# How is Transient Behavior Addressed in Practice? Insights from a Series of Expert Interviews

Samuel Beck
Institute for Visualization and Interactive Systems
University of Stuttgart
Stuttgart, Germany

Sebastian Frank
Alireza Hakamian
Institute for Software Engineering
University of Stuttgart
Stuttgart, Germany

André van Hoorn
Department of Informatics
University of Hamburg
Hamburg, Germany

## ABSTRACT

Transient behavior occurs when a running software system changes from one steady-state to another. In microservice systems, such disruptions can, for example, be caused by continuous deployment, self-adaptation, and various failures. Although transient behavior could be captured in non-functional requirements, little is known of how that is handled in practice. Our objective was to study how architects and engineers approach runtime disruptions, which challenges they face, whether or not they specify transient behavior, and how currently employed tools and methods can be improved. To this end, we conducted semi-structured interviews with five experienced practitioners from major companies in Germany. We found that a big challenge in the industry is a lack of awareness of transient behavior by software stakeholders. Consequently, they often do not consider specifying it in non-functional requirements. Additionally, better tooling is needed to reduce the effort of analyzing transient behavior. We present two prototypes that we developed corresponding to these findings to improve the current situation. Beyond that, the insights we present can serve as pointers for interesting research directions for other researchers.

## CCS CONCEPTS

• **Software and its engineering** → **Extra-functional properties**; **Requirements analysis**; • **Human-centered computing** → Human computer interaction (HCI).

## KEYWORDS

transient behavior, non-functional requirements, microservices

**ACM Reference Format:**
Samuel Beck, Sebastian Frank, Alireza Hakamian, and André van Hoorn. 2022. How is Transient Behavior Addressed in Practice? Insights from a Series of Expert Interviews. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3491204.3527483

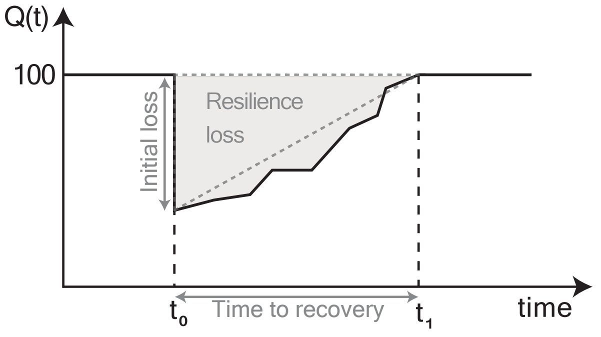


**Figure 1: Bruneau's resilience triangle model [4] on a resilience curve, adopted from Yin et al. [13].**

## 1 INTRODUCTION

Modern microservice-based software systems are subject to frequent changes in production, leading to so-called transient behavior. Changes can arise from outside, e.g., workload peaks and changing user behavior, or inside the system, e.g., autoscaling and deployment of new (versions of) services. User satisfaction can be heavily affected if the system is not resilient to such changes. Figure 1 illustrates a system's transient behavior and resilience [4, 13] by plotting a system's quality of service (QoS) and accumulating resilience loss for a limited time caused by a change at $t_0$. It is crucial not only to comprehend but also to elicit and specify a system's transient behavior in non-functional requirements [6]. However, the elicitation and specification are challenging due to the unexpected changes and the system's complex behavior. For this reason, we want to better understand the problems and challenges that developers, architects, and operators face with eliciting and comprehending non-functional requirements for transient behavior in practice. Our goal is to identify new research directions in this area and potential avenues for improving current practices and tooling. Furthermore, we are interested in whether and how emerging human-computer interaction (HCI) methods can help to tackle the complexity of transient behavior.

We conducted five semi-structured online interviews [12] with experienced software architects from German IT companies. They revealed that requirements for transient behavior are mostly not captured in the industry. Major reasons for that are a lack of understanding from business stakeholders and a perceived low criticality of transient behavior compared to other quality attributes. Regarding the analysis of transient behavior, the interviewees stated that the high effort due to unsuited tooling makes the analysis of transient behavior not worth it for them. They ask for better visualizations and models of transient behavior and improved filtering to identify issues easier. Based on the findings, we developed two early prototypes: TransVis [2] is a tool intended to aid software architects and engineers in the specification and exploration of transient behavior through visualizations and chatbot conversations. Resirio [14] is intended to guide users through an interactive elicitation of resilience scenarios for a microservice architecture model extracted from execution traces.

After introducing the background and related work, this paper details the expert interview series and its findings and outlines the two prototypes.

## 2 BACKGROUND AND RELATED WORK

This section gives a brief introduction to the concept of transient behavior before discussing related work.

### 2.1 Transient Behavior

Transient behavior concerns the time-dependent behavior of a system's QoS, quantified as the probability of being in a specific state $s$ at time $t$ [11]. The concept of transient behavior was originally introduced in electrical engineering, where it describes the state of an electrical power system while it changes from one steady state to another [10]. Causes for changes in microservice-based systems are continuous deployments of services, instance failures, and self-adaptation of the system caused by load changes and resilience mechanisms that limit the effect of failures. The changes disrupt the QoS of impacted services and create transitions between system states, akin to transient behavior in electrical systems. In previous work [2], we modeled transient software behavior using the resilience triangle by Bruneau et al. [4], illustrated in Figure 1 and comprising the following properties: (i) initial loss $X$, (ii) time to recovery $T$, and (iii) loss of resilience $R$. The properties are based on a QoS function $Q(t)$ and an expected QoS value $Q_e$. The initial loss $X$ is defined by the loss of quality at the beginning of an instance of transient behavior. The time to recovery $T$ describes a transient behavior's duration, and the loss of resilience $R$ is approximated by the area of the triangle formed by the previous two.

### 2.2 Related Work

We performed an industry interview to know how engineers deal with transient behavior in their requirements and proposed prototypes to improve current limitations in the industry. To our knowledge, our study is the first attempt to provide empirical evidence on existing challenges of transient behavior in non-functional requirements.

Niedermaier et al. [9] proposed an industry interview to understand existing challenges in observability and monitoring of distributed systems, which use the DevOps paradigm and microservice architectural style. Using a similar method, Eckhardt et al. [6] collected 530 requirements specification from 11 companies to study the distinction between functional and non-functional requirements. They found that 75% of the non-functional requirements describe the system behavior and hence are, in fact, functional. Our main difference to the previous works is that we use the industry interview technique to identify challenges on a different topic: transient behavior specification in non-functional requirements.

**Table 1: Interview design.**

| Hypotheses | Questions (Categories) |
|---|---|
| | General questions |
| $H_1$ $H_2$ | Transient behavior |
| $H_3$ $H_4$ | Specification of transient behavior |
| $H_5$ $H_6$ | Tools/methods for specifying and comprehending transient behavior |

Another method of identifying challenges or open questions is performing a literature review instead of going to the industry. We did not perform a literature review, as our objective was to know how transient behavior is handled in practice.

This paper also presents two prototypes for specifying and understanding transient behavior. Method-wise, related work concerns helping stakeholders of the software systems in requirements engineering phases such as specification using visualization techniques. Shaker et al. [1] performed a systematic literature review on requirements engineering visualization and proposed a classification of works on the subject. For additional related work specific to the prototypes, we refer to [2, 14].

## 3 METHOD

To obtain the desired insights, we interviewed software architects from five different companies in the IT consulting business. We selected candidates who have already accumulated considerable experience in this field and regularly work on projects involving a cloud and microservice environment. We contacted seven software architects and invited each of them to an interview. Five out of those seven accepted the invitation. The interviews were held in 2020 in a semi-structured manner [12] in the form of video calls, each lasting around one hour. Table 1 gives an overview of the interview design, mapping hypotheses to questions.

### 3.1 Hypotheses

Before we conducted the interviews, we created six hypotheses about how engineers think about and handle transient behavior and sought to confirm or reject them through the interviews:

$H_1$: Engineers are familiar with the concept of transient behavior in software.

$H_2$: Engineers understand the transient behavior in their systems.

$H_3$: Engineers think that the transient behavior of software should be specified in non-functional requirements.

$H_4$: Engineers specify the transient behavior of their software systems in non-functional requirements.

$H_5$: Engineers are satisfied with existing methods and processes for analyzing and specifying non-functional requirements for the transient behavior of software.

$H_6$: Engineers collaborate when they are analyzing or specifying non-functional requirements for the transient behavior of software.

### 3.2 Interview Structure

The questions of the interview were split into four categories (Table 1) of (i) general questions about the interviewees' background, (ii) their familiarity and experience with transient behavior, (iii) the specification and comprehension of transient behavior, and (iv) tools and methods for the specification and comprehension of transient behavior and the interviewees' requirements for such. The question categories were time-boxed to ensure that each was covered sufficiently in the interview. The semi-structured nature of the interviews ensures that the interviewees' statements can be compared while allowing us the freedom to go into more details based on participants' answers. This design promoted a more natural discourse in which interviewees could express their thoughts freely. Each interviewee got a short explanation of the concept of transient behavior or time-dependent behavior in software systems and its causes if they were not yet familiar with it. The complete questionnaire is publicly available online as supplementary material [3].

### 3.3 Interview Analysis

The audio was recorded, and a transcript of each interview was produced, which later served as the basis for the analysis. The transcripts of the interviews are publicly available online as supplementary material [3]. To analyze the answers in the transcripts, we first classified them into the categories presented in Section 4. Then, each interviewee's answers in each category were summarized, comprising their main points. For each category, the summaries of all five candidates were then compiled into one report that reflects the interviewees' general opinions on a topic.

## 4 RESULTS

This section describes the results of the conducted interviews. First, we give an overview of the interviewees' backgrounds and their practices for eliciting non-functional requirements. Then, we present their answers that regard the analysis and specification of transient behavior and finish with an overview of their answers about currently employed tools and methods, requirements for an envisioned solution, and their views on novel HCI interfaces.

### 4.1 Background of Interviewees

All five interviewees hold senior positions at different companies in the software consulting industry that mainly work on enterprise applications. All interviewees are experienced software architects and have a deep understanding of microservice architectures. Table 2 presents an overview of the background of the five interviewees.

**Table 2: Background information of the five interviewees.**

| Interviewee | Position | Company |
|---|---|---|
| Stefan Tilkov | CEO | innoQ |
| Uwe Friedrichsen | CTO | codecentric |
| Fabian Keller | Head of cloud innovation | mimacom |
| Matthias Häussler | Senior managing consultant | Novatec Consulting |
| Torsten[1] | Software architect | *Anonymous* |

Stefan Tilkov is the managing director (CEO) and principal consultant at innoQ. He is involved in customer projects, key account management, workshop organization, and architecture consultancy. His company has been actively advocating the microservice architectural style for five to six years.

Uwe Friedrichsen is the chief technical officer (CTO) at codecentric and is responsible for its technological and methodical strategy. He is active as a software architect and develops solutions tailored to his customers' needs. He has been working with service-oriented architectures in the last two decades and has worked with microservices since 2014. Furthermore, he is directly involved in eliciting and documenting non-functional requirements in his projects.

Fabian Keller was recently appointed as the head of cloud innovation at mimacom. He has also been working as a software engineer for four years. His company develops and maintains multiple microservice systems.

Matthias Häussler has been a senior managing consultant at Novatec Consulting for four years. Before that, he worked for almost 17 years at IBM. He is concerned chiefly with cloud-native software development. He mainly works in customer projects and also gives workshops and lectures on this topic. Furthermore, Matthias is the organizer of a local meetup and ambassador for Cloud Foundry. His experience with microservices is mainly limited to transforming monolithic legacy applications into microservice applications. He highlighted during the interview that he is not directly involved with the elicitation and documentation of non-functional requirements in his projects.

Torsten[1] holds a PhD in computer science. His PhD thesis focused on software performance engineering. He has been working as a software architect for one year at a software and IT consulting company whose primary focus is on the tax consultant market. Currently, he is mainly concerned with software performance aspects, microservice systems, and maintainability. He is also directly involved with the elicitation and documentation of non-functional requirements in his projects. Torsten is a fictional name because the interviewee was not permitted to reveal his real identity and company name.

### 4.2 Specification of Non-functional Requirements

All five interviewees stated that they specify some non-functional requirements for their projects. However, the thoroughness with which requirements are captured at the different companies varies. Torsten and Stefan explained that they usually capture non-functional requirements in formal quality scenarios. The three other interviewees explained that they specify non-functional requirements informally, citing the complexity of formal requirements and scenarios as the reason. Fabian expressed that they document both functional and non-function requirements in user stories.

They all described that it is challenging to obtain concrete non-functional requirements from their customers. A lack of general understanding of these requirements and their importance among business stakeholders were generally stated as the main obstacles

[1]Name and organization changed at the interviewee's request.

to a more thorough and precise specification of non-functional requirements. Uwe and Matthias pointed out beyond this that they are often left to guess the non-functional requirements that their customers have for a system. The only way to validate if these requirements meet the customer's expectations is through frequent testing.

## 4.3 Familiarity with the Concept of Transient Behavior

Stefan, Uwe, and Matthias were not familiar with the concept of transient behavior in software systems and its causes. Fabian stated that he was already familiar with the basic concept but enquired about the need for a specification in non-functional requirements thereof. Torsten said that he was already familiar with the concept and its causes. Besides him, the other interviewees asked for a short explanation of the concept.

## 4.4 Causes of Transient Behavior

All participants identified deployments as the most significant cause of transient behavior in their projects. Torsten, Fabian, and Matthias expressed that failures and resilience mechanisms are, in their opinion, potentially the second biggest cause of transient behavior. While this case happens less frequently than deployments, they estimated that the consequences on the system's quality of service might be more critical. However, as Torsten highlighted, the transient behavior introduced in a system by resilience mechanisms is difficult to detect and trace. None of the interviewees claimed that they analyzed behavior introduced this way in detail.

Stefan and Matthias explained that they minimize transient behavior caused by deployments by employing continuous deployment methods. Although frequently deploying small code changes into production means that the system is regularly in a state of transient behavior, this behavior usually has only a small impact on the system's quality of service. On the other hand, big traditional deployments carried out every four to six months can cause transient behavior that substantially impacts the quality of service and potentially lasts for a much longer time. For example, Stefan recounted instances in which the caused transient behavior lasted for weeks.

Matthias pointed out that the use of modern cloud and containerization technologies combined with new architectural styles, like microservices or serverless functions, minimizes the duration and the magnitude of transient behavior, which is an improvement over previous technologies. For this reason, he thinks that transient behavior is not as big a problem for developers, architects, and stakeholders anymore as it used to be in the past. Uwe remarked that a big driver for undesired transient behavior is that engineers often ignore that most actions take a certain amount of time before they are completed and instead assume that they are completed instantaneously.

## 4.5 Identifying, Understanding, and Analyzing Transient Behavior

Torsten and Fabian observed that in their experience, the engineers at their company could identify, understand, and analyze the transient behavior in their software systems. Matthias stated that he thinks the operators tasked with analyzing and understanding the current transient behavior of a system can do so. On the other hand, Uwe estimated, from his experience, that only a fraction of all engineers would be able to complete the task universally and that only a fraction of them are actually analyzing the transient behavior of their software systems.

*4.5.1 Current Situation.* All interviewees mentioned that they are collecting data about the system behavior and its impact on the system during runtime through monitoring and log aggregators. It seems like a large body of data is already available that can be analyzed to understand the various transient behaviors that occur in a system, especially their causes and impacts on the system's quality of service. However, the interviewees also described that the amount of collected monitoring data is so large that it becomes overwhelming to examine it closely and carry out the complex analyses needed to understand the transient behavior fully. Usually, it is not considered critical or critical enough to warrant the required effort. Therefore, such analysis does not take place often. Torsten was the only one who claimed that his company actively attempted to understand and analyze the transient behavior in their software systems.

Torsten, Uwe, and Matthias stated that visualizations reduce some of the complexity of analyzing monitoring data. However, Torsten said that integrating such visualizations into their application monitoring tools is still in a prototypical state. Uwe expects it is challenging to develop visualizations that support engineers in understanding transient behavior in complex scenarios and systems.

Fabian explained that their analysis of monitoring data is error-driven, meaning they only look at the monitoring data after a failure that significantly impacted the system. The negative impact transient behavior has on the system is accepted by both engineers and customers.

Torsten, Stefan, Fabian, and Matthias said that such behavior should ideally already be tested during development to predict and understand its consequences. However, Torsten, Stefan, and Fabian also pointed out that such tests require complicated setups and are time-consuming. Thus, such behavior is only sparsely covered by test scenarios. Stefan explained that they often only discuss specific scenarios in theory for the same reason. They try to reason about if the system would be able to meet the specified quality requirements under these scenarios. Matthias thinks that test-driven development and the scaling and monitoring capacities of modern cloud platforms can be used to keep the impact of transient behavior under control. Small services that can be restarted quickly and straightforwardly mean that stopping and restarting a service that displays undesired behavior is a more straightforward solution than finding the underlying cause of the behavior.

When Torsten and Fabian walked us through their process for analyzing the monitoring data after a failure occurred, they mentioned that the first step is to exchange ideas with other engineers, highlighting that collaboration is critical in this task.

*4.5.2 Challenges.* The interviewees mentioned that the most common challenge for analyzing and understanding transient behavior is that it is not perceived as critical enough to merit analyzing a large amount of monitoring data to understand its causes and consequences. Engineers have many other higher priority responsibilities compared to considering transient behavior. Furthermore, Torsten stated that the biggest challenge is inefficient tooling, making it too time-consuming to analyze transient behavior. Especially lacking filter configurations for monitoring notifications and alerts means that relevant events are either not reported or overlooked between various notifications on irrelevant events. The abundance of notifications also leads to engineers eventually outright ignoring reports. The monitoring tool should only send notifications about critical events in the recipient's area of responsibility to improve this situation.

Another challenge that Uwe observes is to create visualizations of the monitoring data that are still helpful and easy to understand for complex systems and scenarios. Fabian also remarked that it is challenging to reproduce and test many scenarios that cause transient behavior like deployments. Stefan thinks this is complicated because some transient behaviors cannot be located with the help of monitoring tools.

Uwe thinks the main challenge in analyzing transient behavior is that parallel transient behaviors affect each other and form interrelationships between them, too complex to be easily understood by the human mind.

## 4.6 Specification of Transient Behavior

At the beginning of this part of the interview, we asked the interviewees if they think that transient behavior should be specified in non-functional requirements. Only Torsten agreed, in principle. On the other hand, Stefan, Uwe, and Matthias think that it is not necessary to explicitly specify requirements for this kind of software behavior in most projects. However, Stefan and Uwe also added that it depends on the criticality of the respective system. For some projects, it might pay off to conduct a risk assessment of the expected transient behavior concerning its occurrence probability and potential impact on the system. If such an assessment reveals that the system could be critically impacted by transient behavior, this behavior should be specified. Nonetheless, they do not think such a specification is necessary for the enterprise systems that they are developing and operating.

*4.6.1 Current Situation.* Torsten claimed that they already specify some aspects of transient behavior in quality scenarios, the same way as they capture other non-functional requirements. Software architects or test engineers usually create these formal scenarios. The other four interviewees stated that they do not explicitly specify non-functional requirements to capture transient behavior. However, they specify other non-functional requirements that sometimes implicitly also influence a system's transient behavior. Stefan, for example, said that they define informal quality scenarios to capture non-functional requirements that might also affect which transient behavior is acceptable. Fabian also explained that they define requirements like the maximum number of concurrent users that a system must handle. However, they do not specify quality requirements that the system must fulfill while scaling service instances in or out to comply with the other requirement. Matthias thinks that some requirements for transient behavior might be more interesting than others. His example was that startup time and scaling behavior are essential factors of serverless functions that get started and stopped frequently. On the other hand, the behavior during deployment is less critical in his opinion because modern platforms and methods already reduce the impact and extent of transient behavior during that phase.

*4.6.2 Challenges.* The biggest challenge for specifying transient behavior that all five interviewees named is that customers and business stakeholders are not aware of it and lack the knowledge to understand what it is, why it should be specified, and what their requirements are. Thus, it is difficult to find acceptance for the topic of transient behavior among these stakeholders, which corresponds to what they already said about non-functional requirements in general towards the beginning of the interviews. Torsten explained that non-technical stakeholders often have difficulties defining concrete numbers for non-functional requirements in general and transient behavior. Furthermore, they have problems assessing the trade-off between a better solution's development and operational costs and the impact of unspecified transient behavior. Architects and engineers often have to infer the requirements of their customers themselves.

Another challenge that Torsten and Fabian described is that it is difficult to define a response measure for quality scenarios that involve transient behavior, i.e., how the system should react if a specific behavior occurs. Here, input from the business stakeholders is needed again, which is challenging because of the issues mentioned previously.

Furthermore, Stefan, Uwe, and Matthias perceive other non-functional requirements as more important than those that specify transient behavior. Since it is already challenging to elicit these requirements from their customers, they would rather spend their effort on them than on additional requirements for transient behavior. They have other problems that have a higher priority than the specification of transient behavior and are willing to accept the consequences that unspecified transient behavior causes.

Finally, Torsten brought up that it can be challenging to identify quality scenarios for transient behavior and be confident that a scenario captures all aspects of a specific behavior. The fact that it is challenging to trace the causes of transient behavior impedes its formal specification.

## 4.7 Tools and Methods

We asked the interviewees how satisfied they are with the tools and methods they use to analyze and specify transient behavior and how these tools could be further improved. Afterward, we also inquired about the requirements that a new solution for these tasks would have to meet that employs novel HCI interfaces to support engineers.

*4.7.1 Existing Tools.* All five described that they take advantage of application performance monitoring tools to collect data about their software systems during the runtime. Sometimes they use proprietary solutions. Other times the toolset is determined by the customer. In general, none of them has a fixed set of tools for every project. To emphasize this, Matthias explained that they constantly search and try new tools to consult their customers with the latest technologies and methods. These monitoring tools are usually configured to issue alerts when unexpected behavior arises to notify the responsible operator. Torsten also revealed that they use log aggregators and data analytics engines like Elasticsearch[2] to monitor the behavior of an application. In his opinion, these tools lack adequate filters to notify engineers only about relevant events.

Torsten, Stefan, Uwe, and Fabian elaborated on their method for documenting non-functional requirements. They all use standard tools that are simple to use to capture their requirements, like different text processing applications and spreadsheets. Torsten stated that the essential factor in the specification of non-functional requirements is good communication with non-technical stakeholders, not tooling. Before the COVID-19 pandemic that started in 2020, they documented these quality requirements with post-it notes. Fabian said they collect non-functional requirements together with functional requirements in user stories that are tracked in a ticket management tool. However, this makes it difficult to find a specific requirement later on because they are collected in an unstructured fashion, and their tool lacks search functionality. The interviews exposed that a critical aspect of the specification is that it must be simple to capture requirements and change them later.

An interesting approach was presented by Stefan, who mentioned a method based on EventStorming[3] that was developed by one of his co-workers for the collaborative elicitation of quality requirements.

*4.7.2 Requirements for a Potential Solution.* A requirement that all interviewees had for a potential solution that takes advantage of novel HCI interfaces is that it must allow collaboration. For once, virtual collaboration in the form that multiple users can access and edit the same model or file simultaneously is a must. When multiple engineers are working together in person, it would also be helpful if there is a way to collaborate in that scenario, too. Especially since the importance of communication and collaboration have been repeatedly highlighted during the interviews.

Stefan established that a solution must be practice-oriented if it should be adapted in the industry. He means that the tool and the employed interfaces should be easy to learn and use. Uwe stressed this point by remarking that a solution approach should not worsen the current situation by adding additional complexity to the tasks. Furthermore, it must be possible to integrate a solution into the existing processes surrounding the concerned activities. Otherwise, it would find no adoption in practice. Since a big problem in dealing with transient behavior seems to be the lack of acceptance of the topic from business stakeholders, a solution would have to create value directly visible to them. Finally, most projects are entirely managed in version control systems like Git. Any solution would have to support this by ensuring that all created artifacts can be managed there without issues.

For Torsten, a solution must improve the problems he sees with the current tools in use. It needs to include filtering data and incidents to present only relevant data to the engineers who use it. Moreover, data representation and visualization must promote easy and fast identification of relevant behavior, its causes, and its impacts on the system. Fabian would like a filtering or search function not for monitoring data but for the requirements that specify transient behavior. He also thinks it would be helpful if a solution supports engineers in creating scenarios for automatic tests and verification of these requirements.

Uwe already mentioned that a solution would need to include visualizations that are still helpful for complex scenarios and systems. Matthias especially wants visualizations of the duration of transient behavior, a service's or the system's recovery time, and the services affected by it.

Multiple interviewees remarked that a solution that supports stakeholders in the specification of transient behavior must also be easy for non-technical stakeholders involved in this process. While engineers also expect rich options for configuring a solution, especially for analyzing transient behavior, usability should be the focus of a solution that supports various stakeholders in the specification of transient behavior.

*4.7.3 Thoughts on Novel HCI Interfaces.* Stefan, Uwe, and Matthias worry that the utilization of novel HCI interfaces for a solution would not help to increase its usability and effectiveness. Instead, in their opinion, the problems of modeling and visualizing transient behavior need to be solved first, as there is still a lack of capable solutions in this regard. Stefan is concerned that using such HCI methods would make a solution less practical and harder to adapt in the industry.

Torsten, Stefan, Uwe, and Fabian are wary of a solution that employs virtual reality (VR) or augmented reality (AR) because the technology is not yet prevalent. Furthermore, they are afraid that the technology could restrict communication between stakeholders, and there is a context switch when putting on a head-mounted display to start a VR application. Uwe, aware that it is difficult to create useful visualizations for complex transient behavior, is not convinced that VR or AR would solve this problem.

Whereas Fabian also has concerns about VR in its current state, he thinks that a solution that employs this paradigm could have potential in the future when VR is more broadly adopted and the technology's flaws are alleviated. The technology that has the most potential for a solution right now is, in his opinion, chatbots. He argues that bots are already commonly used in software engineering, for example, for notifications about failures or other critical system events. It would be a helpful extension of these bots if they allow him to interact with the information they present, e.g., restarting a service or the presentation of suggestions for possible next steps from the chatbot.

Uwe thinks natural and multimodal user interfaces are the most promising HCI approaches. In his opinion, these user interface paradigms will see broad adoption in the future, as they allow more natural interaction, similar to how humans communicate with each other. For this reason, they could eventually be used to facilitate the handling of such complex topics as transient behavior. However, he fears that many engineers would reject such a solution until these methods are more commonly employed.

Finally, Fabian and Uwe think that speech as a mode is less suited for such a use case because it has a low information density, and the user experience with voice assistants is still lacking, at least for now.

[2]https://www.elastic.co/enterprise-search
[3]https://www.eventstorming.com/

## 5 DISCUSSION

In this section, we discuss insights of our interviews related to the hypotheses presented in Section 3.1 and those identified additionally thanks to the explorative nature of the semi-structured interviews.

*Threats to Validity.* The sample size of our interviews was not big enough to make statements about the general population of engineers. However, the insights that we gained through the interviews helped us to have an impression of the requirements that software engineers have towards transient behavior and knowledge of how they currently handle it. We are only discussing the hypotheses for the same reasons without formally rejecting or accepting them.

### 5.1 Hypotheses

*$H_1$: Engineers are familiar with the concept of transient behavior in software? — Probably not.* While some interviewees, like Torsten, were more familiar with the concept of transient behavior than others, the concept was mostly new to them. When it was explained to them, they quickly grasped it. However, most of them had not heard the term "transient behavior" before we invited them to the interview. Therefore, it seems reasonably safe to say that $H_1$ should be rejected.

*$H_2$: Engineers understand the transient behavior in their systems? — It depends.* It is not easy to provide a general estimate to $H_2$ based on the interviewees' statements. While two of them stated that their engineers would be able to identify, understand, and analyze transient behavior, four interviewees shared that transient behavior is usually not analyzed in their organizations. Furthermore, Uwe estimated that only a fraction of all engineers would be able to analyze complex transient behavior that can cascade from one microservice to another. After refining the hypothesis, future work could attempt to answer this question through quantitative experimentation.

*$H_3$: Engineers think that the transient behavior of software should be specified in non-functional requirements? — It depends.* There is no clear answer to $H_3$. While Torsten thinks that transient behavior should generally be specified, most of the other interviewees clarified that this is often unnecessary. However, they stated that a specification might make sense for critical systems, which could severely be impacted by transient behavior. Therefore, $H_3$ can neither be confirmed nor rejected, as this question is too complex to be covered by a single hypothesis that does not distinguish between different kinds of systems.

*$H_4$: Engineers specify the transient behavior of their software systems in non-functional requirements? — Mainly not.* The division between the interviewees seems similar concerning $H_4$. Torsten claimed that they attempt to specify transient behavior in non-functional requirements in their projects. The other four interviewees stated that they do not explicitly capture non-functional requirements for transient behavior. While several of them mentioned that some of the non-functional requirements they specify also influence transient behavior, these requirements are not created with transient behavior in mind. For this reason, it seems like $H_4$ should mostly be rejected.

*$H_5$: Engineers are satisfied with existing methods and processes for analyzing and specifying non-functional requirements for the transient behavior of software? — Mixed: non-technical stakeholders are a challenge, too.* There is again no clear answer to $H_5$. Torsten stated that tooling is one of their biggest problems in analyzing transient behavior. Similarly, Fabian expressed that reproducing and testing scenarios that involve transient behavior is challenging. On the other hand, Uwe thinks that communication with non-technical stakeholders is a more significant challenge while tooling plays less of a role. He claimed to be satisfied with the current tools and methods they employ.

*$H_6$: Engineers collaborate when they are analyzing or specifying non-functional requirements for the transient behavior of software? — Yes.* The situation is different with $H_6$. All interviewees agreed that collaboration is essential when dealing with the different aspects of transient behavior. It especially plays a big part in the specification because both non-technical and technical stakeholders have to work together. Therefore, we argue that our interviews confirmed $H_6$.

### 5.2 Additional Insights

In addition to the presented findings directly related to the hypotheses, we identified the following insights from the interviews that are related to transient behavior:

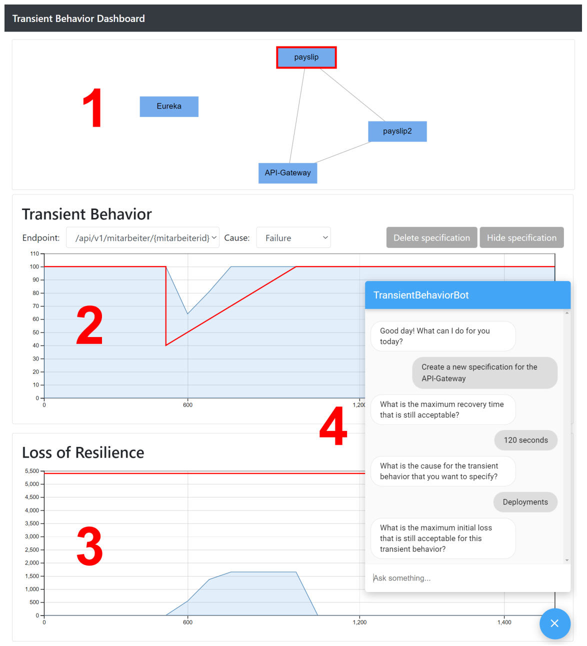
(1) The biggest challenge when eliciting any non-functional requirements is a lack of awareness and understanding from business stakeholders.
(2) Transient behavior is not considered critical enough to be analyzed.
(3) Deployments are seen as the most significant cause of transient behavior in enterprise applications.
(4) Frequently having transient behavior with low quality loss and recovery time is preferred over infrequent transient behavior with high quality loss and recovery time.
(5) Test scenarios should cover transient behavior; however, they rarely do due to the complexity of these scenarios.
(6) Simultaneous transient behaviors can affect each other, making it challenging to understand them.
(7) When specifying non-functional requirements, simple tools are preferred over feature-rich ones.
(8) Novel HCI interfaces are not broadly adopted in software engineering processes.

Moreover, we identified some insights addressing the wider context of application performance management (APM) [8]:

(1) Better analytical solutions are needed to understand large amounts of APM data.
(2) There is a lack of easy-to-understand visualizations for APM data of complex scenarios.
(3) Filtering and easy-to-understand visualizations are key for dealing with large amounts of APM data.

## 6 FOLLOW-UP PROTOTYPES

Our findings reveal that specification and analysis of transient behavior are rarely performed in the industry. The primary reasons for this are the challenge of eliciting requirements from business stakeholders and a lack of effective tooling to analyze transient behavior within a wealth of APM data. We developed two early prototypes to improve this situation and explore potential research directions.



Good day! What can I do for you today?

Create a new specification for the API-Gateway

What is the maximum recovery time that is still acceptable?

120 seconds

What is the cause for the transient behavior that you want to specify?

Deployments

What is the maximum initial loss that is still acceptable for this transient behavior?

**Figure 2: TransVis includes visualizations for a system's (1) architecture, each service's (2) transient behavior and (3) loss of resilience, and a (4) chatbot.**

We have achieved the following complementary intermediate results toward our overall vision of improving the specification and comprehension of transient software behavior:

### 6.1 TransVis

TransVis [2] focuses on the specification and comprehension of transient behavior. TransVis presents a graph-based visualization of the system's services that indicates potential violations. For each service, the violations and measurements can be visually investigated and compared against simple specifications based on Bruneau's resilience triangle model. An integrated chatbot assists the user in their tasks, e.g., adding, deleting, and showing specifications. Figure 2 shows the TransVis dashboard. A user study [2] revealed that the visualizations are effective for specifying and exploring transient behavior. However, the chatbot was rarely utilized by the participants.

### 6.2 Resirio

Resirio [14] is a tool that can import execution traces and conducts a CHAZOPS-based [5] hazard analysis on the extracted architectural model. The results are presented in a graph-based architecture visualization to help users in interactive elicitation of resilience scenarios. The user can then specify the resilience scenario in a conversation with a chatbot. We conducted a user study with participants from industry and academia to evaluate Resirio's usability, effectiveness, and support. The evaluation shows that the developed prototype gives novice requirements engineers a foundation for fast requirements elicitation and that Resirio complements traditional requirements engineering approaches.

## 7 CONCLUSION

Based on the question of how transient behavior is addressed in practice, we interviewed five experienced software architects from the industry. We discovered that transient behavior is seldom specified in requirements due to a lack of knowledge from stakeholders and a low prioritization. Furthermore, we observed that better tooling is needed for the analysis of transient behavior. Besides improved filtering options for incidents, this includes improved visualizations and models for transient behavior. Building on the interview findings, we have provided a brief overview of our tool prototypes, TransVis and Resirio. In addition to the presented tools, we are developing concepts and editors for the interactive transformation of ATAM-based resilience scenarios into more formal representations. This step is intended to extend the requirement elicitation workshop targeting resilience and transient behavior [7].

## ACKNOWLEDGMENTS

We thank the interview participants. This research has received funding from the Baden-Württemberg Stiftung (Orcas project) and the German Federal Ministry of Education and Research (Software Campus 2.0 — Microproject: DiSpel).

## REFERENCES

[1] Zahra Shakeri Hossein Abad, Mohammad Noaeen, and Guenther Ruhe. 2016. Requirements Engineering Visualization: A Systematic Literature Review. In *24th IEEE Int. Requirements Engineering Conference, RE*. IEEE Computer Society, 6–15.
[2] Samuel Beck, Sebastian Frank, Alireza Hakamian, Leonel Merino, and André van Hoorn. 2021. TransVis: Using Visualizations and Chatbots for Supporting Transient Behavior in Microservice Systems. In *2021 Working Conference on Software Visualization (VISSOFT)*. 65–75.
[3] Samuel Beck, Sebastian Frank, Mir Alireza Hakamian, and André van Hoorn. 2022. *How is Transient Behavior Addressed in Practice? Insights from a Series of Expert Interviews – Supplementary Material.* https://doi.org/10.5281/zenodo.6425604
[4] Michel Bruneau et al. 2003. A framework to quantitatively assess and enhance the seismic resilience of communities. *Earthquake spectra* 19, 4 (2003), 733–752.
[5] JV Earthy. 1992. Hazard and operability study as an approach to software safety assessment. In *IEE Colloquium on Hazard Analysis*. IET, 5–1.
[6] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. 2016. Are "non-functional" requirements really non-functional?: an investigation of non-functional requirements in practice. In *Proceedings of the 38th International Conference on Software Engineering, ICSE*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 832–842.
[7] Sebastian Frank, M. Alireza Hakamian, Lion Wagner, Dominik Kesim, Jóakim von Kistowski, and André van Hoorn. 2021. Scenario-based Resilience Evaluation and Improvement of Microservice Architectures: An Experience Report. In *ECSA 2021 Companion Volume*, Vol. 2978.
[8] Christoph Heger, André van Hoorn, Mario Mann, and Dusan Okanovic. 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, ICPE*. ACM, 429–432.
[9] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. 2019. On Observability and Monitoring of Distributed Systems - An Industry Interview Study. In *Proceedings of the 17th International Conference on Service-Oriented Computing, ICSOC*, Vol. 11895. Springer, 36–52.
[10] Valentin Andreevich Venikov. 2014. *Transient Phenomena in Electrical Power Systems: International Series of Monographs on Electronics and Instrumentation, Vol. 24*. Vol. 24. Elsevier.
[11] Chang-Yu Wang, Dimitris Logothetis, Kishor S Trivedi, and Ioannis Viniotis. 1996. Transient behavior of ATM networks under overloads. In *Proceedings of the 1996 IEEE Conference on Computer Communications, INFOCOM*, Vol. 3. IEEE, 978–985.
[12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer.
[13] Kanglin Yin, Qingfeng Du, Wei Wang, Juan Qiu, and Jincheng Xu. 2019. On Representing and Eliciting Resilience Requirements of Microservice Architecture Systems. *arXiv preprint arXiv:1909.13096* (2019).
[14] Christoph Zorn. 2021. *Interactive Elicitation of Resilience Scenarios in Microservice Architectures.* Master's thesis. University of Stuttgart.