TransVis: Using Visualizations and Chatbots for Supporting Transient Behavior in Microservice Systems

Samuel Beck*, Sebastian Frank*, Alireza Hakamian*, Leonel Merino† and André van Hoorn*
*University of Stuttgart, Germany
†Pontifical Catholic University of Chile, Chile

***Abstract*—In a microservice system, runtime changes such as failures, deployments, or self-adaptation can trigger the system to transition from one steady state to another, i.e., exhibiting transient behavior. To assess a system's quality, it is imperative that this transient behavior is specified in non-functional requirements and that stakeholders can analyze whether these requirements are met. Yet, there is little support for either specifying transient behavior as a non-functional requirement or analyzing how such a requirement is met in production. We aim to make these two tasks more accessible by utilizing novel human-computer interaction methods. To this end, we developed *TransVis*, an approach for specifying and analyzing transient behavior based on chatbot interactions and visualizations of the systems' resilience. We examined the effectiveness of our approach by conducting an exploratory expert study on a prototypical implementation. The study revealed that the developed visualizations are effective for specifying and exploring transient behavior. Participants found especially helpful the feature to compare specifications with the actual behavior. However, the integration of a chatbot did not prove effective for our use cases. In conclusion, our approach is capable of supporting stakeholders in the exploration and specification of transient behavior.**

***Index Terms*—transient behavior, microservices, requirements, software quality, visualization, chatbots**

I. Introduction

Modern software systems are complex, highly distributed, and subject to frequent change during runtime. To uphold their service quality is a great challenge under these circumstances. Nonetheless, a multitude of software systems is highly critical, and failures can cause enormous harm and costs [1]–[3]. For this reason, non-functional requirements for various quality attributes of software must be specified and continuously assessed during the runtime. However, the numerous changes that a system must endure complicate this task. For instance, DevOps [4] enables developers to deploy new software releases frequently. Microservice systems [5] are susceptible to service failures that can propagate through the system. To prevent them, resilience mechanisms such as circuit breakers can automatically change the service behavior. Furthermore, systems are exposed to unpredictable workloads and attacks from malicious users.

All these sources of change imply that microservice systems are almost continuously in a state of transient behavior, i.e., in a transition from one steady state to another (see Section II). For that reason, to ensure a system's reliability and performance, this behavior must be specified in non-functional requirements, which stakeholders continuously evaluate. However, due to the complexity of this transient behavior and the lack of specifications, these demanding tasks are often restricted to experts. Czepa and Zdun [6] recently found evidence that both textual and graphical formal specification languages are difficult to use for novice developers. This serves as an incentive to create novel solutions for capturing non-functional requirements for transient behavior. Methods and technologies from human-computer interaction (HCI) have successfully been applied to problems in other domains to increase user performance and user experience. For instance, Merino et al. [7] created CityVR, an interactive software visualization tool that visualizes software in virtual reality as a city metaphor. They observed that developers felt immersed and excited when interacting with the visualization. To name another example, Toxtli et al. [8] deployed a chatbot to help users create, assign, and keep track of tasks in their team. Study participants mostly reported increased productivity due to the chatbot. Current requirement models such as the approach by Yin et al. [9] do not leverage such methods to boost their effectiveness. Consequently, we hypothesize that these benefits of HCI methods such as chatbots can be transferred to transient analysis and help to address the challenges of comprehending and capturing the transient behavior of software systems.

To this end, we created the TransVis approach proposed in Section III that utilizes visualizations and a chatbot to support software architects and DevOps engineers in the specification and analysis of transient behavior. To examine its effectiveness, we developed a prototypical implementation of the approach and conducted an expert study that is presented in Section IV. The study revealed that the TransVis approach is effective for specifying and analyzing transient behavior and is easy to learn and use. Especially the included visualizations are helpful in these tasks. We discuss the results of the study in detail in Section V.

Supplementary material, like code and the study results, is publicly available online [10].

II. Transient Behavior

The term transient behavior has its origin in electrical engineering. Steady-state and transient are two possible solutions to the state-space system model. The transient analysis

captures the time-dependent behavior of the system's quality of service (QoS) [11]. Formally speaking, transient analysis is concerned with the probability of being in a specific state at time t. On the contrary, in steady-state analysis, the notion of time disappears, and the probability distribution of being in each state independently of the initial distribution converges to a unique probability distribution [12]. Venikov defines that transient behavior or a transient state "occurs [in an electrical power system] when the system is changing from one steady state to another" [13]. Software systems are subject to frequent change while operating in production and experience similar transitions between steady states. Frequent deployments, failures, and self-adaptation disrupt the QoS provided by a microservice. Accordingly, the phenomenon of transient behavior can be adopted from electrical engineering to the software domain.

A. Modeling Transient Behavior

Before transient behavior can be specified in non-functional requirements, this abstract concept must be quantified by concrete metrics. To this end, we utilize the resilience triangle proposed by Bruneau et al. [14], illustrated in Figure 1, in an approximated form as presented by Zobel [15]. A resilience triangle visualizes the recovery of a system from a disruption [9]. We propose the use of this visualization to illustrate transient behavior in software systems.

Thereby, transient behavior can be quantified by an initial loss of QoS, the time to recovery from the behavior, and the resulting loss of resilience that the system suffers during the period of transient behavior. To compute these properties, a QoS function Q must be defined. Suitable functions could be the response time or success rate of a service. Also, it is necessary to declare an expected value Q_e for the QoS function based on which the other metrics are computed. The initial loss X denotes the loss of quality at the beginning of the occurrence of transient behavior and is given by Equation 1. $Q_e(t_0)$ is the expected QoS and $Q_a(t_0)$ the actual QoS at t_0.

$$X = Q_e(t_0) - Q_a(t_0) \tag{1}$$

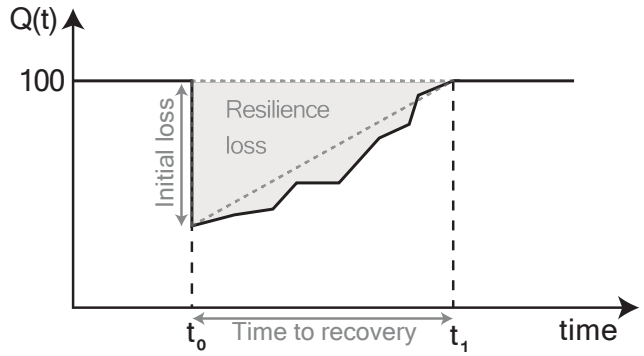

Fig. 1: Bruneau's resilience triangle model [14] on a resilience curve, adopted from Yin et al. [9].

The time to recovery describes a transient behavior's duration and is defined in Equation 2, where t_0 is its start time and t_1 its end time.

$$T = t_1 - t_0 \tag{2}$$

The area of the triangle formed by these metrics approximates the loss of resilience suffered during the behavior. It is given by Equation 3.

$$R = \frac{X \cdot T}{2} \tag{3}$$

Subsequently, we use the initial loss, the time to recovery, and the loss of resilience to quantify transient behavior.

B. Transient Behavior in Practice

In previous work [16], we reported on five expert interviews with experienced software architects in the software consulting business. We investigated how developers, architects, and operators handle transient behavior in practice. We observed that transient behavior is not a well-known concept among many software architects and developers. Furthermore, most companies in the enterprise application domain do not specify transient behavior in non-functional requirements. The reasons are (i) the specification of transient behavior is seen as unnecessary for most projects, and (ii) business stakeholders often lack knowledge about transient behavior and its consequences. With regards to the analysis of transient behavior, faced with an overwhelming amount of data, the interviewees lamented ineffective tooling that lacks filtering of incidents. Motivating this work, they miss a model and visualizations for transient behavior and its consequences.

III. TransVis Approach

We propose the TransVis approach to support software architects and DevOps engineers in the specification and assessment of transient behavior in microservice systems. First, we introduce four use cases that the approach covers. Afterward, we propose a set of visualizations for the introduced metrics and use cases. We also present how they can be combined into a dashboard and detail a prototypical implementation.

A. Use Cases

We propose four use cases for our TransVis approach that build on our previous work [16]. Consequently, they are based on the findings of the expert interviews about transient behavior described in Section II-B.

UC1: Specify the acceptable transient behavior of a service from the software architect's viewpoint.

UC2: Find occurrences of transient behavior in a service from the DevOps engineer's viewpoint.

UC3: Verify the compliance to the requirements for transient behavior of a service from the DevOps engineer's viewpoint.

UC4: Analyze the impact of transient behavior on dependent services from the DevOps engineer's viewpoint.

In the following, we describe each use case and detail the metrics that need to be captured to fulfill them.

1) UC1: Specifying Transient Behavior: Transient behavior's complexity makes its specification via formal requirements difficult. This is aggravated by a lack of knowledge of transient behavior among business stakeholders and its perceived low priority. Indeed, our previous expert interviews [16] have revealed that such requirements are usually not captured in a majority of projects. Facilitating their specification through modern human-computer interfaces and suitable visualizations might reduce the effort to a level where this practice becomes increasingly implemented in real-world software projects.

The quality of a system might be impacted by multiple causes that differ in severity and duration. We expect a different behavior after a service instance failure than during deployment. To take this into account, a specification is always created for a specific cause of transient behavior and one service.

Furthermore, we use the metrics defined in Section II-A to specify what can be considered acceptable transient behavior. For this reason, a specification is given by the acceptable initial loss, the acceptable time to recovery or duration, and the resulting acceptable loss of resilience. These values are defined for a chosen QoS function and the expected QoS value.

2) UC2: Finding Transient Behavior: In line with our previous findings [16], we consider the identification of critical transient behavior that impacts the user experience to be essential. Also, to verify whether a service's transient behavior stays within a defined specification, it is necessary to identify occurrences of transient behavior within the performance measurements of the service. To this end, a QoS function is chosen to illustrate the behavior of the service. Temporary degradations of this metric indicate the occurrence of transient behavior. The bounds of the exhibited transient behavior are marked by the time point t_0 of the initial loss and the time point t_1 at which the QoS completely recovers from the disruption.

3) UC3: Verifying Transient Behavior: Specifications are only valuable if they can be verified. For this reason, another important use case is to verify whether a service complies with its requirements for transient behavior. The precondition of this use case is that we have identified instances of transient behavior in the performance measurements, as described in Section III-A2. Once transient behavior has been detected, whether it complies with the specification is not trivial to answer. The reason for this is that our quantification of transient behavior comprises multiple aspects.

When analyzing transient behavior, the difference between the actual and specified initial loss, time to recovery, and loss of resilience should be considered. If any of these attributes exceeds the specification, the requirements could be considered to be violated. On the other hand, one could argue that only the violation of all three attributes constitutes non-compliance with the requirements. Therefore, the question of requirement violation should be clarified in advance to avoid confusion.

4) UC4: Analyzing the Impact of Transient Behavior: The services of a microservice system are usually interdependent. Because of this, transient behavior manifesting in one service can propagate and cause transient behavior in dependent services, which is of interest to architects and engineers [16].

It is essential to have an overview of the system architecture to answer how transient behavior impacts the behavior of other services. Knowledge about the number of instances, the endpoints, and the dependencies of each service is required. To investigate whether transient behavior in one service leads to transient behavior in dependent services, one needs to observe the QoS of each dependent service precisely at the time the original transient behavior occurs. If an instance of transient behavior coincides in a dependent service, they might be correlated.

B. Visualizations

We consider that software architects and DevOps engineers would benefit from visualizations of the transient behavior of microservices in their daily work. Consequently, we present various visualizations for different aspects of transient behavior designed to support them in the four use cases. For this purpose, we combine the visualizations into a transient behavior dashboard.

1) Architecture Visualization: In our approach, we follow the information-seeking mantra by Shneiderman [17], and consequently, first present the user with an overview of the state of the system. Then, we provide details for single services on demand. The view contains the system's architecture and encodes information about the compliance of observed transient behavior to its requirements, facilitating UC2.

The architecture of a microservice system forms a graph [18] with the services as vertices and the dependencies as edges. A common visualization for graphs is the node-link diagram [19], which we have chosen for our architecture visualization. Figure 2 shows how the visualization looks like for a small example system. The diagram shows the names and interdependencies of the services. The yellow color of the `Passenger Management` service represents a potential violation of the requirements for transient behavior in this service. This makes the state of each service visible at a glance. The visualization allows the selection of single services to show further details about their transient behavior. A selected service is represented by a red frame, such as the `Payment` service in the example.

2) Transient Behavior Visualization: When a service is selected, additional visualizations are displayed that support the user in analyzing transient behavior (UC3) and creating

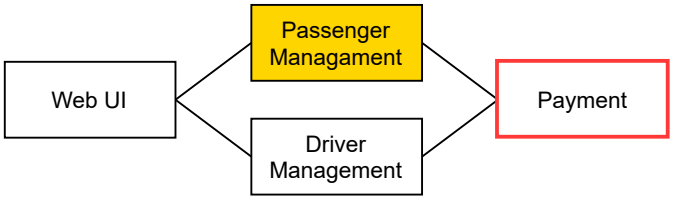

Fig. 2: Architecture visualization.

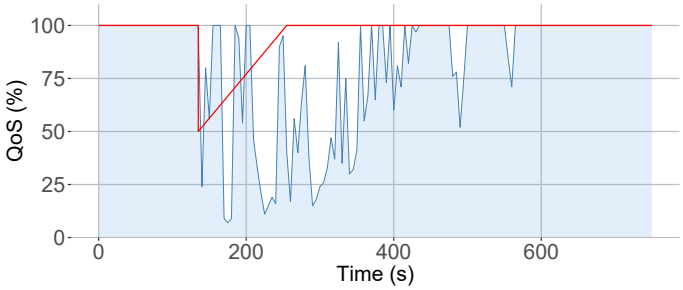

Fig. 3: Transient behavior visualization.

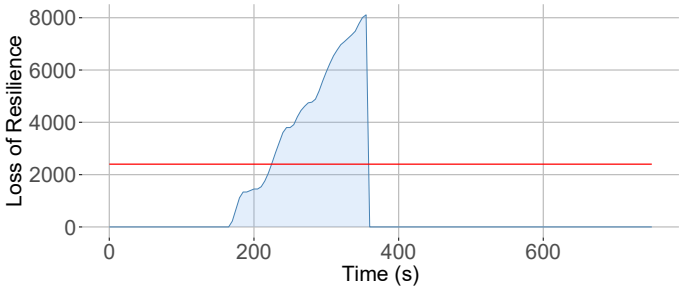

Fig. 4: Loss of resilience visualization.

specifications for it (UC1). The main view is the transient behavior visualization, an area chart that represents the chosen QoS function for a selected service endpoint. The user can change the displayed endpoint. It is plotted on the y-axis on a scale from 0% to 100%, where 100% is the expected QoS. The x-axis represents time in seconds. The resulting visualization is illustrated in Figure 3. An instance of transient behavior can be recognized by a decrease in QoS, marking the initial loss. The time that it takes until the QoS returns to a stable 100% is the time to recover. The white area between the QoS function and the expected QoS (100%) is the loss of resilience.

3) Specification Visualization: In Section II-A we introduced our quantification of transient behavior via resilience triangles. We use this visual mapping to represent specifications for transient behavior in the transient behavior visualization to satisfy UC1. As Figure 3 shows, the initial loss is represented by the left-hand side of the triangle, the time to recovery by the length of the side at the top, and the loss of resilience by its area. To assess the compliance of a service to the requirements for transient behavior, each aspect of transient behavior must be compared with the specification. The visualization allows this by placing a resilience triangle, representing the specification, on top of each occurrence of transient behavior. Now, the initial loss can be compared by comparing the decrease of the QoS function with the side of the triangle. Similarly, the duration of the behavior can be compared by examining whether the QoS function returns to a value of 100% before the plotted specification does. In the example shown, the system's behavior exceeds all three aspects of the specification.

Although the visualization is effective to identify deviations of a system's behavior from its specification, we observe that comparing the loss of resilience is difficult as the visualization offers little support to compare the size of the two areas. Consequently, we designed a visualization of the loss of resilience as a separate chart.

4) Loss of Resilience Visualization: The visualization for the loss of resilience suffered during transient behavior is depicted in Figure 4. The x-axis of the chart represents the time, the y-axis the loss of resilience. It is only plotted when transient behavior is detected, during which the loss of resilience increases until the QoS returns to its expected value and the system reaches a new steady state.

To enable comparison of the loss of resilience with the specification, facilitating UC3, a red line represents the acceptable loss of resilience in the chart. If the loss of resilience exceeds this line, as illustrated, this aspect of the specification is violated. In this case, the service is highlighted with an orange background color in the architecture visualization. This allows the user to identify services that may have violated their specifications quickly.

5) Dashboard: The presented visualizations are integrated into an interactive dashboard that is illustrated in Figure 5. The architecture visualization (1) at the top provides the user with an overview of the system's services and indicates those that experience transient behavior that is potentially outside the requirements' limitations. When the user selects a service, more details are displayed. The transient behavior visualization (2) shows a QoS function over time for a single service endpoint. The visualized endpoint can be changed in a selection box above the chart. Furthermore, the user can examine the transient behavior exhibited by the endpoint by interacting with the chart, i.e., hovering over single data points and geometric zooming. Specifications can be created directly in the dashboard by selecting the cause of the behavior and entering the acceptable initial loss and time to recovery. The resulting acceptable loss of resilience is calculated by the tool. We consider three causes of transient behavior: (i) failures, (ii) deployments, (iii) and load balancing (self-adaptation).

To enable users to interact with the visualizations through natural language, we integrated a chatbot in the dashboard.

C. Chatbot

We observed that engineers can benefit from getting insights about their system through easy-to-use tooling [16]. Using a visualization approach requires users to be familiar with the employed encodings, techniques, interactions, and configurations. Bieliauskas and Schreiber [20] propose the use of conversational interfaces to aid the interaction with complex visualizations. They name accessibility to new users and more natural interaction as benefits of such a combination. Indeed, natural user interfaces, which aim to provide smooth user experiences where technology is becoming invisible to the user, are growing in popularity as they mature [21]. Furthermore, the combination of multiple interaction modes into multimodal interfaces potentially increases task efficiency and enables humans to process information faster and better [22]. For these

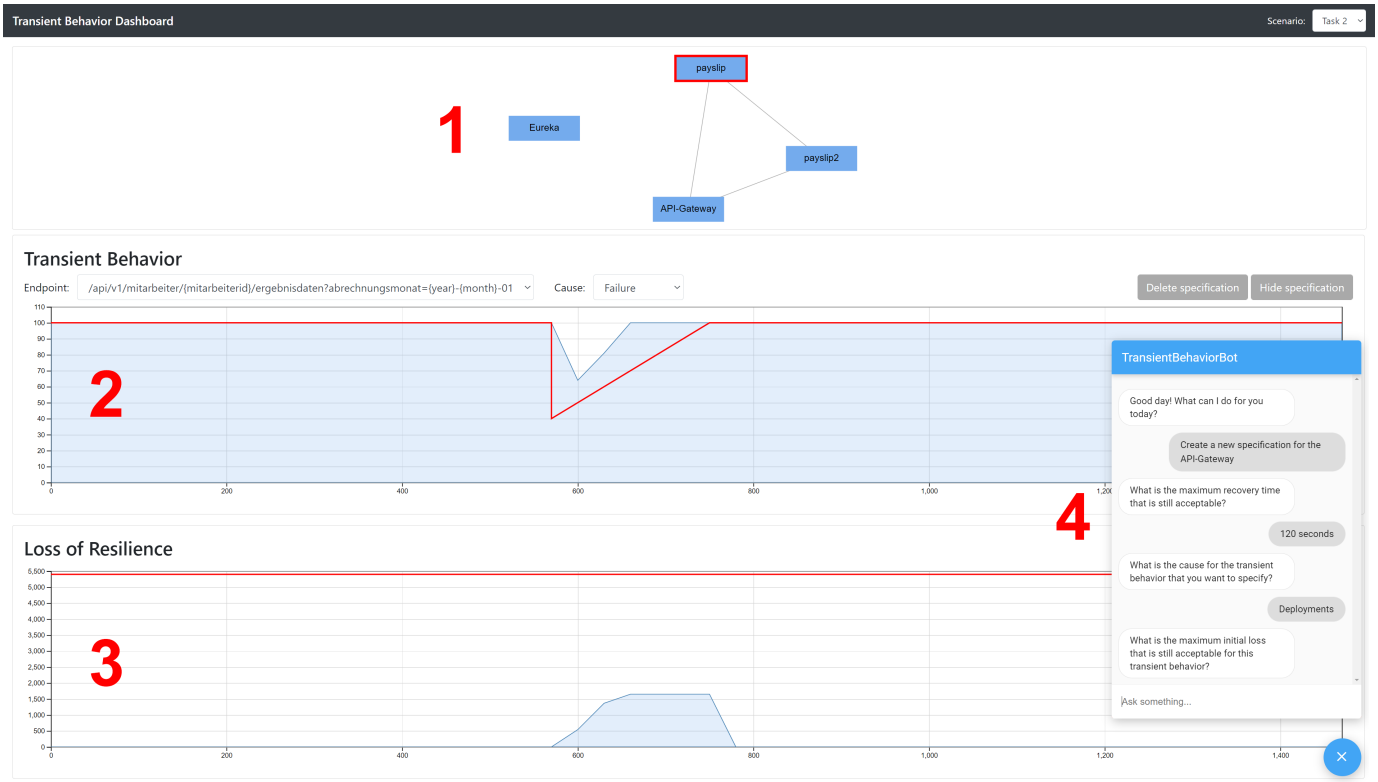

Fig. 5: Screenshot of the dashboard showing transient behavior fulfilling a specification. (1) Architecture visualization, (2) transient behavior visualization, (3) loss of resilience visualization, (4) collapsible chatbot window.

reasons, we decided to extend our interactive visualizations with a chatbot.

Developers already interact frequently with bots [23], they are employed to speed up coding, testing, and deployments [23]. They simulate written or spoken conversations and give the impression of interacting with another human being [24]. Besides answering questions and providing information, they can act as a conversational interface to a more extensive service [25]. Unsurprisingly, they also start to find their way into software engineering [26], [27].

Chatbots rely on natural language processing methods to understand user input. The two main concepts employed are *intents* and *entities*. A chatbot classifies each user input as an intent, i.e., what the user wants to do or know. This is achieved by training a machine learning model with training phrases mapped to the respective intents. Later, this model allows the chatbot to classify similar input phrases to the correct intent. Entities can be thought of as the parameters of an intent. Each entity has a type, such as duration, date, or location. The chatbot is trained to recognize entities in the input by annotating training phrases. It can prompt the user for an entity that is missing from an input phrase [28]. Figure 6 shows how an intent is structured and how a conversation with a chatbot takes place.

We designed our chatbot to enable interaction with the visualizations through natural language. Users can interact with it through a collapsible chat window on the right side

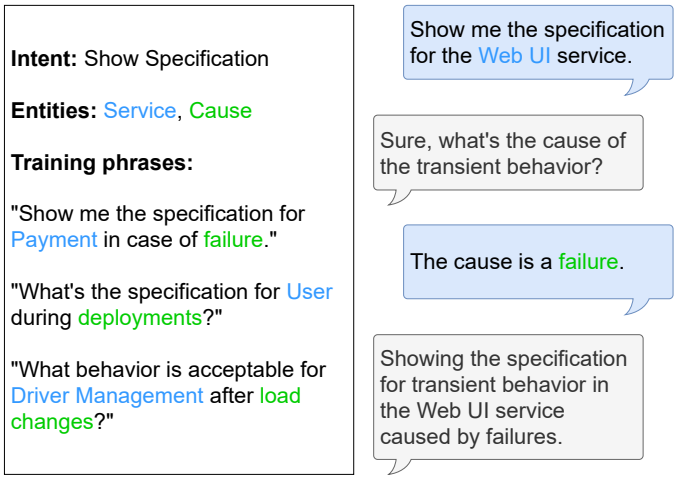

Fig. 6: Intent definition on the left, exemplary chatbot conversation on the right.

of the dashboard. As a result, the chatbot is always readily available while not being obstructing. Furthermore, the chatbot can be used to create, edit, or delete specifications for transient behavior. It supports the following intents.

1) Select Service: Makes it possible to select a service in the architecture visualization to show its endpoints and their transient behavior. It has one entity: the name of the service that the user wants to select. An example phrase would

be "*Please select the Payment service*".

2) Show Specification: Enables the user to ask the chatbot to show the transient behavior specification of a specific service caused by a specific cause. It has two entities: the name of the service and the cause of the transient behavior. An example phrase would be "*Show me the specification for the Web UI service for transient behavior caused by failures*". The definition of this intent and an example conversation are depicted in Figure 6.

3) Add Specification: Permits the creation of a new specification for transient behavior through the chatbot. It has four entities: the name of the service for which the specification will be created, the cause of the specified behavior, the acceptable initial loss, and the acceptable time to recovery. An example phrase would be "*During a deployment, the initial loss in the Driver Management service should not exceed 30% and the duration should not be longer than three minutes*".

4) Edit Specification: Similarly, the acceptable initial loss and time to recovery of an existing specification can be changed. The intent has three entities: the name of the service, the cause of the specified transient behavior, and either the updated acceptable initial loss or time to recovery. An example phrase would be "*Change the acceptable initial loss for transient behavior caused by a failure in the Web UI service to 20%*".

5) Delete Specification: Makes it possible to delete a specification for transient behavior. It has two entities: the name of the respective service and the cause of the specified transient behavior. An example phrase would be "*Remove the specification for transient behavior in the Payment service caused by load changes*".

6) Help: Gives advice to users who need help to use the chatbot and its functionality. This intent has no entities. An example phrase would be "*What can I ask you?*".

D. Implementation

We implemented a prototype[1] to evaluate the feasibility of the proposed approach. For this purpose, we decided to develop a web application with the architecture illustrated in Figure 7. The prototype consists of three components: a central backend component, the dashboard component, and the chatbot component. The Python backend provides Representational State Transfer (REST) interfaces for the dashboard and the chatbot. Furthermore, the components can communicate via WebSockets to allow the backend to issue messages to the dashboard. This is needed to enable interaction between the chatbot and the visualizations.

The backend component is implemented with Django[2], a popular web framework. The QoS data and architectural information about the subject systems used in our expert study are stored in a relational database. The services and their dependencies were entered manually. The QoS data contained information such as success rate and response time for each

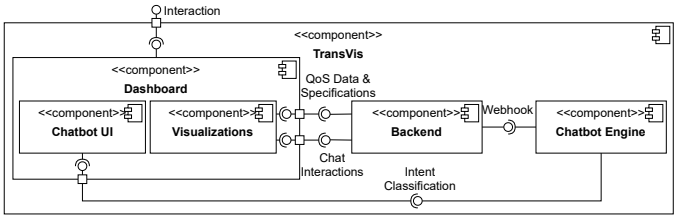

Fig. 7: The architecture of the developed prototype.

service endpoint. When the application starts, the dashboard component requests the architectural information from the backend to display the architecture visualization.

The dashboard is realized as a Hypertext Markup Language (HTML) site that uses JavaScript and the D3.js[3] library to create the visualizations. When a service is selected in the architecture visualization, the component requests the QoS data for this service and presents it in the transient behavior and loss of resilience visualizations. Additionally, the dashboard provides users with the functionality of specifying transient behavior for selected services. For this purpose, it provides a user interface where the cause, acceptable initial loss, and time to recovery that constitute a specification can be defined. The resulting loss of resilience is calculated automatically. Created specifications are also stored in the database. When the dashboard finishes loading, it establishes a WebSocket connection to the backend and listens for messages.

The user can also interact with the dashboard via the chatbot component. The chatbot is implemented with the Dialogflow[4] framework by Google. The framework comes with a free version, provides an easy-to-use web interface for training, and allows webhooks that process intents. Moreover, it offers a chat interface that can conveniently be integrated into an existing website. When natural language input is entered into the chatbot, it is classified by Dialogflow. If an intent is detected, Dialogflow forwards it and all recognized entities to the backend via a webhook. The chatbot will automatically prompt for required entities and only forward input that contains all of them. The backend acts on the intent, for example, by creating a new specification and returns a natural language response containing the requested information to the chatbot, which is displayed to the user. If the intent requires interaction with the visualizations, the backend will send a WebSocket message to the dashboard, which adapts the visualizations accordingly. For example, this is the case when the chatbot is used to select a service or display a specification.

The specifications of transient behavior are drawn directly on top of occurrences of transient behavior in the data. To achieve this, we created an algorithm that detects significant transient behavior in the QoS data. The algorithm iterates over all data items of an endpoint and looks for cases where the QoS drops below the expected value. If this is the case, we must verify whether this is only a minor variance or significant

[1]https://transient-bot.github.io/transient-chatbot-viz/

[2]https://www.djangoproject.com/

[3]https://d3js.org/

[4]https://cloud.google.com/dialogflow/

transient behavior. To this end, the algorithm computes the median QoS value of the subsequent five data items and checks if it falls short of a defined threshold. If this applies, the data item is marked as the beginning of transient behavior, i.e., as the time of the initial loss. Next, the algorithm identifies the end of the behavior by searching for the next data item at which the QoS returns to the expected value without deviation. If the specified time to recovery surpasses the duration of the located transient behavior, the algorithm will assign the specified time to recovery as the duration of the behavior to prevent overlapping specifications in the plot.

IV. Expert Study

We conducted an exploratory expert study to evaluate the effectiveness of the presented TransVis approach. In this section, we will detail the goals and design of the study, the selected participants, and the procedure that we followed during the study.

A. Study Design

Using the template by Wohlin et al. [29], the goals of our study can be summarized as:

> *Analyze* the TransVis approach *for the purpose of* supporting software architects and DevOps engineers in the specification and verification of transient behavior *concerning* its effectiveness and usability *from the point of view of* software architects and DevOps engineers.

We formulate the following research questions to address these goals:

RQ1: How effective is the TransVis approach in supporting architects and engineers in the specification of transient behavior (UC1)?

RQ2: How effective is the TransVis approach in supporting architects and engineers in assessing specifications for transient behavior (UC2 and UC3)?

RQ3: How good is the usability of the TransVis approach?

RQ4: How helpful is the integration of a chatbot in the TransVis approach?

We decided to conduct an exploratory expert study as our research questions require to involve experienced software architects and DevOps engineers. Not only had the participants to be well acquainted with the architecture, quality requirements, and performance monitoring of the test system, but they also needed to be familiar with the concept of transient behavior, which limits the number of potential candidates. Furthermore, Isenberg et al. [30] identified a trend towards experts studies among publications at renowned visualization conferences to examine the value that solutions create for domain experts. A small number like three to five highly knowledgeable participants is sufficient to produce meaningful results. Greenberg and Buxton [31] even warn that quantitative evaluations can mute innovative visions and lack meaningful critique, which is essential for an early academic prototype.

We evaluated our approach on the data from two separate microservice systems. The first is a mockup payroll accounting system implemented by Frank et al. [32]. They created a dataset of performance and reliability measurements during a series of chaos experiments as part of their work on scenario-based resilience evaluation and improvement of microservice architectures. The second subject system is SockShop [33], a demo application that realistically represents a real microservice system [34]. Avritzer et al. [35] provided us with a dataset of performance measurements collected from the system while investigating the impact of security attacks on the performance of different architecture deployment configurations. Both datasets contain measurements during normal execution of the respective system and transient behavior caused by either chaos experiments or security attacks.

B. Participants

Five experts participated in the study. They were selected because they were deeply familiar with one of the subject systems and the concept of transient behavior. Three of the participants hold a PhD in computer science and actively conduct research in the field of performance and reliability engineering. The two remaining participants were master students that have previously researched related topics in resilience engineering. Also, one of the participants was one of the interviewees in the expert interviews on transient behavior described in our previous work [16] and works as a software architect at a large software consulting company.

C. Procedure

We conducted the expert study remotely. To this end, we hosted the prototype online. Participants shared their screen that was also recorded while interacting with the prototype to solve a collection of tasks.

We prepared two tasks for each subject system. Each task consisted of a short scenario and involved creating one or two transient behavior specifications using the prototype and the subsequent analysis of the system's compliance with the specifications. To this end, the scenarios contained the following information: (i) the concerned microservice, (ii) the cause, (iii) the acceptable duration, (iv) and the acceptable QoS degradation of the transient behavior to be specified. In the first task, the specifications are violated by multiple service endpoints, whereas they are all met in the second one.

Before the tasks, we introduced the participants to the topic and the prototype. Afterward, we gave each participant ten minutes to get acquainted with the prototype and resolve all open questions. Once all open questions were resolved, the participants started to solve two tasks with the help of the prototype. They had ten minutes per task and worked on the tasks that regarded the system that they were already familiar with previously. After the time was up, we revealed the solution to the task to them. At the end of each session, the participants had to answer a short questionnaire and report their experience with the prototype.

Large parts of the questionnaire consisted of the System Usability Scale from Brooke [36] and the NASA Task Load

Index [37]. It contained twelve statements about the prototype's usability that had to be rated on a scale from one (strongly disagree) to five (strongly agree). Furthermore, the questionnaire contained four questions designed to assess the participant's cognitive load during the tasks that were also answered on a scale from one to five. Last, we asked four open questions regarding the most and least effective features of the prototype, the helpfulness of the chatbot, and how the participants would have solved the tasks without the prototype. We used the answers to the questionnaire for further discussion with the participants about the prototype.

V. Results

In this section, we present and discuss the results of our study regarding the research questions.

A. Specifying Transient Behavior

The participants successfully created the specifications for all scenarios, which indicates that the approach supports software architects and DevOps engineers in defining and specifying transient behavior. However, the study revealed the limitations of our resilience triangle visualization approach for transient behavior specifications. Two participants commented that they would like to see the concrete values of a specification's parameters directly in the visualization. Such an addition would make it easier to understand a specification and resolve the issue that the parameters of a specification can only be checked through the chatbot. Furthermore, one participant stated that the specification of transient behavior through the three dimensions initial loss, time to recovery, and loss of resilience is not intuitive and has to be learned before it can be applied. Nonetheless, the participants quickly familiarized themselves with the concept. One participant commented that it was not clear to him which cause he should describe in the specification of transient behavior. Another addition that would improve the prototype is an overview of all specifications that have been created. In its current version, such an overview is missing, and only one specification can be examined at a time. Finally, the elicitation of requirements, which our prototype does not support, appears to be the biggest challenge regarding the specification of transient behavior.

B. Analyzing Transient Behavior

Almost all participants were able to correctly identify whether or not the system complied with the specifications. Only in one case, transient behavior was assessed wrongly. In another case, a participant was unsure whether a specification is only fulfilled if all three transient behavior metrics are within the acceptable range.

Plotting the specification on top of significant transient behavior was perceived as helpful for verifying its compliance. The participants used both the transient behavior and loss of resilience visualizations to verify whether a specification was met. They mostly spotted behavior that potentially constitutes a violation in the latter, as such behavior is easily visible by the loss of resilience exceeding the specification. In the next step, the participants examined the transient behavior visualization to compare the extent of the quality degradation and the duration of the transient behavior with the resilience triangle representing the specification. However, it was not always clear to the participants when transient behavior starts and ends. Whereas the initial loss happens instantly in the resilience triangle, the actual QoS can decrease gradually over time, which is exacerbated by a low sampling frequency of the measurements. One participant expressed that it was not clear whether the transient behavior started at the beginning of the initial loss when the degradation exceeded the specified initial loss or at the lowest point of the degradation. Similarly, a service's recovery from disruption is often not linear, as the resilience triangle visualizes it, complicating the comparison of transient behavior with the specification.

Another point that two of the participants mentioned was that they would prefer to specify the quality metric used for the QoS function. They argued that the ambiguity of the abstract QoS function, unknown to the user, made the scenarios more difficult to understand. Moreover, they would like to switch between different metrics to investigate different quality aspects of a system.

The feedback from the participants indicates that the approach is suited for exploration and communication of transient behavior and the analysis of individual specifications. However, engineers missed features to filter or query the data for specification violations. Also, the reporting of violations is essential in production, which the prototype does not support. Future work should provide a way to get a quick overview of transient behavior that exceeds the specifications without manually searching for it.

Finally, all participants stated that they are not aware of any other publicly available system that facilitates the analysis of transient behavior specifications.

C. Usability

To evaluate the usability of our prototype, we asked the participants to rate 15 usability-related statements on a Likert scale from one (*strongly disagree*) to five (*strongly agree*). Figure 8 presents the results of the questionnaire. The results indicate that the participants strongly agreed that the tool was easy to use. Furthermore, four participants strongly agreed that most people would learn to use the tool very quickly. One participant disagreed with this statement and explained that the tool is easy to learn, but the theory of transient behavior is complex, and it is time-consuming to understand the concepts behind the term. Additionally, he claimed that the representation of specifications as resilience triangles is difficult to understand at first. However, the participants agreed that the included visualizations were helpful and supported the users in their tasks. Also, the results suggest that the tasks were not mentally demanding for the participants. In fact, they felt confident using the tool. These findings indicate a high usability of the TransVis approach.

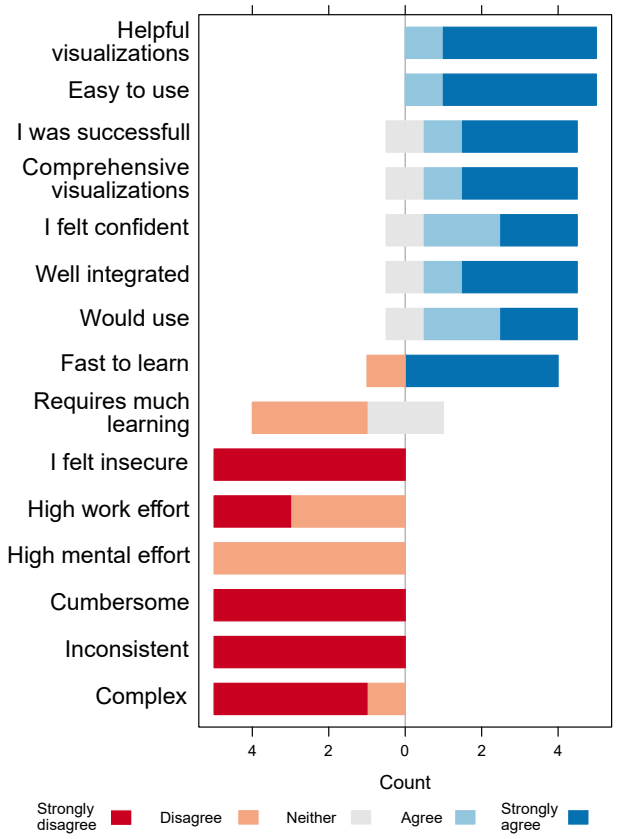

Fig. 8: Results of the questionnaire regarding the usability of the prototype.

D. Helpfulness of Chatbot

All participants but one did not use the chatbot to solve the tasks. They considered that the chatbot interactions required too much time to be a viable alternative to the direct interaction with the visualizations. Furthermore, participants felt that the chatbot did not provide them considerable value beyond controlling the visualizations and creating specifications. Both tasks could be completed faster without using the chatbot. To be helpful, the chatbot would require to provide other distinctive features beyond the functionality provided by the dashboard. The participants attempted to use the chatbot to filter the presented data or directly search for specification violations. If it were to be extended with such features, the chatbot might become more helpful for analyzing transient behavior. Another reason to explain the little value perceived by participants can be the manageable size of the two subject systems and the simplicity of the given tasks. For more complex tasks in larger systems, the chatbot might be a faster method for finding services than the visualization due to the limited screen space.

The participants also complained that it was not evident which input the chatbot understands. This led to trial-and-error approaches when trying to understand the chatbot's features. Additionally, the chatbot occasionally can classify input to the wrong intent or fail to recognize entities.

E. Threats to Validity

In this section, we discuss potential threats to the validity of our results.

Regarding the applicability of the results to the real world, the specified tasks were relatively simple and well defined. This is often not the case in actual software projects, where non-functional requirements are often not sufficiently specified.

Furthermore, the used datasets were created during load tests and chaos experiments and not collected during production. For this reason, the data used in our study might not be representative of the real performance data of a microservice system. Also, the recorded scenarios only contain data of the system's run-time for several minutes. A real-world system's production performance monitoring data is far more extensive, which could substantially increase the difficulty of analyzing it. Therefore, the given tasks might not be representative of problems that are faced in the real world.

There is a possibility that the experts invited to the study are not a representative sample of the population of software architects and DevOps Engineers. That is, they could be more or less skilled than the average software architect or DevOps engineer. Also, their willingness to participate in a user study indicates their high motivation, which could have influenced their performance in the tasks.

Another potential threat to validity is the method of the expert study. The questionnaire might not be extensive enough to capture all aspects of the participants' experience. Many insights were also derived from comments that the participants made about the prototype, which represent their subjective opinion. Moreover, the participants might not have discussed all aspects of the prototype and the task thoroughly.

VI. Related Work

We now analyze related works in two main domains: visualization of software resilience and the application of HCI methods in software engineering.

A. Visualization of Resilience

Work on resilience visualization originates mainly from research on disaster and system resilience. Different solutions for comparing resilience scenarios, as adopted in the TransVis approach, have been proposed.

Zobel [15] extends the resilience triangle from Bruneau et al. [14] with a new approach for visualizing the relationship between the predicted initial loss that a facility suffers during a disaster and the predicted time it needs for recovery. His proposed visualization intends to support decision-makers in disaster planning in comparing the resilience of different critical facilities. We adopted this approach to approximate the loss of resilience by calculating the area of the triangle formed by the initial drop in the QoS and the time needed for a full recovery. To differentiate between scenarios in which different combinations of initial loss and time to recovery result in

the same loss of resilience, Zobel introduces the predicted resilience concept, which can be visualized as a hyperbola. Points on such a hyperbola represent the possible combinations of initial loss and time to recovery that result in the given predicted resilience. However, Zobel does not apply his model and visualization to transient behavior in software systems.

Dessavre et al. [38] propose an extended model for system resilience that introduces stress as an additional dimension of disruptive events. They argue that their model enables comparing the impact of different disruptions and the resilience of different systems. To this end, they define a multidimensional resilience function that incorporates time, stress, and quality of service. Furthermore, they employ a heatmap visualization created by mapping the values of the function to a color scale. Multiple heatmaps can be combined in a single visualization to compare system resilience during different disruptions. By subtracting the resilience values of a dependent system from the original system, dependencies between the systems can be visualized. Adopting such an approach could be interesting for analyzing the impact of transient behavior in a microservice on the rest of the system.

Yin et al. [9] propose the Microservice Resilience Measurement Model (MRMM) as a quantitative metric to measure resilience in microservice systems. Additionally, they build upon this work to introduce a model for representing resilience requirements that consist of (i) a resilience goal, (ii) disruptions, and (iii) resilience mechanisms that mitigate the impact of the disruptions. While their work enables elicitation and modeling of resilience requirements, they highlight that generating such requirements for complex systems is challenging. Moreover, they do not leverage HCI methods to increase the accessibility of their approach.

B. HCI in Software Engineering

Much research has been done on the application of HCI methods in software engineering.

Fittkau et al. [39] propose ExplorViz, which uses a city metaphor to create 3D software visualizations of execution traces and software architectures. The originally web-based tool was extended by a virtual reality component that allowed the exploration of software visualizations in virtual reality through a head-mounted display and gesture-based interaction.

Merino et al. [40] investigated whether the application of immersive augmented reality to 3D software visualizations solved usability issues and increased users' effectiveness. They found out that immersive augmented reality facilitated navigation in 3D visualizations and enhanced the user experience.

Seipel et al. [41] combined software visualization in augmented reality with a conversational interface that can understand spoken natural language. Their work allows users to explore 3D software visualizations through gestures and speech simultaneously.

Okanović et al. [26] propose PerformoBot, a chatbot for configuring and running load tests. PerformoBot guides users in (i) creating load tests through natural language conversations, (ii) automatically running the tests, and (iii) answers respective performance concerns in a generated report. In evaluating their approach through a user study, they found that PerformoBot had a high acceptance rate, especially among novice developers. However, they did not couple the chatbot with interactive performance visualizations.

Bieliauskas and Schreiber [20] propose interaction with software visualizations through a conversational interface. Their approach was able to support group conversations with relevant visualizations. Furthermore, the conversational interface made the visualizations more accessible to new team members. However, their approach focused on visualizations for relations between software components.

While there is existing work that investigates transient analysis or the application of HCI methods in software engineering, to the best of our knowledge, TransVis constitutes the first attempt to apply HCI methods to the transient analysis of microservice systems.

VII. Conclusion

We presented the TransVis approach that employs resilience visualizations and a chatbot for the specification and analysis of transient behavior in microservice systems. The findings of our exploratory expert study suggest that the developed visualizations are effective, especially for exploring transient behavior and comparing a service's QoS with a specified behavior. However, we also found that a chatbot was not beneficial because the direct interaction with the visualizations is a more intuitive and faster method. The study also revealed many opportunities for future work. The prototype could be improved by adding an overview of the specifications and filtering and querying functionalities for critical behavior. Such features could be implemented in the chatbot to increase its effectiveness. Also, the prototype is limited to static datasets. The integration of live performance measurements is essential to make the approach viable for production use and automatic reporting of critical behavior. Furthermore, the chatbot could be improved by extending its functionality to allow answering questions concerning the data. Additionally, we did not explore the application of other human-computer interfaces such as virtual or augmented reality or multi-touch tables to our use cases. An approach employing such interfaces could build upon the developed visualizations. Finally, we see a need for solutions that facilitate the specification and the elicitation of requirements for transient behavior.

Acknowledgment

We thank the study participants. Van Hoorn acknowledges funding by the Baden-Württemberg Stiftung (Orcas project). Frank and Hakamian acknowledge funding by the German Federal Ministry of Education and Research (Software Campus 2.0 — Microproject: DiSpel).

References

[1] M. Krigsman. (2010) Cloud-based IT failure halts Virgin flights. [Online]. Available: https://www.zdnet.com/article/cloud-based-it-failure-halts-virgin-flights/

[2] R. Thomson. (2008) British Airways reveals what went wrong with terminal 5. [Online]. Available: https://www.computerweekly.com/news/2240086013/British-Airways-reveals-what-went-wrong-with-Terminal-5

[3] S. Wolfe. (2018) Amazon's one hour of downtime on prime day may have cost it up to $100 million in lost sales. [Online]. Available: https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

[4] L. Bass, I. Weber, and L. Zhu, *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[5] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[6] C. Czepa and U. Zdun, "How understandable are pattern-based behavioral constraints for novice software designers?" *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 2, pp. 1–38, 2019.

[7] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "CityVR: Gameful software visualization," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 633–637.

[8] C. Toxtli, A. Monroy-Hernández, and J. Cranshaw, "Understanding chatbot-mediated task management," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–6.

[9] K. Yin, Q. Du, W. Wang, J. Qiu, and J. Xu, "On representing and eliciting resilience requirements of microservice architecture systems," *arXiv preprint arXiv:1909.13096*, 2019.

[10] S. Beck, S. Frank, M. A. Hakamian, L. Merino, and A. van Hoorn. (2021) TransVis: Using visualizations and chatbots for supporting transient behavior in microservice systems - supplementary material. [Online]. Available: https://doi.org/10.5281/zenodo.4728953

[11] C.-Y. Wang, D. Logothetis, K. S. Trivedi, and I. Viniotis, "Transient behavior of ATM networks under overloads," in *Proceedings of IEEE INFOCOM'96. Conference on Computer Communications*, vol. 3. IEEE, 1996, pp. 978–985.

[12] W. J. Stewart, *Probability, Markov chains, queues, and simulation*. Princeton university press, 2009.

[13] V. A. Venikov, *Transient Phenomena in Electrical Power Systems: International Series of Monographs on Electronics and Instrumentation, Vol. 24*. Elsevier, 2014, vol. 24.

[14] M. Bruneau, S. E. Chang, R. T. Eguchi, G. C. Lee, T. D. O'Rourke, A. M. Reinhorn, M. Shinozuka, K. Tierney, W. A. Wallace, and D. Von Winterfeldt, "A framework to quantitatively assess and enhance the seismic resilience of communities," *Earthquake spectra*, vol. 19, no. 4, pp. 733–752, 2003.

[15] C. W. Zobel, "Comparative visualization of predicted disaster resilience," in *Proceedings of the 7th International ISCRAM Conference*. ISCRAM, 2010, pp. 1–5.

[16] S. Beck, "Evaluating human-computer interfaces for specification and comprehension of transient behavior in microservice-based software systems," Master's thesis, University of Stuttgart, 2020, relevant chapters are part of the supplementary material [10].

[17] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings 1996 IEEE symposium on visual languages*. IEEE, 1996, pp. 336–343.

[18] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, NJ, 1996, vol. 2.

[19] B. Saket, P. Simonetto, S. Kobourov, and K. Börner, "Node, node-link, and node-link-group diagrams: An evaluation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2231–2240, 2014.

[20] S. Bieliauskas and A. Schreiber, "A conversational user interface for software visualization," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2017, pp. 139–143.

[21] D. Wigdor and D. Wixon, *Brave NUI world: designing natural user interfaces for touch and gesture*. Elsevier, 2011.

[22] M. Turk, "Multimodal interaction: A review," *Pattern Recognition Letters*, vol. 36, pp. 189–195, 2014.

[23] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 928–931.

[24] E. Paikari, J. Choi, S. Kim, S. Baek, M. Kim, S. Lee, C. Han, Y. Kim, K. Ahn, C. Cheong *et al.*, "A chatbot for conflict detection and resolution," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 2019, pp. 29–33.

[25] L. Goasduff. (2019) Chatbots will appeal to modern workers. [Online]. Available: https://www.gartner.com/smarterwithgartner/chatbots-will-appeal-to-modern-workers/

[26] D. Okanović, S. Beck, L. Merz, C. Zorn, L. Merino, A. van Hoorn, and F. Beck, "Can a chatbot support software engineers with load testing? approach and experiences," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 2020, pp. 120–129.

[27] A. Abdellatif, K. Badran, and E. Shihab, "A repository of research articles on software bots," http://papers.botse.org.

[28] Dialogflow concepts. [Online]. Available: https://cloud.google.com/dialogflow/docs/concepts

[29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[30] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, and T. Möller, "A systematic review on the practice of evaluating visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2818–2827, 2013.

[31] S. Greenberg and B. Buxton, "Usability evaluation considered harmful (some of the time)," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 111–120.

[32] S. Frank, A. Hakamian, L. Wagner, D. Kesim, J. von Kistowski, and A. van Hoorn, "Scenario-based resilience evaluation and improvement of microservice architectures: An experience report," in *Companion of the 15th European Conference on Software Architecture (ECSA 2021)*, 2021.

[33] Weaveworks and Container Solutions. (2017) Sockshop: A microservices demo application. [Online]. Available: https://microservices-demo.github.io/

[34] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. IEEE, 2017, pp. 8–13.

[35] A. Avritzer, V. Ferme, A. Janes, B. Russo, A. van Hoorn, H. Schulz, D. Menasché, and V. Rufino, "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests," *Journal of Systems and Software*, p. 110564, 2020.

[36] J. Brooke, "SUS: A quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[37] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (task load index): Results of empirical and theoretical research," in *Advances in psychology*. Elsevier, 1988, vol. 52, pp. 139–183.

[38] D. G. Dessavre, J. E. Ramirez-Marquez, and K. Barker, "Multidimensional approach to complex system resilience analysis," *Reliability Engineering & System Safety*, vol. 149, pp. 34–43, 2016.

[39] F. Fittkau, A. Krause, and W. Hasselbring, "Exploring software cities in virtual reality," in *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. IEEE, 2015, pp. 130–134.

[40] L. Merino, A. Bergel, and O. Nierstrasz, "Overcoming issues of 3D software visualization through immersive augmented reality," in *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2018, pp. 54–64.

[41] P. Seipel, A. Stock, S. Santhanam, A. Baranowski, N. Hochgeschwender, and A. Schreiber, "Speak to your software visualization—exploring component-based software architectures in augmented reality with a conversational interface," in *2019 Working Conference on Software Visualization (VISSOFT)*. IEEE, 2019, pp. 78–82.