Comparing Trace Visualizations for Program Comprehension through Controlled Experiments

Florian Fittkau, Santje Finke, Wilhelm Hasselbring, and Jan Waller
Software Engineering Group, Kiel University, Kiel, Germany
Email: {ffi, sfi, wha, jwa}@informatik.uni-kiel.de

***Abstract*—For efficient and effective program comprehension, it is essential to provide software engineers with appropriate visualizations of the program's execution traces. Empirical studies, such as controlled experiments, are required to assess the effectiveness and efficiency of proposed visualization techniques.**

We present controlled experiments to compare the trace visualization tools EXTRAVIS and ExplorViz in typical program comprehension tasks. We replicate the first controlled experiment with a second one targeting a differently sized software system. In addition to a thorough analysis of the strategies chosen by the participants, we report on common challenges comparing trace visualization techniques. 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.

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 percent 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 percent with ExplorViz.

I. INTRODUCTION

Software maintenance tasks involve program comprehension of existing source code. Static analysis may be applied directly to such code. In contrast, dynamic analysis is applied to the data gathered during runtime of a program resulting in execution traces [1], [2]. This can provide a more accurate picture of a software system by exposing the system's actual behavior. It is essential to provide software engineers with appropriate visualizations of the program's execution traces to support efficient and effective program comprehension.

However, few software visualizations provide empirical evidence for their support in program comprehension [3]. Even less have empirical evidence comparing software visualization techniques [4]. Thus, empirically founded guidelines are often missing, e.g., for which task which visualization technique should be applied [5]–[8].

For these reasons, Cornelissen et al. [9], [10] conducted a controlled experiment which provides quantitative, empirical evidence that the additional availability of a trace visualization tool (EXTRAVIS [11]) can provide benefits with respect to time and correctness over using sole static analysis in typical program comprehension tasks. This kind of empirical evidence is more convincing than just speculation, anecdotes, and common wisdom [12], which provide no reliable sources of knowledge [13].

In this paper, we expand upon the controlled experiment of Cornelissen et al. to compare different trace visualization techniques for program comprehension through controlled experiments. Specifically, we investigate whether EXTRAVIS provides the most efficient and effective solution to typical program comprehension tasks compared to other trace visualization tools. The requirement to provide further comparisons to other trace visualization techniques is also stated by Cornelissen et al. [10]: "To characterize the difference [in performance of the subjects], there is a need for similar experiments involving other trace visualization techniques."

We design such a controlled experiment to compare two trace visualization techniques. For our comparison, we employ EXTRAVIS using circular bundling and a massive sequence view, and ExplorViz [14] – developed by the authors of this paper – using the city metaphor [15]. We conduct two experiments: A first experiment using PMD as the object system and a second one to replicate our results using a smaller-sized object system. To facilitate the verifiability and reproducibility of our results, we provide a package [16] containing all our experimental data including raw data and 80 screen recordings of user sessions using EXTRAVIS and ExplorViz. To the best of our knowledge, our experiments are the first experiments to compare different trace visualization techniques to each other.

In summary, our main contributions are:

1. the reusable design and execution of two controlled experiments comparing the trace visualization tools EXTRAVIS utilizing circular bundling and a massive sequence view and ExplorViz basing on the city metaphor in typical program comprehension tasks,
2. a thorough analysis of the strategies chosen by the participants for each task and visualization technique, and
3. identification of common challenges for controlled experiments comparing trace visualization techniques.

The remainder of this paper is organized as follows. We first present EXTRAVIS in Section II and ExplorViz in Section III. The following Section IV describes our two controlled experiments, including their design, operation, results, discussion, and threats to validity. Related work is discussed in Section V. Finally, we draw the conclusions and illustrate future work in Section VI.

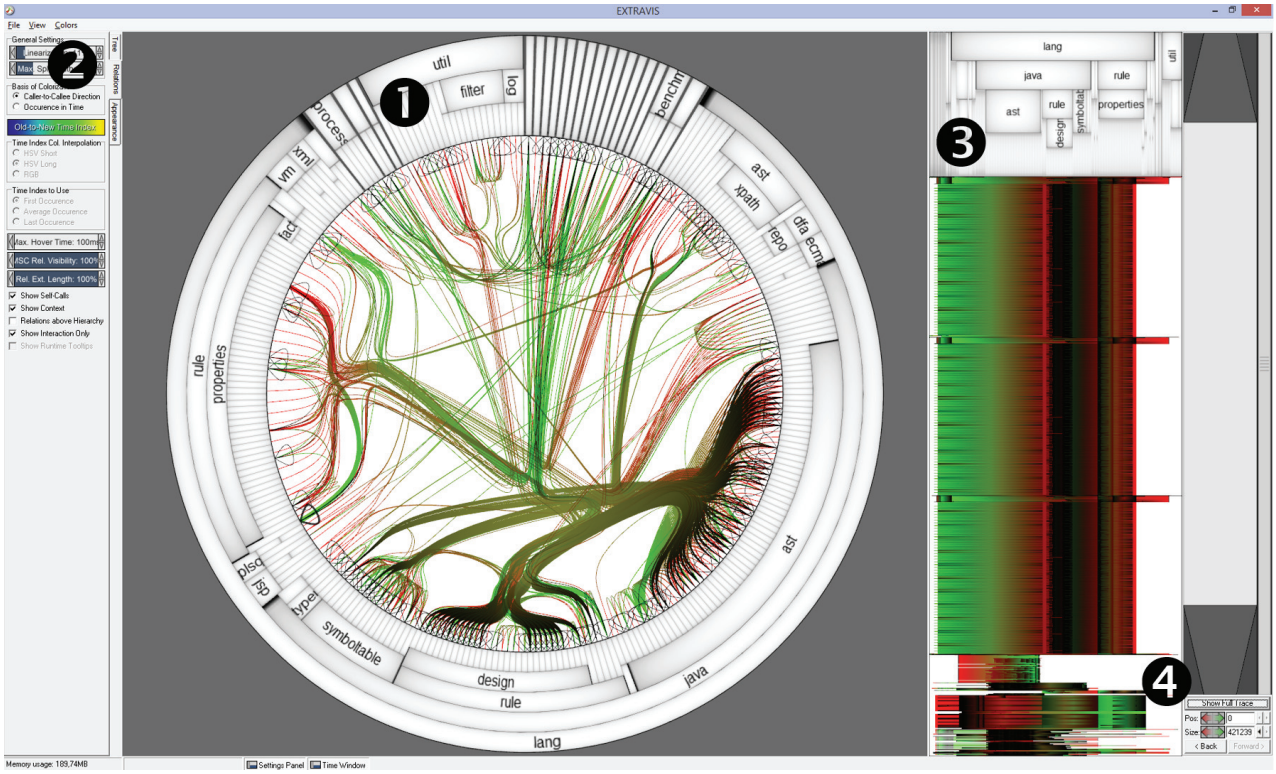

Fig. 1. The recorded execution trace of PMD for the first controlled experiment visualized in EXTRAVIS

II. EXTRAVIS IN A NUTSHELL

In this section, we present the features of EXTRAVIS[1] which forms the baseline of our controlled experiments. This lays the foundation for a later discussion of the chosen strategies of the subjects, which is presented in the corresponding subsection of the controlled experiments (Section IV-E). EXTRAVIS has been developed by Cornelissen et al. [11]. It 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. Those two views are described in the following subsections.

A. Circular Bundle View

The centered visualization of EXTRAVIS is the circular bundle view (❶ in Figure 1). The classes are arranged at the inner circle. Due to the high number of classes in the analyzed software system PMD[2] (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.

[1]http://swerl.tudelft.nl/extravis
[2]http://pmd.sourceforge.net

EXTRAVIS follows a hierarchical, bottom-up strategy [17], 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.

B. 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).

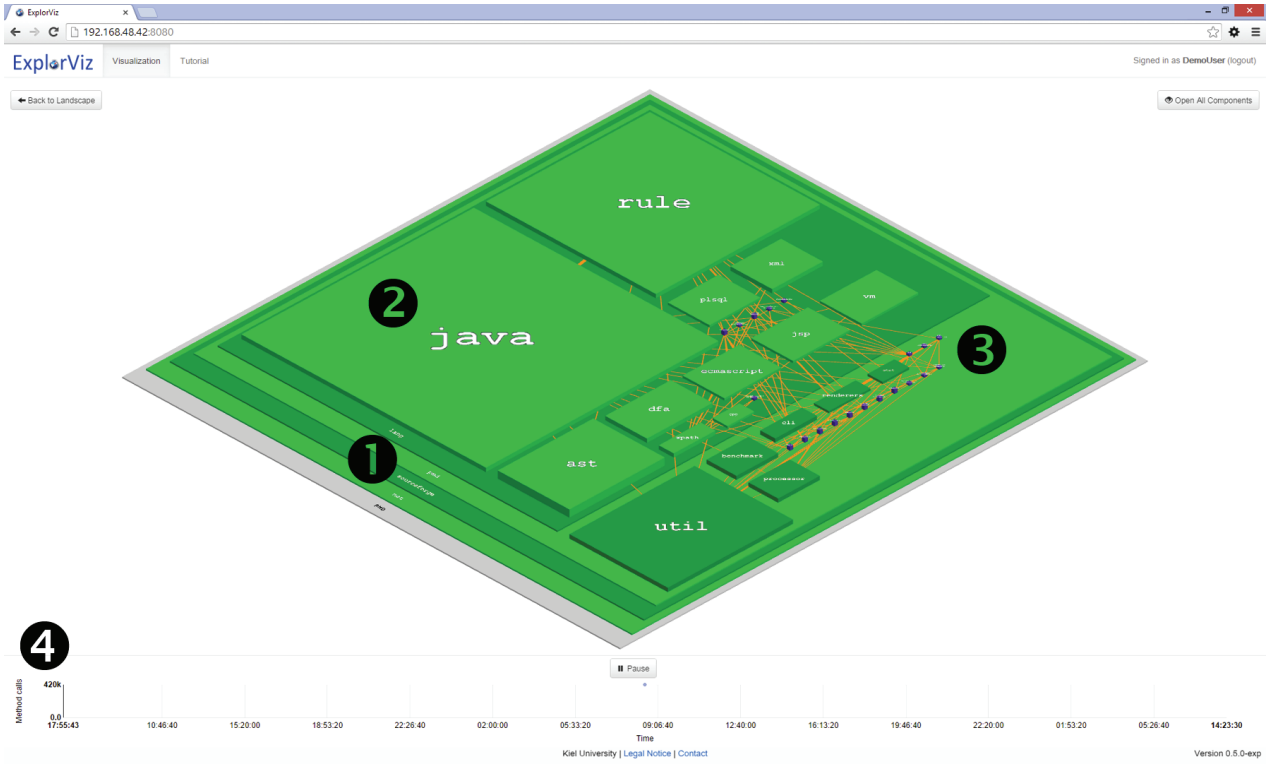

Fig. 2. The recorded execution trace of PMD for the first controlled experiment represented in ExplorViz

III. ExplorViz in a Nutshell

This section introduces our web-based ExplorViz[3] visualization. It has been developed for large software landscapes while providing details on the communication within an application on demand. Therefore, ExplorViz features a landscape-level and an application-level perspective. The experiments presented in Section IV solely use the application-level perspective since only one application is visualized and not a whole software landscape. Thus, we refer to [14] for details on the landscape-level perspective. The application-level perspective of ExplorViz addresses the same level of detail as addressed by Extravis. The city metaphor and the disabled features of ExplorViz for the comparative experiment are described in the following subsections.

A. City Metaphor

Figure 2 displays our application-level perspective. It visualizes the execution trace of PMD used in our first controlled experiment. The flat green boxes (❶) in our 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 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 [18].

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

ExplorViz follows a hierarchical, top-down approach [19]. Therefore, details about the classes and their communication are provided on demand following the Shneiderman mantra [20] of "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 and to obtain details through tooltips.

B. Disabled Features for the Experiments

Clicking on a communication line reveals a dialog where the user can choose which trace to analyze in detail. The trace analysis is conducted through a trace replayer. Here, the user can switch between an automatic play mode and stepping through the trace. We have disabled this feature for the experiments since there is no similar feature in Extravis.

As ExplorViz provides a periodically updated live visualization, a further main feature is time shifting (❹). It is designed for analyzing specific situations of interest, for instance, performance anomalies [21]. Again there is no directly comparable feature in Extravis. Therefore, we have disabled this feature for our experiments.

Source code viewing is considered important [3] especially during a program comprehension process. Therefore, ExplorViz provides the possibility to open a dialog that displays the source code for each class, if available. Similarly to the previous features, we have disabled it for the experiments due to its absence in Extravis.

IV. Controlled Experiments

In this paper we compare the impact of using either EXTRAVIS or ExplorViz for program comprehension. Therefore, we defined typical program comprehension tasks for our object system PMD. To validate our results, we replicated [22] our experiment by conducting a second controlled experiment using a different-sized object system named Babsi.[4] We measured the *time spent* and the *correctness* for each task. These measures are typically used in the context of program comprehension [23]. After the experiments, we analyzed the benefits of using EXTRAVIS with circular bundling or ExplorViz with the city metaphor on the defined tasks.

We describe both controlled experiments by their design, operation, data collection, analysis, results, discussion, and threats to validity. Afterwards, we share lessons learned and give advice on avoiding some observed challenges.

A. Experimental Design

In addition to general software engineering experimentation guidelines [24]–[28], we follow the experimental designs of Wettel et al. [29] and of Cornelissen et al. [9]. Similar to these experiments, we use a *between-subjects* design. Thus, each subject only solves tasks with either EXTRAVIS or ExplorViz and therefore, uses one tool only. Following the GQM approach [30], we define the goal of our experiments as quantifying the impact of using either EXTRAVIS or ExplorViz for program comprehension.

1) Research Questions & Hypotheses: We define three research questions (RQ) for our defined goal:

- **RQ1:** What is the ratio between EXTRAVIS and ExplorViz in the *time required for completing* typical program comprehension tasks?
- **RQ2:** What is the ratio between EXTRAVIS and ExplorViz in the *correctness of solutions* to typical program comprehension tasks?
- **RQ3:** Which typical sources of error exist when solving program comprehension tasks with EXTRAVIS or ExplorViz?

Accordingly, we formulate two null hypotheses:

- **H1$_0$:** There is no difference between EXTRAVIS and ExplorViz in the *time spent* for completing typical program comprehension tasks.
- **H2$_0$:** The *correctness* of solutions to typical program comprehension tasks does not differ between EXTRAVIS and ExplorViz.

We define the following alternative hypotheses:

- **H1** EXTRAVIS and ExplorViz require different times for completing typical program comprehension tasks.
- **H2** The correctness of solutions to typical program comprehension tasks differs between EXTRAVIS and ExplorViz.

For RQ3, we conduct an in-depth analysis of the results and analyze the recorded sessions of each subject in detail.

[4] http://babsi.sourceforge.net

2) Dependent and Independent Variables: The independent variable in both experiments is the employed tool used for the program comprehension tasks, i.e., EXTRAVIS or ExplorViz. We measured the accuracy (*correctness*) and response time (*time spent*) as dependent variables. These are usually investigated in the context of program comprehension [9], [23], [29].

3) Treatment: The control group used EXTRAVIS to solve the given program comprehension tasks. The experimental group solved the tasks with ExplorViz.

4) Tasks: For our task definitions, we stuck to the framework of Pacione et al. [31] which describes categories of typical program comprehension tasks. It focuses on dynamic analysis [29] providing a good match for our task definitions. In addition, Cornelissen et al. [9] used this framework.

For our first experiment, we selected a medium to large-sized object system and adhered to the tasks defined by Cornelissen et al. [9] as close as possible to prevent bias toward ExplorViz. Preliminary experiments with their object system Checkstyle revealed only a small amount of used classes (41). PMD provides similar functionality to Checkstyle, i.e., reporting rule violations on source code. Analyzing a single source code file (`Simple.java` by Cornelissen et al.) with the default `design.xml` of PMD version 5.1.2 revealed 279 used classes and 421.239 method calls in the resulting trace.

Table I shows our tasks including their context and achievable maximum points. We adapted the tasks from Cornelissen et al. to the context of PMD. Notably, we dismissed two original tasks to restrict our experiment to one hour. However, the dismissed tasks are redundant to the remaining tasks with regard to the program comprehension activity categories which are still completely covered. Due to space constraints, we refer to Pacione et al. [31] for an explanation of these categories (A1 to A9). All tasks were given as open questions to prevent guessing. In addition, we changed the order of the tasks compared to Cornelissen et al. since in our experiment no source code access was provided. Our task set starts with less complex tasks (identifying fan-in and fan-out) and ends with complex exploration tasks. This enabled users to get familiar with the visualization in the first tasks and raises the level of complexity in each following task.

To validate our results, we conducted a second controlled experiment as replication. It investigated the influence of the object system's size and design on the results. The visualization of the city metaphor is usually more affected by these factors than using the circular bundling approach. Therefore, we selected a small-sized and not well designed object system. Both criteria are met by Babsi written by undergraduate students. Babsi is an Android app designed to support pharmacists in supervising the prescription of antibiotics. The execution trace generated for our second experiment utilizes all 42 classes and contains 388 method calls.

The tasks for our replication are given in Table II. To enable comparisons of the subjects' performance in our experiments, we kept the tasks as similar as possible. Notably, there is no task similar to T3.1 from our PMD experiment and hence we omitted it in the replication.

TABLE I
DESCRIPTION OF THE PROGRAM COMPREHENSION TASKS FOR THE FIRST EXPERIMENT (PMD)

ID	Category	Description	Score
		Context: Identifying refactoring opportunities	
T1	A{4,8}	Name three classes (from different packages) that have high fan-in (at least 4 incoming communications) and almost no fan-out (outgoing communication).	3
		Context: Understanding the checking process	
T2.1	A{3,4,5}	Write down all constructor/method calls between `RuleChain` and `JavaRuleChainVisitor`.	3
T2.2	A{1,2,5,6}	In general terms, describe the lifecycle of `GodClassRule`: Who creates it, what does it do (on a high level)?	3
		Context: Understanding the violation reporting process	
T3.1	A{1,5}	Which rules are violated by the input file using the design rule set? Hint: Due to dynamic analysis the violation object is created only for those cases.	2
T3.2	A{1,3}	How does the reporting of rule violations work? Where does a rule violation originate and how is it communicated to the user? Write down the classes directly involved in the process. Hint: The output format is set to HTML.	4
		Context: Gaining a general understanding	
T4	A{1,7,9}	Starting from the Mainclass `PMD` – On high level, what are the main abstract steps that are conducted during a PMD checking run. Stick to a maximum of five main steps. Hint: This is an exploration task to get an overview of the system. One strategy is to follow the communication between classes/packages. Keep the handout of PMD in mind.	5

5) Population: We used the computer science students' mailing lists of the Kiel University of Applied Sciences (FH Kiel) and Kiel University (CAU Kiel) to recruit subjects for our first experiment. 30 students have participated in the experiment (6 students from FH Kiel and 24 students from CAU Kiel). Our replication was conducted with 50 students recruited from the CAU Kiel course "Software Project" in summer term 2014 with no overlapping participants.

As motivation, they participated in a lottery for one of five gift cards of 50 €. Additionally, the best three performances received a certificate. The students in the replication had the additional motivation of supporting their task of understanding the software (Babsi) to be used in their course.

The subjects were assigned to the control or experimental groups by random assignment. To validate the equal distribution of experiences, we asked the subjects 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 2.33 versus 2.46 in the experimental group. Their experience with dynamic analysis was 0.41 and 0.69, respectively. Due to the similarity of the self-assessed results, we conclude that the random assignments resulted in a similarly distributed experience between both groups. The same holds for our replication (Java experience: 1.68 and 1.79; dynamic analysis experience: 0.28 and 0.25).

B. Operation

In the following, we detail the operation of our experiments.

1) Generating the Input: We generated the input for ExplorViz directly from the execution of the object systems. ExplorViz persists its data model into files which act as a replay source during the experiments. EXTRAVIS requires files conforming to the Rigi Standard Format (RSF) [33]. To the best of our knowledge, there were no suitable RSF exporter tool for traces of our Java-based object systems. Therefore, we implemented such an exporter in ExplorViz.

Two traces were generated for PMD. The configuration of PMD is conducted in the first trace while the rule checking is performed in the second trace. Both traces are equally important for program comprehension. However, EXTRAVIS is limited to visualize only one trace at a time. Thus, we had to concatenate the two generated traces. Alternatively, the users of EXTRAVIS could have manually loaded each trace when needed. However, this would have hindered the comparison between EXTRAVIS and ExplorViz. Similar circumstances applied to our replication.

2) Tutorials: We provided automated tutorials for EXTRAVIS and ExplorViz where all features were explained. This enhanced the validity of our experiments by eliminating human influences. For ExplorViz, we integrated a guided and interactive tutorial. Since EXTRAVIS is not open-source, we could only provide an illustrated tutorial where the user is not forced to test the functionality. However, we advised the subjects in the control group to interactively test it. Subsequent evaluation of the user recordings showed that all of the subjects have adhered to our advice.

3) Questionnaire: We used an electronic questionnaire rather than sheets of paper. An electronic version provides several advantages for us. First, the timings for each task are automatically recorded and time cheating is impossible. Second, the user is forced to input valid answers for some fields, e.g., perceived difficulty in the debriefing part. Third, we omit a manual and error-prone digitalization of our results.

4) Pilot Study: Before the actual controlled experiment, we conducted a small scale pilot study with experienced colleagues as participants. According to the received feedback, we improved the material and added hints to the tasks which were perceived as too difficult. In addition, a red-green-color-blind impaired colleague used both visualizations to asses any perception difficulties. In the case of ExplorViz, existing arrows in addition to the colors for showing communication directions were sufficient. In the case of EXTRAVIS, we added a tutorial step to change the colors.

TABLE II
DESCRIPTION OF THE PROGRAM COMPREHENSION TASKS FOR OUR REPLICATION (BABSI)

ID	Category	Description	Score
		Context: Identifying refactoring opportunities	
RT1	A{4,8}	Name three classes that have high fan-in (at least 3 incoming communications) and almost no fan-out (outgoing communication).	3
		Context: Understanding the login process	
RT2.1	A{3,4,5}	Write down all constructor/method calls between `gui.MainActivity` and `comm.Sync`.	3
RT2.2	A{1,2,5,6}	In general terms, describe the lifecycle of `data.User`: Who creates it, how is it used? Write down the method calls.	3
		Context: Understanding the antibiotics display process	
RT3	A{1,3}	How does the display of antibiotics work? Where and how are they created? Write down the classes directly involved in the process.	6
		Context: Gaining a general understanding	
RT4	A{1,7,9}	Starting from the Mainclass `gui.MainActivity` - What are the user actions (e.g., Login and Logout) that are performed during this run of Babsi. Write down the classes of the activities/fragment for each user action. Stick to a maximum of seven main steps (excluding Login and Logout). Hint: This is an exploration task to get an overview of the system. One strategy is to follow the communication between classes.	7

5) Procedure: Both experiments took place at CAU Kiel. For the first experiment, each subject had a single session. Therefore, most subjects used the same computer. Only in rare cases, we assigned a second one to deal with time overlaps. In our replication, six to eight computers were concurrently used by the participants in seven sessions. In preliminary experiments, all systems provided similar performance. In all cases, the display resolution was 1920×1080 or 1920×1200.

Each participant received a short written introduction to PMD/Babsi and was given sufficient time for reading before accessing the computer. The subjects were instructed to ask questions in case of encountered challenges at all times. Afterwards, a tutorial for the respective tool was started. Subsequently, the questionnaire part was started with personal questions and experiences, followed by the tasks, and finally debriefing questions.

The less complex tasks (T1, T2.1, T3.1, RT1, RT2.1) have a time allotment of 5 minutes, while the more complex tasks (T2.2, T3.2, T4, RT2.2, RT3, RT4) have 10 minutes. The elapsed time was displayed during the task and highlighted when reaching overtime. The subjects were instructed to adhere to this timing but were not forced to do so.

C. Data Collection

In addition to personal information and experience, we collected several data points during our experiments.

1) Timing and Tracking Information: All our timing information was automatically determined within our electronic questionnaire. Furthermore, we recorded every user session using a screen capture tool (FastStone Capture). These recordings enabled us to reconstruct the user behavior and to look for exceptional cases, e.g., technical problems. In the case of such problems, we manually corrected the timing data.

2) Correctness Information: We conducted a blind review process due to the open questions format. First, we agreed upon sample solutions for each task. A script randomized the order of the answers of the subjects. Thus, no association between answers and group was possible. Then, both reviewers evaluated all solutions independently. Afterwards, small discrepancies in the ratings ware discussed.

3) Qualitative Feedback: The participants were asked to give suggestions to improve their used tool. Due to space constraints, we restrict ourselves to listing the three most mentioned suggestions for each tool. Ten participants suggested hiding not related communication lines when marking a class in EXTRAVIS. Four users missed a textual search feature, which is not available in EXTRAVIS and ExplorViz, and four other users suggested that the performance of fetching called methods should be improved. In the case of ExplorViz, ten subjects suggested to resolve the overlapping of communication lines. Seven users found it difficult to see class names due to overlapping. Five users wished for an opening window containing a list of method names when clicking on communication lines.

D. Analysis and Results

Table III provides descriptive statistics of the overall results related to time spent and correctness for both experiments.

We removed the users with a total score of less than three points from our analysis. This effected five users for our first experiment, i.e., three users from the control group and two users from the experimental group. A single user from the experimental group of our second experiment was effected. In total, three users did not look at the object systems. Hence, they guessed all answers. Two users did not use the "Show full trace" feature in EXTRAVIS, thus analyzing only 0.02% of the trace. One user did not take any look at method names as required for the tasks. For similar reasons, one user for our first experiment and two users in our replication had missing values and are omitted from the overall results but included in the single results.

In Task T3.1, most users searched for a non-existing class `design file` before giving up. This hints at an ambiguous task. Thus, we removed Task T3.1 from our overall analysis.

We use the two-tailed Student's t-test which assumes normal distribution. To test for normal distribution, we use the Shapiro-Wilk test [34] which is considered more powerful [35] than, for instance, the Kolmogorov-Smirnov test [36]. To check for equal or unequal variances, we conduct a Levene test [37].

TABLE III
DESCRIPTIVE STATISTICS OF THE RESULTS RELATED TO TIME SPENT (IN MINUTES) AND CORRECTNESS (IN POINTS)

	PMD				Babsi			
	Time Spent		Correctness		Time Spent		Correctness	
	EXTRAVIS	ExplorViz	EXTRAVIS	ExplorViz	EXTRAVIS	ExplorViz	EXTRAVIS	ExplorViz
mean	47.65	34.27	8.42	13.58	31.55	29.14	9.40	13.04
difference		-28.06%		+61.28%		-7.64%		+38.72%
sd	9.96	3.14	4.29	2.46	7.25	6.48	3.60	3.23
min	23.04	29.43	3	4	18.94	19.38	3	6
median	48.89	33.84	7	14	31.27	27.19	9	13.5
max	65.07	38.99	16	18	43.20	41.56	18	18
Analyzed users	12	12	12	12	24	23	24	23
Shapiro-Wilk W	0.8807	0.9459	0.9055	0.9524	0.9618	0.9297	0.9738	0.9575
Levene F		2.4447		2.0629		0.4642		0.0527
Student's t-test								
df		22		22		45		45
t		4.4377		-3.6170		1.2006		-3.6531
p-value		0.0002		0.0015		0.2362		0.0007

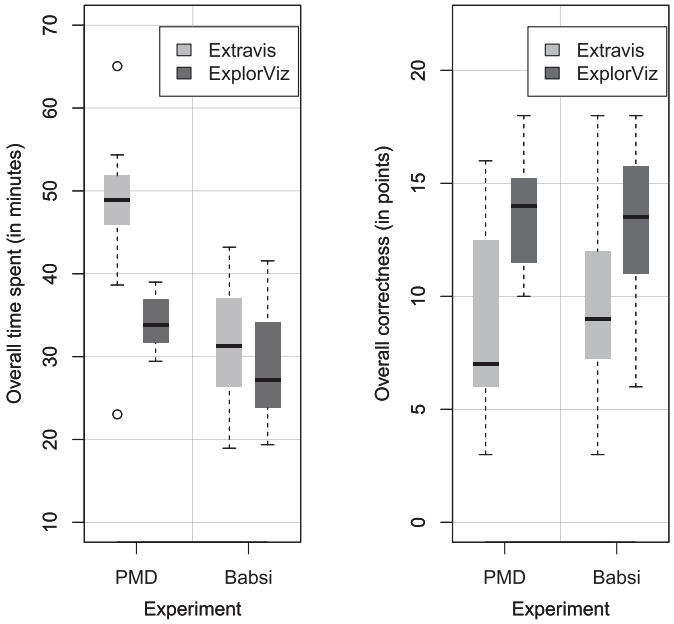

Fig. 3. Overall time spent and correctness for both experiments

For all our analysis tasks, we used the 64-bit R package in version 3.1.1.[5] In addition to the standard 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 in our results. The raw data, the R scripts, and our results are available as part of our experimental data package [16].

RQ1 (Time Spent): We start by checking the null hypothesis $H1_0$ which states that there is no difference in time spent between EXTRAVIS and ExplorViz for completing typical program comprehension tasks.

Figure 3, left-hand side, displays a box plot for the time spent in both experiments. In Table III the differences between the mean values of EXTRAVIS and ExplorViz are shown. For our first experiment, the ExplorViz users required 28.06% less total time for completing the tasks and in our replication they required 7.64% less total time.

The Shapiro-Wilk test for normal distribution in each group and each experiment succeeds and hence we assume normal distribution. The Levene test also succeeds in both experiments and hence we assume equal variances between both groups.

The Student's t-test reveals a probability value of 0.0002 in our first experiment which is lower than the chosen significance level. Therefore, the data allows us to reject the null hypothesis $H1_0$ in favor of the alternative hypothesis H1 for our first experiment. Thus, there is a significant difference in timings (-28.06%) between EXTRAVIS and ExplorViz (t-test t=4.4377, d.f. = 22, P = 0.0002).

In the replication, the Student's t-test reveals a probability value of 0.2362 which is larger than the chosen significance level and we fail to reject the null hypothesis in this case.

RQ2 (Correctness): Next, we check the null hypothesis $H2_0$ which states that there is no difference in correctness of solutions between EXTRAVIS and ExplorViz in completing typical program comprehension tasks.

Figure 3, right-hand side, shows a box plot for the overall correctness in both experiments. Again, Table III shows the differences between the mean values of each group. For our first experiment, the ExplorViz users achieve a 61.28% higher score and in our replication they achieve a 38.78% higher score than the users of EXTRAVIS.

Similar to RQ1, the Shapiro-Wilk and Levene tests succeed for both experiments. The Student's t-test reveals a probability value of 0.0015 for our first experiment and 0.0007 for our replication which is lower than the chosen significance level in both cases. Therefore, the data allows us to reject the null hypothesis $H2_0$ in favor of the alternative hypothesis H2 for both experiment. Hence, there is a significant difference in correctness (+61.28% and +38.72%) between the EXTRAVIS and ExplorViz groups (t-test t=-3.6170, d.f. = 22, P = 0.00015 and t-test t=-3.6531, d.f. = 45, P = 0.0007).

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

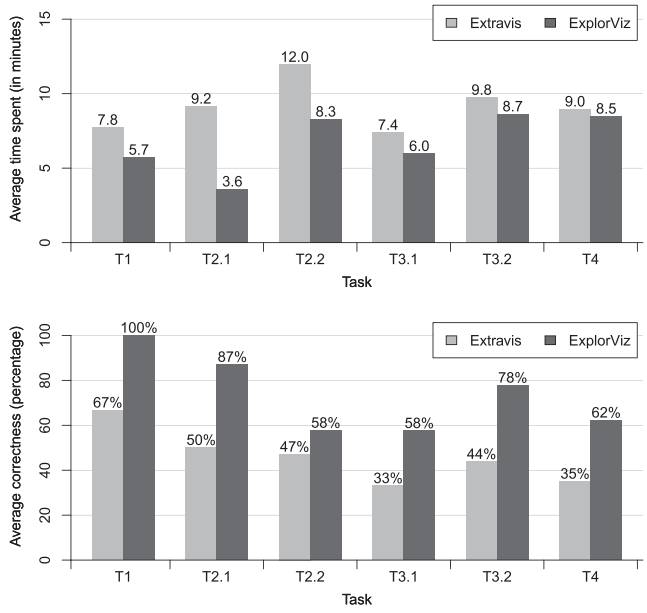

Fig. 4. Average time spent and correctness per task for PMD experiment

E. Discussion

The time spent in our first experiment is significantly lower using ExplorViz. The results for the time spent in our replication are not significant, hence there is no statistical evidence for difference in the time spent. However, due to the median of 31 minutes for EXTRAVIS and 27 minutes for ExplorViz, the box plot, and the fact that our first experiment had a significant difference in time spent, it is unlikely that the time spent with EXTRAVIS is much less than with ExplorViz. Therefore, we can interpret our results such that using either EXTRAVIS or ExplorViz had only a negligible effect in time spent in the replication. Thus, we focus on the correctness in our following discussion.

The analysis of the results reveals a significant higher correctness for users of ExplorViz in both experiments. We conclude 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.

We conducted an in-depth analysis of each user recording to investigate the reasons for the drawn conclusions. Due to space constraints, we focus on our PMD experiment and briefly describe any differences in our replication. The results for each task are shown in Figure 4. For our replication, the graphics for each task are contained in our provided experimental data package. We omit the discussion of Task T3.1 due to its unclear task description.

T1: Most EXTRAVIS users investigated the classes with incoming red lines. They evaluated each of the found classes for the amount of incoming connections by counting the method calls. This was hindered by the large amount of displayed method calls and hence they had to restrict the shown trace using the massive sequence view. A source of error was the used color coding. Some users confused the incoming color and the outgoing color, i.e., they searched for green lines. In the smaller object system of our replication, the method call counting was easier.

Most of the ExplorViz users used random searching for a class. However, the amount of incoming and outgoing method calls of a class was directly visible.

T2.1: The EXTRAVIS users searched for the first class by closing unrelated packages. Then, the second class was searched. Due to the large amount of method calls, the users had to filter them with the massive sequence view. The last call `visitAll` was sometimes missed due to being a thin line in the massive sequence view.

In ExplorViz, the users searched for the first class and marked it. For the second class, the users opened two packages and hovered over the communication to get the method names. With both tools, the users were faster due to a smaller object system in our replication.

T2.2, T3.2, T4: Each of these three tasks required an exploration of the system. For EXTRAVIS, the users started at the class from the task assignment. They marked the class to view its communication and used the massive sequence view to filter it. Therefore, they often divided the trace into arbitrary sequences. Then, the users investigated the method names in each sequence. During this process, some users skipped sequences since EXTRAVIS provides no hints on already viewed sequences. This resulted in misses of method calls and called classes.

The ExplorViz users started with the class described in the task. They looked at the incoming and outgoing method calls. Upon finding a method name of interest, they marked the called class and investigated further communication.

Summary: In our experiments, the massive sequence view of EXTRAVIS led to missing method calls, if the trace is large. This is caused by missing hints on already viewed sequences and by single method calls being visualized by thin lines. Furthermore, the color coding of directions became a source of error. We attribute this circumstance to easily forgettable semantics of color coding while concentrating on a task. Thus, some users had to regularly look up the semantics.

Some users of ExplorViz had difficulties in hovering the communication lines. Some users tried to hover the triangles used for displaying the communication direction instead of the required lines. Furthermore, the overlapping method names and communication lines resulted in taking more time to get the required information.

F. Threats to Validity

In this section, we discuss the threats to internal and external validity [38]–[40] that might have influenced our results.

1) Internal Validity:

a) Subjects: The subjects' experience might not have been fairly distributed across the control group and the experimental group. To alleviate this threat, we randomly assigned the subjects to the groups which resulted in a fair self-assessed experience distribution as described in Section IV-A5.

Another threat is that the subjects might not have been sufficiently competent. Most participants stated regular or advanced programming experience in our first experiment involving PMD. For the replication, most subjects stated beginner or regular programming experience which should be sufficient for the small-sized object system Babsi.

The subjects might have been biased towards ExplorViz since the experiments took place at the university where it is developed. To mitigate this threat, the subjects did not know the goal of the experiments and we did not tell them who developed which tool.

A further threat is that the subjects might not have been properly motivated. In our first experiment, the participants took part voluntarily. Furthermore, we encountered no unmotivated user behavior in the screen recordings – except for the subjects scoring below three points that we have excluded from our analysis (Section IV-D). Therefore, we assume that the remaining subjects were properly motivated. In our replication, the student subjects had the additional motivation to understand the object system. Its maintenance and the extension of Babsi was the main goal of their upcoming course. Hence, we assume that they were properly motivated. This is also confirmed by the screen recordings.

b) Tasks: Since we – as the authors of ExplorViz – created the tasks, they might have been biased towards our tool. We mitigated this threat by adapting the tasks defined by Cornelissen et al. [9] as similar as possible.

The tasks might have been too difficult. This threat is reduced by the fact that users from both groups achieved the maximum score in each task except for the last. Furthermore, the average perceived task difficulty, shown in Table IV, is between easy (2) and difficult (4), and never too difficult (5).

To reduce the threat of incorrect or biased rating of the solutions, we used a blind review process where two reviewers independently reviewed each solution. The seldom discrepancies in the ratings were at most one point which suggests a high inter-rater reliability.

Omitting the unclear Task T3.1 might have biased the results. As can be seen in Figure 4, this task had similar results as the others tasks. Thus, not omitting Task T3.1 would result in only a small difference of one percent in time spent (27 percent) and two percent in correctness (63 percent).

c) Miscellaneous: The results might have been influenced by time constraints that were too loose or too strict. This threat is reduced by the average perceived time pressure which are near an appropriate (3) rating for both groups.

The tutorials might have differed in terms of quality. This might have led to slower subject performance with one tool. In both groups, the subjects had the possibility to continue to use the tutorial until they felt confident in their understanding.

Users of EXTRAVIS had to switch between the questionnaire application and EXTRAVIS. This could have let to a disadvantage of the EXTRAVIS group. This threat is mitigated by the fact that the users had only to switch to enter answers and both applications were tiled on the screen. In addition, the users of ExplorViz had also to click into the input field.

TABLE IV
DEBRIEFING QUESTIONNAIRE RESULTS FOR OUR PMD EXPERIMENT
1 IS BETTER – 5 IS WORSE

	EXTRAVIS		**ExplorViz**	
	mean	stdev.	mean	stdev.
Time pressure (1-5)	3.67	0.78	2.62	0.50
Tool speed (1-5)	3.08	0.90	2.15	0.99
Tutorial helpfulness (1-5)	2.75	1.14	1.92	0.86
Tutorial length (1-5)	3.58	0.67	3.00	0.41
Achieved PMD comprehension (1-5)	3.42	0.79	3.15	0.90
Perceived task difficulty (1-5)				
T1	2.33	0.65	2.42	1.00
T2.1	3.17	0.72	2.31	0.75
T2.2	3.58	0.67	3.46	1.05
T3.1	3.83	0.58	3.54	0.88
T3.2	3.75	0.75	3.38	0.65
T4	3.75	0.87	3.54	0.88

For reasons of fairness, we disabled some features in ExplorViz for the experiment. This might have influenced the results since the participants had less features to learn. The impact of the disabled features should be investigated in a further experiment.

2) External Validity: To counteract the threat of limited representativeness of a single object system, we conducted a replication and varied the system. We chose an object system that differed in size and design quality. However, there might be more attributes of the object system that impacted the results. Therefore, further experiments are required to test the impact of other systems on the results.

Our subjects were only made up of students. Thus, they might have acted different to professional software engineers. Further replications are planned with professionals to quantify the impact of this threat.

The short duration of the tool's usage endangers the generalizability of the results. To investigate the impact of this threat, longer experiments should be conducted.

Another external validity threat concerns the program comprehension tasks, which might not reflect real tasks. We adapted our tasks from Cornelissen et al. [10] who used the framework of Pacione et al. [31]. Thus, this threat is mitigated.

G. Lessons Learned and Challenges Occurred

We consider the user recordings very useful. They enabled us to analyze the users' strategies in detail. Furthermore, we can investigate unsuitable answers as in the case of the users, who did not access the visualization of the correct object system. Thus, the user recordings are a method of quality assurance resulting in more confidence in the data set.

However, we also experienced some challenges during our experiments. Implementing a generator for the input files of tools should be superfluous for using the tool. Therefore, we advise other visualization tool developers, who use externally generated input files, to bundle a working input file generator and a manual with their distribution (if the license permits this). External tools might change or become unavailable, rendering it hard for others to employ the tool.

Furthermore, tutorial material and source code for EXTRAVIS were unfortunately not available. Therefore, we had to create our own tutorial materials which might have influenced the experiment. With the source code, we could have, for instance, integrated an interactive tutorial into EXTRAVIS. To facilitate a better and easier comparison of visualization tools, *research* prototypes should be developed as open source.

Tool developers should contribute to overcome those challenges and to lower the hurdle for comparison experiments.

V. Related Work

In this section, we discuss related work on controlled experiments with software visualization tools in a program comprehension context. Many approaches for software visualization of single applications exist and for reasons of space, we refer to [11] and [14] for a detailed comparison of EXTRAVIS and ExplorViz to other software visualization approaches.

A. Experiments Comparing to an IDE

In this subsection, we list related experiments that compare visualizations to an IDE. In general, contrary to our experiments, we compare two visualization techniques without the use of an IDE to investigate the efficiency and effectiveness of the visualization techniques. Further differentiations are presented in each experiment discussion.

Marcus et al. [41] present a controlled experiment comparing Visual Studio .NET and sv3D for program comprehension. The usage of sv3D led to a significant increase in the time spent. In contrast to our experiments, they compared a visualization basing upon static analysis and not trace visualization.

Quante [42] assessed whether additionally available Dynamic Object Process Graphs provide benefits to program comprehension when using Eclipse. The experiment involved two object systems and for the second system the results were not significant. In contrast to our experiments, the author investigated only the additional availability of a visualization and did not compare two visualizations.

Wettel et al. [29] conducted a controlled experiment to compare the usage of Eclipse and Excel to their tool CodeCity with professionals. They found a significant increase in correctness and decrease of time spent using CodeCity. They investigated a single visualization basing upon static analysis, contrary to our comparison of two trace visualizations.

Sharif et al. [43] present a controlled experiment comparing Eclipse and the additional availability of the SeeIT 3D Eclipse plug-in using eye-tracking. The experimental group with access to SeeIT 3D performed significantly better in overview tasks but required more time in bug fixing tasks. In contrast to our experiments, they investigated only the additional availability of SeeIT 3D basing on static analysis.

Cornelissen et al. [10] performed a controlled experiment for the evaluation of EXTRAVIS to investigate whether the availability of EXTRAVIS in addition to Eclipse provides benefits. The availability of EXTRAVIS led to advantages in time spent and correctness. In contrast, we compare two trace visualization techniques.

B. Experiments Comparing Software Visualizations

Storey et al. [4] compared three static analysis tools in an experiment. The authors performed a detailed discussion of the tools' usage but provided no quantitative results.

Lange and Chaudron [44] investigated the benefits of their enriched UML views by comparing them with traditional UML diagrams. In contrast, we compared two different trace visualization techniques with similar features.

VI. Conclusions and Outlook

In this paper, we presented two controlled experiments with different-sized object systems to compare the trace visualization tools EXTRAVIS and ExplorViz in typical program comprehension tasks. We performed the first experiment with PMD and conducted a replication as our second experiment using the smaller Babsi system.

Our first experiment resulted in a significant decrease of 28 percent of time spent and increase in correctness by 61 percent using ExplorViz for PMD. In our replication with the smaller-sized system, the time spent using either EXTRAVIS or ExplorViz was similar. However, the use of ExplorViz significantly increased the correctness by 39 percent.

Our in-depth analysis of the used strategies revealed sources of error in solving the tasks caused by, for instance, color coding or overlapping communication lines. In addition, we identified common challenges for controlled experiments to compare software visualization techniques. For instance, available visualization tools miss sufficient tutorial material.

Our results provide guidance towards ExplorViz for new users who search for a trace visualization tool and have similar tasks as we examined. Since our experiments investigated first time use, the results might be different in long term usage. This should be addressed in further experiments.

We provide a package containing all our experimental data to facilitate the verifiability and reproducibility for further replications [45]. It contains the employed version of ExplorViz v0.5-exp (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. The package is available online [16] with source code under the Apache 2.0 License and the data under a Creative Commons License (CC BY 3.0).

In our future work, we plan to further replicate our experiments with professional software engineers. Furthermore, several other trace visualization techniques can be compared to EXTRAVIS and ExplorViz, for instance, the technique used by Trümper et al. [46]. Further future work is the development of a validated programming experience questionnaire.

For ExplorViz, we have gained precious in-depth knowledge on how our tool is used by a large group of users. The experiments produced a large amount of valuable qualitative feedback, which we will integrate into future releases of ExplorViz, for instance, improvements of the layout algorithm.

References

[1] S. P. Reiss and A. Tarvo, "What is my program doing? program dynamics in programmer's terms," in *Runtime Verification*, ser. LNCS. Springer Berlin Heidelberg, 2012, vol. 7186, pp. 245–259.

[2] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE TSE*, vol. 35, no. 5, pp. 684–702, 2009.

[3] 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.

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

[5] 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.

[6] W. F. Tichy, "Should computer scientists experiment more?" *IEEE Computer*, vol. 31, no. 5, pp. 32–40, May 1998.

[7] D. I. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal, "A survey of controlled experiments in software engineering," *IEEE TSE*, vol. 31, no. 9, pp. 733–753, 2005.

[8] W. F. Tichy, "Where's the science in software engineering?: Ubiquity symposium: The science in computer science," *Ubiquity*, vol. 2014, no. March, pp. 1:1–1:6, Mar. 2014.

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

[10] B. Cornelissen, A. Zaidman, and A. van Deursen, "A controlled experiment for program comprehension through trace visualization," *IEEE TSE*, vol. 37, no. 3, pp. 341–355, May 2011.

[11] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. Van Wijk, and A. Van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Proc. of the 15th IEEE International Conference on Program Comprehension (ICPC 2007)*. IEEE, 2007, pp. 49–58.

[12] W. F. Tichy, "Hints for reviewing empirical work in software engineering," *Empirical Software Engineering*, vol. 5, no. 4, pp. 309–312, 2000.

[13] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE TSE*, vol. 25, no. 4, pp. 456–473, 1999.

[14] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," in *Proc. of the 1st International Working Conference on Software Visualization (VISSOFT 2013)*, Sep. 2013.

[15] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Proc. of the 4th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, Jun. 2007.

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

[17] B. Shneiderman, "Software psychology: Human factors in computer and information systems," *Winthrop Publishers, Inc.*, 1980.

[18] R. Wettel, "Software systems as cities," Ph.D. dissertation, University of Lugano, 2010.

[19] R. Brooks, "Towards a theory of the comprehension of computer programs," *International Journal of Man-Machine Studies*, vol. 18, no. 6, pp. 543 – 554, 1983.

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

[21] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring, "Self-adaptive software system monitoring for performance anomaly localization," in *Proc. of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011)*. ACM, Jun. 2011, pp. 197–200.

[22] R. M. Lindsay and A. S. Ehrenberg, "The design of replicated studies," *The American Statistician*, vol. 47, no. 3, pp. 217–228, 1993.

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

[24] 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.

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

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

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

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

[29] 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.

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

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

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

[33] K. Wong, "Rigi user's manual – version 5.4.4," last accessed 14-09-05. [Online]. Available: http://www.rigi.cs.uvic.ca/downloads/rigi/doc/rigi-5.4.4-manual.pdf

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

[35] 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.

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

[37] 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.

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

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

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

[41] A. Marcus, D. Comorski, and A. Sergeyev, "Supporting the evolution of a software visualization tool through usability studies," in *Proc. of the 13th International Workshop on Program Comprehension (IWPC 2005)*, May 2005, pp. 307–316.

[42] J. Quante, "Do dynamic object process graphs support program understanding? – A controlled experiment." in *Proc. of the 16th IEEE International Conference on Program Comprehension (ICPC 2008)*, June 2008, pp. 73–82.

[43] B. Sharif, G. Jetty, J. Aponte, and E. Parra, "An empirical study assessing the effect of SeeIT 3D on comprehension," in *Proc. of the 1st IEEE Working Conference on Software Visualization (VISSOFT 2013)*, Sep. 2013, pp. 1–10.

[44] C. Lange and M. R. V. Chaudron, "Interactive views to improve the comprehension of UML models – An experimental validation," in *Proc. of the 15th IEEE International Conference on Program Comprehension (ICPC 2007)*, June 2007, pp. 221–230.

[45] T. Crick, B. A. Hall, and S. Ishtiaq, "Can I implement your algorithm?: A model for reproducible research software," in *Proc. of the 2nd Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2)*. arXiv, Nov. 2014, pp. 1–4.

[46] J. Trümper, J. Bohnet, and J. Döllner, "Understanding complex multithreaded software systems by using trace visualization," in *Proc. of the 5th International Symposium on Software Visualization (SOFTVIS 2010)*. ACM, 2010, pp. 133–142.