

Effectiveness of 3D Code Visualization Techniques over Time

Alice Truong
University of Zurich
Zurich, Switzerland
alice.truong@uzh.ch

Sarah Zurmühle
University of Zurich
Zurich, Switzerland
sarah.zurmuehle2@uzh.ch

ABSTRACT

3D code visualization tools are widely discussed in the literature. They provide a way to represent software structure and components. Enhancing program comprehension and representing software structure is a relevant topic for software developers and 3D code visualization techniques are a possibility to contribute to this research field. There are multiple studies that analyze how 3D code visualization techniques work and if they are effective to use. Our study focuses on the effectiveness of three such techniques: *CodeCity*, *Code Park* and *Evo-Streets*. We try to find out if these techniques are better suited in solving comprehension tasks than using no such tools. Furthermore, we try to analyze the usage of these tools over time to be able to draw a conclusion about the effectiveness of these techniques in a long-term spectrum. There are no studies known to us so far that analyze the effectiveness of 3D code visualization tools over time. We hypothesize that our chosen techniques increase the efficiency of developers in completing comprehension tasks over multiple usage iterations. Software developers get demotivated quickly using new tools if they do not show a positive effect immediately. If our hypothesis is correct, it might convince software developers to use such techniques, even though the positive effect is not visible instantaneously.

KEYWORDS

3D Code Visualization, Code Park, CodeCity, Evo-Streets, Time Interval, Apache Software Foundation, Learning Curve

ACM Reference Format:

Alice Truong and Sarah Zurmühle. 2018. Effectiveness of 3D Code Visualization Techniques over Time. In *Proceedings of Data Science for Software Engineering (DSforSE'18)*, Alice Truong and Sarah Zurmühle (Eds.). ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The usage of 3D code visualization has become a frequently discussed topic. They enable developers to represent their software structure and components. Due to their potential in helping developers to better understand software systems, these techniques have become especially relevant. There are multiple studies that research the usage and effectiveness of such methods (e.g. the studies of Merino et al. [5], Khaloo et al. [3] or Steinbrückner and Lewerentz [11]). These studies mainly analyze the different techniques

with a single experiment and draw their conclusion from these results. Furthermore, most studies only discuss one tool per study or experiment and do not compare multiple techniques with each other.

However, in this study, we try to analyze different 3D code visualization techniques over time intervals. This allows us to measure how the efficiency of users develop over time while using a specific 3D code visualization technique. At the end, the convergence of the different techniques can be analyzed and compared. More precisely, the goal of this study is to investigate whether 3D code visualization techniques increase developers' efficiency in completing comprehension tasks in a long-term usage. Additionally, we try to find out whether different visualization techniques can have different learning curves.

We plan to conduct a controlled experiment to check our hypotheses. We will analyze the effectiveness and the learning curves of three different 3D visualization techniques: *Code Park*, *CodeCity* and *Evo-Streets*. These three tools share a common feature; they map code parts to elements of a city, e.g. streets or buildings. We decided to use multiple visualization tools, so that we can compare them to each other. In our experiment we ask the participants to perform comprehension tasks. Separate groups of participants will perform the tasks for each visualization technique. To effectively examine the long-term effect of using the 3D visualization techniques, the experiment will be conducted in five sessions and for each session the participants have to solve the tasks on a different software project.

We believe that developers need time to get familiar with specific visualization techniques. With the amount of times they use a visualization technique, the efficiency in completing comprehension tasks should increase. Therefore, the effect of using 3D code visualizations might not show itself immediately, but in a long-term spectrum. If this is the case, showing the results to developers might motivate them to get to know and use certain 3D software visualization types. When comparing the different 3D visualization techniques, we believe that they take different amounts of time to learn and get used to them. So, they will show different learning curves. If this hypothesis is correct, we may provide developers recommendations about which techniques should be used for which kind of projects. For example, 3D visualization types with a steeper learning curve may be more suitable for long-term projects.

This paper is organized as follows: In section 2 the related work is presented along with a description of the used 3D code visualization techniques including their advantages and disadvantages. Section 3 describes our methodology approach entailing our research questions, hypotheses, research method, experimental approach, data collection and threats to validity. Finally in section 4 we draw conclusions.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

DSforSE'18, Fall 2018, Zurich, Switzerland

© 2018 Copyright held by the owner/author(s).

ACM ISBN 123-4567-24-567/08/06.

https://doi.org/10.475/123_4

2 RELATED WORK

3D Software visualization techniques provide a way to represent software structure and components. They are especially important due to their potential in helping developers to better understand software systems. For an overview of the general topic of 3D software visualization we refer to Teyseyre and Campo's [13] paper.

The general field of representing software structure and program comprehension is a relevant topic. There are multiple studies that tackle this topic. For instance, in the study of Maletic and Marcus [4], they investigated a cheap but still sufficient way to extract relevant semantic information (domain issues of software systems). This combination of semantic and structural information are important to support comprehension tasks of software systems. On the other hand, Storey et al. [12] followed a more general approach. They describe the hierarchy of cognitive issues that may arise while working on software exploration tools, analyzed cognitive design elements and concluded on how these elements can be used for software exploration. Furthermore, Valverde and Solé [15] created a network approach for object-oriented software systems with small-world behaviour (small distance between classes) which spots universal software organization patterns. Object-oriented software often build a heterogeneous and hierarchical network. Because of this fact, software developers find it difficult to get a clear overview of the individual components of software systems [9]. Therefore, the results of Valverde and Solé [15] and other authors that analyze software structures are relevant in the field of software comprehension.

More specifically, in the field of 3D code visualization techniques, there exist multiple studies that research the usage and effectiveness of such tools as well. For example, Merino et al. [5] investigated if the usage of augmented reality can overcome the usability issues of 3D visualizations (navigation, selection, occlusion and text readability) and if it can increase developers' efficiency. Rüdél et al. [6] compared among other things the efficiency of *Evo-Streets* (see section 2.3) in two variants of immersive virtual reality systems. On the other hand, Wettel et al. [21] conducted a controlled experiment to find out if their *city metaphor* [18] approach implemented in the tool *CodeCity* (see section 2.1) is at least as efficient as the state of the art at the time of the study. Wettel and Lanza [19] used a *city metaphor* approach as well to test their hypothesis. They argued that habitability (the feeling that software is an actual physical space) is a very relevant aspect in the field of software comprehension and that 3D visualizations enable to enhance it.

These kind of studies drew a conclusion about the effectiveness of specific 3D visualization techniques on their experiments which they conducted at one time. Furthermore, we could not find any studies that compare multiple 3D code visualization techniques with each other. There are studies that use two tools for their experiment, e.g. Merino et al. [5], but they did not compare them.

However, with the results of our study we can draw conclusions about the effectiveness of the examined 3D code visualization techniques over time. So our findings are also valid for the usage of 3D visualization tools in a long-term usage. If our hypothesis that

the effectiveness of using these techniques increases over time is correct, this study may encourage developers to actively use the 3D visualizations. We believe that, if there is a long-term increase of effectiveness, the results can motivate developers to get to know and use certain 3D software visualizations types, even if the time it takes to get familiar with the visualizations doesn't pay off right from the start.

By comparing different visualization techniques, we can additionally investigate whether their learning curves differ. We believe, that some techniques may take more time to learn, but may have a better long-term effect than other techniques. So, by examining the learning curves of the different techniques and finding different learning curves, we can provide visualization type recommendations for short-term and long-term projects.

For our experiment, we will use three different 3D code visualization techniques: *CodeCity*, *Code Park* and *Evo-Street*. There exist other techniques as well, however, we choose these three techniques because they have some similarities (mapping of code parts to elements of a city) and therefore are easy to compare. When the three visualization techniques are too different, there exists the threat that the study gets biased because of the many differences of the analyzed tools. Therefore, we decided to use similar 3D code visualization techniques. In the following subsections, we will discuss the three tools we choose by comparing their advantages and disadvantages.

2.1 CodeCity

The 3D code visualization tool *CodeCity* represents software based on a city metaphor. Buildings in the city are used to visualize classes and districts the packages. The width and height of the buildings are used to encode software metrics. *CodeCity* maps the amount of attributes for the classes on the width and length and the number of methods is mapped to the height [20, 21].

One advantage of using a city metaphor for code visualization is that the users have a certain degree of familiarity with cities. The buildings and street mappings have an understandable concept and a clear orientation. Furthermore, a software system is comparable to a large city by the means that both need a step by step approach to fully understand the whole construct. This mapping avoids oversimplification of software systems which can lead to errors while working with them [20].

On the other hand, the city metaphor is bound to elements which are connected to a city and nothing more. So, the mapping between the code and visual elements is limited. Furthermore, *CodeCity* struggles with some usability issues, namely navigation inside the visualization, selection of elements (especially when they are small or occluded), bigger buildings occluding smaller buildings and text readability in 3D space [5].

2.2 Code Park

The 3D code visualization tool *Code Park* represents code in an environment which resembles a 3D game. It tries to be easily understandable and appealing by inexperienced users, e.g. students. The code is mapped to a 3D room-like environment where each room is

mapped to a class. The size and color of the room represents the actual size of the class in the code. On the wall inside the room are all constituent parts of the class presented, written as programming code. The color theme used for the walls is the same as Microsoft Visual Studio's dark theme. The user can use code overview to interact with the code. Also, the vision of the user is in first-person (inside a room) or bird's eye view mode (outside a room to look down on all available rooms) [3].

Compared to other visualization techniques, Code Park's goal is to enable the user to get familiar with the code base of a software system. It additionally shows the source code integrated in the 3D environment together with metaphorical representations. To achieve this, this technique uses methods to reduce the user's cognitive load. So, the user can view actual code while using the visual mappings to get an overview of the whole project, which is the main advantage of this visualization technique [3].

However, in our opinion, with the structure of Code Park, it is only possible to see the size of a class and which classes are grouped together, but it is not possible to see how the classes are connected to each other when only locking at the rooms. God classes and anemic classes cannot be spotted immediately because the rooms' sizes are only mapped to the general size of the class and not according to the amount of methods or attributes.

2.3 Evo-Streets

Similar as CodeCity the visualization technique *Evo-Streets* uses a city metaphor for the visualization of software systems. Classes are also represented by buildings. However, instead of displaying packages as districts, *Evo-Streets* uses a hierarchical street system to represent the structure. The hierarchy of the structure is modeled in a tree structure with each street representing a certain subsystem. These subsystems can contain further subsystems which are displayed in branching streets [11].

Compared to CodeCity and Code Park, *Evo-Streets* introduces a hierarchical street mapping which enables the user to better see the relations between classes in the software system. Decomposition hierarchy and element properties can be easily represented. Also, a hierarchical street system is adaptable to changes because new elements can be easily added to the tree structure. This feature also enables to track the development time of e.g. module creation. The usage of tertiary models, e.g. coloring of the elements in the tree, enlarges the possibility to visualize different software characteristics [11].

However, the tree-structure of *Evo-Street* have the same occlusion-problem like CodeCity: elements block the view of other elements. This problem occurs especially in large software systems [2]. We also think, that a tree structure might get confusing when it represents a large software project. If one branch has multiple leaves and subbranches, it gets difficult to keep an overview over the whole project.

2.4 Comparison

To sum up, while CodeCity uses a strict city metaphor, Code Park also focuses on integrating actual programming code inside the

visualization. The street connection of the different buildings in CodeCity resemble more the hierarchical structure of *Evo-Streets*, however, *Evo-Streets* tries to strictly stick to a tree structure, while CodeCity is more focused on representing an actual city with districts. Furthermore, Code Park does not use any connections between their rooms to represent relationships. An overview of the advantages and disadvantages of these three techniques is listed in Table 1. Furthermore, Table 1 includes an image which shows what the visualizations look like for each technique .

3 METHODOLOGY

We plan to conduct a controlled experiment to find out if 3D code visualization techniques can increase developers' efficiency in a long-term usage and whether different visualization techniques can have different long-term effects. We will analyze the effectiveness and the learning curves of three different 3D visualization techniques: *CodeCity*, *Code Park* and *Evo-Streets*. The performance of participants using visualization techniques is additionally compared to a baseline, in which only the code is provided and no visualization technique is used.

3.1 Research Questions

The overall goal of this study is to find out if 3D code visualization techniques can increase developers' efficiency (completion time and correctness of comprehension tasks) in a long-term usage. Additionally, we investigate whether different visualization techniques can have different long-term effects. Accordingly, we defined the following research questions:

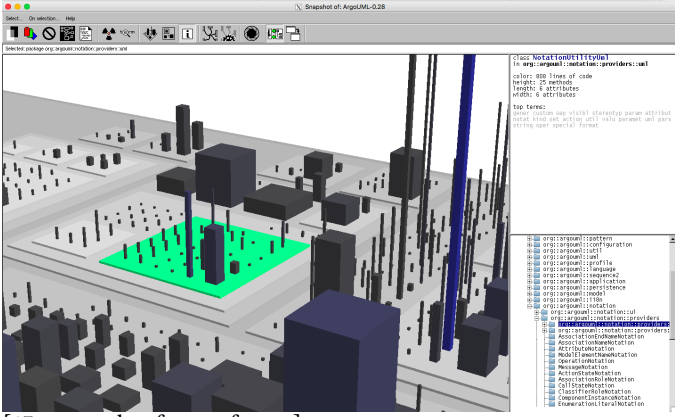
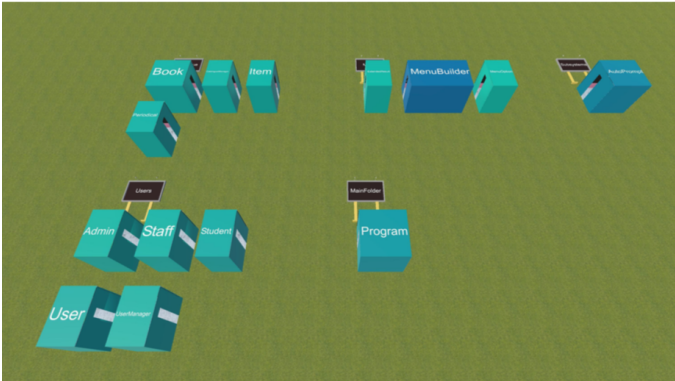
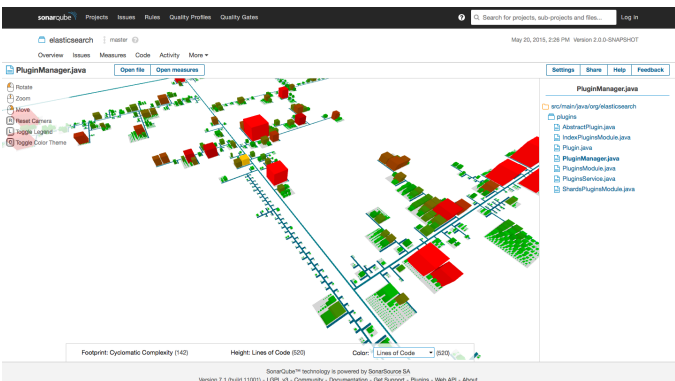
RQ1 *Does the long-term usage of 3D visualization techniques increase the developers' efficiency in completing comprehension tasks?*

We believe that developers need time to get familiar to specific visualization techniques. Therefore, we try to find out whether the efficiency in completing comprehension tasks increases with the amount of times they use a visualization technique. If this can successfully be shown, developers will be more motivated to learn and use certain 3D software visualization types.

RQ2 *Do different visualization techniques have different learning curves?*

When comparing the different 3D visualization techniques, we believe that they take different amounts of time to learn. So, the visualization techniques will show different learning curves. A learning curve in this case graphically represents the efficiency of completing comprehension tasks on the y-axis and the number of usage times of a visualization technique on the x-axis (see Figure 1). Ideally, it shows how the more a developer uses a visualization tool, the better they get at performing comprehension tasks. If we can show that visualization techniques have different learning curves, we may provide developers recommendations about which

Table 1: Comparison of the three 3D code visualization techniques used in this study

Name	Appearance	Advantages	Disadvantages
CodeCity	 <p>[17, screenshot from software]</p>	<ul style="list-style-type: none"> familiar design understandable concept clear orientation no oversimplification of software systems [20] 	<ul style="list-style-type: none"> limited mapping from software elements to components usability issues: navigation, selection, occlusion, text readability [5]
Code Park	 <p>[3, p. 1]</p>	<ul style="list-style-type: none"> source code integrated in the 3D environment reduction of user's cognitive load [3] 	<ul style="list-style-type: none"> limited functionality of size of classes and class groupings god classes and anemic classes cannot be spotted immediately
Evo-Streets	 <p>[10, Screenshot of live demo]</p>	<ul style="list-style-type: none"> hierarchical street mapping enables better recognition of relations easily adaptable to changes track of development time [11] 	<ul style="list-style-type: none"> problem of occlusion [2] confusing structure if hierarchical tree gets too large

techniques should be used for which kind of projects. For example, 3D visualization types with a steeper learning curve in the beginning may be more suitable for long-term projects than short-term projects.

3.2 Hypotheses

Learning and getting familiar with 3D software visualization techniques takes time. We hypothesize that the efficiency in completing comprehension tasks increases with the amount of time developers use a specific technique. Therefore, the effect of using 3D code visualization might not show itself immediately, but in a long-term

spectrum. Additionally, we believe that different 3D code visualization techniques take different amounts of time to learn and get used to, which leads to different learning curves. Consequently, we define the following two hypotheses for the experiment:

H1 *Using a 3D visualization technique, such as CodeCity, Code Park or Evo-Streets, increases developers' efficiency in completing comprehension tasks in the long term.*

H2 *Different 3D visualization techniques, such as CodeCity, Code Park and Evo-Streets have different learning curves.*

3.3 Research Method

Participants are asked to perform predefined comprehension tasks on our website. We choose five different open-source projects that the participants of our study should analyze and use to solve code comprehension tasks.

Design. Our experiment follows a mixed factorial design. The 3D visualization type is a between-subjects variable, which means that we use separate groups of participants for each visualization technique and the baseline. To effectively examine the long-term effect of using the 3D visualization techniques, we have the software project as a within-subjects variable. For each visualization technique a participant group has to solve comprehension tasks for five different open-source projects. For the participants it means, that each of them have to solve the predefined comprehension tasks using the assigned visualization technique for each open-source project. The experiment will be conducted with five internet-based sessions with a total of 60 participants, one for each open-source project: In a session the participants have to solve the comprehension tasks for one project using the assigned visualization type on their own computer.

Because the participants can do the tasks on their personal computers wherever they please, we also need a controlled lab experiment as a comparison and baseline. This ensures that the results are not too biased by the participants' different environments and devices. We plan to conduct the controlled lab study with 12 participants. They have to solve the tasks of the first iteration of the main experiment at the same time and in the same place. We decided to test the lab experiment for the first iteration, because if there is a big difference between the lab and the main experiment, there would still be enough time to adapt the main experiment if needed. If we conduct the lab experiment at a later stage, this would most likely not be possible anymore.

Procedure. Every participant receives the same written instructions on the experiment procedure and on how to submit solutions before they start to solve the tasks. The 3D visualization of the code is provided on our website and they have to commit their results on the platform. After solving the tasks in the first session, they get another software project and are asked to solve the same comprehension tasks in the second session. This continues until they have solved the tasks for all of the five open-source projects. Since each visualization type is performed five times (five different

projects) we can compare the performance change of the participants over the usage time and check whether the visualization types have different learning curves. By using different projects for each iteration we aim to eliminate the possible learning effects of the project. Therefore, if the participants solve the tasks quicker in later iterations, it will be attributed to the fact that they know the visualization type better and not to the increased knowledge about the project.

Training. Before the first session we plan to train the participants on how to use their assigned visualization type, so that they will not be completely clueless on how to start in the first session. It is important that the treatment groups are trained the same way, so that it will not affect the results of the experiment.

The training session will be internet-based as well. Because of this design, all participants will not be able to ask questions during the training and experiment. This is unfortunately a drawback of internet-based sessions. The positive aspect of this approach is the fact that participants can solve the tasks whenever they have time. This will increase the probability that we will find enough people.

Duration. Each of the five sessions will be available on different days in order to avoid the fatigue effect which would have had an undesired impact on the results. The training will be on the same day as the first session. The experiment will be conducted in ten days if we plan a one day break between consecutive sessions. We plan to design the sessions so that they on average will not take longer than 30 minutes. The exact time needed however depends on the task completion times of the participants. There is no time restriction for solving the tasks because this would bias the results. So, the average time each participant has to spent would in total be around 2.5 hours.

3.4 Participants

A challenge to conducting this experiment could be getting access to sufficient participants. We will need many participants in order to have a high statistical power and get reliable results. In total, our experiment needs four groups of participants: One for the base condition and three for the different 3D visualization techniques. Due to this challenge, we plan to do an internet-based study with online-hired participants. Conducting the experiment remotely also comes with the advantage that the participants can work on their own computer and in a more natural environment. Giving the participant the feeling of working in a natural environment can reduce or eliminate the Hawthorne effect. In total we plan to hire 60 professional developers as participants for the experiment, which will result in 15 developers per treatment group.

We will select the participants to have a similar distribution of age, gender and programming background in order to control these potential confounding variables. The programming experience will be checked with specific programming tasks, so no participants that lie about their experience are recruited. We will randomly assign the participants to the treatment groups, to distribute variation in participant behavior across different visualization types evenly. With these factors controlled, possible learning curve differences

Table 2: The five Apache projects which we intent to use for the experiment.

Name	Functionality
Apache Tez	building data-flow driven processing run-times [7]
Apache Bigtop	building Hadoop rpm and deb packages [14]
Apache REEF	building run-time management support for the monitoring of tasks [16]
Apache Storm	stream processing [1]
Apache Zeppelin	web-based notebook for data analysis and visualization [8]

found between different visualization techniques will be attributable to the technique itself and not to the participant characteristics. Since each participant only uses one specific visualization technique we can effectively check whether the efficiency of solving comprehension tasks increases with the amount of time they use the visualization.

It is unlikely that we will find enough people that will participate in an experiment that lasts for ten days for free. Therefore, we plan to pay our participants to give them enough motivation to take part in our study. We will ask the developers of the three visualization techniques if they are interested in collaborating with us. Our study will give publicity to them and if our hypothesis is true, it will also advertise their products. So, by funding the payment of our participants, these developers can promote their products and will likely get more people to use their software.

3.5 Open-Source Projects

For our experiment we plan to only use open-source projects from *the Apache Software Foundation*¹. We picked projects which have a similar purpose, namely Big Data analysis. A list of all projects with a short description of their main functionality is shown in Table 2. Using projects only from *the Apache Software Foundation* with similar goals may not provide a varied test-set, however using data from multiple different sources with different purposes can also lead to biases. If the software projects which the participants have to use are too different, we cannot draw a clear conclusion about the usage of the software visualization tools. The reason for this effect includes the fact that the observed behavior could also be a result of the adaption of the user to the new structure and design of the software. Because we think that this drawback is more crucial, we decided to only analyze software with similar goals from the same origin, in our case *the Apache Software Foundation*.

¹<https://www.apache.org>

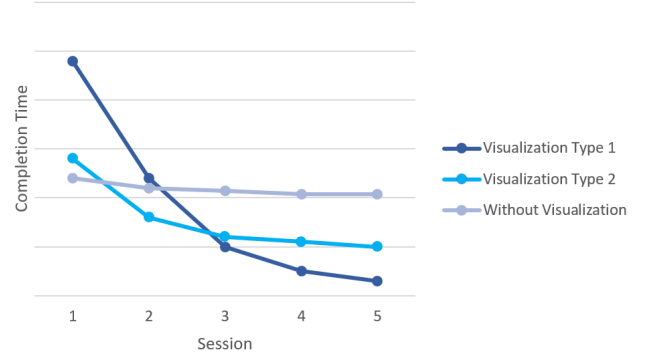


Figure 1: Learning Curves Example

3.6 Data

To measure the efficiency of solving the comprehension tasks we use the completion time and the correctness of each task. The completion time is measured per task, so we measure the time they need from the start of the task until the solution is submitted for each task separately. The completion time and the correctness can on the one hand be used to compare different visualization techniques and on the other to check the long-term effect and the learning curve of each visualization technique.

According to our hypothesis H1, which states that the developers efficiency increases over time, the completion time should decrease and the number of correctly solved tasks should increase with each session when using a 3D visualization type to solve the tasks. Figure 1 illustrates our expectation of possible learning curves measured with the completion time. We expect each curve for conditions in which the participants use a visualization technique to be downward sloping with a flat slope toward the end: There is less to learn about the visualization technique after each session, so the amount improvement should decrease over time. For the baseline condition in which no visualization technique is used, we expect the completion time to slightly decrease as well, due to the gained experience of solving the same tasks in each session. However, the amount of decrease in the baseline condition should be significantly different from the decrease in the other conditions.

Figure 1 additionally shows an example of our hypothesis H2, which states that different visualization types have different learning curves. We believe that different 3D visualization techniques take different amounts of time to learn and get used to. So, they will show different learning curves. For instance, *Visualization Type 1* in the figure takes more effort to learn, however may be more efficient for later sessions than *Visualization Type 2*. By conducting this controlled experiment, we will be able to compare the learning curves of CodeCity, Code Park and Evo-Streets and for example find out which techniques have a steeper learning curve at the beginning, but are more effective in the later sessions. 3D tools with such curves will be more suitable for long-term projects. Learning curves that are flatter at the beginning indicate that developers need less time to learn and get familiar with the visualization type. 3D techniques with such curves might be more suitable for short-term projects.

3.7 Data Analysis

This section will describe the analysis and tests we will perform to check whether our two hypotheses are correct. The data analysis will be described separately for each hypothesis.

H1. H1 states that using a 3D visualization technique increases developers' efficiency when completing comprehension tasks. We plan to do the following analysis for each 3D visualization technique group and the baseline group, which did not use any 3D visualization technique:

Firstly, we will focus on the measure of the completion time. When the results show that the mean completion time decreases after each session, we will perform a repeated measures ANOVA to check whether the differences between the means of completion times are significant across different sessions. Figure 2 shows an overview of this analysis of our experiment. The session is the within-subjects factor with five levels, one open-source project for each session. Since the sessions were conducted on different days, they also represent the timeline. The ANOVA will test whether at least two means of the factor completion time are significantly different. With the ANOVA, however, we will not be able to tell whether the means of all five levels of the variable session are different from each other or just some of them. Therefore, we plan to do post hoc pairwise t-tests, if the ANOVA showed that at least two means are significantly different from each other. The post hoc tests allow us to compare each pair of levels, so we will see which sessions yield significant different means from which sessions. For example, it is possible that only the first session and last session show a significant difference in the mean, so using the post hoc tests allows us to detect this relation. The whole analysis will be repeated with the second measure, the number of correctly answered tasks, to check whether the mean significantly differs over the sessions. The number of correctly answered questions should increase over time according to our hypothesis H1.

Before we perform the repeated measure ANOVA, we have to make sure that the normality assumption holds. This will be done with a Shapiro-Wilk test. If the measure does not have an underlying normal distribution, we plan to run nonparametric tests which do not require the normality assumption to hold: We will run Friedman tests instead of ANOVA and Wilcoxon matched-pairs signed-ranks tests instead of the post hoc pairwise t-tests.

H2. H2 claims that different 3D visualization techniques have different learning curves. As described in section 3.6, a learning curve graphically represents the efficiency of completing comprehension tasks on the y-axis and the number of usage times of a visualization technique on the x-axis.

For the first step, we will compute the learning curves for each visualization type condition and the baseline as in Figure 1. We will check whether the learning curves of the conditions look differently. This graph can also be used for the hypothesis H1: The completion time should decrease over sessions and the number of correct answers increase for all three visualization types.

To statistically check whether the curves differ, we decided to investigate the interaction effect between the sessions and the 3D visualization techniques. Again, the session is the within-subjects

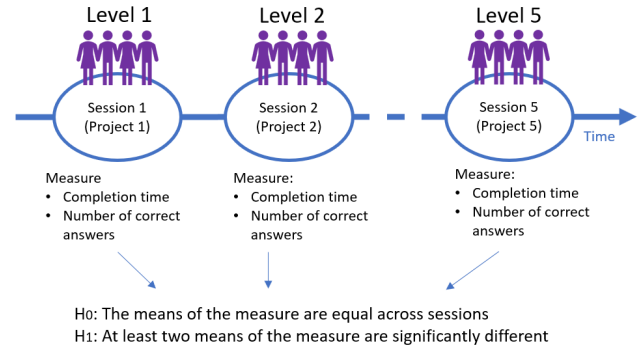


Figure 2: ANOVA Overview

factor with five levels, one open-source project for each session. This time we will include the between-subjects factor, 3D visualization type, which has four levels: CodeCity, Code Park, Evo-Streets and the baseline with no usage of visualizations. The curves look differently, if there is an interaction effect, since it means that the effect of a level of a session depends on the level of the visualization type. If we can reveal an interaction effect, we will be able to claim that the visualization types show different learning curves. As in H1 we will use the repeated measures ANOVA, however, with two named factors instead of only one.

As mentioned for H1 we will use the Shapiro-Wilk test to check whether we have an underlying normal distribution. If not, we decided to use the Aligned Rank Transform (ART) procedure, which is according to Wobbrock et al. [22] a nonparametric equivalent of a repeated measures ANOVA that produces reliable results for main effects as well as interaction effects. We cannot use the Friedman test as we did for H1, since we have to investigate two factors instead of only one and the interaction effect.

3.8 Threats to Validity

In the following section we will discuss the threats to the construct, internal and external validity of our experimental design.

Construct Validity. The learning curves of each 3D visualization type consist of only five measured data points, since we work with the measurements of the five different projects and sessions. However, it is debatable whether the measurements done in ten days and five sessions are sufficient to measure the long-term effect and whether the resulting learning curves are representative. Better measurements of the long-term effect and learning curves could be achieved by carrying out more sessions over a longer period of time.

Internal Validity. Due to the fact that the participants are online-hired and will work remotely in their own environment there is a risk, that developers may lie about their age in order to participate in the study. The programming experience could also be manipulated, if they ask other people to complete the programming tasks which are used to check their experience in the recruitment process.

Allowing participants to work on their own computer and in a natural environment has the advantage of reducing the Hawthorne

effect, however, we will not be able to control the computers, graphics and therefore, the qualities of the visualizations. Also, the download and submitting time of tasks can slightly vary across participants depending on the computer and internet speed.

Another issue that might arise is, that the participants can get distracted when working on their own computer and environment. For example, we cannot ensure that participants start to reply to incoming email during a session, which results in a higher completion time.

All these problems can distort the results of the experiment and constitute potential alternative explanations of our results. However, since we randomly assign the participants to the treatment groups this will not be a big issue. Randomly assigning the participants allows us to distribute variation in participant and computer behavior across different 3D visualization types evenly. So, errors regarding completion time resulted from these named problems should also be evenly distributed in the four groups. Additionally, we conduct a controlled lab experiment with 12 participants as a comparison, to check whether the measured data in the lab experiment deviates from the results collected from the online experiment. If the results do not deviate too much, we may assume that the participants' different environments and devices did not distort our results.

Another threat to the internal validity is the fact that the training session is performed on the same day as the first session. It is possible that the participants get tired in the first session and have longer completion times and more mistakes due to the fatigue effect. Therefore, if the completion time increases from the first session to the second, we have to consider the fact that the increase or part of it might be due to the fatigue effect.

External Validity. It is important to mention that our experiment focuses on three similar 3D visualization techniques: *CodeCity*, *Code Park* and *Evo-Streets*. The extent to which our findings generalize to 3D code visualization techniques other than these three is limited.

To avoid the learning effect, we use different open-source projects for the sessions. We selected the projects to be of similar size and to have the same purpose, namely Big Data analysis. This allows us to better draw conclusions about the long-term effects which are not biased by the project characteristics. However, this method comes with the disadvantage, that our results might not be generalizable to the long-term effects of the usage of the visualization types with projects that have different purposes or sizes.

4 CONCLUSION

We expect that 3D code visualization techniques are less efficient than the base case without those techniques in the first two iterations of our experiment. However, in the last three iterations the 3D code visualization tools will show a steady efficiency increase which is going to be higher than the base case. These findings would confirm our first hypothesis.

We furthermore expect that different 3D visualization techniques will show different learning curves, but the difference is not going to be very significant. The 3D code visualization techniques we will use for our study have some similar features, e.g. that properties

are mapped to buildings in the visualization. So, we believe that these common features will lead to similar learning curves of the user, however they will not be the same, because each technique has its own unique features to learn.

Our findings will strengthen the trust in 3D code visualization techniques. If project managers know that 3D tools will not be as efficient in an early stage of introducing them to developers, but will increase and overpower conventional techniques without such tools over time, they are more likely to use them.

The findings of our study about the learning curves are going to be useful for developers to decide which 3D code visualization technique they want to use for their specific case. If they know that one technique has a higher learning curve than another, they might decide to use this tool instead. On the other hand, if they already have a preference for one technique, they can use the learning curve findings to discuss if they want to stick with their favorite or if they want to switch to another tool with a higher curve.

This study is similar to other studies that feature efficiency in 3D code visualization techniques, like for example in the study of Merino et al. [5]. However, because our study tackles the usage of 3D tools over time, it provides an extension to other studies. Developers can use our findings in combination with the findings of other studies to gain a wide overview of the advantages and disadvantages of using 3D code visualization techniques for their specific type of work.

For example: A developer whose goal is to use *CodeCity* for the first time in a project can use the study of Wetzel et al. [21] to decide if 3D visualization tools will bring an overall benefit to their project, additionally the study of Merino et al. [5] can be considered to decide if they want to include augmented reality for their usage and lastly they can use the findings of our study to estimate in which time spectrum the increased efficiency is presumably going to show its effects.

However, our study is not complete on its own. We only considered three different 3D code visualization techniques which had some common features. For a future study, more 3D code visualization tools, which have not much or no similarity to each other, could be analyzed and compared. Furthermore, we only considered the factor of efficiency. But other factors would also be interesting for a comparison, for example how happy the developers are while using the tools or the adaption rate to switch to another similar technique.

To sum up, our study provides a useful insight into the usage of 3D code visualization techniques while using them in a long-term spectrum. The findings of the study can be combined with other study's results to gain an even better overview and understanding of such tools. We hope that developers, project managers and everyone else who works in a software team will get encouraged to use 3D code visualization techniques more in the future.

REFERENCES

- [1] R. Evans. 2015. Apache Storm, a Hands on Tutorial. In *2015 IEEE International Conference on Cloud Engineering*. 2–2. <https://doi.org/10.1109/IC2E.2015.67>

- [2] S. Hahn, M. Trapp, N. Wuttke, and J. Döllner. 2015. Thread City: Combined Visualization of Structure and Activity for the Exploration of Multi-threaded Software Systems. In *2015 19th International Conference on Information Visualisation*. 101–106. <https://doi.org/10.1109/IV.2015.28>
- [3] Pooya Khaloo, Mehran Maghousi, Eugene Taranta, David Bettner, and Joseph Laviola. 2017. Code Park: A New 3D Code Visualization Tool. In *Software Visualization (VISOFT), 2017 IEEE Working Conference on*. IEEE, 43–53.
- [4] Jonathan I. Maletic and Andrian Marcus. 2001. Supporting Program Comprehension Using Semantic and Structural Information. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE '01)*. IEEE Computer Society, Washington, DC, USA, 103–112. <http://dl.acm.org/citation.cfm?id=381473.381484>
- [5] Leonel Merino, Alexandre Bergel, and Oscar Nierstrasz. 2018. Overcoming issues of 3D software visualization through immersive augmented reality. *Proc. of VISOFT, page in review*. IEEE (2018).
- [6] M. Rüdel, J. Ganser, and R. Koschke. 2018. A Controlled Experiment on Spatial Orientation in VR-Based Software Cities. In *2018 IEEE Working Conference on Software Visualization (VISOFT)*. 21–31. <https://doi.org/10.1109/VISOFT.2018.00011>
- [7] Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, and Carlo Curino. 2015. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 1357–1369. <https://doi.org/10.1145/2723372.2742790>
- [8] M. A. Sharma and M. O. Joshi. 2016. Openstack Ceilometer Data Analytics and Predictions. In *2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 182–183. <https://doi.org/10.1109/CCEM.2016.045>
- [9] Martin Shepperd. 1995. *Fundamentals of software measurement*. Prentice-Hall.
- [10] SoftVis3D. 2015. Live Demo: Evostreet layout of elasticsearch. (2015). https://softvis3d.com/sonar/project/extension/softvis3d/overview_page?id=org.elasticsearch:elasticsearch&metricFootprint=complexity&metricHeight=ncloc&metricColor=none&layout=evostreet&scale=logarithmic&cameraX=0&cameraY=8866&cameraZ=7993 Last Visited: 2019-01-14.
- [11] Frank Steinbrückner and Claus Lewerentz. 2010. Representing Development History in Software Cities. In *Proceedings of the 5th International Symposium on Software Visualization (SOFTVIS '10)*. ACM, New York, NY, USA, 193–202. <https://doi.org/10.1145/1879211.1879239>
- [12] M.-A.D Storey, F.D Fracchia, and H.A Müller. 1999. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Systems and Software* 44, 3 (1999), 171 – 185. [https://doi.org/10.1016/S0164-1212\(98\)10055-9](https://doi.org/10.1016/S0164-1212(98)10055-9)
- [13] Alfredo Raúl Teyseyre and Marcelo R Campo. 2009. An overview of 3D software visualization. *IEEE Trans. Vis. Comput. Graph.* 15, 1 (2009), 87–105.
- [14] K. Tsakalozos, C. Johns, K. Monroe, P. VanderGiessen, A. Mcleod, and A. Rosales. 2016. Open big data infrastructures to everyone. In *2016 IEEE International Conference on Big Data (Big Data)*. 2127–2129. <https://doi.org/10.1109/BigData.2016.7840841>
- [15] Sergi Valverde and Ricard V Solé. 2003. Hierarchical small worlds in software architecture. *arXiv preprint cond-mat/0307278* (2003).
- [16] Markus Weimer, Yingda Chen, Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Yunseong Lee, Tony Majestro, Dahlia Malkhi, Sergiy Matusevych, Brandon Myers, Shravan Narayanamurthy, Raghu Ramakrishnan, Sriram Rao, Russel Sears, Beysim Sezgin, and Julia Wang. 2015. REEF: Retainable Evaluator Execution Framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 1343–1355. <https://doi.org/10.1145/2723372.2742793>
- [17] Richard Wettel. 2017. Download Release 1.4.3. (2017). <https://wettel.github.io/codcity-download.html> Last Visited: 2019-01-14.
- [18] Richard Wettel and Michele Lanza. 2007. Program comprehension through software habitability. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*. IEEE, 231–240.
- [19] R. Wettel and M. Lanza. 2007. Program Comprehension through Software Habitability. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*. 231–240. <https://doi.org/10.1109/ICPC.2007.30>
- [20] Richard Wettel and Michele Lanza. 2008. CodeCity: 3D Visualization of Large-scale Software. In *Companion of the 30th International Conference on Software Engineering (ICSE Companion '08)*. ACM, New York, NY, USA, 921–922. <https://doi.org/10.1145/1370175.1370188>
- [21] Richard Wettel, Michele Lanza, and Romain Robbes. 2011. Software systems as cities: A controlled experiment. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 551–560.
- [22] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The Aligned Rank Transform for Nonparametric Factorial Analyses Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 143–146. <https://doi.org/10.1145/1978942.1978963>