A Controlled Experiment on the Usability of Automated Reflexion Mapping Suggestions Integrated in Code Cities

Leon Ehrhardt
University of Bremen, Germany
leeeon333@gmail.com

Rainer Koschke
University of Bremen, Germany
https://orcid.org/0000-0003-4094-3444

***Abstract*—Reflexion modeling allows developers to reconstruct and verify their software architecture against its implementation. One step of this method is the mapping of implementation components onto architecture components. Researchers have tried to automate the process of recommending suggestions where to map implementation components. This paper describes how such techniques can be integrated in an interactive approach that is based on software visualization using the code-city metaphor and evaluates the usability of this approach through a controlled experiment.**

During our user study, effects of reduced cognitive effort and time savings were measured when automated suggestions were available during reflexion modeling, although no statistical significance of the observed differences could be established.

***Index Terms*—reflexion modeling, architecture recovery, semi-automatic mapping, code city metaphor, usability.**

I. Introduction

Gail Murphy's seminal work on software reflexion models focuses on bridging the gap between high-level architectural models and the actual source code of software systems [1]. This approach helps software engineers compare their high-level design intentions with the implemented code, identifying discrepancies and areas of agreement. The process involves:

1) Architects create a *high-level model* representing their understanding of the system's architecture.
2) Architects *map source code* onto the high-level model to specify which elements of the source code implement which parts of the high-level model.
3) An automated analysis generates a *reflexion model* that highlights where the high-level model and the source code align and where they differ.

The first two steps can cause a lot of manuals work [2]. Because the high-level model is a concept in the mind of architects, there is little to automate. Yet, many researchers have attempted to semi-automate the mapping process [3]–[9]. Mapping suggestions can be derived from, for instance, naming conventions or a partial mapping. Current research for evaluating those automated mapping suggestions has mostly focused on comparing automated suggestions against an oracle mapping. How the automated mappings can be integrated in an interactive process and whether they are actually useful is less researched.

This paper complements this research by pondering on how automated mapping suggestions can be integrated into the code-city metaphor. A *code city* is a software visualization that depicts software as a kind of city in 3D [10]. Specifically, we integrate automated mapping suggestions in our collaborative visualization platform SEE (for Software Engineering Experience). We will report on a controlled experiment in which we evaluate the effort, time saving, and usability of this integration.

Section II will first summarize the relevant research on this subject. After that we describe in Section III the reflexion-modeling approach in SEE including the integration of automated mappings. The design of our controlled experiment will be described in Section IV and its results discussed in Section V. Section VI, finally, summarizes the conclusions.

II. Related Work

There are different approaches to architecture conformance checking. Because our approach is based on reflexion modeling, we will focus on this approach and refer the reader to existing overviews on alternative approaches [11], [12].

Murphy et al. introducted reflexion modeling in 1995 [1]. Although originally intended for reconstructing or verifying architectures, other researchers have used it for various related purposes. Buckley et al. used it for encapsulating user-targeted components, as a prelude to component recovery, reuse, and refactoring [13]. Ackermann et al. [14] adopted reflexion modeling to support the compliance checking of behaviors of systems of systems in terms of sequencing properties. We used it to extract state machines from code [15]. We also extended reflexion modeling in combination with clone detection to consolidate software variants into product lines [16]. Another use of reflexion modeling in the context of product lines was reported by Tekinerdogan et al. [17]. Çilden et al. [18] used it to check applications for OSGi compliance. Le Gear et al. [19] proposed a technique called *software reconnexion*—based on reflexion modeling—that uses a reuse perspective of software. which contains core elements of the subject system. Based on these, the user is prompted during the early iterations of the reflexion-modeling process. Reconnexion aims at reducing

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – KO 2342/8-1.

the technique's dependency upon documentation and domain knowledge. Herold et al. [20] have proposed a technique to detect typical causes of violations in reflexion models.

There were also a few algorithmic additions to the original reflexion analysis. The original method did not allow hierarchical architecture components. We provided a formalization of an extension that can be applied to architecture models with arbitrary many hierarchical levels [21]. Moreover, we described an incremental analysis to compute the reflexion model [22]. Our algorithm repeats the analysis only for those parts that are actually influenced by a change, which is particularly advantageous in interactive usage. SEE uses this incremental algorithm to provide what-if information in real-time while a user hovers a dragged source-code element on an architecture component. The what-if information shows how the reflexion model would change if a user mapped a given source-code element onto a particular architecture component. Bittencourt [23] has proposed three changes to improve integration of the technique into software development by (1) expressing architectural rules as design tests that may be checked in a testing framework, (2) by supporting semi-automated changes to the mapping between source code and the model when code changes occur, and (3) by deriving automated suggestions of high-level model changes based on improving software cohesion and coupling.

Romanelli et al. [24] presented visual support for reflexion modeling. The source-code elements are visualized as a 2D tree map. These elements can be mapped to architecture components using regular expressions as in the original approach by Murphy. Yet, the hits of those regular expressions are hightlighted in the tree map giving an overview on their location and hierarchy. The architecture and the resulting reflexion model are drawn as box-and-arrow diagrams as we do. The difference to our approach is that we are using a 3D visualization and the same kind of visualization for both the implementation and architecture in an integrated manner, allowing users to intuitively drag source-code elements onto architecture components. We express the mapping through spatial enclosing. This gives users an immediate overview what has already been mapped and where it has been mapped.

There are many papers on automating the process of recommending mappings for reflexion modeling [3]–[9]. These papers are described in greater detail in the other paper we submitted to this workshop [25]. Moreover, the approach used to generate mapping suggestions is not truly relevant for the way we integrated automated mappings into SEE. For these reasons, we refer the reader to the other paper. We will go into the details of only the approach used in our study.

SEE uses the *HugMe* method [3] as a general framework for how to come up with mapping recommendations based on a partial mapping. *HugMe* combines various clustering techniques to group related source code artifacts. Then it selects clusters based on a so called *attract* function that computes the attraction between a source-code element to be mapped and existing clusters. The candidate set of all mapping suggestions is then formed by all clusters whose attraction is greater than the mean attraction over all clusters plus its standard deviation. *HugMe* itself does not dictate what *attract* function to use. Because we found in our evaluation in another of ours [25] that *ADC-Attract* has the best performance for *PetClinic*. *ADC-Attract* calculates the attraction of a code entity to be mapped for each architecture component based on its coupling—taking into account required coupling according to the architecture model—and the similarity of term frequencies (identifiers) in the source code. We will use *HugMe* in combination with *ADC-Attract*, but expect the findings of our study to hold for other approaches to recommend mappings, too, as long as these have similar performance.

There are many tools implementing reflexion modeling, for instance, the commercial Axivion Suite[1] and ConQAT[2] or the research tools *iCIA* [26], *JITTAC* [27], [28], or *SAVE* [11] (the latter two tools allow real-time incremental changes for what-if analyses). However, there are only a few tools—all of them research tools—which offer automated mappings interactively. For instance, a tool by Biehl and Löwe [29] generates mapping suggestions in the context of model-driven software development (MDSD) by deducing information from the implementation, design documents, and model transformations. Another tool by Kim et al. [26]—again in the context of MDSD—leverages the tracing information left in the generated code sceletons to establish the mapping for all unchanged generated code. The creation of mappings for manually changed code is based on an improved *Count Attract* function. Both tools are designed specifically for an MDSC context.

III. Reflexion Modeling Using Code Cities

SEE (for (Software Engineering Experience) is a versatile software-visualization platform that uses the "software-as-a-city" metaphor [10]. It enables users, such as software architects and developers, to collaborate from different locations, when they need to discuss aspects of their software at a higher level. The users of SEE interact in a shared virtual room, each one represented by an avatar, allowing them to see and communicate with each other via an integrated voice chat and body gestures. The virtual rooms of SEE can be entered from various hardware devices, including desktop computers, tablets, and virtual reality systems (VR).

The programs the users of SEE want to discuss are visualized as code cities in those virtual rooms residing on tables where users can group around (cf. Fig. 1). Anslow et al. [30] have explored this kind of setting for the physical world with a large multi-touch 2D display laid on a table around which participants in the same room can group. They offered different kinds of software visualizations (yet no code cities). In a study with 42 professional developers—working either as pairs or individually–they found that the multi-touch table encouraged the participants to work together and collaborate with each other [31]. SEE is a 3D virtualization of this setting, where code cities are the primary means of visualization. We note

[1] https://www.axivion.com
[2] https://en.wikipedia.org/wiki/ConQAT

that SEE offers also 2D scatter plots within the virtual scene to visualize additional metrics (a block in a city has only three dimensions and one color). In addition, SEE provides a shared whiteboard that can be used by participants of a session to create their own sketches about their software. Yet, metric scatterplots and whiteboard were not used in our study).

In SEE, a program's components are visualized in a code city as blocks for atomic components or areas for components consisting of other components, where an area is simply a block with zero height[3]. The three dimensions of an atomic block and its color are determined by code metrics that can be selected by a user. The size of an area, on the other hand, is determined automatically such that all its nested components fit into the area. Thus, the hierarchy of the program is depicted through spatial enclosing. The layout of a code city is computed by automated hierarchical layouts. SEE offers tree maps, circle and rectangle packing, and a balloon layout.

Dependencies among the program's components are depicted by edges. To reduce visual clutter, edges are hierarchically bundled, as proposed by Holten [32], and the direction of the edges is indicated through a color gradient rather than an arrow head. In addition, SEE offers an option to show edges only on demand when a block is hovered over by the user.

The code cities are dynamic, enabling users to highlight and move and modify parts of the visualized code cities in real-time, visible to all participants. Because the code city and the avatars of the users are in the same scene, a gesture of a user can be traced meaningfully. For instance, if a user points to a block in the code city, a laser beam will be drawn from the avatar's hand to the pointed block that can be seen by all participants. This distinguishes SEE from general video conference systems, such as Zoom, Teams, etc., where the video tiles of the particpants are totally disconnected from the shared screen. Moreover, all users can walk freely through the virtual room and view the code city from any angle independently of all other participants. Every user can manipulate the code city, for instance, by moving blocks to group components semantically—which is neither possible in video conference systems where only one person can interact with the shared content. SEE makes sure that the effect of such manipulations will be propagated to all connected client computers such that all participants always have the same consistent view of the world.

A key use case for SEE is supporting the reflexion modeling [1]. To do that, the components of the implementation—extracted through a static analysis—need to be mapped onto architecture components—manually modeled by an architect. After that, an algorithm can compute the convergences, divergences, and absences between implementation and architecture. A *convergence* is an architecture dependency that has at least one corresponding implementation dependency. Any such corresponding implementation dependency is said to be *allowed*. A *divergence*, on the other hand, is an implementation dependency that is not allowed according to the architecture, that is, there is no convergent architecture dependency. An *absence* is an architecture dependency that has no corresponding implementation dependency. Here, an implementation dependency is missing. This paper will focus on this use case and its implementation within SEE with the new addition of automated mapping suggestions.

The same kind of visualization as a code city used for the implementation can be used for the architecture model, too. Typically, however, users would model the architecture manually in SEE and, thus, create the layout themselves, although architectures can also be imported and automatically laid out by SEE if needed. Intuitively, if architects model the architecture manually, they will group elements semantically as predicted by the laws of Gestalt [33], while automated layouts make their decisions based only on non-semantic constraints such as spatial fit. In most cases, architecture components do not have any metrics associated (although they could in SEE), such that their height is zero and their width and depth are determined by architects, again allowing them to underlay these dimensions with their own meaning, for instance, to draw more prominent components larger or to use similar dimensions for semantically similar components. For the expected dependencies among architecture components, again edges can be drawn. Because the code city of an architecture is generally much smaller than a code city for an implementation, users can most often afford to show all edges permanently.

For the use case of reflexion modeling, the two code cities for the implementation and architecture are put on a table side by side. The components of an implementation (both blocks or areas) can be dragged from the implementation city onto architecture components. While dragging an implementation component over an architecture component, an incremental reflexion analysis [22] calculates the changes of edges as if the dragged component were mapped onto the hovered architecture component and those changes are immediately shown. This allows a user to quickly run a what-if analysis: "What would happen if I mapped this component here?" The changed edges receive a shiny effect such that they can be distinguished from previously existing, unchanged edges. The mapping is finalized when the dragged component is dropped.

The suggested mappings are visualized whenever a user picks up an implementation component. Then the suggested architecture areas it can be mapped to will be highlighted through a shiny effect. Currently, the highlighting is the same for all candidates independent from their attraction.

If it is necessary for users to see the source code when deciding where to map an implementation component, they can open a code window for a selected component. The code window uses syntax highlighting for better readability as well as scrollbars if the complete source code does not fit into the window. The window can be dragged freely in the scene and on demand be shared with other participants connected via different SEE clients. In addition, SEE offers a classic tree view (as offered by IDEs) to show the components of a city and their nesting hierarchy. This tree view also allows a user

[3]https://youtu.be/jQlBIcRubZw

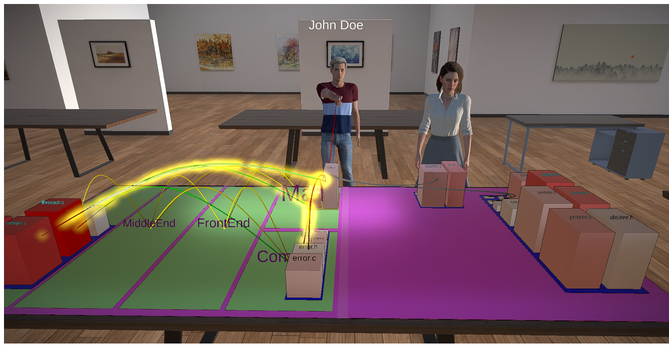

Fig. 1. Example scene for reflexion modeling

to search for the names of components using fuzzy search.

An example of a virtual room for reflexion modeling is shown in Figure 1. There are three participants present in the room, where the screenshot is taken from the first-person perspective of one of the avatars. The scene shows a partial mapping for a compiler architecture and its implementation. The architecture is shown on the left, where green areas represent architecture components. The implementation components are either on the right at their original location if not yet mapped or on the left if they are already mapped. The architecture dependencies, in this example, are either convergences drawn as green edges or absences drawn as yellow edges. The edges added as a result of the most recent mapping step are highlighted by a shiny effect.

The implementation of SEE is based on the game engine Unity and developed in C#. Its source code is published under the MIT license and publicly available on GitHub[4].

IV. Design of the Usability Study

We hypothesize that automated suggestions make it easier for the user to establish the mapping in reflexion modeling as integrated in SEE. Hence, the aim of our study is to investigate the usability of our approach, where we adopt the more comprehensive notion of usability by the standard ISO/IEC 9241: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."

The effectiveness of the automated mappings in terms of recall and precision are evaluated in great detail in the other paper we submitted to this workshop [25]. That is why this study focuses on effort (how much work is needed by the user to conduct the mapping with and without suggestions), efficiency (how much time can be saved), and satisfaction (how pleased is a user with our implementation of reflexion modeling). The concrete measures for these aspects follow in Section IV-A.

The "product"—as mentioned by ISO/IEC 9241's definition—whose usability is to be investigated is our implementation of reflexion modeling using code cities, enriched by automated suggestions. The "users" will be developers and software architects; further details on the experimental subjects representing these users follow in Section IV-C. The "goal" of developers and architects using reflexion modeling in general is to verify or reconstruct a static software architecture that truly describes a given implementation including all deviations. In our study, the subjects are given the more specific goal to extend a partial mapping as one step in the overall process of reflexion modeling, as this is the subject of our research. The "context of use" manifests in the concrete tasks for the subjects of our study along with the project for which to apply reflexion modeling. The concrete goal and context of use within our study will be described in Section IV-B.

The overall usability in the sense of ISO/IEC 9241 in general is always determined by a combination of both functional aspects—will a software provide the expected output given some input—and aspects of interaction and representation—how can users achieve their goals and how comprehensible is the output? The functional aspects in our case include the automated suggestions and the incremental calculation of reflexion results based on a partial mapping. Representational aspects include the visualization of implementation and architecture as code cities, using spatial enclosing for the representation of both the component hierarchy and the mapping, the representation of actual and specified dependencies as colored edges (where actual dependencies are shown only on demand), the highlighting of changed dependencies and mapping candidates through a glow effect, the immediate feedback on what the results would be if a grabbed block were put on an architecture area and more. The interactions offered by our approach are the hovering over blocks of interest to show their dependencies and allowing a user to drag blocks representing implementation components onto architecture components by way of direct manipulation.

As mentioned in Section II, there are other tools implementing reflexion modeling. Although functionally similar, most of these do not offer automated mapping suggestions interactively, which is the focus of our study. There are only a few tools, for instance, one by Biehl and Löwe [29] and one by Kim et al. [26], which offer automated mappings interactively. Both of these tools, however, are specifically designed for model-driven software development (MDSD), where they can leverage information that is only available in an MDSC context. Our tool aims at a broader use outside of MDSC. Moreover, the interactions and representations of tools offering automated mappings differ very much from our approach. If we compared our approach to these tools, any observed difference could result not only from the availability of automated mappings but also from interactive or representational differences among the tools. Yet, our primary interest is whether the automated mappings help. That is why we decided not to compare our tool to other tools also offering automated mappings. Instead, to isolate the contribution of automated mapping suggestions in our controlled experiment, we will compare our tool once with and once without mapping suggestions, where all other aspects are kept alike.

[4] https://github.com/uni-bremen-agst/SEE

The Master Thesis by Erhardt provides more details on the study including a detailed task description [34]. The collected data and evaluation scripts in R are available, too[5].

A. Dependent variables

In this section, we will first go into the details of the measured dependent variables regarding effort, efficiency, and usability.

Effort: One measure for the perceived effort in fulfilling tasks is the NASA Task Load Index (NASA-TLX) score [35], which is a well established index in human-computer interaction [36] even though not originally designed for software. This index is implemented by way of a questionnaire consisting of six questions asking a subject for the demands imposed on the subject (mental, physical, and temporal demands) and for the interaction of the subject with the task (perceived effort, performance, and frustration)[6]. Each question is answered on an ordinal scale with 21 gradations from *very low* to *very high* [35]. Scale ratings are scored based on where the user marked the scale. Tick marks range from 0 to 100 by 5 point increments. Lower values are better. The values for the six different aspects are combined into a single value as weighted average of the aspects. The weights are determined by showing each subject all 15 pairwise comparisons of the six dimensions and asking them to select which better represents their experience of demand. The number of times a dimension was selected determines its final weight.

A simplified variant—waiving the cross-validation of the individual subscales—is the Raw NASA-TLX (Raw-TLX) index. It uses the identical questions. The only difference is that the six ratings are simply averaged or added to create an estimate of the overall workload. Various studies found a high correlation of the subscales [37] of the questionnaire. Moreover, Hart could not find an advantage of the original TLX over Raw-TLX (neither vice versa) regarding the overall rating outcome in a meta analysis of 29 studies in which Raw-TLX was compared to the original version [38].

The Raw-TLX has the advantage that it puts less burden on the subjects by not forcing a subject to make all 15 pairwise comparisons of the six dimensions. That is why we used Raw-TLX in our study. We will report the results for each of the six questions individually and the total Raw-TLX as the unweighted average of the individual ratings.

We made one more simplification by reducing the 21 gradations of the original NASA-TLX to only 10 to further simplify the decision making of the respondents. Yet, the selections of the users on this scale are still scaled to the original value range of 0–100.

In our study, we distinguish between the conditions with suggestions (S) and without suggestions (NS). Hence, our null hypothesis $H_0(TLX)$ regarding effort that our study attempts to falsify is as follows: The effort during reflexion modeling is the same or higher with suggestions than without suggestions: $\textit{Raw-TLX}_S \geq \textit{Raw-TLX}_{NS}$.

Efficiency: To capture efficiency, we measure the duration of reflexion modeling for a user. We call this duration t_S for the condition with suggestions and t_{NS} for the condition without suggestions. Consequently, our null hypothesis, $H_0(D)$, to be falsified can be stated as follows: The duration of reflexion modeling with suggestions is greater than or equal to the duration without suggestions: $t_S \geq t_{NS}$.

Usability: The perceived usability is measured by the *System Usability Scale (SUS)* [39]. SUS is based on a questionnaire with ten different questions answered on a five-point Likert scale from strongly disagree to strongly agree.

The exact questions are listed in Table III, although we need to note that we used a German translation [40] because our participants were Germans. The SUS questions are alternated between positively and negatively worded, which is recommended by the body of research on questionnaire design to avoid bias. The weight of each positively worded answer is the position on the five-point Likert scale minus 1 and then multiplied by 2.5. To be able to quantitatively compare positively and negatively worded questions, the weight of the latter is 5 minus the scale position and then multiplied by 2.5. That is, the weights of all positively and negatively worded questions is in the range from 0 to 10. The ten individual weights are summed up to the total SUS value. The factor of 2.5 for the weights is used to obtain a maximal total SUS value of 100, which allows a direct interpretation as a percentage. The minimal value is of course 0. The higher the value, the better is the perceived usability.

Unlike for effort and efficiency, we do not compare the two conditions with and without suggestions for SUS. The interaction for a user to conduct reflexion modeling is almost identical; the only difference is that the suggested mapping candidate is visually highlighted when suggestions are available. This little detail is hardly worth to be evaluated separately. Instead we were interested in the perceived usability for our reflexion modeling as a whole. Section V-C will report the SUS for only one condition where suggestions are available.

There are many alternative post-study usability questionnaires, such as PSSUQ [41], CSUQ [42], TAM [43], or UEQ [44]. We opted for SUS because it is a widely used and well evaluated questionnaire [36], the number of questions asked is still feasible for respondents not to fatigue, it is free to use (no license fee) [36], it is applicable to small sample sizes [45], it is technology agnostic [36] and there is a recommended German translation [40], [46].

In addition to the standardized SUS questions, the subjects were asked for free-form qualitative feedback on the approach.

B. Tasks

Our study aims at the evaluation of the automated suggestions for mappings in the course of reflexion modeling. Thus, a subject in our study was confronted with a software system in which it is necessary to assign *orphaned* (i.e., not yet mapped) modules to architecture layers. An architecture

[5] https://doi.org/10.5281/zenodo.14718505

[6] The exact questions can be found here: https://humansystems.arc.nasa.gov/groups/TLX/downloads/TLXScale.pdf.

model and a visualization of the implementation exist already. Both are represented as code cities side by side as explained in Section III. Because the automated mapping suggestions assume a partial mapping, some of the implementation modules are already mapped onto architecture components. The tasks require a subject to assign orphaned modules onto the architecture components.

Because we want to evaluate the added value of automated mappings versus no suggestions, our experimental design has one factor with two levels resulting in two treatments: (S) with automated suggestions and (*NS*) without automated suggestions (suggestions were turned off).

In order to maximize the data gain per subject and be able to make an intra-personal comparison, every subject received both treatments in our study. Thus, we had to create two tasks for each subject. Task T_S represents the treatment S, and task T_{NS} the treatment *NS*. All other aspects of T_S and T_{NS} needed to be as equals as possible.

As subject received two treatments, they were randomly assigned to one of two groups in order to balance carry-over and learning effects. One group started with task T_S and the other group with task T_{NS} to alternate the order in which suggestions are available.

In both tasks, the subjects were to conduct the mapping as part of reflexion modeling for a real system. To keep T_S and T_{NS} as comparable as possible, we used the same system in both tasks, however, different orphaned modules were to be mapped onto different architecture components.

A system suitable for our study should be large enough; otherwise automated suggestions are not needed. Yet, a suitable system can only be so large that a subject without prior knowledge is still capable to decide where to map modules. Otherwise, the mappings could possibly be arbitrary due to a lack of understanding. Equally, the structure of the system must not be trivial. Otherwise, the user would be underchallenged with the task, which would mean that no benefits could be expected from the automated suggestions. That means, there must be sufficiently many classes and dependencies in the system. This requirement is also necessary simply to be able to calculate an initial mapping. Without initial mappings, no automated suggestions can be made by *HugMe*. In addition, the subjects should be familiar with the programming language used, if they needed to consult the source code to make their mapping decisions. To increase the pool of subjects we can draw from, the programming language should be popular. The same holds true for the application domain of a suitable system.

For alle these reasons, we selected the spring framework fork of the Spring sample project *PetClinic*[7]. Written in Java, it illustrates the main concepts of developing a web application based on the Spring framework. *PetClinic* consists of 47 Java files. An informal box diagram exists that sketches its four-tear layered architecture[8], consisting of controller classes, which

[7]https://github.com/spring-petclinic/spring-framework-petclinic

[8]https://de.slideshare.net/slideshow/spring-framework-petclinic\-sample-application/71978076#4

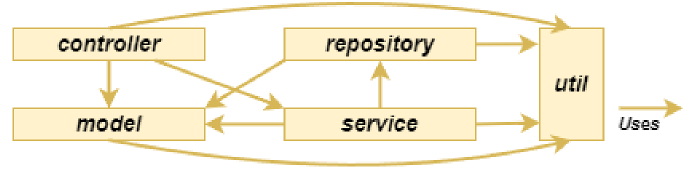

Fig. 2. Architecture of *PetClinic* as used in the study

are used from the view of the web page; service classes, which implement the application logic; and repository classes, which execute database accesses. The informal architecture diagram visualizes those layers as stabled boxes as it is common for layered architectures. The diagram does not have any arrows to describe explicit dependencies. Instead, we reverse engineered them from the code and the usual interpretation of layered architectures, where lower layers must not access higher layers. The 'Controller' layer uses the 'Service' layer and the 'Service' layer uses the 'Repository' layer. The implementation of *PetClient* has no component for the 'View' because *PetClient* is a server application and the view is rendered on the client side in a connected browser.

We added two more architectural components to the simple layered architecture sketched by the mentioned box diagram. All layers mentioned above use common data objects, which we summarized as a 'Model' component. In addition, general-purpose classes are assigned to the 'Util' component based on their namespace. The architecture we used in the study is shown in Figure 2 and its visualization in SEE can be seen in Figure 3. All blocks in the implementation had the same zero height and the same surface are to reduce the chances of distraction.

Based on the namespaces of the modules in the sample project, we constructed an oracle mapping to be able to make automated suggestions. In our study, the ADC-Attract function was used, as it achieved the best mapping according to the oracle. In the course of the study, it can therefore be assumed that the automated suggestions make sense.

The users had to map classes. To prevent further learning effects, the orphaned classes for the fist task were removed for the second task from the code city for the implementation, so that completely unknown classes were to be mapped in the subsequent tasks solved by the same user. A balanced selection was made so that the subjects in the study had to assign the same number of classes to the same architectural layers. The specific classes for the tasks are listed in Table I.

The depdencency graph of the implementation of *PetClinic* was gathered through a static analysis implemented by a commercial tool developed by Axivion. Third-party components used by the files of *PetClinic* such as Java libraries or Spring classes were removed from the dependency graph as they do not belong to *PetClinic* and because the architecture modeled only *PetClinic* and not its surroundings.

Before solving the tasks, all respondents were asked to provide brief demographic information and details of their technical skills. This information was collected in order to

Orphaned classes of the task	Layer
Orphaned classes of task A	
petclinic.web.PetTypeFormatter	controller
petclinic.repository.jdbc.JdbcVisitRowMapper	repository
petclinic.repository.jdbc.JdbcPetRowMapper	repository
petclinic.web.PetController	controller
petclinic.util.CallMonitoringAspect	util
petclinic.model.Pet	model
petclinic.model.Visit	model
Orphaned classes of task B	
petclinic.web.PetValidator	controller
petclinic.repository.jdbc.OneToManyResultSetExtractor	repository
petclinic.repository.jdbc.JdbcPetVisitExtractor	repository
petclinic.web.VetController	controller
petclinic.util.EntityUtils	util
petclinic.model.Vets	model
petclinic.model.NamedEntity	model

TABLE I
OVERVIEW ON ORPHANED MODULES

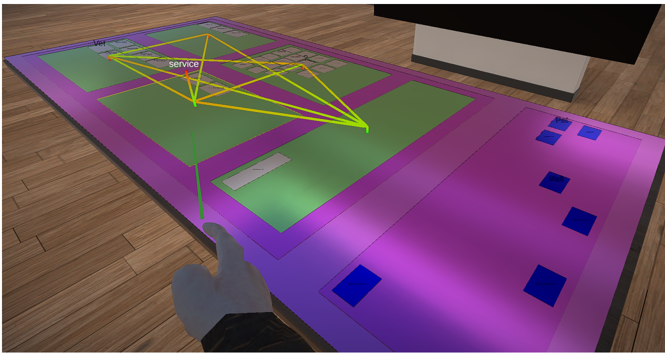

Fig. 3. Code city for the tasks

better categorize the sample of this study. The respondents were then told that they would be put in the role of a software architect. Subsequently, reflexion modeling was then explained in textual and verbal form. Before starting the task, the subjects had the opportunity to familiarize themselves with SEE and reflexion modeling on a second project. They could run the reflexion analysis and use the tree view and the code windows. In particular, they got to know the highlighting of the automated mapping suggestions. As soon as they felt confident with SEE, they were introduced to the architecture of *PetClinic*. A subject was instructed to assign the orphaned classes listed in Table I at their own discretion within 20 minutes, using the SEE features discussed. At the start of the tasks, the time was recorded via a KoboToolBox form. After completing each task, the subjects were asked to stop the time recording and answer the Raw-TLX questionnaire. After both tasks, they had to fill in the SUS questionnaire and were asked for written qualitative feedback.

C. Sampling and Demographics

Fourteen subjects volunteered for the study originally. Only software developers were considered, as a basic understanding of programming concepts is a prerequisite for completing the tasks. All subjects were either professional developers or computer-science students. We gained them through convenience sampling, that is, we invited professionals and students we knew personally.

One participant was very tired during the study, because the study was conducted late in the evening. In addition, many bugs occurred during his session (nodes fell under the table and could no longer be controlled). The problems were so extreme that we decided to remove the subject from the evaluation as a measurement error. This left us with $n = 13$ subjects.

The age of the subjects ranged between 24 and 36 years. Unfortunately, we did not succeed in including female developers. All subjects were familiar with Java development. Four of the subjects stated that they had experience with Spring. Five subjects had already used SEE before. Four subjects were themselves involved in the development of SEE, although not in the implementation of reflexion modeling.

V. RESULTS

A. Effort in terms of Raw-TLX

Table II shows the Raw-TLX. We report the mean values even though we are aware that the answers on the Likert scale are only ordinal data and only the medians would be valid according to measurement theory. The reason for that is that the authors of the NASA-TLX already suggest to compute averages and to sum them up to the total TLX index, which assumes an interval scale.

On all subscales except for the *Performance* subscale, an effect of reduced cognitive effort is observed in the group with suggestions (the lower the index, the better). The average total score of approximately 36 in the group with suggestions is around seven points below the value of the comparison group with approximately 43 scores. The standard deviations in both groups fluctuate by 18.5 scores.

To verify whether the observed differences are statistically significant, we applied a significance test. A t-test can be used only if normal distribution can be assumed or if the sample is large enough (more than 30 data points). Our sample is small and the Shapiro-Wilk test yielded a p value of 0.0091. This test calculates the p value for the null hypothesis that the data has normality. A p value greater than the significance level implies that the distribution of the data is not significantly different from normal distribution. With a p value of 0.0091 we cannot assume normality.

Instead, we use the paired asymptotic two-sample Fisher-Pitman permutation test. This test does not assume a particular distribution of the population and works for all ordinal data. A paired variant is to be used because our two samples are not independent since every subject contributed to data of both samples. In that case, the differences of the paired data are used as the rank. A tie may occur when the difference between a pair of observations is zero. This means that the two values being compared are equal, resulting in no difference to rank. In our sample with $n = 13$, this happened twice, that is, in about 15 % of the cases. Because of this relative high percentage of

Subscale	T_S	T_{NS}
Mental Demand	35.04 (±26.78)	50.43 (±29.26)
Physical Demand	25.64 (±29.88)	34.19 (±34.39)
Temporal Demand	14.53 (±18.36)	22.22 (±27.59)
Performance	72.65 (±29.61)	62.39 (±18.45)
Effort	31.62 (±22.61)	44.44 (±26.45)
Frustration	35.04 (±26.78)	44.44 (±25.26)
Total Raw-TLX	35.75 (±18.42)	43.02 (±18.74)

TABLE II
AVERAGED RESULTS OF THE NASA-TLX QUESTIONNAIRE PER TASK (THE VALUES IN BRACKETS ARE THE STANDARD DEVIATION)

ties, we did not use the Wilcoxon-Pratt signed-rank test, which would have otherwise been an option. The statistics literature recommends permutation tests for such cases.

The p value for the paired asymptotic two-sample Fisher-Pitman permutation test is 0.05208, which is extremely close to, yet above our significance level of $\alpha = 0.05$. That is why we cannot reject the null hypothesis $H_0(\mathit{TLX})$.

In meta-analyses for reference values of the NASA-TLX questionnaire, Morten et al. [47] found an average TLX value of 42(±13), while Grier et al. [48] found an average value of 45(±14.99) for Raw-TLX. It is noticeable that T_S, with an average value of approximately 36 is far below these averages, where lower is better. The Raw-TLX for T_{NS}, with an average value of 43, is similar to the values reported in the literature.

In summary, even though the differences between T_S and T_{NS} regarding Raw-TLX is not statistically significant ($p = 0.05208$) if an $\alpha = 0.05$ is used, we can at least conclude that the perceived effort for the mapping step in our reflexion modeling is lower than average applications.

B. Efficiency in terms of duration

The subjects took 8 minutes and 40 seconds for task T_S and 9 minutes and 33 seconds for task T_{NS} on average. That is, the processing of task T_S, in which the suggestions were active, took about one minute less on average. The measurements for task T_S varied by around 3 minutes and 54 seconds. The standard deviation for task T_{NS} is slightly higher with 4 minutes and 8 seconds.

Here, too, an effect is visible that speaks against the corresponding null hypothesis $H_0(D)$ at first sight. For the same reasons stated above, the t-test could not be used. The Shapiro-Wilk test yielded a p value of 0.01574 for normality. Unlike for the evaluation of Raw-TLX, there were no ties for the paired comparison of duration, that is why we ran the exact Wilcoxon signed-rank test to check whether the differences are statistically significant. This test yielded a p value of 0.2939, hence, the differences are not significantly different. In summary, although a difference in favor of mapping suggestions could be observed, hypothesis $H_0(D)$ cannot be rejected.

C. Usability in terms of SUS

This section deals with the investigation of perceived usability of our reflexion modeling with suggestions in SEE as measured by SUS. No distinction is made here between the two tasks because the only difference in SEE, when suggestions are enabled, is the highlighting of mapping target canidates. All other interactions are the same. This difference is very minor and we are rather interested in the overall usability of our reflexion modeling.

Questions	**Mean weight**
Q1: I think that I would like to use this system frequently.	5.77 (±2.58)
Q2: I found the system unnecessarily complex.	7.69 (±2.79)
Q3: I thought the system was easy to use.	5.38 (±3.51)
Q4: I think that I would need the support of a technical person to be able to use this system.	7.5 (±3.39)
Q5: I found the various functions in this system were well integrated.	6.15 (±2.42)
Q6: I thought there was too much inconsistency in this system.	6.92 (±2.53)
Q7: I would imagine that most people would learn to use this system very quickly.	6.73 (±3.73)
Q8: I found the system very cumbersome to use.	7.31 (±2.97)
Q9: I felt very confident using the system.	6.35 (±2.82)
Q10: I needed to learn a lot of things before I could get going with this system.	7.69 (±2.79)
Total average SUS	**67.5** (±21.41)

TABLE III
MEAN WEIGHTS OF THE SUS RESPONSES (THE VALUES IN BRACKETS ARE THE STANDARD DEVIATION)

Table III shows the averaged SUS weights per question and the total SUS (the values for negatively worded questions are already inversed; all respones are multiplied by 2.5 as explained in Section IV-A).

We note—analogously to Section V-A—that the reponses for the SUS questions are given on a Likert scale and, hence, only ordinal, so that averaging them is actually against the principles of measurement theory. Yet, such is the procedure for SUS recommended and practiced in the scientific literature. Calculating the total SUS by summing up the individual weights already assumes an interval scale.

The SUS score of the sample reaches 67.5 points. In many studies, a SUS score of 68 is given as a reference value for average usability [49]. This value is close to the value that could be measured in the sample. Despite a small sample, an impression of the usability of reflexion modeling in SEE can be gained here. Nielsen argues that five subjects suffice to find major usability issues [50].

Beyond the quantified usability in terms of SUS, we also collected qualitative data as free-form feedback from the subjects. This helps us to identify possible problems which could have had an influence on the SUS score and the other dependent variables. The qualitative feedback from our subjects on the use of reflexion modeling in the code-city metaphor reveals several recurring themes and issues. The most important points are summarized below.

1) User interface and interaction: Change of interaction modes. Two subjects found switching interaction modes via the space bar unintuitive. The context menu was preferred. *Conflicts with control elements.* Two subjects reported problems with controlling the camera and displaying context menus by right-clicking. When turning the camera, the context menu was opened unintentionally. *Difficulties with the selection.* Three subjects mentioned problems when selecting objects in the code city. A spongy selection was criticized. In another run, nodes were overlaid by the layout after a movement and could no longer be selected. In addition, the freezing of the

'Show-In-City' animation was mentioned, which blocked the selection of a node. *Legend and orientation.* Two respondents wanted a clear legend for control commands. *Code window.* The code window caused problems for two subjects because it allowed unintentional interactions with the code city in the background the window was in front of.

2) Visualization and layout: Edges and colors. Five subjects reported problems with the visibility and color display of the edges. The edges were sometimes described as too bright or difficult to see. Among them were two subjects who wished a clearer representation of the edge directions. One respondent wanted architecture and implementation edges in different colors. We assumed here that the coloring of the convergent state was meant, as the edges are colored differently before a state change. Two subjects stated that the architectural edges were helpful. One subject's interactions with nodes were blocked by edges. *Understandability of the recommendations.* Two subjects had difficulty with understanding the recommendations and the user interface. *Confusion caused by the layout.* Some respondents mentioned confusion due to the layout. *Visualization of the suggestions.* One respondent stated that he felt supported by the suggestions. Another emphasized that the blink animation of searched and found blocks is lost in the edges of the city.

3) Technical problems: Crashes and errors. Three subjects experienced crashes during reflexion modeling, whereby the user was flooded by a high number of pop-up error messages. *Missing functions.* Some subjects wished for additional functions, such as fixed perspectives or automatic assignment of implementation modules.

D. Threats to validity

In our user study, several factors exist that could have influenced the validity of the results and must be taken into account. We distinguish between internal, external, and construct validity.

Internal validity: Internal validity looks at whether the variables were measured correctly. A potential influencing factor is the Hawthorne effect, as the study was supervised, which may have influenced the participants' behavior. Errors in our implementation may have influenced the SUS score. Furthermore, errors and problems that only occurred in the first or second tasks may have affected the RAW-TLX measurement between tasks. The results also showed a remarkably low NASA-TLX value for the domain. Possibly the time window of 20 minutes allowed to the subjects to complete the tasks was too generous. In addition, the mapping suggestions in the study were highly precise. For suggestions with a higher error rate, the suggestions could possibly offer users less reduced cognitive effort in comparison. The tasks might have had different levels of difficulty due to the implementation details of *PetClinic*, which limits the comparability of the results between the groups.

External validity: External validity looks at the general transferability of the results. As the study is limited by a small sample, the generalizability of the results is restricted. Our subjects may not be representative for the population as we used only convenience sampling. In particular, we did not succeed in involving female developers. The task of our study might have been too simple to demonstrate advantages of automated suggestions. In addition, in practical applications, the reduction of cognitive effort through suggestions depends heavily on the precision and quantity of the suggestions. This in turn depends on an existing system and its initial mappings. Whether and how often suggestions can be used to reduce the workload for other systems must be investigated by future studies. Furthermore, the results are influenced by the concrete visualization of the suggestions. Suggestions of the same attract function in the same system with a different visualization could burden the user less or more. The impression of the users was largely gained from first-time users of SEE. If users have more experience in using SEE, stronger or weaker effects on effort, duration, and usability could arise.

Construct validity: Construct validity concerns how well our measures reflect a concept that is not directly measurable. We used NASA Raw-TLX as a measure for effort and SUS as a measure for usability. Although they are widely used in the scientific literature, they may not necessarily reflect effort and usability, respectively, well enough.

VI. Conclusion

The automated mapping suggestions based on attract functions were successfully integrated for real-time visualization into our implementation of reflexion modeling in SEE. During our user study, effects of reduced cognitive effort and time savings were measured when automated suggestions were available during reflexion modeling, although no statistical significance of the observed differences could be found. Despite technical difficulties during use, the implementation of reflexion modeling in SEE showed an average usability according to SUS. Accordingly, further improvement appears to be necessary in order to increase its usability above average. Our participants gave us many concrete hints. This includes better visualization of edge directions and edge states, a more reliable and precise selection of elements in the code city, and better integration of the suggestions with the rest of the UI.

References

[1] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between source and high-level models," in *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1995, pp. 18–28.

[2] R. Koschke, "Industrial experience on code clean-up using architectural conformance checking," in *Proceedings of the 12th European Conference on Software Architecture (ECSA): Companion Proceedings*, 2018.

[3] A. Christl, R. Koschke, and M.-A. Storey, "Automated clustering to support the reflexion method," *Information and Software Technology*, vol. 49, no. 3, pp. 255–274, 2007.

[4] M. Bibi, O. Maqbool, and J. Kanwal, "Supervised learning for orphan adoption problem in software architecture recovery," *Malaysian Journal of Computer Science*, vol. 29, pp. 287–313, 2016.

[5] R. A. Bittencourt, G. J. d. Santos, D. D. S. Guerrero, and G. C. Murphy, "Improving automated mapping in reflexion models using information retrieval techniques," in *IEEE Working Conference on Reverse Engineering*, 2010, pp. 163–172.

[6] T. Olsson, M. Ericsson, and A. Wingkvist, “To automatically map source code entities to architectural modules with naive bayes,” *Journal of Systems and Software*, vol. 183, p. 111095, 2022.

[7] Z. T. Sinkala and S. Herold, “InMap: Automated interactive code-to-architecture mapping recommendations,” in *IEEE International Conference on Software Architecture*, March 2021, pp. 173–183.

[8] D. Link, P. Behnamghader, R. Moazeni, and B. Boehm, “Recover and relax: Concern-oriented software architecture recovery for systems development and maintenance,” in *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, 2019, pp. 64–73.

[9] S. M. Naim, K. Damevski, and M. S. Hossain, “Reconstructing and evolving software architectures using a coordinated clustering framework,” *Automated Software Engineering*, vol. 24, pp. 543–572, 2017.

[10] K. Andrews, J. Wolte, and M. Pichler, “Information pyramids: A new approach to visualising large hierarchies,” in *IEEE Conference on Visualization*, 1997, pp. 49–52.

[11] J. Knodel and D. Popescu, “A comparison of static architecture compliance checking approaches,” in *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Jan 2007.

[12] L. Passos, R. Terra, M. T. Valente, R. Diniz, and N. Mendonça, “Static architecture-conformance checking: An illustrative overview,” *IEEE Software*, vol. 27, no. 5, pp. 82–89, Sep. 2010.

[13] J. Buckley, A. LeGear, C. Exton, R. Cadogan, T. Johnston, B. Looby, and R. Koschke, “Encapsulating targeted component abstractions using software reflexion modelling,” *Journal on Software Maintenance and Evolution*, vol. 20, no. 2, pp. 107–134, March–April 2008.

[14] C. Ackermann, M. Lindvall, and R. Cleaveland, “Towards behavioral reflexion models,” in *International Symposium on Software Reliability Engineering*, Nov 2009, pp. 175–184.

[15] W. Said, J. Quante, and R. Koschke, “Reflexion models for state machine extraction and verification,” in *IEEE International Conference on Software Maintenance and Evolution*, Sep. 2018, pp. 149–159.

[16] R. Koschke, P. Frenzel, A. P. Breu, and K. Angstmann, “Extending the reflexion method for consolidating software variants into product lines,” *Software Quality Journal*, vol. 17, no. 4, pp. 331–366, Dec. 2009.

[17] B. Tekinerdogan, E. Çilden, Ö. Ö. Erdogan, and O. Aktug, “Architecture conformance analysis approach within the context of multiple product line engineering,” in *Australian Software Engineering Conference*, April 2014, pp. 25–28.

[18] E. Çilden and H. Oğuztüzün, “A reflexion model based architecture conformance analysis toolkit for OSGi-compliant applications,” in *IEEE International Conference on Software Architecture Workshops (ICSAW)*, April 2017, pp. 263–266.

[19] A. Le Gear, J. Buckley, J. Collins, and K. O’Dea, “Software reconnexion: understanding software using a variation on software reconnaissance and reflexion modelling,” in *International Symposium on Empirical Software Engineering*, Nov 2005.

[20] S. Herold, M. English, J. Buckley, S. Counsell, and M. O. Cinnéide, “Detection of violation causes in reflexion models,” in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, March 2015, pp. 565–569.

[21] R. Koschke and D. Simon, “Hierarchical reflexion models,” in *IEEE Working Conference on Reverse Engineering*, Nov. 2003, pp. 36–45.

[22] R. Koschke, “Incremental reflexion analysis,” *Journal on Software Maintenance and Evolution*, vol. 25, no. 6, pp. 601–637, 2013.

[23] R. A. Bittencourt, “Conformance checking during software evolution,” in *IEEE Working Conference on Reverse Engineering*, Oct 2010, pp. 289–292.

[24] M. Romanelli, A. Mocci, and M. Lanza, “Towards visual reflexion models,” in *International Conference on Program Comprehension*, May 2015, pp. 277–280.

[25] L. Erhardt and R. Koschke, “Automated mapping suggestions for the reflexion analysis,” in *Workshop on Software Architecture Erosion and Architectural Consistency (SAEroCon)*, ser. ICSA 2025 Companion Proceedings, 2025.

[26] T.-h. Kim, K. Kim, and W. Kim, “An interactive change impact analysis based on an architectural reflexion model approach,” in *IEEE 34th Annual Computer Software and Applications Conference*, 2010, pp. 297–302.

[27] J. Buckley, N. Ali, M. English, J. Rosik, and S. Herold, “Real-time reflexion modelling in architecture reconciliation: A multi case study,” *Information and Software Technology*, vol. 61, pp. 107–123, 2015.

[28] J. Buckley, S. Mooney, J. Rosik, and N. Ali, “JITTAC: A just-in-time tool for architectural consistency,” in *ACM/IEEE International Conference on Software Engineering*, May 2013, pp. 1291–1294.

[29] M. Biehl and W. Löwe, “Automated architecture consistency checking for model driven software development,” in *Proceedings of the 5th International Conference on the Quality of Software Architectures (QoSA): Architectures for Adaptive Software Systems*. Springer-Verlag, 2009, p. 36–51.

[30] C. Anslow, S. Marshall, J. Noble, and R. Biddle, “SourceVis: Collaborative software visualization for co-located environments,” in *IEEE Working Conference on Software Visualization*, Sep. 2013, pp. 1–10.

[31] C. Anslow, “Collaborative software visualization in co-located environments,” PhD Dissertation, Victoria University of Wellington, New Zealand, 2013.

[32] D. H. R. Holten, “Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, Sep. 2006.

[33] M. Wertheimer, “Untersuchungen zur Lehre von der Gestalt,” *Zeitschrift für Psychologische Forschung und ihre Grenzwissenschaften*, vol. 4, pp. 300–350, 1923.

[34] L. Erhardt, “Ein Vergleich von Attract-Funktionen für die Reflexionsanalyse hinsichtlich Struktur, Code-Termen und interaktive Nutzung,” Master’s Thesis, Department for Mathematics and Informatics, University of Bremen, Germany, Aug. 2024.

[35] NASA, *Task Load Index (NASA-TLX)*, 1st ed., NASA Ames Research Cener, 1998.

[36] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale,” *International Journal of Human-Computer Interaction*, vol. 24, no. 6, pp. 574–594, Jul. 2008.

[37] J. C. Byers, A. C. Bittner, and S. G. Hill, “Traditional and raw task load index (TLX) correlations: Are paired comparisons necessary?” in *Advances in Industrial Ergonomics and Safety I*, 1989, pp. 481–485.

[38] S. Hart, “NASA task load index (NASA-TLX); 20 years later,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 50, no. 9, 10 2006, pp. 904–908.

[39] J. Brooke, *Usability Evaluation in Industry*. Taylor and Francis, 1996, ch. SUS: A quick and dirty usability scale.

[40] B. Rummel and E. Rügenhagen, “System usability scale–jetzt auch auf Deutsch,” https://community.sap.com/t5/additional-blogs-by-sap/system-usability-scale-jetzt-auch-auf-deutsch/ba-p/13487686, 2016.

[41] J. R. Lewis, “Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ,” in *Proc. of the Human Factors and Ergonomics Society Annual Meeting*, vol. 36, no. 16, 1992, pp. 1259–1260.

[42] ——, “Computer system usability questionnaire,” *International Journal of Human-Computer Interaction*, 1995.

[43] F. D. Davis, “Perceived usefulness, perceived ease of use, and user acceptance of information technology,” *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[44] B. Laugwitz, T. Held, and M. Schrepp, “Construction and evaluation of a user experience questionnaire,” in *HCI and Usability for Education and Work*, A. Holzinger, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 63–76.

[45] T. S. Tullis and J. N. Stetson, “A comparison of questionnaires for assessing website usability,” in *Usability professional association conference*, vol. 1. Minneapolis, USA, 2004, pp. 1–12.

[46] T. Brix, A. Janssen, M. Storck, and J. Varghese, “Comparison of German translations of the system usability scale— which to take?” *Studies in health technology and informatics*, vol. 307, pp. 96–101, 09 2023.

[47] M. Hertzum, “Reference values and subscale patterns for the task load index (TLX): A meta-analytic review,” *Ergonomics*, vol. 64, pp. 8–9, 01 2021.

[48] R. Grier, “How high is high? a meta-analysis of NASA-TLX global workload scores,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 59, no. 1, 10 2015, pp. 1727–1731.

[49] J. Sauro, “5 ways to interpret a SUS score,” 2018. [Online]. Available: https://measuringu.com/interpret-sus-score/

[50] J. Nielsen, “Why you only need to test with 5 users,” 2000. [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/