Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment

Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring
Software Engineering Group, Kiel University, Kiel, Germany
Email: {ffi, akr, wha}@informatik.uni-kiel.de

***Abstract*—In many enterprises the number of deployed applications is constantly increasing. Those applications – often several hundreds – form large software landscapes. The comprehension of such landscapes is frequently impeded due to, for instance, architectural erosion, personnel turnover, or changing requirements. Therefore, an efficient and effective way to comprehend such software landscapes is required. The current state of the art often visualizes software landscapes via flat graph-based representations of nodes, applications, and their communication.**

In our ExplorViz visualization, we introduce hierarchical abstractions aiming at solving typical system comprehension tasks fast and accurately for large software landscapes. To evaluate our hierarchical approach, we conduct a controlled experiment comparing our hierarchical landscape visualization to a flat, state-of-the-art visualization. In addition, we thoroughly analyze the strategies employed by the participants and provide a package containing all our experimental data to facilitate the verifiability, reproducibility, and further extensibility of our results.

We observed a statistically significant increase of 14 % in task correctness of the hierarchical visualization group compared to the flat visualization group in our experiment. The time spent on the system comprehension tasks did not show any significant differences. The results backup our claim that our hierarchical concept enhances the current state of the art in landscape visualization.

I. Introduction

While program comprehension has been researched extensively, system comprehension has received much less attention [1]. 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 [2]. Therefore, system comprehension – in our terminology the comprehension of such landscapes – is a crucial part of the maintenance process [3].

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 [4], which provides live trace visualization for large software landscapes, introduces three hierarchical abstractions [2]. 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 lines.

While these abstractions seem reasonable, it should still be evaluated whether they provide any benefits concerning the comprehension process [5]–[7]. 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.

In this paper, we present a controlled experiment to compare a flat, state-of-the-art landscape visualization to our hierarchical visualization in the context of system comprehension. In contrast to former publications [2], [4], we perform a quantitative evaluation of our landscape perspective. Beneath evaluating if a hierarchical visualization provides benefits, we conducted this experiment to get input for improving our ExplorViz tool[1]. 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 technical 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 [8] containing all our experimental data including source code for both visualizations, raw data, and 29 recordings of the participant sessions.

In summary, our main contributions are:

1. the reusable design and execution of a controlled experiment comparing a flat landscape visualization to our hierarchical landscape visualization in typical system comprehension tasks, and
2. a thorough analysis of typical sources of error and the strategies chosen by the participants for each task.

The remainder of this paper is organized as follows. Section II describes the flat, state-of-the-art visualization of software landscapes used in our experiment. Then, Section III introduces our hierarchical landscape visualization. Afterwards, Section IV presents a controlled experiment to evaluate the impact of using a hierarchical visualization instead of a flat one for system comprehension. Related work is discussed in Section V. Finally, we draw the conclusions and illustrate future work in Section VI.

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

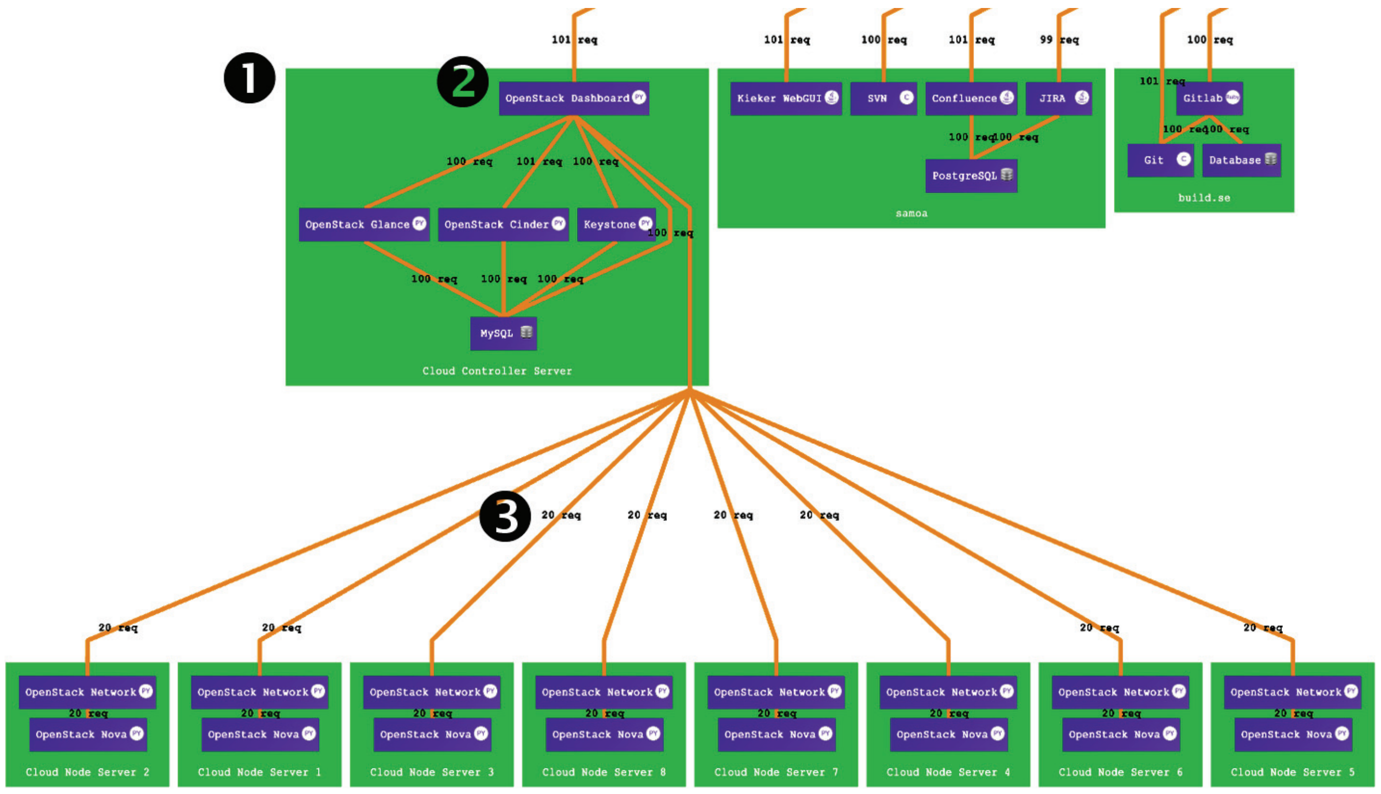

Fig. 1. An excerpt (29 of 140 applications) of the model of the technical IT infrastructure of the Kiel University in the flat landscape visualization

II. 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,[2] Foglight,[3] or Dynatrace.[4] 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 Figure 1 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[5] [9], 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.

III. Hierarchical Landscape Visualization

This section briefly presents our hierarchical software landscape visualization. The hierarchical visualization [2], [10] is part of our web-based ExplorViz tool [4] which enables live trace visualization in large software landscapes.

[2]http://www.appdynamics.com
[3]http://www.foglight.com
[4]http://www.dynatrace.com

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

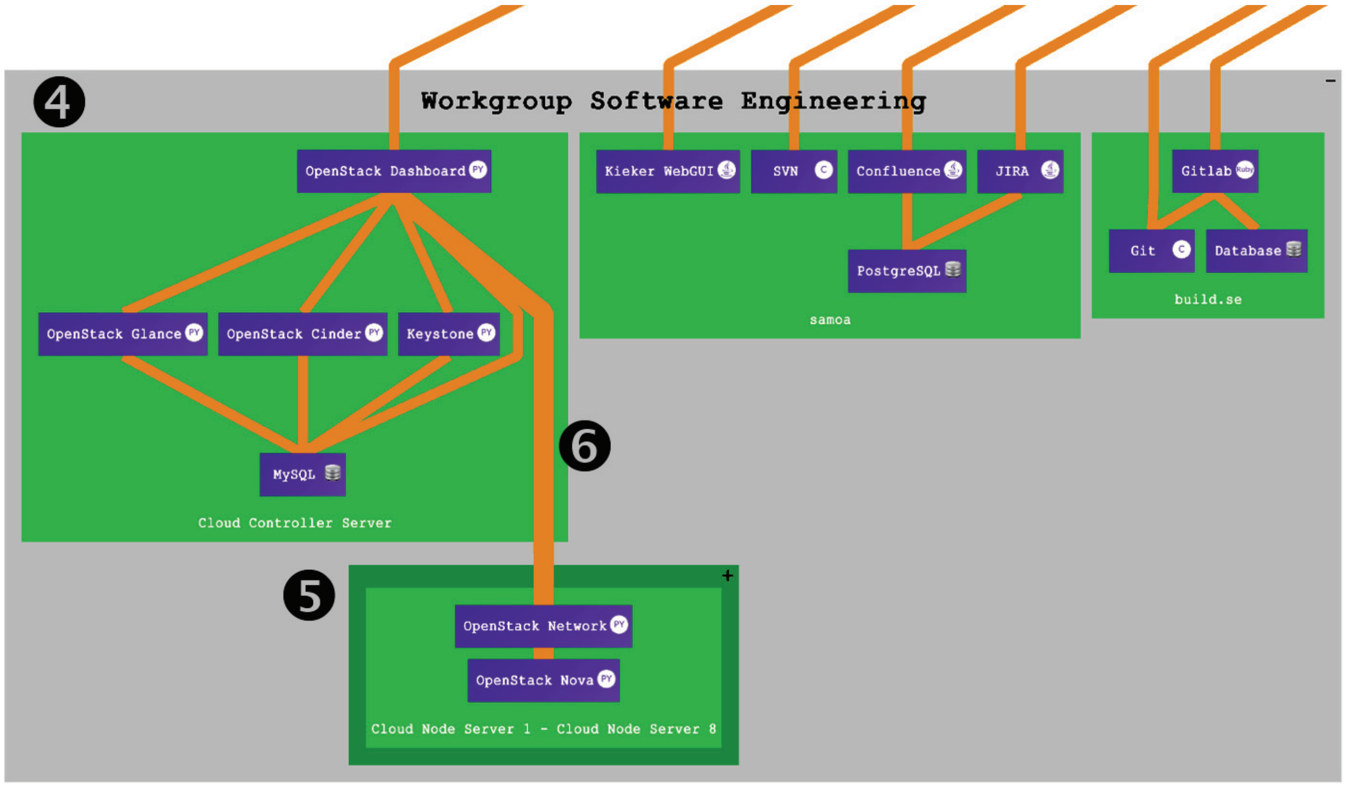

Fig. 2. An excerpt of the model of the technical IT infrastructure of the Kiel University in the hierarchical landscape visualization

Our hierarchical visualization is shown in Figure 2. 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.

IV. Controlled Experiment

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 typical system comprehension tasks. As object landscape we use a model of the technical IT infrastructure of the Kiel University and measure the *time spent* and *correctness* for each task which are typical metrics in the context of program comprehension [11]. 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.

A. Experimental Design

In addition to general software engineering experimentation guidelines [12]–[16], we follow the designs of Wettel et al. [17] and of Cornelissen et al. [18]. Similar to them, we use a *between-subjects* design. Thus, each subject solves tasks either using the flat or the hierarchical visualization. Following GQM [19], 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.

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* typical system comprehension tasks?
- **RQ2:** What is the ratio between Flat Group and Hierarchical Group in the *correctness of solutions* to typical system comprehension tasks?
- **RQ3:** Which typical 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 typical system comprehension tasks.
- **H2** The correctness of solutions to typical 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.

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 [11], [17], [18] and thus should also be applicable in the context of system comprehension.

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) Tasks: We selected a medium-sized software landscape (140 applications). Our model of the technical 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 actual tasks.

In Table I, 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.

5) Population: The 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 [20] 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 both group. Since the experience was self-assessed, we assume that random assignment succeeded.

B. Operation

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 visualization as DSL. Afterwards, we exported the model as a 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.

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 [2] 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.

3) Questionnaire: Both groups answered the questions on an electronic questionnaire. 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) 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.

TABLE I
DESCRIPTION OF THE SYSTEM COMPREHENSION TASKS FOR THE EXPERIMENT

ID	Description	Score
T1	*Context: Identification of Critical Dependencies* 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
T2	*Context: Potential Bottleneck Detection* Name the Top 3 communications with the highest request count in descending order. Write down the start application and the end application.	4
T3	*Context: Scalability Evaluation* 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
T4	*Context: Service Analysis* 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
T5	*Context: Risk Management* 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

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 1920×1200. Before the experiment took place, we benchmarked the computer to ensure that both types of visualization run smoothly.

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.

TABLE II
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

C. Data Collection

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 cases, for instance, technical problems encountered by the participant. The recordings become important in the case of technical problems since the timing data must manually be corrected and it must be reconstructed how long the subject actually worked on the task.

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.

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.

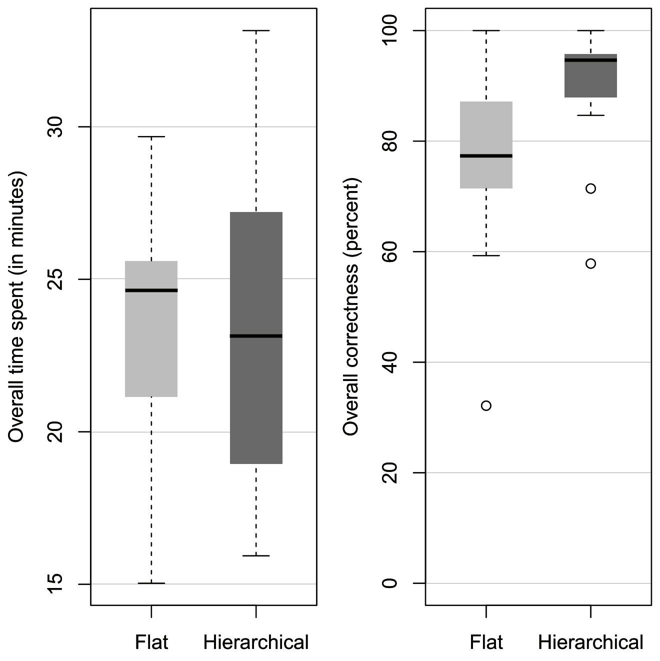

Fig. 3. Overall time spent and correctness for our experiment

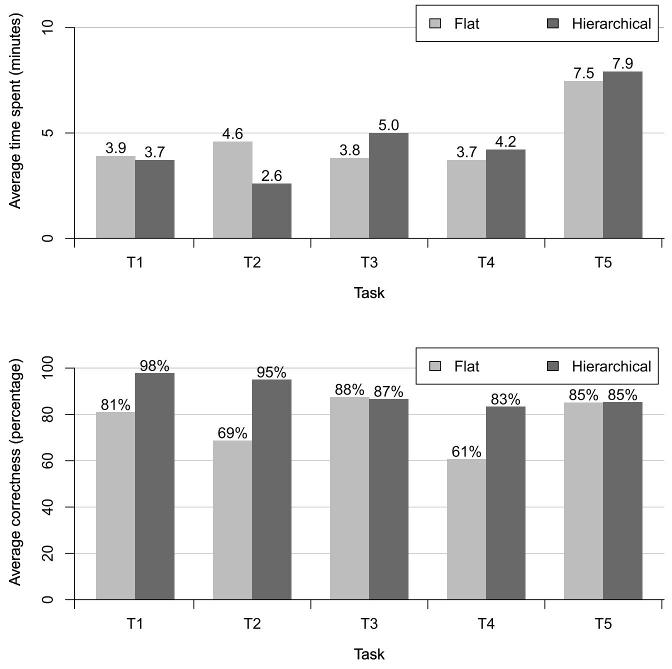

Fig. 4. Average time spent per task and average correctness per task

D. Analysis and Results

Descriptive statistics for the results of our experiment are shown in Table II. 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 [21] which is considered more powerful [22] than, for instance, the Kolmogorov-Smirnov test [23]. We conduct a Levene test [24] to check for equal or unequal variances.

We used the 64-bit R package in version 3.1.3.[6] 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 $\alpha = .05$ to check for significance. The raw data, R scripts, and results are available as part of our experimental data package [8].

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 Figure 3. Table II 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 Figure 3.

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).

E. Discussion

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, is typically out of interest. In terms of task correctness, the Hierarchical

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

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.

T1: 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: 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: 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: 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, e.g., `LDAP`.

In average, the Hierarchical Group required 30 seconds 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 six minutes).

One source of error in this task was overlooking the connection to `LDAP` and thus not detected that `WWWPrint` requires authentication. 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: 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 seconds *lower* than the time spent by the Flat Group, again influenced by User 25 who took about 16 minutes.

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.

TABLE III
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

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.

F. Threats to Validity

In this section, we discuss the threats to internal and external validity [25]–[27] that might have influenced our results.

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

a) 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 IV-A5.

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.

b) 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 III. 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.

c) 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, the average perceived time pressure was slightly above little (2) for both groups. Therefore, we assume that the time pressure was well fitted for the tasks.

2) External Validity: Our experiment only involved one single object landscape. Since this is typically 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. [28]. 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.

V. Related Work

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

A. Landscape Visualizations

Software landscape visualizations are mostly found in application performance management (APM) tools, for instance, AppDynamics, Foglight, or Dynatrace. Most of the APM tools provide their own visualization. However, they often resemble the familiar look of UML deployment diagrams.

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

B. Experiments Comparing to the State of the Art

Marcus et al. [29] 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 [30] 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 [18] 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. [17] 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.

C. Experiments Comparing Software Visualizations

Storey et al. [6] compared three software visualizations in an experiment. They present a detailed discussion of the tools' usage but provide no quantitative results.

Lange and Chaudron [31] investigated the benefits of their enriched UML views by comparing them to traditional UML diagrams. Contrary, we compare two landscape visualizations.

In [32], we present a controlled experiment comparing the execution trace visualizations Extravis using circular bundling and ExplorViz using a 3D city metaphor in typical program comprehension tasks. In contrast, we investigate the impact of using either a flat or a hierarchical software landscape visualization for system comprehension.

VI. Conclusions And Outlook

In this paper, we presented two different types of software landscape visualizations, i.e., a flat and our hierarchical one. The flat visualization was derived from concepts currently found in different, 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 lines representing the request count. We conducted a controlled experiment to investigate which visualization type supports solving typical 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. 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.

To facilitate the verifiability and reproducibility for replications and further experiments [33], we provide a package containing all our experimental data. Included are the employed version of ExplorViz v1.0-exp (including source code and manual), input files, tutorial materials, questionnaires, R scripts, dataset of the raw data and results, and 29 screen recordings of the participant sessions. The package is available online [8] with source code under the Apache 2.0 License and the data under a Creative Commons License (CC BY 3.0).

As future work, our experiment could be replicated for higher external validity. 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 a software landscape in general, future work could investigate the usage of existing visual metaphors, e.g., the city metaphor [4], or even new visual metaphors to further enhance the visualization. Afterwards, the resulting visualizations should be compared to each other and to our hierarchical software landscape visualization in controlled experiments where our design can be reused to ensure lower setup costs of such experiments.

The experiment gave us precious insights 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.

References

[1] I. Hadar and O. Hazzan, "On the contribution of UML diagrams to software system comprehension," *Journal of Object Technology*, vol. 3, no. 1, pp. 143–156, Jan. 2004.

[2] F. Fittkau, S. Roth, and W. Hasselbring, "ExplorViz: Visual runtime behavior analysis of enterprise application landscapes," in *Proceedings of the 23rd European Conference on Information Systems (ECIS 2015)*. AIS, May 2015.

[3] C. Tjortjis, N. Gold, P. Layzell, and 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.

[4] F. Fittkau, J. Waller, C. Wulf, and 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)*, Sep. 2013.

[5] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003.

[6] M.-A. Storey, K. Wong, and 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.

[7] 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.

[8] F. Fittkau, A. Krause, and W. Hasselbring, "Experimental data for: Hierarchical software landscape visualization for system comprehension: A controlled experiment," zenodo.org, doi: 10.5281/zenodo.18853.

[9] C. D. Schulze, M. Spönemann, and R. von Hanxleden, "Drawing layered graphs with port constraints," *Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout*, vol. 25, no. 2, pp. 89–106, 2014.

[10] F. Fittkau, P. Stelzer, and W. Hasselbring, "Live visualization of large software landscapes for ensuring architecture conformance," in *ECSAW 2nd International Workshop on Software Engineering for Systems-of-Systems 2014 (SESoS 2014)*. ACM, Aug. 2014.

[11] V. Rajlich and 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.

[12] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE TSE*, vol. 28, no. 8, pp. 721–734, 2002.

[13] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering," in *Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005)*. IEEE, 2005.

[14] G. A. Di Lucca and M. Di 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.

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

[16] M. Sensalire, P. Ogao, and 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)*, Sep. 2009, pp. 19–26.

[17] R. Wettel, M. Lanza, and 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.

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

[19] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE TSE*, vol. SE-10, no. 6, Nov. 1984.

[20] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, vol. 22, no. 140, pp. 5–55, 1932.

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

[22] N. Razali and Y. B. Wah, "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests," *Journal of Statistical Modeling and Analytics*, vol. 2, no. 1, pp. 21–33, 2011.

[23] E. Pearson and H. Hartley, *Biometrika Tables for Statisticians*, 2nd ed. Cambridge University Press, 1972.

[24] H. Levene, "Robust tests for equality of variances," *Contributions to probability and statistics: Essays in honor of Harold Hotelling*, vol. 2, pp. 278–292, 1960.

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

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

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

[28] M. J. Pacione, M. Roper, and M. Wood, "A novel software visualisation model to support software comprehension," in *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*, Nov. 2004.

[29] A. Marcus, D. Comorski, and 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)*, May 2005, pp. 307–316.

[30] 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)*, June 2008, pp. 73–82.

[31] C. Lange and 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)*, June 2007, pp. 221–230.

[32] F. Fittkau, S. Finke, W. Hasselbring, and 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, May 2015.

[33] T. Crick, B. A. Hall, and 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, Nov. 2014, pp. 1–4.