traces. In: 13th European Performance Engineering Workshop (EPEW). vol. 9951, pp. 94–108 (2016)
24. Pohl, K.: Requirements Engineering - Fundamentals, Principles, and Techniques. Springer (2010)
25. Rajender Kumar Surana, C.S., Shriya, Gupta, D.B., Shankar, S.P.: Intelligent chatbot for requirements elicitation and classification. In: 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT). pp. 866–870. IEEE (2019)
26. Rietz, T.: Designing a conversational requirements elicitation system for end-users. In: 2019 IEEE 27th International Requirements Engineering Conference (RE). pp. 452–457. IEEE (2019)
27. S. Frank et al.: Supplementary material (2022), https://doi.org/10.5281/zenodo.6077724; Code Ocean capsule: https://doi.org/10.24433/CO.0520280.v1
28. Yin, K., Du, Q., Wang, W., Qiu, J., Xu, J.: On representing and eliciting resilience requirements of microservice architecture systems. CoRR **abs/1909.13096** (2020), https://arxiv.org/abs/1909.13096v3
29. Zhou, X., et al.: Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 683–694. ACM (2019)
30. Zorn, C.: Interactive elicitation of resilience scenarios in microservice architectures. Master's thesis, University of Stuttgart (2021)