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. There are multiple studies that analyze how these techniques work and if they are effective to use. Our study focuses on the effectiveness of three 3D code visualization techniques: *CodeCity, Code Park* and *Evo-Streets*. 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.

KEYWORDS

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

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, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The usage of 3D code visualization has become a frequent discussed topic. There are multiple studies that research the usage and effectiveness of such methods (e.g. [4, 12, 14]). These studies mainly study the different techniques with an experiment at one time and draw their conclusion from these results. Furthermore, most studies only discuss one tool 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 increases 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 to 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 visualizations 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.

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 advise the reader to read [9]'s paper.

There are multiple studies that research the usage and effectiveness of 3D code visualizations. For example, [4] 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 developer's efficiency. [5] 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, [14] conducted an controlled experiment to find out if their *city metaphor* [12] 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. These kind of studies made 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. [4], 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

DSforSE'18, Fall 2018, Zurich, Switzerland

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

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 they take 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.* We will discuss these different techniques in the following subsections 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 [13, 14].

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 [13].

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 [4].

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 were 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 in 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 lock 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. The main advantage of this visualization technique is, that the user can view actual code while using the visual mappings to get an overview of the whole project [3].

However, 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 [8].

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 [8].

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 leafs 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.

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: *Code Park, CodeCity* 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?
- RQ2 Do different visualization techniques have different learning curves?

3.2 Hypotheses

We believe that developers need time in order to learn and get familiar to specific visualization techniques. With the amount times they use a specific technique, the efficiency in completing comprehension tasks may increase. Therefore, the effect of using 3D code visualization might not show itself immediately, but in a long-term spectrum. Additionally, we hypothesize 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 3D visualization techniques in the long term increases developers efficiency in completing comprehension tasks.
- H2 Different visualization techniques 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, 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.

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 they solved 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 better know the visualization type 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.

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 one day break between consecutive sessions. Each session will take about one hour, however, it depends on the task completion times of the participants.

3.4 Participants

A challenge to conducting this experiment can 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

Table 1: The five Apache projects which we intent to use for the experiment.

Name	Functionality
Apache Tez	building data-flow driven processing run-times [6]
Apache Bigtop	building Hadoop rpm and deb packages [10]
Apache REEF	building run-time management support for the monitoring of tasks [11]
Apache Storm	stream processing [1]
Apache Zeppelin	web-based notebook for data analysis and visualization [7]

80 developers as participants for the experiment, which will result in 20 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 is 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 found between different visualization techniques will be attributable to the technique itself and not to the participant characteristics. Since each participant only uses only one specific visualization technique we can effectively check whether the the efficiency of solving comprehension tasks increases with the amount of times they use the visualization.

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 1. Using projects from the same software company with similar goals may not provide a varied test-set, however using data from multiple different companies 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 behaviour 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 software company.

3.6 Data Collection

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 the hypotheses, we expect the completion time for the same tasks to decrease and the number of correctly solved tasks to increase with the number of times they solve the task with the same visualization type.

[1]https://www.apache.org

3.7 Threats of Validity

In the following section we will discuss the threats of 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 on their own environment there is a risk, 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.
These two problems can 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 behavior across different 3D visualization types evenly.
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.

External Validity. It is important to mention that our experiment focuses on three similar 3D visualization techniques: *Code Park, CodeCity* 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. This findings would strengthen our first hypothesis.

We furthermore expect, that different 3D visualization techniques will show different learning curves, but the difference is not going to be much 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 this 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 [4]. 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 his project can use the study of [14] to decide if 3D visualization tools will bring an overall benefit to their project, additionally the study of [4] 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 completed in it's 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. DÃűllner. 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 Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. 2017. Code Park: A New 3D Code Visualization Tool. In *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE, 43–53.

[4] Leonel Merino, Alexandre Bergel, and Oscar Nierstrasz. 2018. Overcoming issues of 3D software visualization through immersive augmented reality. *Proc. of VISSOFT, page in review. IEEE* (2018).

[5] M. RÃijdel, 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 (VISSOFT)*. 21–31. https://doi.org/10.1109/VISSOFT.2018.00011

[6] 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

[7] M. A. Sharma and M. O. Joshi. 2016. Openstack Ceilometer Data Analytics amp;amp; Predictions. In *2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 182–183. https://doi.org/10.1109/CCEM.2016.045

[8] 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

[9] Alfredo Raúl Teyseyre and Marcelo R Campo. 2009. An overview of 3D software visualization. *IEEE Trans. Vis. Comput. Graph.* 15, 1 (2009), 87–105.

[10] 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

[11] 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

[12] 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.

[13] 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

[14] 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.