INSTITUT FÜR INFORMATIK

Research Perspective on Supporting Software Engineering via Physical 3D Models

Florian Fittkau, Erik Koppenhagen, and Wilhelm Hasselbring

Bericht Nr. 1507
June 2015
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Abstract

Building architects, but also civil or mechanical engineers often build from their designs physical 3D models for a better presentation, comprehension, and communication among stakeholders. Software engineers usually create visualizations of their software designs as digital objects to be presented on a screen only. 3D software visualization metaphors, such as the software city metaphor, provide a basis for exporting those on-screen software visualizations into physical models. This can be achieved by 3D-printers to transfer the advantages of real, physical, tangible architecture models from traditional engineering disciplines to software engineering.

We present a new research perspective of using physical models to support software engineering. Furthermore, we describe four envisioned usage scenarios for physical models which provide a plethora of new research topics. As proof of concept, we investigate the first usage scenario by evaluating the impact of using physical models on program comprehension in teams through a first controlled experiment.

Our experiment reveals that the usage of physical models has a diverging influence. However, they improve the team-based program comprehension process for specific tasks by initiating gesture-based interaction. Therefore, the experiment shows that physical models can provide a promising future research direction.

1 Introduction

As stated by [Ball and Eick 1996] in 1996, “software is intangible, having no physical shape or size.” However, effective program comprehension requires abstractions from source code in larger projects. To overcome this challenge, many software visualization techniques exist. Such software visualizations often rely on metaphors, such as the 3D software city metaphor [Knight and Munro 2000].

Although 3D visualizations can deliver more information compared to 2D visualizations due to its further dimension, it is often difficult for users to navigate in 3D spaces using a 2D screen and a 2D input device [Teyseyre and Campo 2009]. As a consequence, users may get disoriented [Herndon et al. 1994] and thus the advantages of the third dimension may be abolished.

One solution candidate for the navigation issue is Virtual Reality (VR). It provides benefits via stereoscopy in combination with specialized hardware [Ware et al. 1993; Ware and Mitchell 2005]. However, it relies on – sometimes expensive – extra equipment which has to be purchased and might not function in every environment.

Traditional engineering disciplines overcome these issues by building physical 3D models. Beneath resolving navigation issues, the physical models are used for better presentation, comprehension, and communication among stakeholders.

In this paper, we present a new research perspective to transfer these advantages to software engineering by building physical models following the 3D software city metaphor through 3D-printing. Furthermore, we describe four envisioned, potential usage scenarios for physical models and formulate possible research questions. To the best of our knowledge, we are the first to envision and create such physical models of software visualizations.

As a proof of concept to show the potential of using physical models, we conduct a first controlled experiment for our first envisioned usage scenario, i.e., in the context of program comprehension. Gestures support in thinking and communication processes [Goldin-Meadow 2005] and thus can enhance program comprehension in groups. We hypothesize that physical models support in gesticulation. In our experiment, we compare the usage of an on-screen model to the usage of a physical model for program comprehension in small teams (pairs). To facilitate the verifiability and reproducibility of our results, we provide a package [Fittkau et al. 2015] containing all our experimental data including the raw data and 112 recordings of the participant sessions.

In summary, our main contributions are:

1. a new research perspective to transfer the advantages of physical models to software engineering through 3D-printing,
2. four envisioned usage scenarios for physical models, and
3. the reusable design and execution with 56 teams of a controlled experiment comparing the usage of an on-screen model to the usage of a physical model in typical program comprehension tasks.

The remainder of this paper is organized as follows. The envisioned usage scenarios and possible research questions for physical models are described in Section 2. Then, Section 3 introduces our virtual models and how we create physical models of them. Section 4 presents a first controlled experiment to evaluate the impact of using physical models for program comprehension. Related work is discussed in Section 5. Finally, we draw the conclusions and illustrate further future work in Section 6.

2 Envisioned Usage Scenarios

Physical models provide a plethora of future research possibilities. We envision several potential usage scenarios for physical models which we will outline in the following and propose selected research questions.

2.1 Program Comprehension in Teams

Gestures support in thinking and communication processes [Goldin-Meadow 2005]. Since physical models are more accessible than 2D screens and provide a natural interaction possibility, they might increase the gesticulation of users. This might lead to faster and better understanding when applied in a team-based program comprehension scenario due to its supporting nature. Furthermore, the advantages might increase when applied in larger teams. Since software systems are often changing, the model should only be printed for special occasions, e.g., a new developer team or upcoming major refactorings.

Potential Research Questions

In which scenarios/tasks do physical models provide benefits? Does the additional usage of physical models provide advantages compared to sole on-screen tooling? How large is the impact of gesticulation on correctness and time spent in team-based program comprehension tasks?

2.2 Educational Visualization

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

Potential Research Questions

Which software visualization metaphor provides the best basis for representing design patterns? How large is the impact of using physical models for educational purposes? How to display the dynamic behavior in physical models which is often defined by a design pattern?

2.3 Effort Visualization in Customer Dialog

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

Potential Research Questions

Do customers accept physical models to show the effort? Does the usage of physical models increase the conviction of effort in a customer dialog? How much impact does a physical model provide in this process (e.g., measured in amount of money for the effort)?

2.4 Saving Digital Heritage

We envision physical models to be a step toward saving the digital heritage of software visualizations. Compared to programs, physical models do not depend on the availability of SDKs, library versions, or hardware and thus are less vulnerable to changes of external environment. Often it is uncertain, if the code can still be run in thirty years. In contrast, depending on the material (e.g., resin or metal), physical models might last hundreds of years. One might argue that pictures of the visualizations are sufficient. However, they do not provide interaction possibilities and can suffer from occlusion. Contrary, physical models still provide the possibility to interact (e.g., rotate) avoiding possible occlusion.

Potential Research Questions

How to provide an omni accessible archive for physical models? How to convert 2D software visualizations to 3D physical models?

3 From Virtual to Physical Models

3.1 Our Virtual Models in a Nutshell

This section briefly introduces our web-based ExplorViz[1] visualization which provides the 3D model data for 3D-printing and forms the basis for our controlled experiment.

Figure 1 displays our application-level perspective utilizing the city metaphor. Compared with previous publications [Fittkau et al. 2013], we changed the on-screen visualization to investigate the impact of physical models. These changes are described in Section 4. The visualization is constructed from an execution trace of PMD[2] used as the object system in our controlled experiment. The flat green platforms (❶) in our visualization represent packages showing their contained elements. The green boxes on the top layer are packages (❷) hiding their internal details. The two green tones serve to differentiate the hierarchy levels. Classes are visualized by purple boxes (❸). The height of classes maps to the active instance count. The layout is a modified version of the layout used in CodeCity [Wettel 2010]. The user can rotate, pan, and zoom the visualized model.

3.2 Creating Physical 3D-Printed Models

After creating the on-screen model in ExplorViz by monitoring a PMD run, we exported the model as an OpenSCAD[3] script file. To fit the build platform of our low-budget 3D-printer (a Prusa i3), we split the exported model into twelve jigsaw

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

[2]`http://pmd.sourceforge.net`

[3]`http://www.openscad.org`

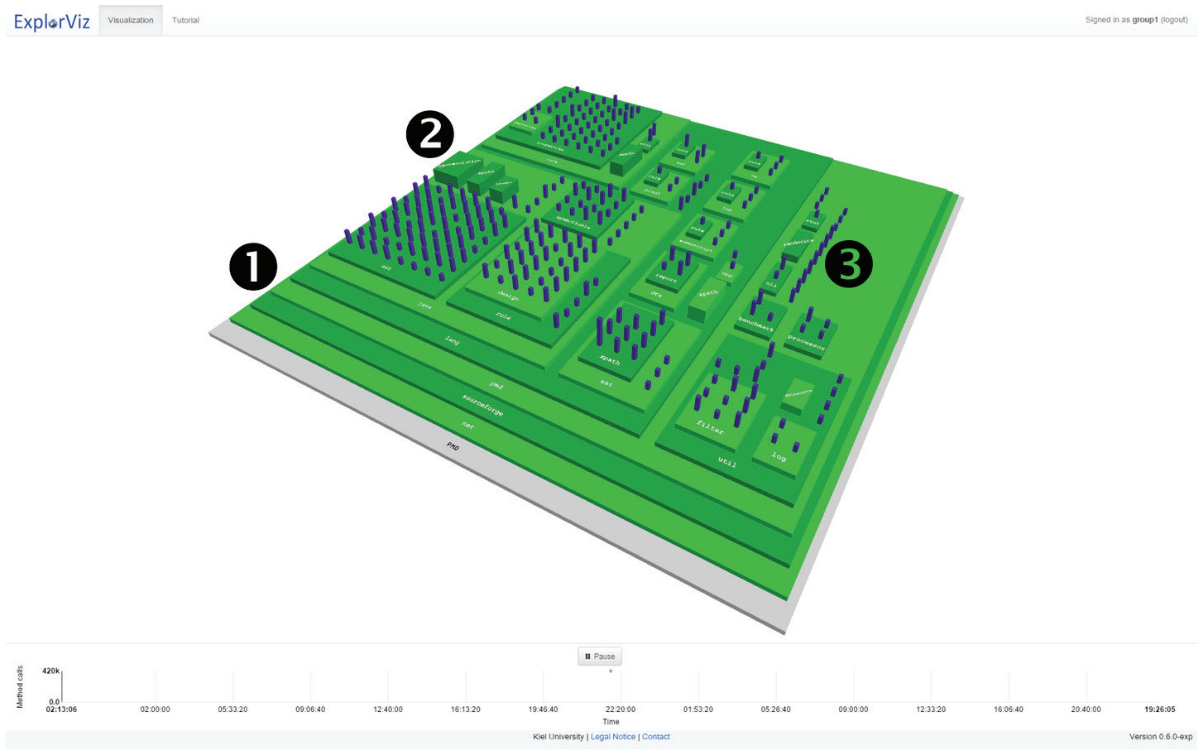

Figure 1. On-screen, web-based application-level perspective of ExplorViz visualizing PMD

pieces and exported these as STL[4] files as input for the 3D-printer. After printing each piece, we glued them together. The fully assembled physical PMD model is 334 mm wide and 354 mm deep. At last, we manually painted the single-colored model. The overall building time of the physical PMD model sums up to 58 hours and the costs of material are about 9 € on our low-budget, self-built, and rather slow 3D-printer. On a modern multi-color printer, 3D-printing the model would take only a small fraction of the building time (i.e., a few hours) and save several of the described working steps.

4 Proof of Concept: First Evaluation for Program Comprehension

Beneath introducing physical models in this paper, we present a first evaluation for the envisioned physical models as proof of concept. We compare the impact of using either an on-screen model or a physical model to solve typical program

[4] http://www.ennex.com/~fabbers/StL.asp

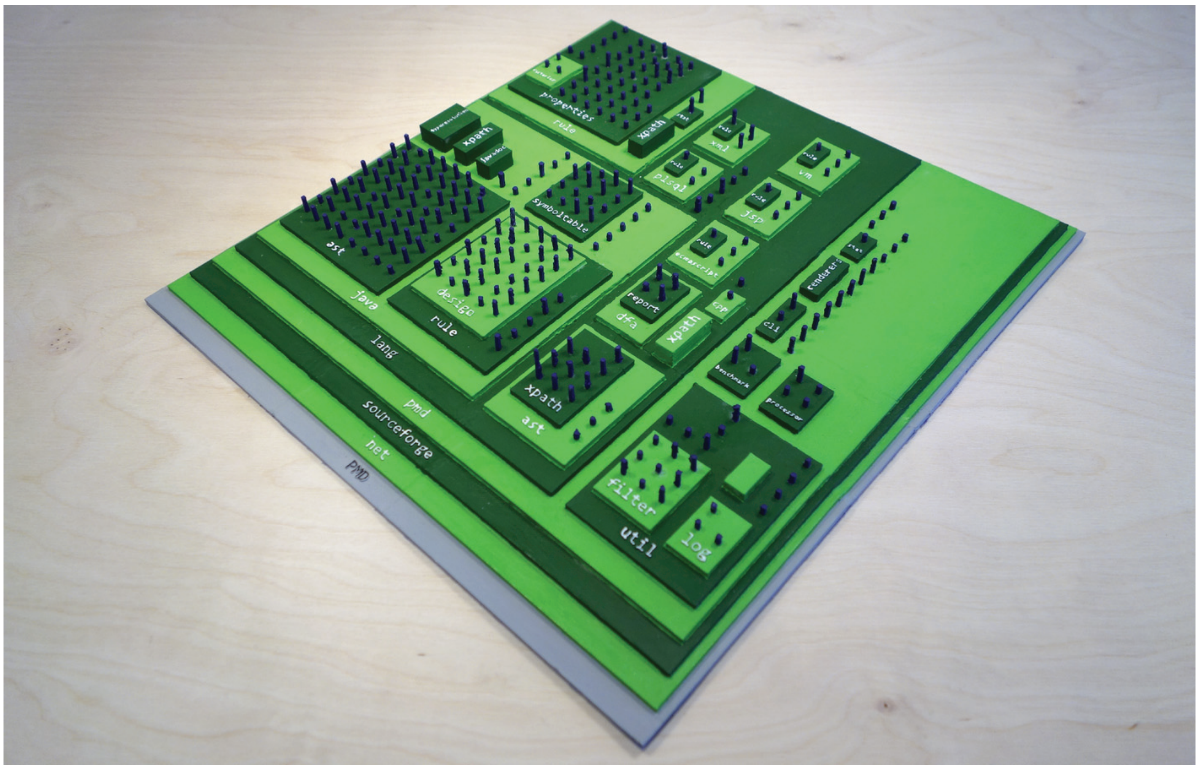

Figure 2. Physical 3D-printed and manually painted city metaphor model of PMD (334 mm wide and 354 mm deep)

comprehension tasks in a team-based scenario. As object system we use PMD and measure the *time spent* and *correctness* for each task typically used in the context of program comprehension [Rajlich and Cowan 1997]. Afterwards, we analyze the employed strategies and possible differences between both groups.

Since we disabled some interaction possibilities of the on-screen model, the comparison seems unfair at first sight. However, our goal of the evaluation is to investigate if there *is* any difference in using physical models compared to on-screen visualization. If we would not have disabled some interaction possibilities, we would have tested for the current shortcomings of our physical model, rather than investigating if physical models provide any benefits. Further experiments should examine, for instance, if the combination of using a physical model and an on-screen model compared to solely using an on-screen model provides further advantages.

We describe the design of our controlled experiment, its operation, data collection, analysis, results, discussion, and threats to validity.

4.1 Experimental Design

In addition to general software engineering experimentation guidelines [Kitchenham et al. 2002; Jedlitschka and Pfahl 2005; Di Lucca and Di Penta 2006; Di Penta et al. 2007; Sensalire et al. 2009], we follow the experimental designs of Wettel et al. [Wettel et al. 2011] and of Cornelissen et al. [Cornelissen et al. 2009]. Similar to them, we use a *between-subjects* design. Thus, each subject only solves tasks with either using the on-screen or the physical model. Following GQM [Basili and Weiss 1984], we define the goal of our experiment as quantifying the impact of using either the on-screen or the physical model for program comprehension.

We name the control group On-Screen Model and the experimental group Physical Model. Due to space constraints, we abbreviate the groups as On-Screen and Physical in figures and tables. To circumvent confusion with the control group and experimental group, we name each group of two participants (a pair) as a team and not as a group.

4.1.1 Research Questions & Hypotheses

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

▻ **RQ1:** What is the ratio between On-Screen Model and Physical Model in the *time required for completing*
typical program comprehension tasks?

▻ **RQ2:** What is the ratio between On-Screen Model and Physical Model in the *correctness of solutions* to typical program comprehension tasks?

▻ **RQ3:** Which typical program comprehension tasks benefit more from using the on-screen model and which benefit more from using the physical model?

Accordingly, we formulate two hypotheses:

▻ **H1** On-Screen Model and Physical Model require different times for completing typical program comprehension tasks.

▻ **H2** The correctness of solutions to typical program comprehension tasks differs between On-Screen Model and Physical Model.

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 team in detail.

4.1.2 Dependent and Independent Variables

The independent variable is the model employed for the program comprehension tasks, i.e., on-screen model or physical model. We measured the accuracy (*correctness*) and response time (*time spent*) as dependent variables. These are usually investigated in the context of program comprehension [Wettel et al. 2011; Rajlich and Cowan 1997; Cornelissen et al. 2009].

4.1.3 Treatment

The control group used the on-screen model to solve the given program comprehension tasks. The experimental group solved the tasks utilizing the physical model.

4.1.4 Tasks

We used the framework of Pacione et al. [Pacione et al. 2004] to create our task set. The framework describes categories of typical program comprehension tasks.

We selected a medium to large-sized object system (PMD) for our experiment since it provides a well designed software architecture. The function of PMD is reporting rule violations on source code. Therefore, it takes source code and a rule configuration as input parameters. To generate our city metaphor visualization of PMD, we monitor the analysis run of PMD on a simple source code file (`Simple.java` by Cornelissen et al.) and the default `design.xml` of PMD version 5.1.2. The resulting model visualizes 279 used classes of PMD.

In Table 1, our defined tasks including their context and achievable maximum points is displayed. Furthermore, the covered categories of the framework of [Pacione et al. 2004] are shown. Due to space constraints, we refer to their paper for a detailed explanation of the categories. To prevent guessing, all tasks were given as open questions. Our task set starts with less complex tasks (identifying the class with the most instances) and ends with a more complex design understanding task. This enabled each team to get familiar with the model in the first tasks and raises the level of complexity in each following task.

4.1.5 Population

The participants were students from the course “Software Engineering”. They received one exercise point for successfully solving each task. For the weekly assigned exercise points, they received bonus points for their exam at the end of the term. To further motivate the participants for the experiment, they could win one of ten gift

Table 1. Description of the Program Comprehension Tasks for the experiment

ID	Category	Description	Score
T1	A{3,5,6,8}	*Context: Metric-Based Analysis* Find the package containing the one class having the most instances in the application. How is the package named? How many classes (and subpackages if existing) does it contain? Please write down the full package path.	2
T2	A{6,8}	*Context: Structural Understanding* What are the names of the three packages directly containing the most classes (without their subpackages)? Please order your answer by beginning with the package containing the most classes and write down the full path.	4
T3	A{1,3,7}	*Context: Concept Location* Assuming a good design, which package could contain the *Main* class of the application? Give reasons for your answer.	2
T4	A{3,4}	*Context: Structural Understanding* Which package name occurs the most in the application? In addition, shortly describe the distribution of these packages in the system. Hint: Have a look at the different levels of the packages. There are exactly two types of distribution.	3
T5	A{1,2,3,9}	*Context: Design Understanding* What is the purpose of the `lang` package and what can you say about its content regarding PMD? Are there any special packages? Do they differ by size? Ignore the `xpath` and `dfa` packages and name three facts in your answer. Hint: Remember the received paper about the introduction to PMD.	3

cards over 10 € for the sole participation and the best eight teams each received a gift card over 20 €.

The teams were assigned to the control or experimental groups by random assignment. To validate the equal distribution of experiences, we asked the teams to perform a self-assessment on a 5-point Likert Scale [Likert 1932] ranging from 0 (no experience) to 4 (expert with years of experience) before the experiment. The average programming experience in the control group was 1.89 versus 1.8 in the experimental group. The average software architecture experience was 1.11 in the control group and 1.00 in the experimental group. Since the experience was self-assessed and both values are within a close range, we assume that random assignment succeeded.

4.2 Operation

Next, the operation of our experiment is detailed.

4.2.1 Generating the Input

We generated the input for the control group directly from the execution of PMD. ExplorViz persists its data model into files which act as a replay source during the experiment. How we build the physical model from this on-screen model, was already detailed in Section 3.

4.2.2 Tutorials

We provided automated tutorials for both groups of the experiment. This enhanced the validity of our experiments by eliminating human influences. For the tutorial system, we used a small mockup of Neo4J to make the teams familiar with the visualization. For the control group, we integrated an interactive tutorial such that the teams can directly test the functionality on-screen. As tutorial, the experimental group received a physical tutorial model which can be seen in Figure 3. Their explanation text was shown on a screen in addition to a photo of the model.

Both groups had the same explanation text for the tutorial except how the control group could interact with the on-screen model, i.e., rotation, panning, and zooming.

4.2.3 Disabled Features of the On-Screen Model

To investigate the impact of physical models, we disabled some features in the on-screen visualization. Otherwise, we would only have tested the current limitations of the physical models.

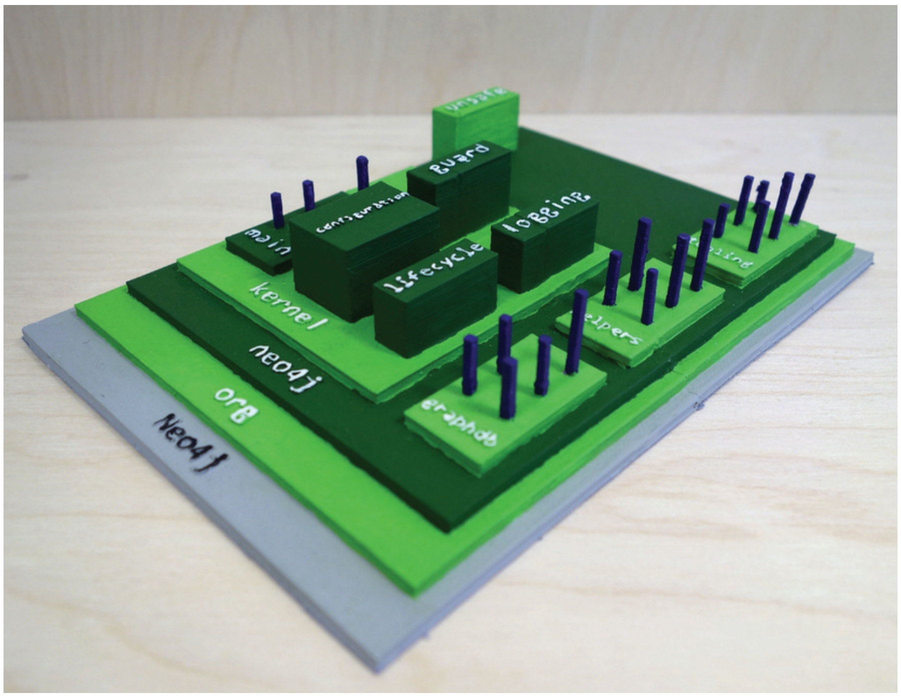

Figure 3. Physical model of a mockup of Neo4J used during the tutorial

For the experiment, we hide class names and communication lines, since the physical model cannot provide this information at the moment. In ExplorViz, classes can be highlighted to see its communication. We also disabled this feature, since the physical model does not provide such capabilities. The interactive opening and closing of packages to show or hide their internal details is also disabled.

As ExplorViz provides a periodically updated live visualization, a further main feature is time shifting. It is designed for analyzing specific situations of interest, for instance, performance anomalies [Ehlers et al. 2011]. Since the physical model is static and having multiple points in time would require multiple physical models, we disabled this feature for our experiment.

Source code viewing is considered important [Koschke 2003]. 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 the physical model.

4.2.4 Questionnaire

We utilize an electronic questionnaire for both experiment groups. An electronic version provides three advantages over using sheets of paper for us. First, the teams are forced to input valid answers for category fields, e.g., existing experience. Second, manual digitalization can be error-prone in itself and we avoid this by direct electronic capture. Lastly, the timings are automatically recorded for each task and thus time cheating by the teams is impossible.

4.2.5 Pilot Study

Before the controlled experiment took place, we conducted a pilot study with four experienced colleagues as participants forming two teams. The received feedback helped us in improving the material. Furthermore, we added hints to the tasks which were perceived as too difficult.

4.2.6 Procedure

Our experiment took place at the Kiel University. Each team had a single session of up to one hour. Therefore, all teams used the same computer. The display resolution was set to 1920 × 1200 and prior benchmarking had shown that ExplorViz runs smoothly on this computer.

In order to have recordings of the conducted gesticulation, a Full HD camera recorded all team sessions. At the beginning, the control and the experimental group were told that the table is recorded such that only hands are visible on the video. Furthermore, we explained them that their faces – if they came into the recording area – get pixelated. Afterwards, each team received a sheet of paper containing a short introduction to PMD and its features. They were given sufficient time for reading before accessing the computer. After telling the teams that they can ask questions at all times, a tutorial for the respective model was started and the Physical Model group were given the physical tutorial model. Subsequently, the questionnaire part was started with personal questions and experiences. Afterwards, the program comprehension tasks begun and the experimental group were handed the physical PMD model. The session ended with the debriefing questions.

The less complex tasks (T1, T2, T3) had a time allotment of 5 minutes and the more complex tasks (T4, T5) had 10 minutes. During the task, the elapsed time was displayed beneath the task description. The teams were instructed to adhere to this timing but were not forced to end the task. If they reached overtime, the timer was only highlighted.

4.3 Data Collection

We collected several data points during our experiment.

Table 2. Descriptive Statistics of the Results Related to Time Spent (in Minutes) and Correctness (in Points)

	Time Spent		**Correctness**	
	On-Screen	Physical	On-Screen	Physical
mean	28.11	29.39	11.28	11.62
difference		+4.55%		+3.01%
sd	5.94	8.46	1.88	1.47
min	16.24	14.54	7	8.5
median	28.46	27.72	11	12
max	40.97	53.48	14	14
Shapiro-Wilk W	0.9894	0.9436	0.9326	0.9566
Levene F		0.6387		0.7095
Student's t-test				
df		50		50
t		-0.6326		-0.7283
p-value		0.5299		0.4698

4.3.1 Timing and Tracking Information

Our electronic questionnaire automatically determined the timing information for each task. In addition to the camera recordings, we directly capture the screen of every team using a screen capture tool. The screen recordings enabled us to reconstruct the behavior of the teams and to look for exceptional cases, for instance, technical problems. If we found such a case, we had to manually correct the timing data.

4.3.2 Correctness Information

Since we chose an open question format for each task, we conducted a blind review process to rate the answers. After agreeing upon sample solutions, a script randomized the order of the solutions. Thus during the review process, no association between the answers and a team was possible for the reviewers. After both reviewers evaluated all solutions independently, any discrepancies in the ratings were discussed and resolved.

4.3.3 Qualitative Feedback

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

1) Three teams from the control group missed a feature to highlight entities. 2) Furthermore, three teams would like to have more readable labels at a higher zoom distance. 3) One team disliked that the camera was not resetable.

1) In the experimental group, two teams wanted to have a legend explaining the meaning of the entities. 2) One team would like to have more readable labels. 3) Another team proposed that the amount of instances of a class should be color-coded.

4.4 Analysis and Results

Table 2 provides descriptive statistics of the overall results related to time spent and correctness for our experiment.

We removed the teams with a total score of less than six points from our analysis. This effected one team from the control group and three teams from the experimental group. After watching the recorded sessions, we came to the conclusion that the teams did not understand the semantics of the visualization and thus their answers seem guessed.

For our analysis, we use the two-tailed Student's t-test. To test for normal distribution, we use the Shapiro-Wilk test [Shapiro and Wilk 1965] which is considered more powerful [Razali and Wah 2011] than, for instance, the Kolmogorov-Smirnov test [Pearson and Hartley 1972]. We conduct a Levene test [Levene 1960] to check for equal or unequal variances.

We used the 64-bit R package in version 3.1.2.[5] for the analysis. 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. The raw data, the R scripts, and our results are available as part of our experimental data package [Fittkau et al. 2015].

RQ1 (Time Spent)

We start by checking the null hypothesis $H1_0$. On the left side of Figure 4 a box plot for the time spent is displayed. In Table 2 the differences between the mean values of On-Screen Model and Physical Model are shown.

The Shapiro-Wilk test for normal distribution in each group succeeds and hence we assume normal distribution. The Levene test also succeeds and hence we assume equal variances between both groups. The Student's t-test reveals a probability value of 0.5299 which is larger than the chosen significance level and we fail to reject the null hypothesis $H1_0$.

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

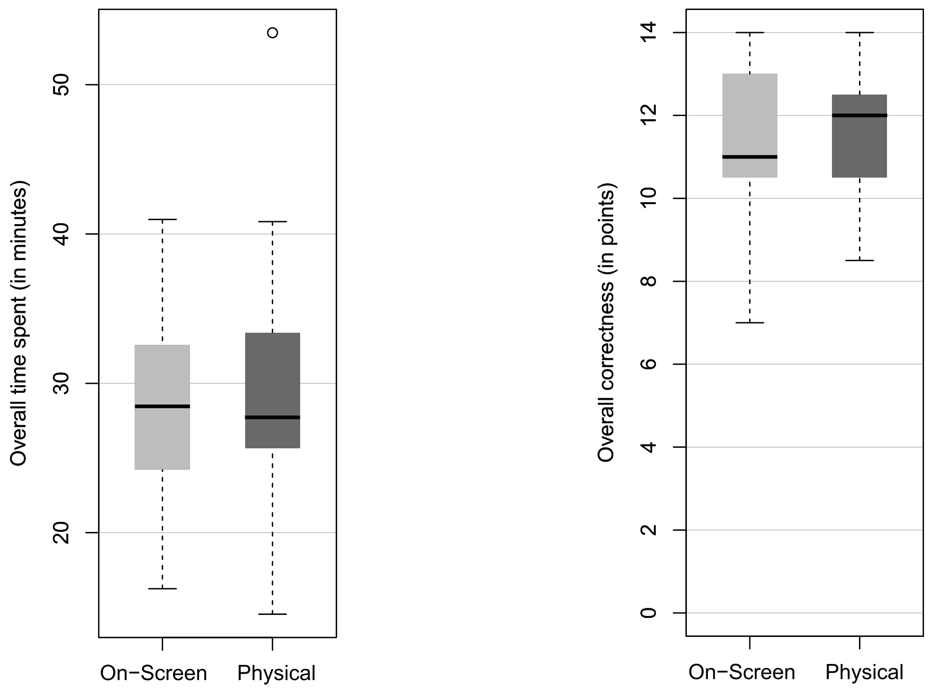

Figure 4. Overall time spent and correctness for our experiment

RQ2 (Correctness)

Next, we check the null hypothesis $H2_0$. On the right side of Figure 4 a box plot for the overall correctness is shown. Similar to RQ1, the Shapiro-Wilk tests and Levene test succeed. The Student's t-test reveals a probability value of 0.4698 which is larger than the chosen significance level and we fail to reject the null hypothesis $H2_0$.

4.5 Discussion

The time spent in the experimental group is slightly higher than the time spent in the control group. However, since we failed to reject $H1_0$, this could also be caused due to randomness. The analysis also reveals a slightly higher correctness in the experimental group. However, since we failed to reject $H2_0$, this could also be caused due to randomness.

Since time spent and correctness of the solutions are in a close range, we conclude that there is only a non-determining overall impact when using physical models for our task set. However, analyzing each task reveals that there is an actual impact in four tasks as can be seen in Figure 5. However, the effects of each task compensate each other for our chosen task set resulting in a close ranged overall result.

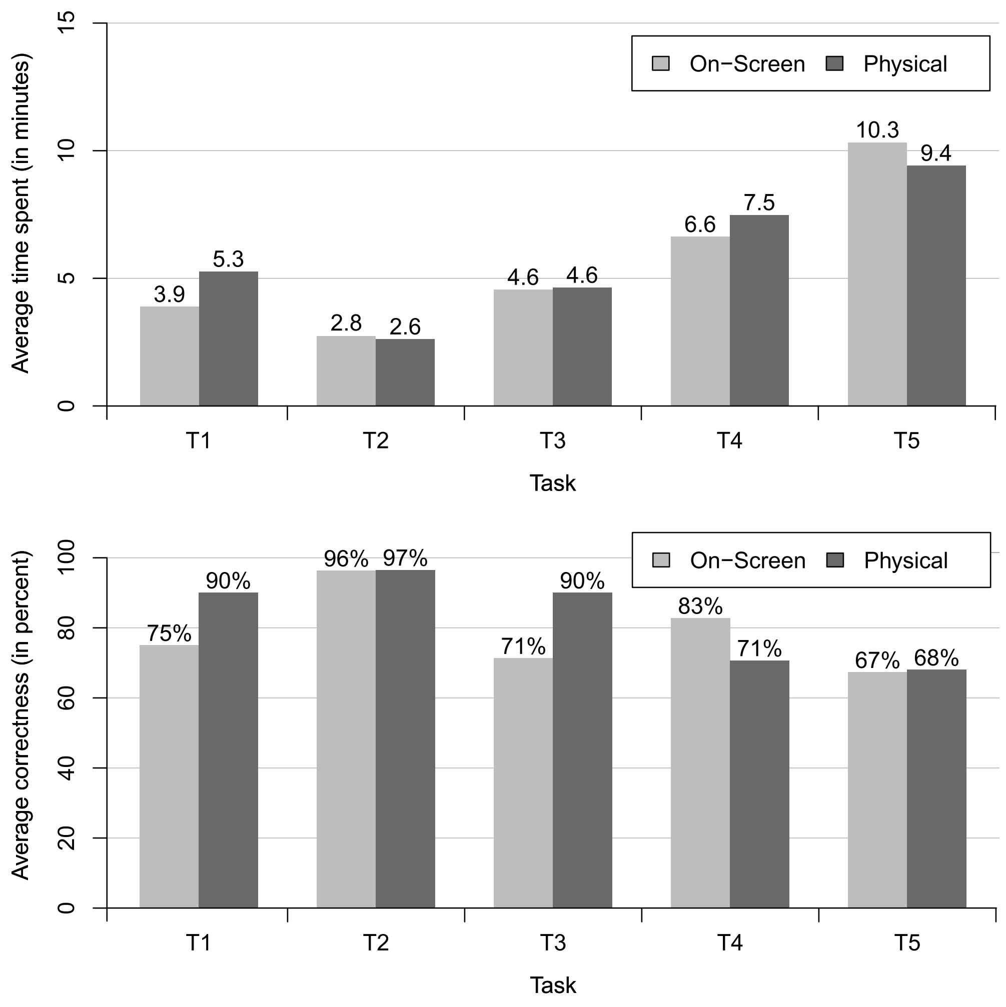

Figure 5. Average time spent per task and average correctness per task

To investigate the impact of using physical models in each task and the reasons for this impact, we conducted an in-depth analysis of the camera and screen recordings. In the following, used strategies and possible differences between the two groups are described.

T1

The control group often used the rotation feature to find the highest class. However, they sometimes rotated the model longer than required for the task. Since the experimental group has never seen anything similar to our physical models, the participants often first investigated the whole model. We assume this as the reason for the increased time needed for this task. The correctness in the experimental group is increased by 15%. We attribute this fact to an easier navigation to find the highest class. Furthermore, the control group often missed the second part of the task, i.e., counting the classes in the contained package. In comparison, the experimental group was more attentive to the task description.

T2

Often the On-Screen Model group turned the model to have a bird's-eye view for a better view of the packages and their contained classes. Furthermore, often the gestures (if any were conducted) were only implied such that the monitor screen was not touched. In the Physical Model group, the gesticulation was more pronounced. One team partner often touched the physical model and showed his teammate the package he was talking about. Furthermore, the package path was often dictated to the partner for writing down the answer, while going through the path having the fingers on the labels (see, for instance, camera recording of Team 6). However, the task might have been too small or easy to show any effect of the gesticulation in the experimental group.

T3

We often observed decisions via exclusion procedure in both groups. However, the experimental group outperformed the control group by 19% in average correctness while using roughly the same time. In the control group, the team partner often had to search for the package name that the teammate was talking about. In contrast, the experiment group often used gestures to show the package under investigation. Furthermore, all packages could be clearly seen without scrolling or rotating. A further reason for the outperformance might be that the package hierarchies were more clear to perceive.

T4

Finding the most occurring package name was more difficult for the experimental group, which needed more time for a lower correctness. We assume that the uneven size of the labels on the physical model causes problems in reading. Especially, the solution package name "rule" is harder to read on the physical model. In contrast, the labels in the on-screen model all have the same size.

T5

The control group required more time for roughly the same correctness score as the experimental group. We often observed that both team partners used gestures during discussion in the Physical Model group – also in parallel. In contrast, it was harder for the On-Screen Model group to use gestures in parallel since one arm often occludes the screen. These parallel gestures could have increased the efficiency to solve the task in the experimental group.

Summary

In summary, we observed a higher amount of gestures in the experimental group compared to the control group. These were used to communicate with the team partner and reading the package paths. Difficulties with the physical model were encountered due to less readable labels.

To summarize the impact of using physical models in our task set: Two tasks (T3 and T5) were positively influenced by the physical model. In contrast, T4 was negatively influenced. T2 did not show any differences in using either the on-screen or the physical model. The achieved correctness in T1 increased with the physical model but also the time spent increased, leading to no clear statement of the impact.

4.6 Threats to Validity

In this section, we discuss the threats to internal and external validity [Shadish et al. 2002; Juristo and Moreno 2010; Wohlin et al. 2012] that might have influenced our results.

4.6.1 Internal Validity

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

Subjects One threat is that the subjects' experience might not have been fairly distributed across the control group and the experimental group. To mitigate this threat, we randomly assigned the subjects to the groups. These random assignment resulted in a fair self-assessed experience distribution as described in Section 4.1.5.

Another threat is that the subjects might not have been properly motivated. In addition to the lottery, the students received only bonus points for the participation and thus were not forced to take part. Furthermore, we encountered no unmotivated

Table 3. Debriefing Questionnaire Results for our Experiment 1 is better – 5 is worse

	On-Screen		Physical	
	mean	stdev.	mean	stdev.
Time pressure (1-5)	2.04	0.65	2.04	0.79
Tool speed (1-5)	1.48	0.64	–	–
Tutorial helpfulness (1-5)	2.11	0.85	2.20	1.08
Tutorial length (1-5)	3.30	0.47	3.20	0.50
Achieved PMD comprehension (1-5)	3.00	0.83	3.08	0.76
Perceived task difficulty (1-5)				
T1	1.74	0.66	2.12	0.73
T2	2.07	0.73	2.08	0.57
T3	2.74	0.71	2.88	0.53
T4	3.56	0.70	3.68	0.48
T5	3.41	0.69	3.60	0.65

user behavior in the screen recordings with the exception of the four teams scoring below six points that we have excluded from our analysis (Section 4.4). Hence, we assume that the remaining teams were properly motivated.

A further threat is that the subjects might not have been sufficiently competent. At the beginning of the questionnaire, most teams stated beginner or regular programming experience. This should be sufficient for our rather easy task set.

Tasks The tasks might have been biased towards one kind of model. We mitigated this threat by disabling some features in the on-screen model to provide equal functionality.

A further threat is the incorrect or biased rating of the solutions. We reduced this threat by conducting a blind review where two reviewers independently reviewed each solution. There were seldom discrepancies in the ratings with at most one point suggesting a high inter-rater reliability.

Another threat is that the tasks might have been too difficult. Teams from both groups achieved the maximum score in each task. Furthermore, the average perceived task difficulty, shown in Table 3, is never between difficult (4) and too difficult (5).

Miscellaneous The results might have been influenced by time constraints that were too loose or strict. Contrary, the average perceived time pressure was slightly above little (2) for both groups and thus the time pressure well fitted.

The experimental group had to switch between the physical model and the keyboard. This could have led to a disadvantage of this group. This threat is mitigated by the fact that one team member could stay at the keyboard and the other one could tell him what he should write.

Another threat concerns the possible different quality of the tutorials. In both groups, the teams had the possibility to continue to use the tutorial until they felt confident in their understanding of the semantics. Furthermore, both groups had the same tutorial text except the explanation of interaction with the on-screen model.

4.6.2 External Validity

Experimenting with only one single object system is not representative for all available systems. Therefore, further experiments with different object systems should be conducted. Another threat concerns the program comprehension tasks, which might not reflect real tasks. We used the framework of [Pacione et al. 2004] to define the task set and to mitigate this threat.

Our subjects were made up of students. Therefore, they might have acted different to professional software engineers. Replications with professionals should be conducted to quantify this impact. Furthermore, the teams were made up of only two persons. The impact of using physical models in larger teams should be investigated in further experiments.

5 Related Work

Since we are – to the best of our knowledge – the first creating physical models of software visualizations, there is only related work to our visualization and evaluation methodology. First, we discuss related work concerning the city metaphor. Afterwards, we describe virtual reality approaches, which also aim for a more natural impression and intuitive navigation in software visualizations. Finally, software visualization experiments, which compare themselves to other software visualizations, are discussed. Since out of our focus, we do not discuss experiments comparing software visualizations to IDEs and refer to [Wettel et al. 2011] for more details.

5.1 City Metaphor

Knight and Munro [Knight and Munro 2000] were the first to transfer the city metaphor to software visualization. In their visualization *Software World*, the buildings represent methods, and classes are mapped to districts.

Panas et al. [Panas et al. 2003] developed a city metaphor where classes are mapped to buildings and a city represents one package. They focused on providing a visualization looking as real as possible, e.g., with trees and photo-realistic textures.

Wettel and Lanza [Wettel and Lanza 2007] created the tool *CodeCity* which also uses the city metaphor. In addition to Panas et al. [Panas et al. 2003], they use the width and depth of the buildings to represent, for instance, the amount of attributes of a class. Furthermore, the package hierarchy is represented by stacking districts similar to our ExplorViz visualization.

EvoSpaces, developed by Alam and Dugerdil [Alam and Dugerdil 2007], introduces a day and night mode utilizing the city metaphor. While the day mode provides information gathered from static analysis, the night mode displays dynamic information as connections between each building.

In contrast to the aforementioned approaches, we enable the user to create physical models from the on-screen presentation to facilitate, for instance, more intuitive navigation.

5.2 Virtual Reality for Software Visualization

Imsovision [Maletic et al. 2001] aims at representing object-oriented software in a virtual reality environment. The user is tracked with electromagnetic sensors attached to the shutter glasses and a *wand* which is used for the 3D navigation.

SykscrapAR [Souza et al. 2012] is an augmented reality approach employing the city metaphor to visualize software evolution. The user can interact with a physical marker platform in an intuitive way while the actual visualization can be seen only on the monitor.

Contrary to both approaches, our physical models do not require any additional devices once they are created.

5.3 Experiments Comparing Software Visualizations

Storey et al. [Storey et al. 1997] compared three software visualizations in an experiment. The authors performed a detailed discussion of the tools' usage but provided no quantitative results.

Lange and Chaudron [Lange and Chaudron 2007] investigated the benefits of their enriched UML views by comparing them with traditional UML diagrams. In contrast, we compare the impact of using physical models on program comprehension.

6 Conclusions And Outlook

In this paper, we presented the vision of transferring the advantages of physical, tangible models to support software engineering. We described four potential usage scenarios, and investigated – as a proof of concept – the impact of using physical models on program comprehension in a team-based scenario by conducting a first controlled experiment.

In our experiment, we identified two tasks that benefit from using physical models in comparison to using on-screen models. However, our experiment resulted in no overall impact neither on time spent nor on correctness of solutions to the program comprehension tasks, since the effects of each task compensate each other for our chosen task set.

Our in-depth analysis of the strategies supports our hypothesis that physical models provide an appropriate, complementary communication basis and increase interaction when solving comprehension tasks in teams. In the analysis, we observed an increase in the amount of performed gestures when using the physical model.

We provide a package containing all our experimental data to facilitate the verifiability and reproducibility for replications and further experiments [Crick et al. 2014]. It contains the employed version of ExplorViz v0.6-exp (including source code and manual), input files, STL files for 3D-printing the used models, tutorial materials, questionnaires, R scripts, dataset of the raw data and results, and 112 screen and camera recordings of the participant sessions. The package is available online [Fittkau et al. 2015] with source code under the Apache 2.0 License and the data under a Creative Commons License (CC BY 3.0).

In the context of program comprehension, future work is manifold. To validate our observations in the controlled experiment for program comprehension and provide a higher external validity, further replications should be conducted. Further approaches to overcome current limitations of our physical models should be investigated. For instance, the naive mapping of communication lines to the physical models creates a set of confusing, overlying lines – since missing the possibility to interactively hide communication. Furthermore, our experiment design should be tested with a larger team size where gesticulation and communication can have a higher impact, and with professionals as subjects. Further experiments should examine the combination of using a physical model and an on-screen model compared to solely using an on-screen model. Likewise, physical models should be compared to Virtual Reality.

On a more general perspective, we only investigated a part of the first usage scenario of the four envisioned scenarios revealing a promising future research direction. The other scenarios should also be evaluated and investigated in, for instance, controlled experiments.

Another direction is the creation of physical models of other 3D software visualization metaphors, e.g., trees [Kleiberg et al. 2001], spheres [Balzer et al. 2004], or solar systems [Graham et al. 2004]. Every usage scenario might reveal different results with different representation of physical models.

Bibliography

[Alam and Dugerdil 2007] S. Alam and P. Dugerdil. EvoSpaces visualization tool: Exploring software architecture in 3D. In: *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE 2007)*. 2007, pages 269–270. (Cited on page 23)

[Ball and Eick 1996] T. Ball and S. G. Eick. Software visualization in the large. *Computer* 29.4 (1996), pages 33–43. (Cited on page 3)

[Balzer et al. 2004] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz. Software landscapes: visualizing the structure of large software systems. In: *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association. 2004, pages 261–266. (Cited on page 25)

[Basili and Weiss 1984] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE TSE* SE-10.6 (Nov. 1984). (Cited on page 9)

[Cornelissen et al. 2009] 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)*. 2009, pages 100–109. (Cited on pages 9, 10)

[Crick et al. 2014] 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, pages 1–4. (Cited on page 24)

[Di Lucca and Di Penta 2006] 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, pages 229–234. (Cited on page 9)

[Di Penta et al. 2007] 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)*. 2007, pages 281–285. (Cited on page 9)

[Ehlers et al. 2011] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In: *Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011)*. ACM, June 2011, pages 197–200. (Cited on page 13)

[Fittkau et al. 2013] 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)*. Sept. 2013. (Cited on page 6)

[Fittkau et al. 2015] F. Fittkau, E. Koppenhagen, and W. Hasselbring. Experimental Data for: Research Perspective on Supporting Software Engineering via Physical 3D Models. Zenodo.org. doi: 10.5281/zenodo.18378. June 2015. (Cited on pages 3, 16, and 24)

[Goldin-Meadow 2005] S. Goldin-Meadow. Hearing gesture: How our hands help us think. Harvard University Press, 2005. (Cited on pages 3, 4)

[Graham et al. 2004] H. Graham, H. Y. Yang, and R. Berrigan. A solar system metaphor for 3D visualisation of object oriented software metrics. In: *Proceedings of the Australasian Symposium on Information Visualisation (APVIS 2004)*. Australian Computer Society, Inc., 2004, pages 53–59. (Cited on page 25)

[Herndon et al. 1994] K. P. Herndon, A. van Dam, and M. Gleicher. The challenges of 3D interaction: A CHI ’94 Workshop. *SIGCHI Bull.* 26.4 (Oct. 1994), pages 36–43. (Cited on page 3)

[Jedlitschka and Pfahl 2005] 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. (Cited on page 9)

[Juristo and Moreno 2010] N. Juristo and A. M. Moreno. Basics of software engineering experimentation. Springer, 2010. (Cited on page 20)

[Kitchenham et al. 2002] 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* 28.8 (2002), pages 721–734. (Cited on page 9)

[Kleiberg et al. 2001] E. Kleiberg, H. Van De Wetering, and J. J. Van Wijk. Botanical visualization of huge hierarchies. In: *IEEE Symposium on Information Visualization*. IEEE Computer Society. 2001. (Cited on page 25)

[Knight and Munro 2000] C. Knight and M. Munro. Virtual but visible software. In: *Proceedings of the IEEE International Conference on Information Visualization (IV 2000)*. IEEE, 2000, pages 198–205. (Cited on pages 3 and 22)

[Koschke 2003] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey. *Journal of Software Maintenance and Evolution: Research and Practice* 15.2 (2003), pages 87–109. (Cited on page 13)

[Lange and Chaudron 2007] 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)*. 2007, pages 221–230. (Cited on page 23)

[Levene 1960] H. Levene. Robust tests for equality of variances. *Contributions to probability and statistics: Essays in honor of Harold Hotelling* 2 (1960), pages 278–292. (Cited on page 16)

[Likert 1932] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology* 22.140 (1932), pages 5–55. (Cited on page 12)

[Maletic et al. 2001] J. I. Maletic, J. Leigh, A. Marcus, and G. Dunlap. Visualizing object-oriented software in virtual reality. In: *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*. Society Press, 2001, pages 26–35. (Cited on page 23)

[Pacione et al. 2004] 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. (Cited on pages 10 and 22)

[Panas et al. 2003] T. Panas, R. Berrigan, and J. Grundy. A 3D metaphor for software production visualization. In: *Proceedings of the 7th International Conference on Information Visualization (IV 2003)*. IEEE Computer Society, 2003, pages 314–320. (Cited on page 23)

[Pearson and Hartley 1972] E. Pearson and H. Hartley. Biometrika Tables for Statisticians. 2nd edition. Cambridge University Press, 1972. (Cited on page 16)

[Rajlich and Cowan 1997] 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, pages 160–161. (Cited on pages 8 and 10)

[Razali and Wah 2011] N. Razali and Y. B. Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics* 2.1 (2011), pages 21–33. (Cited on page 16)

[Sensalire et al. 2009] 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)*. Sept. 2009, pages 19–26. (Cited on page 9)

[Shadish et al. 2002] W. R. Shadish, T. D. Cook, and D. T. Campbell. Experimental and quasi-experimental designs for generalized causal inference. Wadsworth – Cengage Learning, 2002. (Cited on page 20)

[Shapiro and Wilk 1965] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika* (1965), pages 591–611. (Cited on page 16)

[Souza et al. 2012] R. Souza, B. Silva, T. Mendes, and M. Mendonca. SkyscrapAR: an augmented reality visualization for software evolution. In: *Proceedings of the II Brazilian Workshop on Software Visualization (WBVS 2012)*. 2012. (Cited on page 23)

[Storey et al. 1997] 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, pages 12–21. (Cited on page 23)

[Teyseyre and Campo 2009] A. Teyseyre and M. Campo. An overview of 3D software visualization. *IEEE Transactions on Visualization and Computer Graphics* 15.1 (Jan. 2009), pages 87–105. (Cited on page 3)

[Ware and Mitchell 2005] C. Ware and P. Mitchell. Reevaluating stereo and motion cues for visualizing graphs in three dimensions. In: *Proceedings of the 2nd Symposium on Applied Perception in Graphics and Visualization (APGV 2005)*. ACM, 2005, pages 51–58. (Cited on page 3)

[Ware et al. 1993] C. Ware, K. Arthur, and K. S. Booth. Fish tank virtual reality. In: *Proceedings of the INTERACT 1993 and Conference on Human Factors in Computing Systems (CHI 1993)*. ACM, 1993, pages 37–42. (Cited on page 3)

[Wettel 2010] R. Wettel. Software Systems as Cities. PhD thesis. University of Lugano, 2010. (Cited on page 6)

[Wettel and Lanza 2007] R. Wettel and M. Lanza. Visualizing software systems as cities. In: *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*. IEEE. 2007, pages 92–99. (Cited on page 23)

[Wettel et al. 2011] 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, pages 551–560. (Cited on pages 9, 10, and 22)

[Wohlin et al. 2012] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. Experimentation in software engineering. Springer, 2012. (Cited on page 20)