Information and Software Technology 87 (2017) 259–277

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software landscape and application visualization for system comprehension with ExplorViz

Florian Fittkau[1,*], Alexander Krause, Wilhelm Hasselbring

Software Engineering Group, Kiel University, D-24098 Kiel, Germany

ARTICLE INFO

Article history:
Received 20 December 2015
Revised 12 July 2016
Accepted 14 July 2016
Available online 15 July 2016

Keywords:
Software visualization
Dynamic analysis
System comprehension

ABSTRACT

Context: The number of software applications deployed in organizations is constantly increasing. Those applications – often several hundreds – form large software landscapes.

Objective: The comprehension of such landscapes and their applications is often impeded by, for instance, architectural erosion, personnel turnover, or changing requirements. Therefore, an efficient and effective way to comprehend such software landscapes is required.

Method: In our ExplorViz visualization, we introduce hierarchical abstractions aiming at solving system comprehension tasks fast and accurately for large software landscapes. Besides hierarchical visualization on the landscape level, ExplorViz provides multi-level visualization from the landscape to the level of individual applications. The 3D application-level visualization is empirically evaluated with a comparison to the Extravis approach, with physical models and in virtual reality. To evaluate ExplorViz, we conducted four controlled experiments. We provide packages containing all our experimental data to facilitate the verifiability, reproducibility, and further extensibility of our results.

Results: We observed a statistical significant increase in task correctness of the hierarchical visualization compared to the flat visualization. The time spent did not show any significant differences. For the comparison with Extravis, we observed that solving program comprehension tasks using ExplorViz leads to a significant increase in correctness and in less or similar time spent. The physical models improved the team-based program comprehension process for specific tasks by initiating gesture-based interaction, but not for all tasks. The participants of our virtual reality experiment with ExplorViz rated the realized gestures for translation, rotation, and selection as highly usable. However, our zooming gesture was less favored.

Conclusion: The results backup our claim that our hierarchical and multi-level approach enhances the current state of the art in landscape and application visualization for better software system comprehension, including new forms of interaction with physical models and virtual reality.

© 2016 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license.
(http://creativecommons.org/licenses/by-nc-nd/4.0/)

1. Introduction

While program comprehension has been researched extensively [1], system comprehension has received much less attention [2]. From a historical point of view, program comprehension became important when programs reached more than a few hundreds lines of code. Today's IT infrastructures in enterprises often consist of several hundreds of applications forming large software landscapes [3]. Therefore, system comprehension – in our terminology the comprehension of such landscapes – is a crucial part of the maintenance process [4]. This circumstance is intensified by, for instance, Cloud Computing which provides scalability through replication of nodes and thus increases the number of deployed applications.

One way to achieve system comprehension is software landscape visualization. Current software landscape visualizations are mostly found in application performance management (APM) tools. While surveying them, we observed that these tools often use a flat graph-based representation of nodes, applications, and their communication.

In contrast, our ExplorViz approach [5], which provides live trace visualization for large software landscapes, introduces three

* Corresponding author.
E-mail addresses: florian.fittkau@gmx.de (F. Fittkau), akr@informatik.uni-kiel.de (A. Krause), hasselbring@email.uni-kiel.de (W. Hasselbring).
URL: http://se.informatik.uni-kiel.de (W. Hasselbring)
[1] Present address: PPI AG, Wall 55, 24103 Kiel, Germany.

http://dx.doi.org/10.1016/j.infsof.2016.07.004
0950-5849/© 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license. (http://creativecommons.org/licenses/by-nc-nd/4.0/)

hierarchical abstractions [3]. First, there are *systems* which consist of one or more server nodes. Second, especially designed for cloud environments and their horizontal scalability, our hierarchical visualization features *node groups* which cluster nodes that are running the same application configuration. Third, the *amount of communication* between the applications is represented by the thickness of the communication links.

Besides such hierarchical visualizations on the landscape level, the ExplorViz approach offers multi-layer monitoring from landscape level to application level [5].

While these visualizations seem reasonable, it should still be evaluated whether they provide any benefits concerning the comprehension process [6–8]. For example, the users might not understand the abstractions, or the abstractions might not support or might even hinder the user in solving system comprehension tasks.

The main contributions of the present paper in this context are:

1. An introduction to the hierarchical and multi-layer visualization of large software landscapes with ExplorViz, including its meta model and the process of generating these models from monitoring traces.
2. The reusable design and execution of a controlled experiment comparing a flat landscape visualization to our hierarchical landscape visualization in system comprehension tasks, with an emphasis on the methodology of how we operate a controlled experiment. This includes a thorough analysis of typical sources of error and the strategies chosen by the participants for each task.
3. For evaluating the application-level visualization of ExplorViz, we summarize the controlled experiments for comparing ExplorViz with the Extravis trace visualization approach, for employing physical 3D-printed ExplorViz models, and for exploring 3D ExplorViz models in virtual reality.

Beneath evaluating whether a hierarchical and multi-level visualization provides benefits, we conducted these experiments to get input for improving our ExplorViz tool.[2]

The remainder of this article is organized as follows. Our live trace visualization ExplorViz – building the foundation – is described in Section 2. Section 3 presents the flat, state-of-the-art visualization of software landscapes and introduces our hierarchical landscape visualization. Afterwards, Section 4 presents a controlled experiment to evaluate the impact of using a hierarchical visualization for system comprehension on the landscape level instead of a flat visualization. Section 5 presents the empirical evaluation of application-level visualizations in ExplorViz. Related work is discussed in Section 6. Finally, we draw the conclusions and illustrate future work in Section 7.

2. ExplorViz

ExplorViz offers live monitoring of large software landscapes [3]. In particular, the ExplorViz approach offers hierarchical visualization [9] and multi-layer monitoring from landscape level to application level [5]. ExplorViz provides mechanisms to acquire up-to-date (live) information that is consistent with the actual enterprise application landscape and information systems. Our tool introspects applications and information systems to a fine-grained level, and facilitates to improve the resource capacity [10,11] in enterprise application landscapes by means of a control center [12].

We introduce ExplorViz via modeling the software landscape of the Kiel Data Management Infrastructure at the Helmholtz Centre for Ocean Research Kiel (GEOMAR) in Section 2.1. We introduce the application level in Section 2.2. ExplorViz generates these landscape and application level visualizations from the information contained in the monitoring traces. Section 2.3 introduces the underlying meta model of ExplorViz. The trace-to-model mapping, i.e. how these visualizations are generated from the monitoring log, is presented in Section 2.4.

2.1. Landscape-level visualization

Fig. 1 shows the modeled infrastructure of the GEOMAR's Kiel Data Management Infrastructure for ocean science[3] on the landscape level. The large gray boxes with, e.g., PubFlow (❶) [13], represent the systems present in the software landscape. They can also be minimized such that only the system and its communication are visible, without their interior. Thus, providing abstraction on the level of systems and only visualizing systems currently in focus.

The smaller green boxes in one system represent the contained node groups (❷) or nodes (❸). Node groups are labeled with a textual representation of their contained nodes, for example, "10.0.0.1 –10.0.0.7". We introduced node groups because in cloud computing, for instance, nodes are scaled for performance reasons, but typically keep their application configuration. For providing an overview, these nodes are grouped. However, they can be extended with the *plus* symbol near the node group.

A node can contain different applications (❹). The communication between applications is visualized by lines. In accordance to their call count, the line thickness changes, i.e., higher amount of communication leads to thicker communication lines (❺). The user can navigate to the application-level perspective by choosing one application.

As already stated, we feature a time shift mode to analyze specific situations (❻). To provide an indication, when large amounts of calls are processed, the call count of the entire landscape is shown on the y-axis. A configurable time window is shown on the x-axis.

We employ auto-layout algorithms to ensure that the user does not need to manually layout the nodes which can be infeasible in large software landscapes. The employed flow-based auto-layout, named KLay Layered[4] [14], orders the nodes and applications in accordance to our defined communication flow direction, i.e., from left to right. Future work should make the flow direction configurable in the GUI.

2.2. Application-level visualization

Fig. 2 displays the application-level perspective of the source code analyzer PMD[5] with ExplorViz. The flat green boxes (❶) in our 3D visualization represent packages showing their contained elements. The green boxes on the top layer are packages (❷) hiding their internal details. They can be opened or closed interactively. Classes are visualized by purple boxes and the communication link is displayed by orange lines (❸). The width of the line corresponds to the call frequency of the represented methods. The height of classes maps to the active instance count. The layout is a modified version of the layout used in CodeCity [15]. While CodeCity visualizes the static software structure, ExplorViz provides dynamic live visualization of monitoring logs.

ExplorViz follows a hierarchical, top-down approach [16]. Therefore, details about the classes and their communication links are provided on demand following the Shneiderman mantra [17] of

[2] http://www.explorviz.net.

[3] https://portal.geomar.de.

[4] http://rtsys.informatik.uni-kiel.de/confluence/x/joAN.

[5] https://pmd.github.io/.

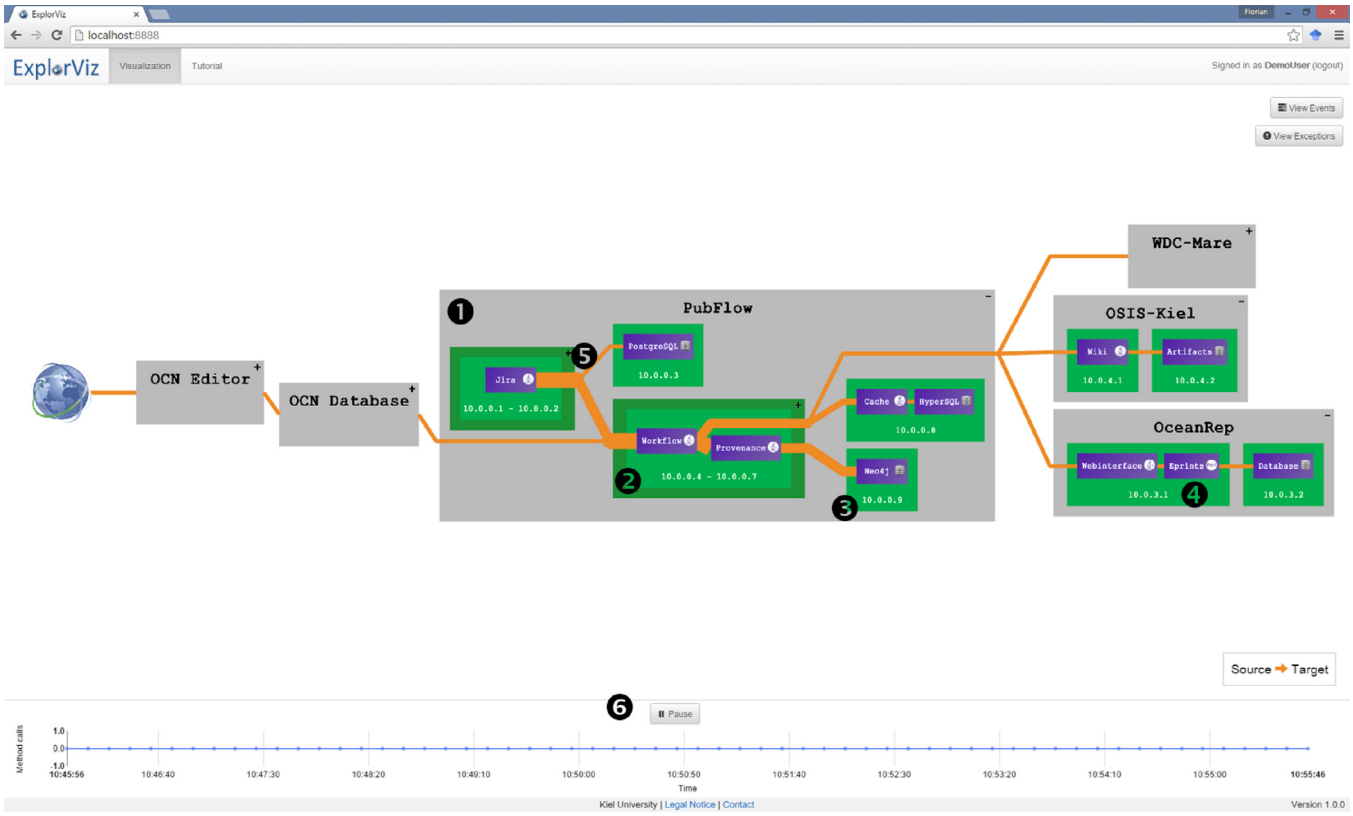


Fig. 1. Landscape-level perspective visualization of the Kiel Data Management Infrastructure for ocean science in ExplorViz.

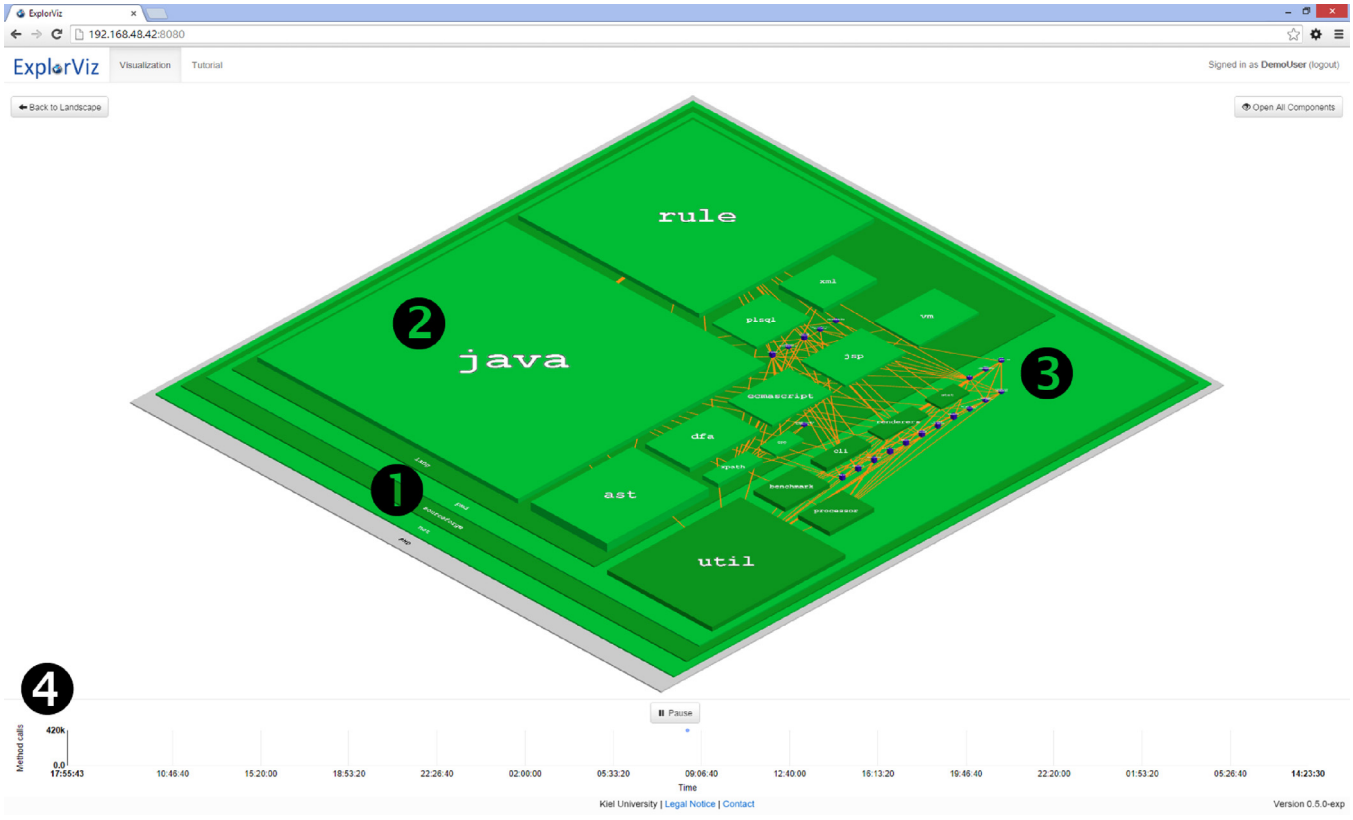


Fig. 2. Application-level perspective visualization of PMD in ExplorViz.

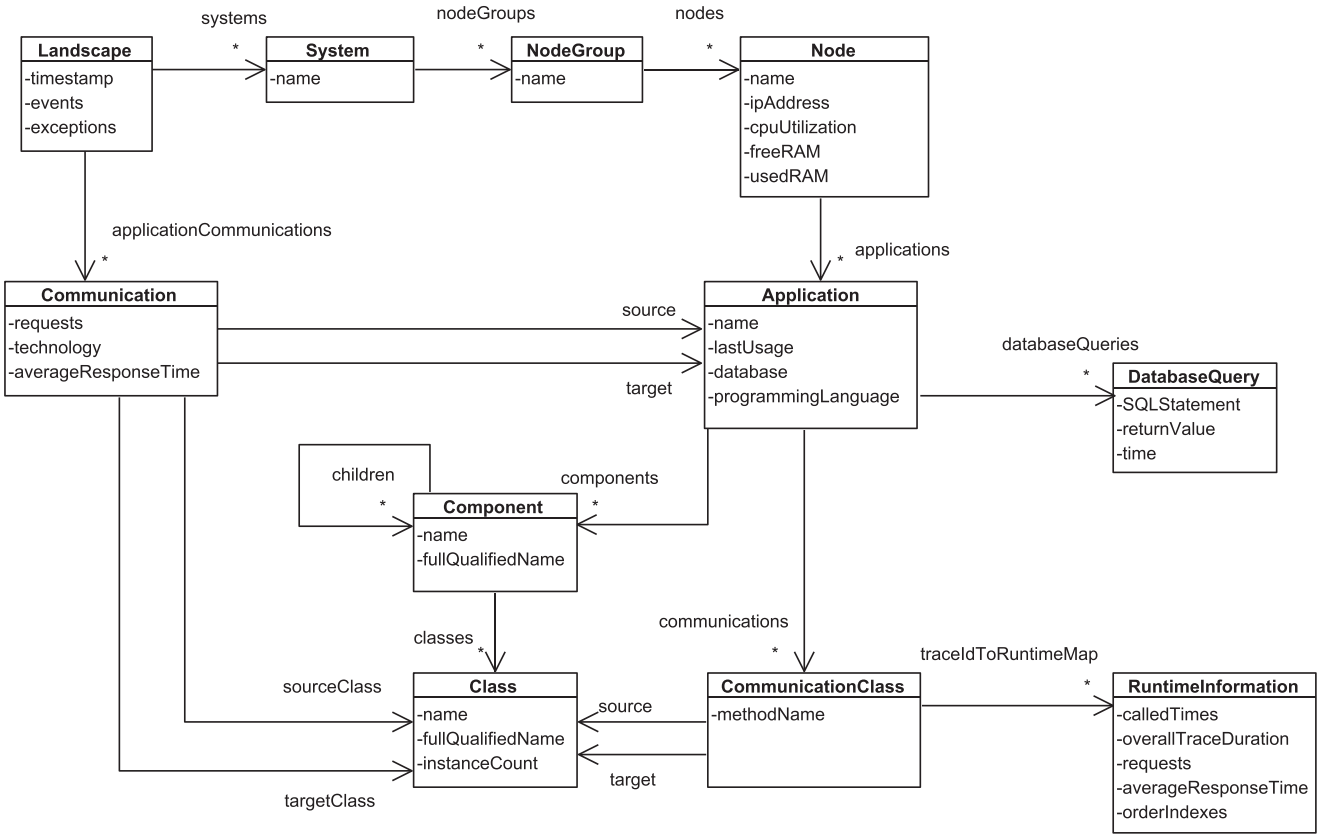


Fig. 3. ExplorViz meta model.

"Overview first, zoom and filter, then details-on-demand." To explore the relationships between classes, the user can mark classes by clicking on them to highlight their incoming and outgoing communication links and to obtain details through tooltips.

2.3. The ExplorViz meta model

Fig. 3 displays the ExplorViz meta model. A *Landscape* class represents the top-level entity of the meta model. It contains a *timestamp* attribute to represent the timestamp of the ExplorViz model's creation. Furthermore, it has a list of landscape *events*, e.g., "Node Demo1 has been added" and a list of *exceptions* which occurred in the landscape. The *Landscape* class has a reference list of *Systems*. In our terms, a *System* represents a logical union of multiple applications and servers. The *System* class contains a *name* attribute representing the actual name of the system. Furthermore, *System* holds a reference list of *NodeGroups*.

A *NodeGroup* forms a logical abstraction from the servers and applications by representing servers which have the same application configuration. An equal application configuration typically occurs in, for instance, cloud environments. A *NodeGroup* contains an attribute *name* which is formed by the range of the lowest IP address and the highest IP address in it. Furthermore, A *NodeGroup* has a reference list to *Nodes*. One *Node* has a *name* (hostname) and an IP address. In addition, it contains attributes to represent the current utilization of the server and the current average CPU utilization. A *Node* holds a reference list of *Applications* running on it.

An *Application* has the attributes *name, last usage* representing the last timestamp where activity was monitored, whether it is a *database* or not, and the *programming language*. In addition, it holds a list of *DatabaseQueries* which represent one database query, i.e., its *SQL statement*, the *return value*, and the execution *time*. An *Application* contains a reference list to *Components*. *Components* represent logical organization units of classes [18]. For example, packages can be *Components* in the context of Java. A *Component* has the attributes *name* and *full qualified name*. The latter represents the full name including the names of parent *Components*. Furthermore, a *Component* contains a list of *Components*, i.e., its children, and a list of *Classes*.

A *Class* represents the lowest level entity in our meta model. It has a *name*, a *full qualified name* representing its name and including parent components' names, and an *instance count* attribute representing how many instances were active.

The *Landscape* class also holds a list of *Communication* links which exist between *Applications*. In addition, the *source* and *target* class are referenced by the *Communication* link. A *Communication* link has the attributes *requests* which can also be zero, an attribute *technology* meaning the actual technology used to conduct the Remote Procedure Call (RPC), and an *average response time* of the RPC.

An *Application* also contains a list of *Communication* links but on the *Class* level. The *CommunicationClass* class contains a *method name* attribute and has a *source* and *target Class*. To be able to visualize the actual traces in the application-level perspective, it also holds a reference map between a trace identifier and a *Runtime Information*. A *Runtime Information* provides the information about how many *times* the trace was *called*, the *overall trace duration*, the *request* amount of the method, the *average response time*, and a list of *order indexes* representing the position in the trace.

2.4. Trace-to-model mapping

Monitoring the software landscape generates trace information from the running systems. As a prerequisite, the software needs to be instrumented with monitoring probes [19–21]. After processing

these traces from the monitoring sources, the model – corresponding to the meta model of the previous subsection – is generated from the information contained in the monitoring traces. Subsequently, the ExplorViz model provides the required information to generate the visualizations.

In this subsection, we describe how the information contained in the monitoring traces are used to create a ExplorViz model. Each monitoring trace includes *HostApplicationMetaDataRecords* which contain information about the originating system, the IP address and hostname of the server, the application name, and the programming language. Since the traces are generated in a distributed system and potentially each monitoring record might be processed on different analysis servers, every operation monitoring record contains such information. The contained information are used to create the counterpart in the ExplorViz model. For example, the existing systems are iterated and if the current system is not found, a new instance for the new system is created. This procedure also applies to nodes and application instances. After inserting a new node or application into the model, the node groups get updated accordingly.

For generating the components, classes, and communication on the application level, the operation monitoring records of the trace are iterated. The contained method signature is used to create or update components and classes. The method call is used to create or update the communication links on the class level. In addition, the *Runtime Information* is created as part of this communication link. As explained in Section 2.2, this runtime information may be retrieved by selecting the corresponding orange communications links.

Beneath this normal mapping of the traces, there exist separate records for special purposes. The special caller and callee records for RPC monitoring are matched to create the communication on the landscape level. Another special record is the *SystemMonitoringRecord* which contains information about the CPU utilization and RAM usage. This record is used to update the utilization of a node. Furthermore, the record for monitoring database calls is used to generate a list of database queries for each application.

3. Flat vs. hierarchical landscape visualization

In Section 4, we will present a controlled experiment to compare a flat, state-of-the-art landscape visualization to our hierarchical visualization in the context of system comprehension. These flat and hierarchical visualizations are now introduced in Sections 3.1 and 3.2, respectively.

3.1. Flat landscape visualization

This section introduces the flat software landscape visualization used in the experiment by the control group to solve system comprehension tasks.

Current landscape visualizations can mostly be found in application performance management (APM) tools, for instance, AppDynamics,[6] Foglight,[7] or Dynatrace.[8] Those tools are often driven by commercial interest and thus are not free to use or – if they have an evaluation phase – it is explicitly prohibited to conduct studies with these versions. Therefore, we had to create our own implementation of the visualization which follows the concepts of current landscape visualizations available in APM tools. By implementing the visualization into our ExplorViz tool, we also assure that interaction capabilities are the same between both groups in the experiment. This also leads to a higher reliability of the presented results.

After surveying the available visualizations, we implemented the visualization depicted in Fig. 4 which is a mixture of the concepts we rated as best suitable for system comprehension. The figure shows an excerpt of the used object landscape, i.e., a model of the technical IT infrastructure of the Kiel University. Nodes are visualized as green boxes (❶) with white labels representing the hostname of each node at the bottom. The applications running on the nodes are visualized by purple boxes (❷). A white label shows the application name at the center. Besides the label, the programming language or – in the special case of a database – a database symbol is depicted. The communication between applications is represented by orange lines (❸). The conducted request count is shown next to a line in black letters in the abbreviated form of, e.g., *10 req*.

We employ auto-layout algorithms to ensure that the user does not need to manually layout the nodes which can be infeasible in large software landscapes. The employed flow-based auto-layout, named KLay Layered[9] [14], orders the nodes and applications in accordance to our defined communication flow direction, i.e., from top to bottom. In our object landscape, all communication paths originate at a `Network` entity visualized as a globe which is not shown in the picture. The cropped lines at the top directly lead into this globe entity.

All entities provide more information on demand by means of a tooltip when hovering over them with the mouse. Further interaction possibilities include the dragging of the view for navigation purposes, and zooming for an overview of the landscape.

3.2. Hierarchical landscape visualization

This section briefly presents our hierarchical software landscape visualization. The hierarchical visualization [3,22] is part of our web-based ExplorViz tool [5] which enables live trace visualization in large software landscapes.

Our hierarchical visualization is shown in Fig. 5. It visualizes nodes and applications in the same way as the flat visualization does. However, it also features hierarchies, i.e., systems and node groups. In our terms, systems (❹) are made up of one or more nodes which form a semantic union. Systems are visualized as gray boxes with their name labeled at the top. Nodes running the same application configuration form a node group (❺) represented by a dark green frame. In a node group, the hostnames are grouped to a joint label, for example, `Cloud Node Server 1 - Cloud Node Server 8`. Furthermore, systems and node groups are interactively extensible and collapsible to get a quick overview and details on demand. By default, systems are opened and node groups are collapsed.

A further difference to the flat visualization is the display of communication lines (❻) in the hierarchical visualization. This provides a further abstraction from the flat visualization which represents the request count through labels. In our hierarchical visualization, the thickness of the communication lines follows its conducted request count. For instance, the orange line displayed at ❻ is thicker than the other communication lines, indicating that more request are conducted between the connected applications. The thickness of the lines is determined by linearly grouping the request count into four categories. The actual number of request can also be viewed on demand – like in the flat visualization – by hovering over the communication line.

[6] http://www.appdynamics.com.
[7] http://www.foglight.com.
[8] http://www.dynatrace.com.
[9] http://rtsys.informatik.uni-kiel.de/confluence/x/joAN.

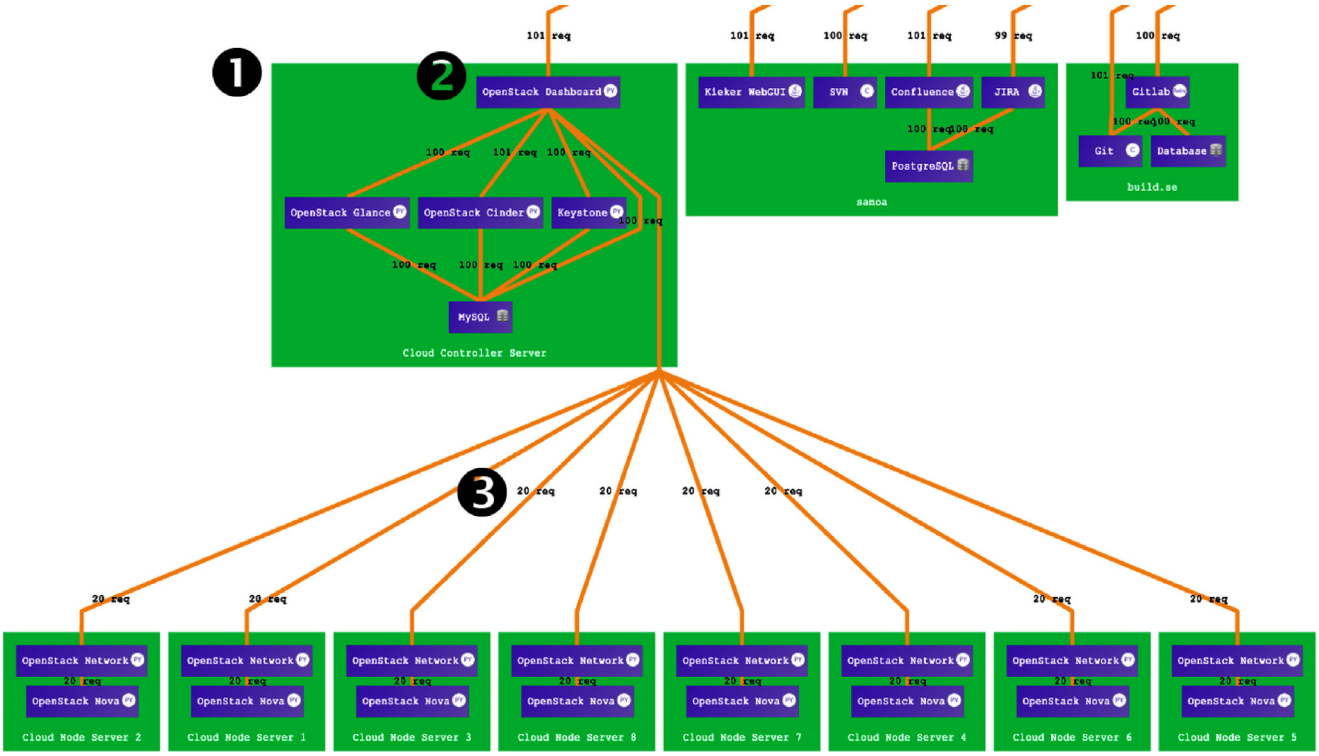


Fig. 4. An excerpt (29 of 140 applications) of the model of the technical IT infrastructure of the Kiel University in the flat landscape visualization.

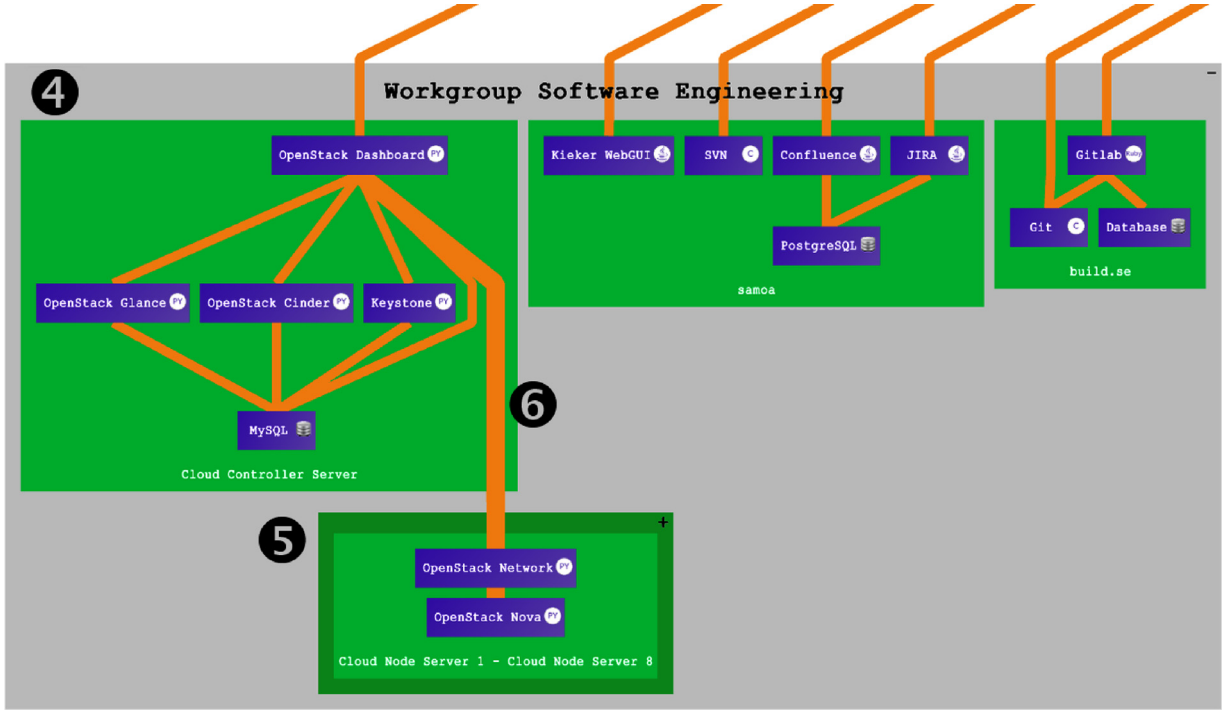


Fig. 5. An excerpt of the model of the technical IT infrastructure of the Kiel University in the hierarchical landscape visualization.

4. Evaluation of landscape-level visualization with ExplorViz

In this section, we present our controlled experiment which compares the usage of a flat, state-of-the-art landscape visualization to our hierarchical landscape visualization in system comprehension tasks. The experiment was first described in [9] and this section is a substantial extension to this paper by describing how we generate the landscape model and providing details on our developed tutorial and questionnaire mode used to operate the controlled experiments. To the best of our knowledge, we are the first to conduct such a controlled experiment comparing two software landscape visualizations.

In our experiment, 29 participants solved five system comprehension tasks. We used a medium-sized model of the IT infrastructure of the Kiel University containing 140 applications as object landscape. To facilitate the verifiability and reproducibility of our

results, we provide a package [23] containing all our experimental data including source code for both visualizations, input files, tutorial material, questionnaires, R scripts, raw data, and 29 recordings of the participant sessions. The package is available with source code under the Apache 2.0 License and the data under a Creative Commons License (CC BY 3.0).

For our object landscape we measure the *time spent* and *correctness* for each task which are typical metrics in the context of program comprehension [24]. Afterwards, we analyze the employed strategies and possible differences between both groups.

We describe the design of our controlled experiment, its operation, data collection, analysis, results, discussion (including reasoning about the different performances in each task), and threats to validity.

4.1. Experimental design

In addition to general software engineering experimentation guidelines [25–29], we follow the designs of Wettel et al. [30] and of Cornelissen et al. [1]. Similar to them, we use a *between-subjects* design. Thus, each subject solves tasks either using the flat or the hierarchical visualization. Following GQM [31], we define the goal of our experiment as quantifying the impact of using either the flat visualization or the hierarchical one for system comprehension.

We name the control group *Flat Group* and the experimental group *Hierarchical Group*. Due to space constraints, we abbreviate the groups as *Flat* and *Hierarchical* in figures and tables.

4.1.1. Research questions & hypotheses

We define three research questions (RQ) for our defined goal:

- **RQ1:** What is the ratio between Flat Group and Hierarchical Group in the *time required for completing* system comprehension tasks?
- **RQ2:** What is the ratio between Flat Group and Hierarchical Group in the *correctness of solutions* to system comprehension tasks?
- **RQ3:** Which sources of error exist when solving system comprehension tasks with either of the two visualization types?

Accordingly, we formulate two hypotheses:

- **H1** Flat Group and Hierarchical Group require different times for completing system comprehension tasks.
- **H2** The correctness of solutions to system comprehension tasks differs between Flat Group and Hierarchical Group .

The null hypotheses $H1_0$ and $H2_0$ follow analogously. For RQ3, we conduct an in-depth analysis of the results and analyze the recorded sessions of each participant in detail.

4.1.2. Dependent and independent variables

The independent variable is the visualization used for the system comprehension tasks, i.e., flat or hierarchical visualization. We measured the accuracy (*correctness*) and response time (*time spent*) as dependent variables. These are usually investigated in the context of program comprehension [1,24,30] and thus should also be applicable in the context of system comprehension.

4.1.3. Treatment

The control group used the flat visualization to solve the given system comprehension tasks. The experimental group solved the tasks utilizing the hierarchical visualization which includes the abstractions of systems, node groups, and the communication lines representing the conducted requests by the line's thickness.

4.1.4. Tasks

We selected a medium-sized software landscape (140 applications). Our model of the IT infrastructure of the Kiel University landscape represents services of our working group, computing center services, examination services, information services, system operating group services, and management services. We modeled the landscape by available information from the Internet and prior knowledge, and thus the model might not reflect the real deployment in detail. However, this is unimportant for the tasks.

In Table 1, our defined tasks including their context and achievable maximum points are displayed. To prevent guessing, all tasks were given as open questions. Our task set starts with less complex tasks (identifying applications with a high fan-in) and ends with a more complex risk management task. This enabled each subject to get familiar with the visualization in the first tasks and raises the level of complexity in the following ones. We chose only five tasks since we aimed to stay in a one hour time slot and prevent exhaustion issues.

4.1.5. Population

The 29 subjects were students from the master course "Software Engineering for Parallel and Distributed Systems." For successfully solving one task, they received bonus points for the final exam of the course. As further motivation, the students could win one of ten gift cards over 10 € for the sole participation and the best five subjects each received a gift card over 30 € .

The subjects were assigned to the Flat Group or Hierarchical Group by random assignment. To validate the equal distribution of experiences, we asked the participants to perform a self-assessment on a 5-point Likert Scale [32] ranging from 0 (no experience) to 4 (expert with years of experience) before the experiment. The average programming experience in the control group was 2.5 versus 2.6 in the experimental group. The average dynamic analysis experience was between no experience and beginner in

Table 1
Description of the system comprehension tasks for the experiment.

ID	Description	Score
	Context: Identification of critical dependencies	
T1	Name three applications that have a high fan-in (at least two incoming communication lines). The two incoming communication lines should be on one node and not distributed over multiple nodes.	3
	Context: Potential Bottleneck Detection	
T2	Name the Top 3 communications with the highest request count in descending order. Write down the start application and the end application.	4
	Context: Scalability Evaluation	
T3	Which applications are duplicated on multiple nodes? The answer should contain all 8 duplicated applications which are all named differently. Hint: The hostname of the nodes, where the applications are running, are numbered, e.g., Server 1, Server 2,...	4
	Context: Service Analysis	
T4	What is the purpose of the WWWPRINT application in your opinion? How does the process might work to achieve the functionality for the user?	4
	Context: Risk Management	
T5	What are the consequences of a failure of the LDAP application? Name all affected applications and briefly describe their purposes. Hint: Remember the received paper about the introduction to the university landscape.	7

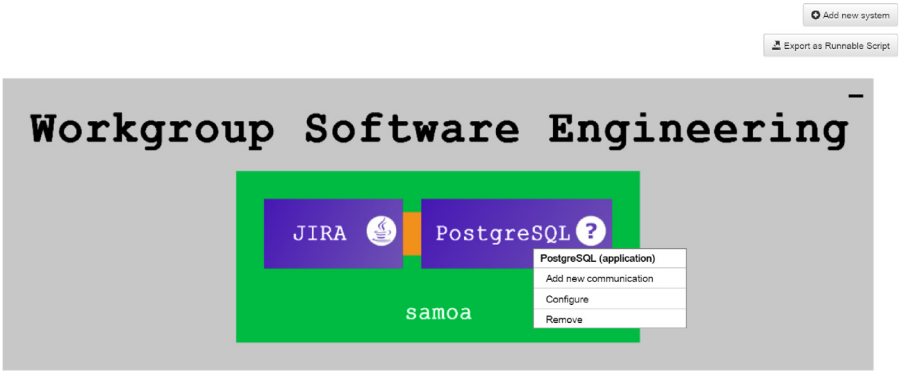


Fig. 6. Modeling the object landscape.

both group. Since the experience was self-assessed, we assume that random assignment succeeded.

4.2. Operation

4.2.1. Generating the input

We generated the input for the experiment by modeling the object landscape in ExplorViz by means of a modeling editor using our ExplorViz visualization language (see Fig. 6). Afterwards, we exported the model as a Ruby script file. This file contains one entry for each application that should be started. We have written a small configurable RPC application which acts as a server and connects to different servers configurable on the command line. This small application can pass off as the application names from the modeled landscapes which is also a part of one entry in the script. Therefore, the script imitates the real object landscape without having the need to instrument the productive applications. After executing the script and receiving the monitored data of the remote procedure calls, ExplorViz persists its created landscape model into a file which acts as a replay source during the experiment.

4.2.2. Tutorials

We provided automated tutorials for both groups of the experiment. This enhanced the validity of our experiments by eliminating human influences. For the tutorial system, we used a small-sized model of the Kiel Data Management Infrastructure for ocean science [3] to make the subjects familiar with the visualization. Both groups had the same explanation text for the tutorial except information about the abstractions in the hierarchical visualization which were only available to the associated group. As example, Fig. 7 displays the second tutorial step for the Hierarchical Group . The message dialog contains a description and an action which is to be completed by the subject. To help the user, the red arrow points at the designated entity which is to be interacted with. The tutorial was done right before the study.

4.2.3. Questionnaire

Both groups answered the questions on an electronic questionnaire. Fig. 8 shows the first question displayed in our electronic questionnaire. It starts with the task description and important words and phrases are emphasized. If there is more than one required answer, the dialog displays n input fields. In the example, these are the three expected answers. Underneath is an elapsed time counter which shows the maximum time and switches to a red color when the time is exceeded. The Next Button takes the user to the next question. Notably, we do not provide a Back Button since the recorded time for the question might overlap with the other recorded time of other questions.

An electronic version provides three advantages over using sheets of paper for us. First, time cheating by the subjects is impossible since the timings are automatically recorded. Second, we avoid a possible error-prone manual digitalization by direct electronic capture. Lastly, the participants are forced to input valid answers for category fields, e.g., their experience.

4.2.4. Pilot study

To check whether the material and questions are understandable for the target population, we conducted a pilot study with two master students as participants before the actual experiment. After this study, we improved the materials based on the feedback. In addition, we added hints to the tasks which were perceived as too difficult or which were misunderstood. While the hint for Task 3 might hinder the visual query in the Hierarchical Group , the hint for Task 5 does not favor any group.

4.2.5. Procedure

Our experiment took place at the Kiel University. Each participant had a single session of up to 45 minutes. All subjects used the same computer which had a display resolution of 1920 $\times$ 1200. Before the experiment took place, we benchmarked the computer to ensure that both types of visualization run smoothly. We run the study one student at a time.

At the beginning of each session, each subject received a sheet of paper containing a short introduction to the object landscape and a description of selected applications which might be unknown. We gave the subjects sufficient time for reading before they could access the computer. After telling the participants that they can ask questions at all times, a tutorial for the respective visualization type was started. Subsequently, the questionnaire part was started with personal questions and experiences. Afterwards, the system comprehension tasks begun. The session ended with the debriefing questions.

The less complex tasks (T1, T2, T3, T4) had a time allotment of 5 minutes. The more complex task T5 had an allotment of 10 minutes. The elapsed time was displayed beneath the task description during each task. The subjects were instructed to adhere to this timing. However, if they reached overtime, the timer was only highlighted in red and they were not forced to end the task.

4.3. Data collection

4.3.1. Timing and tracking information

The timing information for each task is automatically determined by our electronic questionnaire. In addition, the computer screen of every session is captured using a screen capture tool. With the screen recordings, we could analyze the behavior of each participant. Furthermore, it enabled us to look for exceptional

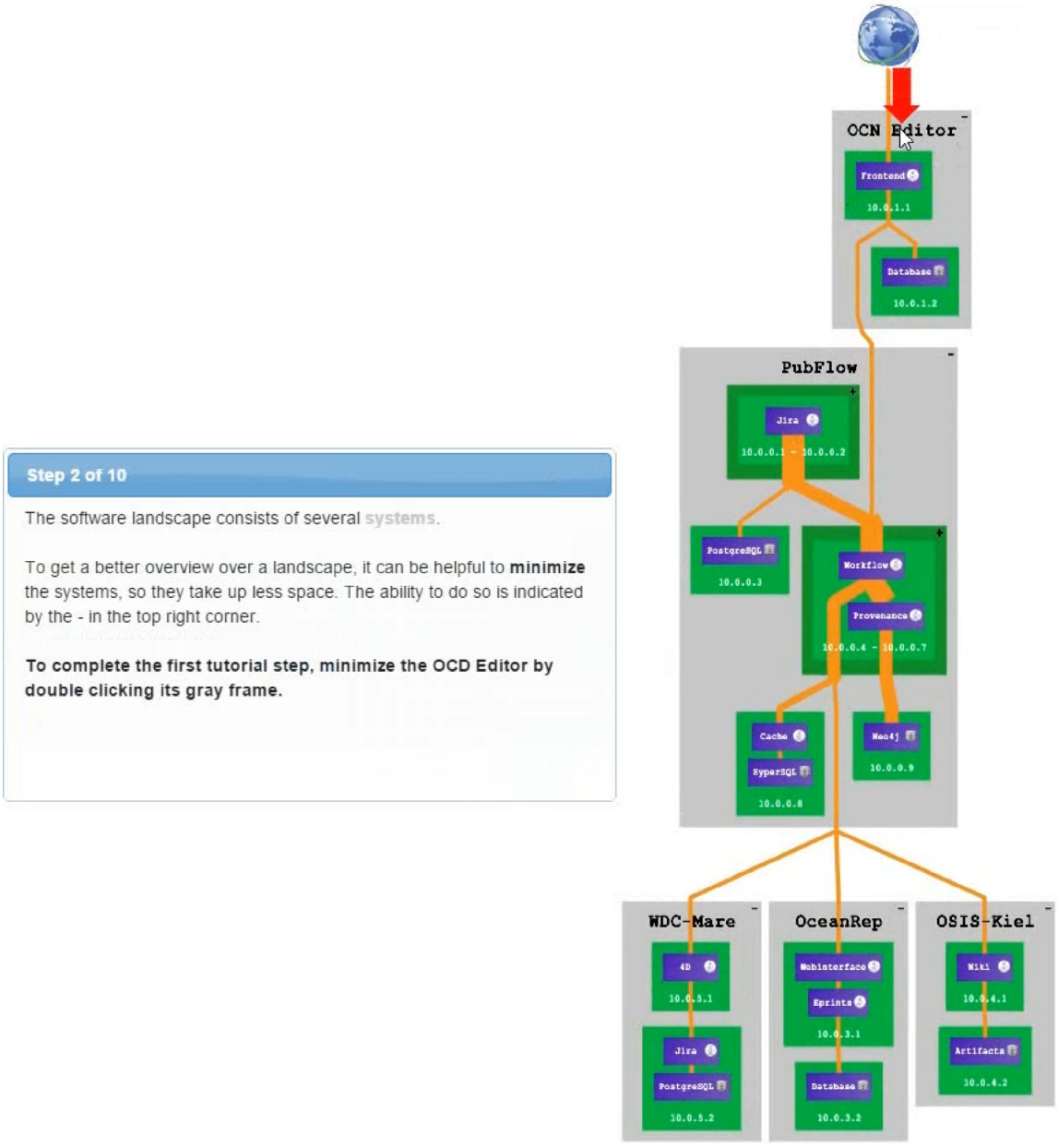


Fig. 7. Second tutorial step of the Hierarchical Group.

cases, for instance, tool-related problems encountered by the participants such as accidentally closing the web browser. The recordings become important in the case of tool-related problems since the timing data must manually be corrected and it must be reconstructed how long the subject actually worked on the task. The recordings are provided in the replication package [23], such that other researchers may re-check these manual corrections.

4.3.2. Correctness information

The open question format implies to conduct a blind review for rating the given answers. The two reviewers first agreed upon sample solutions and a maximum score for each task. A script randomized the order of the solutions so that no association between the answers and the originating group could be drawn. Then, both reviewers evaluated all solutions independently. Afterwards, any discrepancies in the ratings were discussed and resolved.

4.3.3. Qualitative feedback

The participants were asked to give suggestions to improve the visualization they used for solving the tasks. Due to space constraints, we only list the Top 3 for each group.

In the Flat Group , five users noted that some labels representing the request count overlapped such that they were forced to get the count by hovering over the communication line. Two users suggested to implement a sortable table for Task T2. Furthermore, two subjects disliked that the font size is not increasing when zooming out.

In the Hierarchical Group , three subjects suggested to use animations for opening and closing the systems or node groups. Two users would like to be able to highlight nodes or connections. As in the flat visualization group, one subject disliked that the font size is not increasing when zooming out.

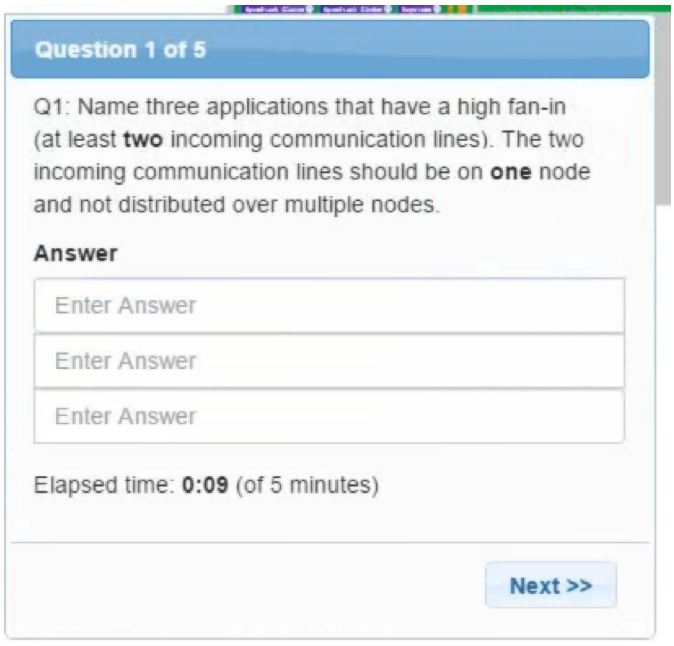


Fig. 8. First question displayed in our electronic questionnaire.

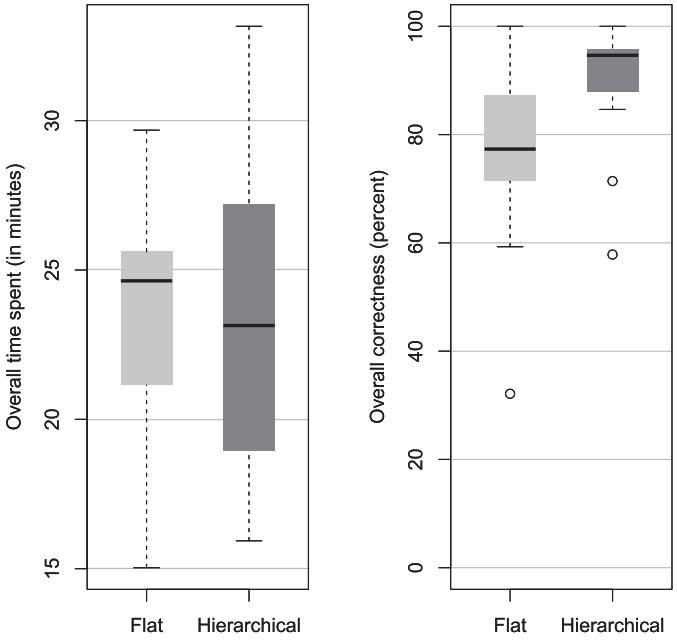


Fig. 9. Overall time spent and correctness for our experiment.

Table 2
Descriptive statistics of the results related to time spent (in minutes) and correctness (in points).

	Time spent		Correctness	
	Flat	Hierarchical	Flat	Hierarchical
Mean	23.49	23.45	17.07	19.5
Difference		−0.17%		+14.24%
Effect size d		0.0093		0.7827
sd	3.87	5.29	3.27	2.93
min	15.03	15.93	9	11
Median	24.64	23.14	17.25	20.5
max	29.68	33.16	22	22
Shapiro–Wilk W	0.9232	0.9605	0.9156	0.7933
Levene F		2.1048		1.2307
Student's *t*-test				
df		27		27
t		0.0251		−2.4102
p-value		0.9802		0.02303

4.4. Analysis and results

Descriptive statistics for the results of our experiment are shown in Table 2. Although we had three outliers, we did not remove any subjects from our analysis since the errors were comprehensible and did not result from exceptional cases. We use the two-tailed Student's *t*-test for our analysis which assumes normal distribution and depends on equal or unequal variances. To test for normal distribution, we use the Shapiro–Wilk test [33] which is considered more powerful [34] than, for instance, the Kolmogorov–Smirnov test [35]. We conduct a Levene test [36] to check for equal or unequal variances. The effect size is a quantitative measure of the strength of a phenomenon. For each type of effect-size, a larger absolute value always indicates a stronger effect. In our evaluation, the effect size for correctness is an order of magnitude higher than the effect size for the time spent.

We used the 64-bit R package in version 3.1.3.[10] for the analysis. As supplementary packages, we utilize `gplots` and `lawstat` for drawing bar plots and for importing Levene's test functionality, respectively. Furthermore, we chose $\alpha = 0.05$ to check for significance. The raw data, R scripts, and results are available as part of our experimental data package [23].

RQ1 (Time spent)

We start by checking the null hypothesis $H1_0$ which states that there is no difference between the flat and the hierarchical visualization in respect to the time spent on the system comprehension tasks. The box plot for the time spent is displayed on the left side of Fig. 9. Table 2 shows the differences between the mean values of Flat Group and Hierarchical Group .

The Shapiro–Wilk test for normal distribution in each group succeeds and hence we assume normal distribution of our data in each group. The Levene test also succeeds and thus we assume equal variances between the Flat Group and the Hierarchical Group. The Student's *t*-test reveals a probability value of 0.98 which is above our chosen significance level of 0.05. Therefore, we fail to reject the null hypothesis $H1_0$.

RQ2 (Correctness)

Next, we check the null hypothesis $H2_0$ which states that there is no difference between the two groups in respect to correctness of the solutions. A box plot for the overall correctness in each group is shown on the right side of Fig. 9.

The Shapiro–Wilk test for the Flat Group succeeds and hence we assume normal distribution in this group. The test fails for the Hierarchical Group . Therefore, we plotted a histogram and looked at the actual distribution. Most points are near 100% and the rest follows a normal distribution to the left side. Since 100% imposes a natural cutoff for the task correctness and the rest of the values are normal distributed, we also assume normal distribution for this group. The Levene test succeeds and thus we assume equal variances between both groups.

The Student's *t*-test reveals a probability value of 0.02 which is below our chosen significance level of 0.05. As a result, we reject $H2_0$ in favor of the alternative hypothesis H2 (*t*-test $t = -2.4102$, $df = 27$, $p = 0.02303$).

[10] http://www.r-project.org.

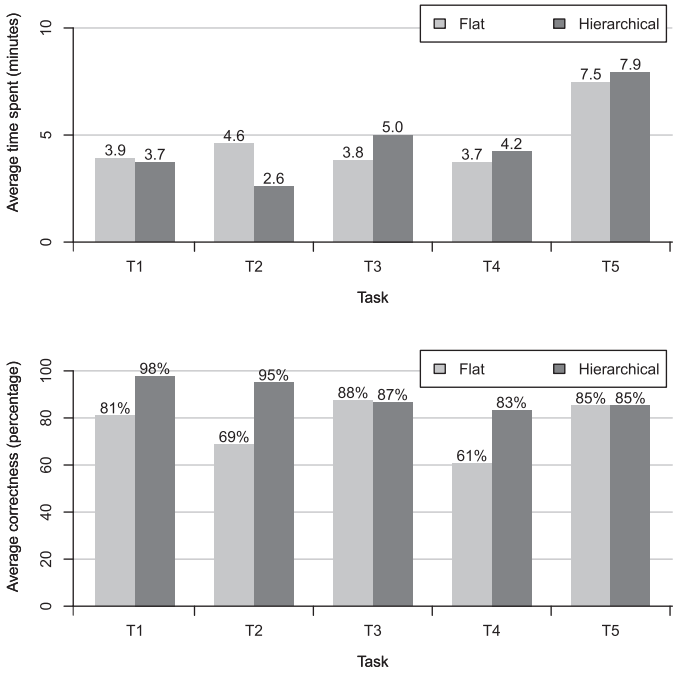


Fig. 10. Average time spent per task and average correctness per task.

4.5. Task-based analysis

The results for the time spent are not statistically significant. Hence, there is no statistical evidence for a difference in the time spent meaning it could be equal or even be different. However, it is likely that the impact of using a flat or hierarchical visualization is negligible in terms of time spent. Whether one group took a few seconds less, usually not of interest. In terms of task correctness, the Hierarchical Group outperformed the Flat Group by 14%. This difference is statistically significant in our experiment.

Since the time spent is negligibly different or equal, and the correctness of the solutions are higher in the hierarchical visualization, we conclude that using the hierarchical visualization provides more benefits than the flat visualization.

To investigate the reasons for this circumstance, we conducted an in-depth analysis of the recorded user sessions and looked for the employed strategies and typical sources of error. In the following, these findings are described (Fig. 10).

T1 (Identification of critical dependencies)

Both groups used the same strategy to find the three applications with a high fan-in. At first, the subjects got a general idea of the software landscape looking at its coarse-grained structure. Then, they zoomed in such that they can read the application labels and moved the view until they discovered the wanted applications. Some of the participants began their search from one side (left or right) such that they only needed to go over the landscape once. Others started at a random position and therefore, had to go over the landscape twice if they did not find the wanted applications.

A source of error in this task was the distinction between applications and nodes. We observed this confusion more in the Flat Group than in the Hierarchical Group which could be a reason for the 17% higher task correctness in the hierarchical visualization group. Since the hierarchical visualization group uses more hierarchies, the participants in this group might be more aware of the differences between each abstraction level.

A further possible reason for the higher task correctness might be that the hierarchical landscape visualization is more compact since node groups are closed and thus take less space.

T2 (Potential bottleneck detection)

Since the presentation of request labels are different in each experiment group, each group used a different strategy.

Subjects in the Flat Group again started from one side of the landscape visualization searching for the label with the highest request count. Sometimes the labels overlapped and the participants hovered over the communication line to get the number as popup.

The Hierarchical Group zoomed out to get an overview where the thickest communication lines are located. They hovered over these lines to get the actual request count and to form the descending order. Interestingly, the subjects often only distinguished between small and larger lines (4 steps of line thickness were visualized). Therefore, they also looked at medium sized lines instead of only looking at the largest lines.

In respect to time spent and task correctness, the Hierarchical Group outperformed the Flat Group in both metrics. One reason for this circumstance might be that in the flat visualization the manual search for the highest request count can be error-prone in respect to finding and in respect to the descending order.

T3 (Scalability evaluation)

In this task, both groups used the same strategy to find duplicated applications at the beginning. Participants from both groups formed the visual query for applications that are named equally and run on different nodes. The Flat Group succeeded with this query since the visualization only contains nodes and applications and no closed node group entities. In contrast, the Hierarchical Group did not find such applications since the node groups are closed by default. Often they realized this circumstance after going through the whole landscape without finding any duplicate applications and then they looked for the node groups. Only a few subjects in the hierarchical visualization looked for the node group entity right from the start.

From our expectations, the Hierarchical Group should have outperformed the Flat Group . However, the opposite happened. While the task correctness is roughly equal, the time spent was larger due to the wrong visual query in the beginning. Therefore, the introduced node groups abstraction confused the subjects in this task. We attribute this to a first time use and properly this behavior changes in long-term usage.

A further reason for the good performance of the Flat Group originates from our layout which visually grouped nodes running duplicated applications instead of distributing the nodes over the whole landscape. Otherwise, the comparison of applications would have been much harder in this group.

T4 (Service analysis)

Both groups followed the same strategy for describing the purpose of the `WWWPrint` application. First, the subjects had to search the application. After finding it, they looked at the communication lines and the connected applications. Then, they reasoned about the purpose on the basis of the application names and their connections. Additionally, the introduction sheet provided hints about the meaning of certain terms, e.g., `LDAP`.

In average, the Hierarchical Group required 30 s more time for this task. Since the visualization of the `WWWPrint` node is similar – except communication lines –, we expected an equal timing for this task. Therefore, we also looked at the median which actually reveals an roughly equal time spent. The average is influenced by two outliers (User 5 and User 25 – both taking around 6 min.).

One source of error in this task was overlooking the connection to `LDAP` and thus not detected that `WWWPrint` requires authen-

tication. We observed this behavior more often in the Flat Group than in the Hierarchical Group which might be a reason for the higher task correctness.

T5 (Risk management)

Again, both groups had the same strategy. First, they searched for the LDAP application. Afterwards, they followed the communication lines backward to find the services which would fail when LDAP fails.

Similar to Task T4, we expected an equal or lower time spent in this task since the layout is more compact in the Hierarchical Group . However, the time spent is 25 seconds higher in average. In the median, it is actually 25 s *lower* than the time spent by the Flat Group , again influenced by User 25 who took about 16 min.

A typical source of error in this task was not describing the purpose of the potentially failing services. We did not observe any difference in the occurrence of this behavior between the two groups which possibly led to the similar task correctness.

Summary

In summary, we observed three issues leading to a higher time spent or lower task correctness in the Flat Group . The subjects mistook nodes for applications. This happened also in the Hierarchical Group but less frequently. Furthermore, when space became narrow, the request labels overlapped. This led to manually hovering over the connection to get the actual request count. The third issue is related to the layout that was inherently larger due to the absence of abstractions, i.e., especially a node group abstraction. Therefore, the Flat Group often required more time to find entities.

Subjects in the Hierarchical Group often did not utilize the node group abstraction efficiently right from the start. Therefore, this abstraction imposes a non-zero learning curve.

One general issue which affected both groups was that some participants mixed up the direction of the communication which goes from top to bottom in our layout. They sometimes thought it would go from bottom to top. This issue could probably be solved by an always visible legend.

4.6. Threats to validity

In this section, we discuss the threats to internal and external validity [37–39] that might have influenced our results.

4.6.1. Internal validity

We split the internal validity into three parts for our experiment: threats concerning the subjects, the tasks, and miscellaneous threats.

Subjects. The subjects might not have been sufficiently competent. Most participants rated themselves as having regular programming experience which should be sufficient for our task set.

A further threat is that the experience of the subjects might not have been fairly distributed across the Flat Group and the Hierarchical Group . This threat is mitigated by randomly assigning the participants to each group. We checked that the random assignment resulted in a fairly distributed self-assessed experience. The concrete numbers were already described in Section 4.1.5.

The subjects might not have been properly motivated which imposes another threat to validity of our experiment. The students were not forced to take part in the experimenter since in addition to the lottery, the students received only bonus points which are not required to pass the exam. Furthermore, while watching the recorded user sessions, we did not encounter any unmotivated user behavior.

Table 3
Debriefing questionnaire results for our experiment 1 is better – 5 is worse.

	Flat		Hierarchical	
	mean	stdev.	mean	stdev.
Time pressure (1–5)	2.14	0.77	2.20	0.94
Tool speed (1–5)	2.07	1.00	1.60	0.83
Tutorial helpfulness (1–5)	2.21	0.58	1.6	0.51
Tutorial length (1–5)	3.21	0.70	3.00	0.65
Achieved comprehension (1–5)	2.50	0.85	2.20	0.68
Perceived task difficulty (1–5)				
T1	2.36	0.84	2.20	0.77
T2	2.64	0.93	2.00	0.53
T3	2.64	0.63	3.00	0.76
T4	3.00	0.78	2.93	0.70
T5	2.93	0.73	3.00	0.53

Tasks. One task-related threat is that the solutions were incorrectly rated or a reviewer might have been biased towards one experiment group. We mitigated this threat by employing a blind review process. Before the actual reviewing process took place, the solutions were mixed by a script such that no trace to the originating group was possible for the reviewers. Then, two reviewers independently reviewed each solution. Afterwards, the seldom discrepancies in the ratings were discussed. These discrepancies were at most one point suggesting a high inter-rater reliability.

The tasks might have been too difficult which imposes another threat to validity. However, subjects from both groups achieved the maximum score in each task. The average perceived task difficulty is shown in Table 3. Since the average rating of each task is never difficult (4) or too difficult (5), we conclude that the difficulty of each task was appropriate.

Another threat is that the tasks might have been biased towards one type of visualization. Since the average perceived task difficulty only differs significantly in T2 and T3 between both groups, at least the other tasks are not biased towards one type of visualization. Task T2 was perceived easier in the Hierarchical Group and Task T3 was perceived harder in this group. Therefore, we conclude that this potential bias is fairly distributed between the two experiment groups.

Miscellaneous. The possible different quality of the tutorials impose another threat to validity. In both groups, the teams had the possibility to continue to use the tutorial until they felt confident in their understanding of the semantics. In addition, both groups had the same tutorial text except the hierarchical abstractions in the Hierarchical Group.

The too loose or strict time constraints might have influenced the results of our experiment. However, both groups had the same average perceived time pressure (Level 2). Therefore, we assume that the time pressure was well fitted for the tasks.

4.6.2. External validity

Our experiment only involved one single object landscape. Since this is usually not representative for all available software landscapes, further experiments with different object landscapes should be conducted.

Another threat concerns the system comprehension tasks, which might not reflect real tasks. Unfortunately, we did not find any task frameworks for composing system comprehension tasks for software landscapes. We also took a look at program comprehension task frameworks, e.g., the framework by Pacione et al. [40]. However, we could not adapt the tasks in a reasonable way. Therefore, we used our present knowledge about software landscapes to made up tasks in interesting contexts from real usage scenarios.

Only students participated in our experiment. Professionals might act differently which could result in a different outcome

of our experiment. To investigate the impact of this threat, further controlled experiments should be conducted. To lower the setup effort for such experiments, our experimental design can be reused.

5. Evaluation of application-level visualization in ExplorViz

For evaluating the application-level visualization of ExplorViz, we summarize the results of controlled experiments for comparing ExplorViz with the Extravis trace visualization approach in Section 5.1, for employing physical 3D-printed ExplorViz models in Section 5.2, and for exploring 3D ExplorViz models in virtual reality in Section 5.3.

5.1. Comparing ExplorViz with the Extravis trace visualization approach

We performed controlled experiments with 80 subjects to compare the trace visualization tools Extravis and ExplorViz for program comprehension tasks [41]. We replicated the first controlled experiment with a second one targeting a differently sized software system.

Cornelissen et al. [1,42] conducted a controlled experiment which provides quantitative, empirical evidence that the additional availability of a trace visualization tool (Extravis [43]) can provide benefits with respect to time and correctness over using sole static analysis in program comprehension tasks. In our experiments, we expanded upon the controlled experiment of Cornelissen et al. to compare different trace visualization techniques for program comprehension through controlled experiments. Specifically, we investigated whether Extravis provides the most efficient and effective solution to typical program comprehension tasks compared to other trace visualization tools.

For our comparison, we employ Extravis using circular bundling and a massive sequence view with the application-level visualization in ExplorViz, as introduced in Section 2.2. Extravis focuses on the visualization of one large execution trace. For this purpose, it utilizes two interactive, linked views: the circular bundle view and the massive sequence view.

Circular bundle view. The centered visualization of Extravis is the circular bundle view (❶ in Fig. 11). The classes are arranged at the inner circle. Due to the high number of classes in the analyzed software system PMD[11] (279 visualized classes), the names of the classes are only visible through tooltips on the respective entity. The outer circles represent the packages of PMD. In the inner field of the circle, the method calls between classes is represented by lines. The names of the method calls are visible by hovering over these lines. Extravis utilizes color coding for the direction of the visualized communication. In its default setting, green represents outgoing calls and red expresses incoming calls. The width of each line corresponds to the call frequency of the method. The application-level perspective visualization of PMD in ExplorViz was displayed in Fig. 2, Section 2.2.

Extravis follows a hierarchical, bottom-up strategy [44], i.e., all packages show their internal details at the beginning. It is possible to close packages and thus hide the contained classes to gain further insights into the global structure of the visualized software system. Furthermore, edge bundling provides hints about strong relationships between packages. The communication between two classes can be filtered by marking both classes. This selection highlights the method calls in the massive sequence view. In addition to displaying the communication direction, Extravis enables switching to a chronological trace analysis (❷) by changing the semantics of the line colors. In this mode, color is globally used for representing the occurrence in time of the method call in the trace. In its default setting, dark blue represents the oldest method call and yellow corresponds to the newest method call.

Massive sequence view. The massive sequence view (❸) visualizes the method calls over time similar to a compressed UML sequence diagram. On top, the classes and packages are displayed and their method calls are listed beneath. The direction of the communication is color coded as in the circular bundle view. The massive sequence view enables to filter the method calls according to a time window from point *A* in a trace to point *B* in a trace. This filtering restricts the massive sequence view and the circular bundle view to only contain method calls within the selected time window. A further feature of Extravis is a history of the previously selected time windows (❹).

Experiment results. Fig. 12, left-hand side, displays a box plot for the time spent in both experiments. Fig. 12, right-hand side, shows a box plot for the overall correctness in both experiments. The analysis of the results reveals a significant higher correctness for users of ExplorViz in both experiments. We concluded that the effect of using ExplorViz for solving typical program comprehension tasks leads to a significant increase in correctness and in less or similar time spent on the tasks in comparison to using Extravis. Although subjects spent similar time on program comprehension tasks with both tools for a small-sized system, analyzing a larger software system resulted in a significant efficiency advantage of 28% less time spent by using ExplorViz. Concerning the effectiveness (correct solutions for program comprehension tasks), we observed a significant improvement of correctness for both object system sizes of 39 and 61% with ExplorViz. For a detailed presentation and analysis of these experiments refer to [41].

Besides our own replication of the first experiment, we provide a package containing all our experimental data to facilitate the verifiability, reproducibility and further extensibility of our presented results [45]. It contains the employed version of ExplorViz (including source code and manual), input files, tutorial materials, questionnaires, R scripts, datasets of the raw data and results, and 80 screen recordings of the user sessions. We explicitly invite other researches to compare their trace visualizations with ExplorViz and we provide as complete material as possible to lower the effort for setting up similar experiments.

5.2. Employing physical 3D-printed ExplorViz models

Although 3D visualizations can deliver more information compared to 2D visualizations due to its additional dimension, it is often difficult for users to navigate in 3D spaces using a 2D screen and a 2D input device [46]. As a consequence, users may get disoriented [47] and thus the advantages of the third dimension may be abolished.

Traditional engineering disciplines overcome these issues by building solid, physical 3D models of their designs. Beneath resolving navigation issues, the physical models are used for better presentation, comprehension, and communication among stakeholders. We intend to transfer these advantages to software engineering by building physical models following the 3D software city metaphor through 3D printing.

Physical models take significant effort to be produced [48], but they provide a plethora of future research possibilities. We envision several potential usage scenarios for physical models which we will discuss in the following subsections.

[11] https://pmd.github.io/.

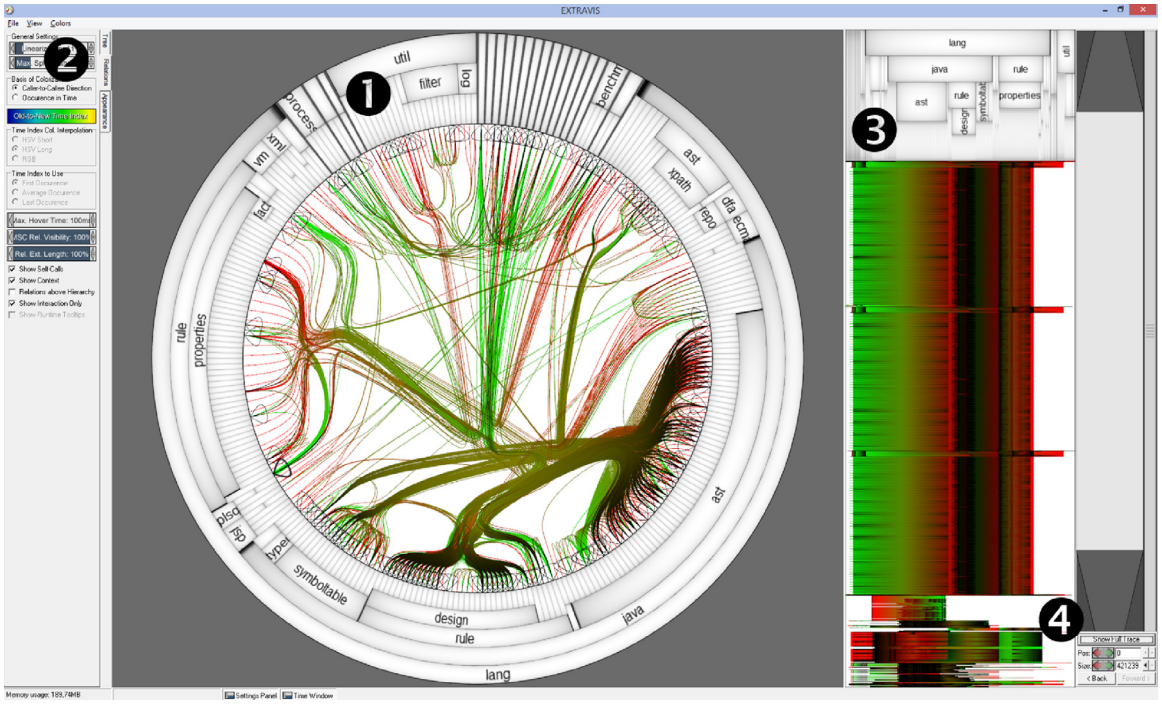


Fig. 11. The recorded execution trace of PMD for the first controlled experiment visualized in Extravis.

5.2.1. Program comprehension in teams

Gestures support in thinking and communication processes [49]. Since physical models are more accessible than 2D screens and provide a natural interaction possibility, they might increase the gesticulation of users. This might lead to faster and better understanding when applied in a team-based program comprehension scenario due to its supporting nature. Furthermore, the advantages might increase when applied in larger teams. Since software systems are often changing, the model should only be printed for special occasions, e.g., a new developer team or upcoming major refactorings.

We evaluated the impact of using physical 3D ExplorViz models on program comprehension in teams through a controlled experiment [50]. Fig. 13 displays our physical 3D-printed and manually painted ExplorViz model of PMD that we employed for the experiment. In our experiment, we compare the usage of the on-screen model of Fig. 2 (visualized on a plain 2D screen) to the usage of a physical model in Fig. 13 for program comprehension in small teams. To investigate the impact of using physical models in each task and the reasons for this impact, we conducted an in depth analysis of the camera and screen recordings of 112 subjects. Fig. 14, left-hand side, displays a box plot for the time spent, and

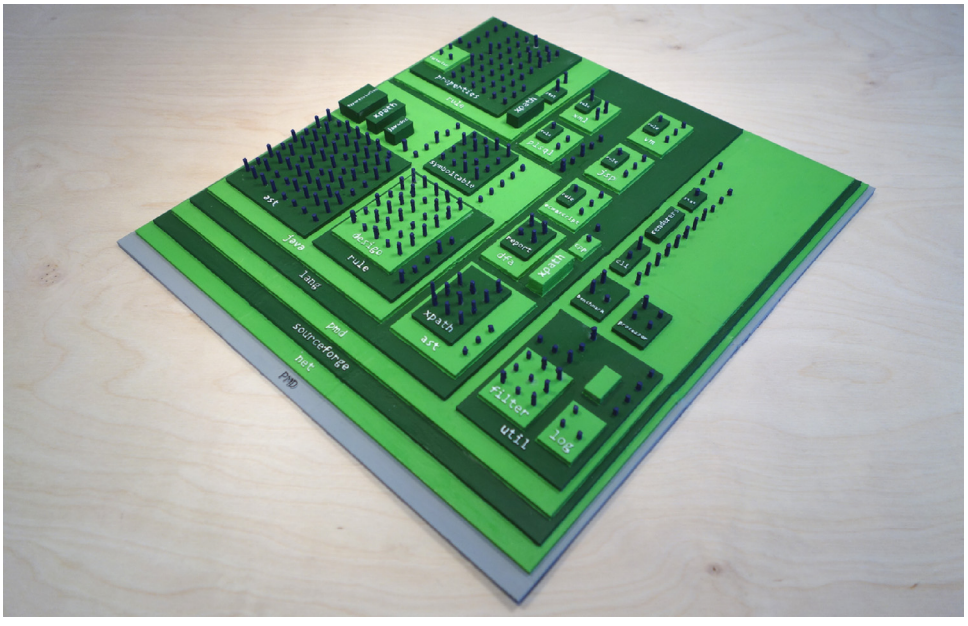


Fig. 13. Physical 3D-printed and manually painted ExplorViz model of PMD (334 mm wide and 354 mm deep).

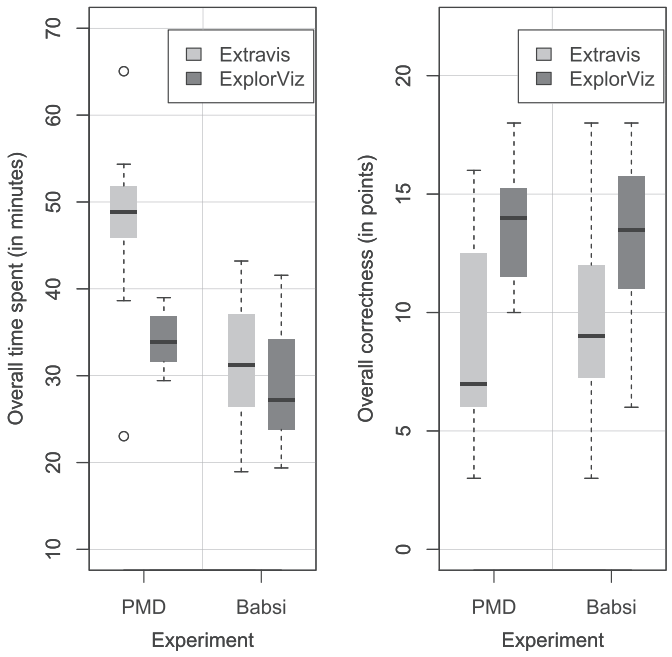


Fig. 12. Overall time spent and correctness for comparing ExplorViz with Extravis.

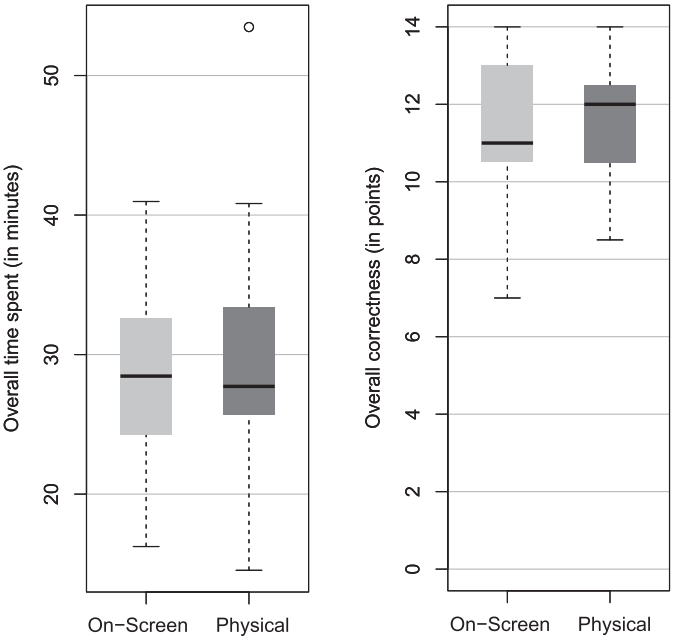


Fig. 14. Overall time spent and correctness for our experiment on using physical 3D ExplorViz models.

Fig. 14, right-hand side, shows a box plot for the overall correctness in the experiment.

To summarize the impact of using physical models in our task set: Two tasks (concept location and design understanding) were positively influenced by the physical model. In contrast, structural understanding was negatively influenced. The achieved correctness in metric-based analysis increased with the physical model but also the time spent increased, leading to no clear statement of the impact. We observed that the physical models improved the team-based program comprehension process for specific tasks by initiating gesture-based interaction. For further additional details about the experiment, we refer to [48,50].

To facilitate the verifiability and reproducibility of our results, we provide a package containing all our data including the raw data and 112 recordings of the participant sessions [51].

5.2.2. Educational visualization

A further usage scenario is the usage of 3D models for educational purposes. Like an anatomic skeleton model used in a biology course, 3D models of design patterns, architectural styles, or reference architectures could be 3D-printed. Advantages include the possibly increased interest of the students and due to a 3D visualization and the possibility to touch the model, there might be a higher chance to remain in memory. Further interaction possibilities, e.g., plugging mechanisms, with the 3D model could be developed to support the learning process of the students.

5.2.3. Effort visualization in customer dialog

A further potential field of application are dialogs with customers. Customers often see the GUI as the program since the actual program logic code is often invisible for them. Therefore, the – possible large – effort to add a feature or to refactor the code is also often invisible for them. Presenting a physical 3D model of the status quo and another physical 3D model of the desired state, might convince the customer of the effort of the required change. This could also be achieved with two on-screen software visualizations but a touchable and *solid* 3D model might provide higher conviction.

5.3. Exploring 3D ExplorViz models in virtual reality

Another solution candidate for the above-mentioned navigation issue in 3D spaces is Virtual Reality (VR) [52]. VR can employ the natural perception of spatial locality of users and thus provide advantages for 3D visualization [53,54]. In addition to stereoscopic [55,56] display, natural interaction beyond the 2D mouse provides advantages, e.g., creativity can be enhanced by walking around [57]. However, it relies on – sometimes expensive – extra equipment which has to be purchased and might not function in every environment.

We extended ExplorViz with a VR approach to explore ExplorViz visualizations following the 3D software city metaphor by using a head-mounted display and gesture-based interaction. As head-mounted display, we employed the *Oculus Rift*,[12] Development Kit Version 1. Fig. 15 shows the user's view. Head rotation leads to viewpoint rotation in the virtual space. Hence, users only need to rotate their head to view surrounding model elements instead of moving them to the center of a firm viewpoint.

For gesture recognition, we use the Microsoft *Kinect v2 for Windows*.[13] It contains a depth camera and can be used as body tracker. There are two basic concepts for designing gesture-based movement actions. The first concept is commonly used by control sticks in game controllers. A user performs a gesture and holds the position at a boundary. While she or he holds this position, the movement is conducted continuously into the implied direction. The second concept is a direct mapping between the hand movement and the movement actions in the model, similar to how a computer mouse works. In our prior tests, users familiarized with a direct mapping faster than with the first concept. Furthermore, participants working with the continuous movement sometimes tried to manipulate the model as if they would use a direct mapping approach. Thus, we discarded the first concept and designed our gestures with a direct mapping of hand to manipulation.

In an iterative process, we realized the following gestures for interaction with the ExplorViz visualization in VR. Fig. 16a shows

[12] http://www.oculus.com.
[13] http://www.microsoft.com/en-us/kinectforwindows.

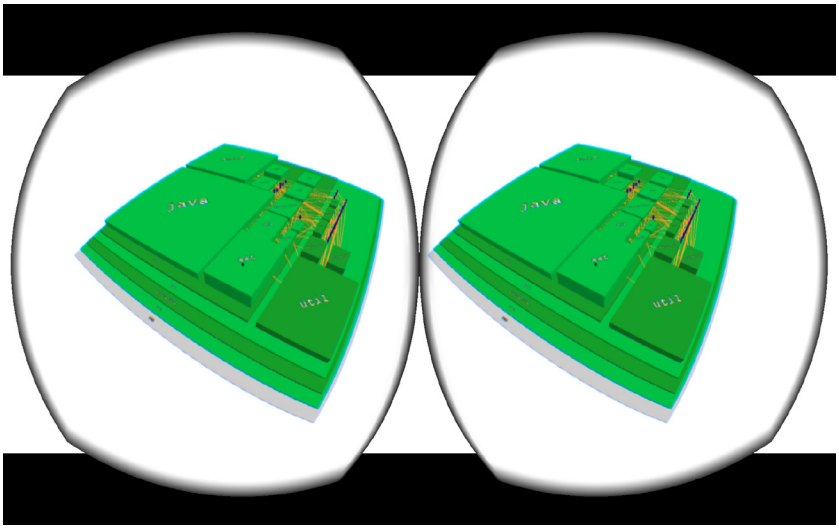
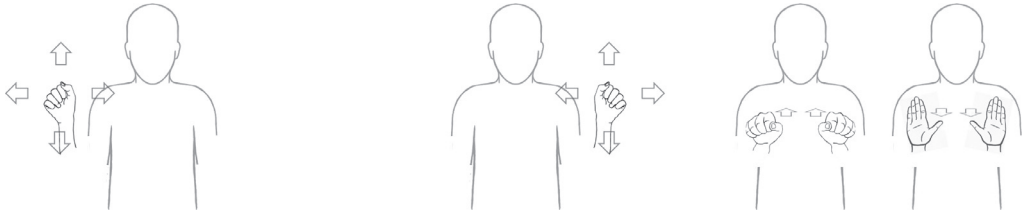


Fig. 15. View on the software city of PMD through the Oculus Rift.



(a) Gesture for moving the model

(b) Gesture for rotating the model

(c) Gesture for zooming in and out

Fig. 16. Gesture concepts for interacting with the software city model.

the gesture for moving the model. The user lifts the right hand, clenches into a fist, and then moves the object. Fig. 16b shows the design rotating the model. It is very similar to the translation gesture and only differs in using the left hand. Fig. 16c shows the design for zooming. The gesture is derived from real life interaction similar to rowing. The gesture maps to pulling and pushing the model towards or away from the viewer. To select an entity in the software city model, the user raises his right hand, and quickly closes and opens it.

As evaluation, we conducted structured interviews where eleven participants had to solve three program comprehension tasks and rate the usability of the used gestures and general VR experience for program comprehension. The participants of our interviews rated the developed gestures for translation, rotation, and selection as highly usable. However, our zooming gesture was less favored. In general, the subjects see potential for virtual reality in program comprehension. For further additional details about the experiment and the empirical evaluation, we refer to [52].

To facilitate the verifiability and reproducibility of our results, we provide a package containing all our evaluation data including source code, raw data, and eleven video recordings of the participant sessions [58].

6. Related work

In this section, we describe related work concerning landscape visualization, application visualization and experiments in the context of software visualizations.

6.1. Landscape-level visualizations

Web Services Navigator [59] provides 2D graph visualizations of the communication of web services. *Streamsight* [60] visualizes cooperating distributed components of streaming applications. In contrast to both tools, our approach aims at general software landscapes and introduces interactive explorable hierarchies to provide a higher visual scalability.

APM tools also often provide a visualization of a software landscape. Most APM tools do not provide different abstraction levels at the software landscape visualization. Furthermore, if the APM-tool allows detailed analysis of one application, the used visualization often is a tree-based viewer, which can hinder the analysis of traces with thousands of events.

Briand et al. [61] utilize UML sequence diagrams to visualize Remote Procedure Calls by adding a hostname to the object representation. Single method calls are shown in the diagrams. In contrast, we provide a single relation entity between communicating applications and therefore a higher visual scalability.

RanCorr [62] visualizes the dependencies between applications in a root cause analysis scenario. The root cause probability of each application is visualized by a color-coding. In contrast to them, our approach uses hierarchies to provide a higher visual scalability.

VisuSniff [63] shows the communication between servers. In contrast to our approach, they visualize every communication path on each port and protocol. Therefore, communicating servers often have several connected communication lines and thus the visualization does not scale to large software landscapes.

In relation to our hierarchical visualization, we did not find any tool providing the same hierarchy concepts as we use in our visualization.

6.2. Application-level visualizations

Knight and Munro [64] were the first to transfer the city metaphor to software visualization. In their visualization *Software World*, the buildings represent methods, and classes are mapped to districts.

Panas et al. [65] developed a city metaphor where classes are mapped to buildings and a city represents one package. They focused on providing a visualization looking as real as possible, e.g., with trees and photo-realistic textures.

Wettel and Lanza [66] created the tool *CodeCity* which also uses the city metaphor. In addition to Panas et al. [65], they use the width and depth of the buildings to represent, for instance, the amount of attributes of a class. Furthermore, the package hierarchy is represented by stacking districts similar to our ExplorViz visualization.

EvoSpaces, developed by Alam and Dugerdil [67], introduces a day and night mode utilizing the city metaphor. While the day mode provides information gathered from static analysis, the night mode displays dynamic information as connections between each building.

Imsovision [68] aims at representing object-oriented software in a virtual reality environment. The user is tracked with electromagnetic sensors attached to the shutter glasses and a *wand* which is used for the 3D navigation.

SkyscrapAR [69] is an augmented reality approach employing the city metaphor to visualize software evolution. The user can interact with a physical marker platform in an intuitive way while the actual visualization can be seen only on the monitor.

In contrast to the aforementioned approaches, ExplorViz enables the user to create physical models and to use virtual-reality equipment for intuitive navigation.

6.3. Experiments comparing to the state of the art

Marcus et al. [70] conducted an experiment comparing their sv3D tool to an IDE and a text file containing metrics values. The experiment resulted in sv3D not decreasing the task correctness. In respect to time, the sv3D group performed worse than the control group which – according to the authors – originates from using a new technology.

Quante [71] performed a controlled experiment for the evaluation of dynamic object process graphs. They failed to reject the null hypothesis that the availability of dynamic object process graphs support program comprehension in general.

Cornelissen [1] investigated the impact of using solely Eclipse to using Eclipse with additional access to the trace visualization Extravis for solving program comprehension tasks. The group with additional access to Extravis had a decrease in time spent and an increase in task correctness.

Wettel et al. [30] compare the usage of CodeCity to using Eclipse and Excel on two object systems (Azureus and Findbugs) in a controlled experiment. The CodeCity group achieved a statistically significant decrease in time completion and an increase in task correctness.

In contrast to the above mentioned experiments, our experiment operates on the software landscape level and not on the application level.

6.4. Experiments comparing software visualizations

Storey et al. [7] compared three software visualizations in an experiment. They present a detailed discussion of the tools' usage but provide no quantitative results.

Lange and Chaudron [72] investigated the benefits of their enriched UML views by comparing them to traditional UML diagrams. Contrary, we compare two landscape visualizations.

7. Conclusions and outlook

ExplorViz provides a hierarchical and multi-level visualization approach for solving system comprehension tasks for large software landscapes and individual software applications. To evaluate our hierarchical and multi-level visualization approach, we conducted four controlled experiments.

For evaluating our hierarchical landscape abstraction, we presented two different types of software landscape visualizations, i.e., a flat and a hierarchical one. The flat visualization was derived from concepts currently found in commercial APM tools. Our hierarchical landscape visualization extends this state-of-the-art visualization by abstractions, i.e., systems, node groups, and thickness of communication links representing the request count. We conducted a controlled experiment to investigate which visualization type supports solving system comprehension tasks more effectively or efficiently. Our experiment resulted in a statistically significant increase of 14% task correctness in the hierarchical visualization group for system comprehension tasks. The time used by the participants on the tasks did not differ significantly. To evaluate the benefits of a visualization, both metrics must be combined. Since the time spent is approximately equal and the task correctness is improved by our hierarchical visualization, it provides a valuable enhancement to the current state of the art in landscape visualizations in the context of system comprehension. During our analysis, we identified some challenges encountered by the participants in both visualizations types. Some subjects mistook nodes for applications. This happened more frequently in the Flat Group than in the Hierarchical Group . Furthermore, some participants from the Hierarchical Group did not utilize the node group abstraction efficiently right from the start. A further challenge was imposed by the flow-based layout. Some participants from both groups sometimes mixed up the direction of the communication.

The ExplorViz 3D application-level visualization of applications is empirically evaluated with a comparison to the Extravis approach, with physical models and in virtual reality. For the comparison with Extravis, we observed that the effect of using ExplorViz for solving program comprehension tasks leads to a significant increase in correctness and in less or similar time spent on the tasks in comparison to using Extravis. We observed that the physical models improved the team-based program comprehension process for specific tasks, but not for all tasks, by initiating gesture-based interaction. The participants of our virtual reality experiment with ExplorViz rated the realized gestures for translation, rotation, and selection as highly usable. However, our zooming gesture was less favored. In general, the subjects see potential for virtual reality in program comprehension.

The results backup our claim that our hierarchical and multi-level approach enhances the current state of the art in landscape and application visualization for better software system comprehension, including new forms of interaction with physical models and virtual reality.

To facilitate the verifiability and reproducibility for replications and further experiments [73], we provide packages containing all our experimental data for the four experiments [23,45,51,58].

The experiments gave us precious insights on how users actually perceive and interact with the visualization. Based on the results, we decided to enhance our visualization by an always visible legend and to enhance our tutorial with more introductions to the abstractions to flatten the learning curve.

As future work, our experiment could be replicated for increased external validity [74]. Especially, it should be conducted with professionals as test subjects. Since our experiments investigated first time use, the results might be different in long term usage. This should be addressed in further experiments.

For our visualization of software landscapes and applications, future work could investigate the usage of new visual metaphors to further enhance the visualization. New perspectives on employing physical models [50] and virtual reality [52] could be further explored. The resulting visualizations should be compared to each other and to our hierarchical software landscape visualization in controlled experiments where our experiment design can be reused to ensure lower setup costs of such experiments.

ExplorViz provides a component-based [18] 3D visualization of applications. Microservices-based [75] applications are structured into independently deployable, distributed self-contained smaller systems. Such architectures blur the boundaries of landscapes and applications. The new (commercial) monitoring tool Instana,[14] for instance, is specifically designed for monitoring such microservices-based systems. Instana provides a 3D visualization on the landscape level. It would be interesting to investigate how 3D visualizations could be beneficial on ExplorViz' landscape level, in combination with the 3D visualization on the application level.

References

[1] B. Cornelissen, A. Zaidman, A. van Deursen, B. van Rompaey, Trace visualization for program comprehension: a controlled experiment, in: Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC 2009), 2009, pp. 100–109.

[2] I. Hadar, O. Hazzan, On the contribution of UML diagrams to software system comprehension, J. Obj. Technol. 3 (1) (2004) 143–156.

[3] F. Fittkau, S. Roth, W. Hasselbring, ExplorViz: visual runtime behavior analysis of enterprise application landscapes, in: Proceedings of the 23rd European Conference on Information Systems (ECIS 2015), AIS, 2015, pp. 1–13.

[4] C. Tjortjis, N. Gold, P. Layzell, K. Bennett, From system comprehension to program comprehension, in: Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), IEEE, 2002, pp. 427–432.

[5] F. Fittkau, J. Waller, C. Wulf, W. Hasselbring, Live trace visualization for comprehending large software landscapes: the ExplorViz approach, in: Proceedings of the 1st International Working Conference on Software Visualization (VISSOFT 2013), 2013, pp. 1–4.

[6] R. Koschke, Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey, J. Softw. Maintenance. Evol. 15 (2) (2003) 87–109.

[7] M.-A. Storey, K. Wong, H.A. Müller, How do program understanding tools affect how programmers understand programs? in: Proceedings of the 4h Working Conference on Reverse Engineering (WCRE 1997), IEEE, 1997, pp. 12–21.

[8] V.R. Basili, The role of controlled experiments in software engineering research, in: Empirical Software Engineering Issues: Critical Assessment and Future Directions, Springer, 2007, pp. 33–37.

[9] F. Fittkau, A. Krause, W. Hasselbring, Hierarchical software landscape visualization for system comprehension: a controlled experiment, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015), IEEE, 2015, pp. 36–45.

[10] A. van Hoorn, M. Rohr, I.A. Gul, W. Hasselbring, An adaptation framework enabling resource-efficient operation of software systems, in: Proceedings of the Warm Up Workshop (WUP 2009) for ICSE 2010, ACM, 2009, pp. 37–40.

[11] F. Fittkau, W. Hasselbring, Elastic application-level monitoring for large software landscapes in the cloud, in: Service Oriented and Cloud Computing, Vol. 9306 of Lecture Notes in Computer Science, Springer-Verlag, 2015, pp. 80–94.

[12] F. Fittkau, A. van Hoorn, W. Hasselbring, Towards a dependability control center for large software landscapes, in: Proceedings of the 10th European Dependable Computing Conference (EDCC 2014), IEEE, 2014, pp. 58–61.

[13] P.C. Brauer, W. Hasselbring, PubFlow: A scientific data publication framework for marine science, in: Proceedings of the International Conference on Marine Data and Information Systems (IMDIS 2013), Vol. 54, Lucca, Italia, 2013, pp. 29–31.

[14] C.D. Schulze, M. Spönemann, R. von Hanxleden, Drawing layered graphs with port constraints, J. Visual Lang. Comput., Spec. Issue Diagram Aesthet. Layout 25 (2) (2014) 89–106.

[15] R. Wettel, Software systems as cities, University of Lugano, 2010 Ph.d. thesis.

[16] R. Brooks, Towards a theory of the comprehension of computer programs, Int. J. Man-Mach. Stud. 18 (6) (1983) 543–554.

[17] B. Shneiderman, The eyes have it: A task by data type taxonomy for information visualizations, in: Proceedings of the IEEE Symposium on Visual Languages, IEEE, 1996, pp. 336–343.

[18] W. Hasselbring, Component-based software engineering, in: Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, 2002, pp. 289–305.

[19] M. Boskovic, W. Hasselbring, Model driven performance measurement and assessment with MoDePeMART, in: Proc. Model Driven Engineering Languages and Systems, 12th International Conference (MODELS 2009), Vol. 5795 of Lecture Notes in Computer Science, Springer-Verlag, Denver, Colorado, USA, 2009, pp. 62–76.

[20] A. van Hoorn, J. Waller, W. Hasselbring, Kieker: A framework for application performance monitoring and dynamic software analysis, in: Proceedings of the 3rd Int. Conf. on Performance Engineering (ICPE 2012), ACM, 2012, pp. 247–248.

[21] H. Knoche, A. van Hoorn, W. Goerigk, W. Hasselbring, Automated source-level instrumentation for dynamic dependency analysis of COBOL systems, in: Proceedings of the 14. Workshop Software-Reengineering (WSR '12), Softwaretechnik-Trends 32(2), 2012, pp. 45–46.

[22] F. Fittkau, P. Stelzer, W. Hasselbring, Live visualization of large software landscapes for ensuring architecture conformance, in: Proceedings of the 2014 European Conference on Software Architecture Workshops (ECSAW 2014), ACM, 2014 28:1–28:4, doi:10.1145/2642803.2642831.

[23] F. Fittkau, A. Krause, W. Hasselbring, Experimental data for: Hierarchical software landscape visualization for system comprehension: A controlled experiment, 2015, 10.5281/zenodo.18853.

[24] V. Rajlich, G.S. Cowan, Towards standard for experiments in program comprehension, in: Proceedings of the 5th International Workshop on Program Comprehension (IWPC 1997), IEEE, 1997, pp. 160–161.

[25] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K.E. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE TSE 28 (8) (2002) 721–734.

[26] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005), IEEE, 2005, pp. 95–104.

[27] G.A.D. Lucca, M.D. Penta, Experimental settings in program comprehension: Challenges and open issues, in: Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC 2006), IEEE, 2006, pp. 229–234.

[28] M.D. Penta, R.E.K. Stirewalt, E. Kraemer, Designing your next empirical study on program comprehension, in: Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC 2007), 2007, pp. 281–285.

[29] M. Sensalire, P. Ogao, A. Telea, Evaluation of software visualization tools: Lessons learned, in: Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2009), 2009, pp. 19–26.

[30] R. Wettel, M. Lanza, R. Robbes, Software systems as cities: a controlled experiment, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), ACM, 2011, pp. 551–560.

[31] V.R. Basili, D.M. Weiss, A methodology for collecting valid software engineering data, IEEE TSE 10 (6) (1984) 728–738.

[32] R. Likert, A technique for the measurement of attitudes, Archives of Psychology 22 (140) (1932) 5–55.

[33] S.S. Shapiro, M.B. Wilk, An analysis of variance test for normality (complete samples), Biometrika (1965) 591–611.

[34] N. Razali, Y.B. Wah, Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests, J. Stat. Model. Anal. 2 (1) (2011) 21–33.

[35] E. Pearson, H. Hartley, Biometrika Tables for Statisticians, 2nd edition, Cambridge University Press, 1972.

[36] H. Levene, Robust tests for equality of variances, Contrib. Probab. Stat. 2 (1960) 278–292.

[37] W.R. Shadish, T.D. Cook, D.T. Campbell, Experimental and quasi-experimental designs for generalized causal inference, Wadsworth – Cengage Learning, 2002.

[38] N. Juristo, A.M. Moreno, Basics of software engineering experimentation, Springer, 2010.

[39] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer, 2012.

[40] M.J. Pacione, M. Roper, M. Wood, A novel software visualisation model to support software comprehension, in: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004), 2004, pp. 70–79.

[41] F. Fittkau, S. Finke, W. Hasselbring, J. Waller, Comparing trace visualizations for program comprehension through controlled experiments, in: Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015), IEEE, 2015, pp. 266–276.

[42] B. Cornelissen, A. Zaidman, A. van Deursen, A controlled experiment for program comprehension through trace visualization, IEEE TSE 37 (3) (2011) 341–355.

[43] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J.J.V. Wijk, A.V. Deursen, Understanding execution traces using massive sequence and circular bundle views, IEEE (2007) 49–58.

[44] B. Shneiderman, Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers, Inc., 1980.

[45] F. Fittkau, S. Finke, W. Hasselbring, J. Waller, Experimental data for: Comparing trace visualizations for program comprehension through controlled experiments, 2015, 10.5281/zenodo.11611.

[46] A. Teyseyre, M. Campo, An overview of 3D software visualization, IEEE Trans. Visual. Comput. Graphics 15 (1) (2009) 87–105.

[14] https://www.instana.com/.

[47] K.P. Herndon, A. van Dam, M. Gleicher, The challenges of 3D interaction: A CHI '94 Workshop, SIGCHI Bull 26 (4) (1994) 36–43.
[48] F. Fittkau, E. Koppenhagen, W. Hasselbring, Research perspective on supporting software engineering via physical 3D models, Tech. rep., Kiel University, 1507. (Jun. 2015). URL http://eprints.uni-kiel.de/28949/
[49] S. Goldin-Meadow, Hearing gesture: How our hands help us think, Harvard University Press, 2005.
[50] F. Fittkau, E. Koppenhagen, W. Hasselbring, Research perspective on supporting software engineering via physical 3D models, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015), IEEE, 2015, pp. 125–129.
[51] F. Fittkau, E. Koppenhagen, W. Hasselbring, Experimental data for: Research perspective on supporting software engineering via physical 3D models, 2015, 10.5281/zenodo.18378.
[52] F. Fittkau, A. Krause, W. Hasselbring, Exploring software cities in virtual reality, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015), IEEE, 2015, pp. 130–134.
[53] A. Elliott, B. Peiris, C. Parnin, Virtual reality in software engineering: affordances, applications, and challenges, in: Proc. of 37th Int Conf. on Software Engineering (ICSE 2015), IEEE, 2015, pp. 547–550.
[54] D. Delimarschi, G. Swartzendruber, H. Kagdi, Enabling integrated development environments with natural user interface interactions, in: Proceedings of the 22nd International Conference on Program Comprehension (ICPC 2014), ACM, 2014, pp. 126–129.
[55] C. Ware, K. Arthur, K.S. Booth, Fish tank virtual reality, in: Proceedings of the INTERACT 1993 and Conference on Human Factors in Computing Systems (CHI 1993), ACM, 1993, pp. 37–42.
[56] C. Ware, P. Mitchell, Reevaluating stereo and motion cues for visualizing graphs in three dimensions, in: Proceedings of the 2nd Symposium on Applied Perception in Graphics and Visualization (APGV 2005), ACM, 2005, pp. 51–58.
[57] M. Oppezzo, D.L. Schwartz, Give your ideas some legs: the positive effect of walking on creative thinking, J. Exp. Psychol. Learn., Mem., Cognit. 40 (4) (2014) 1142–1152.
[58] F. Fittkau, A. Krause, W. Hasselbring, Experimental data for: Exploring software cities in virtual reality, 2015, 10.5281/zenodo.23168.
[59] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, J.F. Morar, Web services navigator: Visualizing the execution of web services, IBM Syst. J. 44 (4) (2005) 821–845.
[60] W. De Pauw, H. Andrade, L. Amini, Streamsight: A visualization tool for large-scale streaming applications, in: Proceedings of the 4th ACM Symposium on Software Visualization (SoftVis 2008), ACM, 2008, pp. 125–134.
[61] L.C. Briand, Y. Labiche, J. Leduc, Toward the reverse engineering of UML sequence diagrams for distributed Java software, IEEE Transaction on Software Engineering 32 (9) (2006) 642–663.
[62] N.S. Marwede, M. Rohr, A. van Hoorn, W. Hasselbring, Automatic failure diagnosis in distributed large-scale software systems based on timing behavior anomaly correlation, in: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), IEEE, 2009, pp. 47–57.
[63] R. Oechsle, O. Gronz, M. Schler, VisuSniff: A tool for the visualization of network traffic, in: Proceedings of the Second Program Visualization Workshop, ACM, 2002, pp. 118–124.
[64] C. Knight, M. Munro, Virtual but visible software, in: Proceedings of the IEEE International Conference on Information Visualization (IV 2000), IEEE, 2000, pp. 198–205.
[65] T. Panas, R. Berrigan, J. Grundy, A 3D metaphor for software production visualization, in: Proceedings of the 7th International Conference on Information Visualization (IV 2003), IEEE Comput. Soc., 2003, pp. 314–320.
[66] R. Wettel, M. Lanza, Visualizing software systems as cities, in: Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007), IEEE, 2007, pp. 92–99.
[67] S. Alam, P. Dugerdil, Evospaces visualization tool: Exploring software architecture in 3D, in: Proceedings of the 14th Working Conference on Reverse Engineering (WCRE 2007), 2007, pp. 269–270.
[68] J.I. Maletic, J. Leigh, A. Marcus, G. Dunlap, Visualizing object-oriented software in virtual reality, in: Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001), Society Press, 2001, pp. 26–35.
[69] R. Souza, B. Silva, T. Mendes, M. Mendonca, SkyscrapAR: an augmented reality visualization for software evolution, in: Proceedings of the II Brazilian Workshop on Software Visualization (WBVS 2012), 2012, pp. 17–24.
[70] A. Marcus, D. Comorski, A. Sergeyev, Supporting the evolution of a software visualization tool through usability studies, in: Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005), 2005, pp. 307–316.
[71] J. Quante, Do dynamic object process graphs support program understanding? – a controlled experiment., in: Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC 2008), 2008, pp. 73–82.
[72] C. Lange, M.R.V. Chaudron, Interactive views to improve the comprehension of UML models – An experimental validation, in: Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC 2007), 2007, pp. 221–230.
[73] T. Crick, B.A. Hall, S. Ishtiaq, Can I implement your algorithm?: a model for reproducible research software, in: Proceedings of the 2nd Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2), arXiv, 2014, pp. 1–4.
[74] J. Siegmund, N. Siegmund, S. Apel, Views on internal and external validity in empirical software engineering, in: IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015), 2015, pp. 9–19.
[75] W. Hasselbring, Microservices for scalability: keynote talk abstract, in: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE 2016), ACM, New York, NY, USA, 2016, pp. 133–134, doi:10.1145/2851553.2858659.