Visualization for Software Developers

Timon Ruth [TR]
Institute of Computer science
University of Goettingen
Goettingen, Germany
timonniklas.ruth@stud.uni-goettingen.de

Surendhar Muthukumar [SM]
Institute of Computer science
University of Goettingen
Goettingen, Germany
surendhar.m@stud.uni-goettingen.de

Abstract*—[TR][SM] Software is becoming more complex in modern times. This increases the difficulty of understanding and working with code which can be either created by individual developers themselves or even by others. Hence to make this process easier and time efficient, software visualisation techniques shall play vital role. These tools provides features to simplify code understanding and interactions, that are advantageous over normal IDEs or text editors. The presented work analysed multiple visualization techniques and tools to evaluate the extent of their advantages when used. The study found that these tools are helpful in multiple stages of software development like designing, developing, debugging, security and privacy. It also found that despite of its advantages the visualizations are not completely adapted by current codebases for multiple reasons.

I. Introduction

As computers got more complex over the years so did their software. [SM] Modern software is larger and more complex than its predecessors, making it challenging for programmers to obtain a global understanding of the program's state and behavior. The causality to the complex programs and systems are generally increased user bases, need for enhancements in current available systems, need for new applications that support businesses,etc... Tools that analyze the dynamic behavior of programs have traditionally focused on identifying performance problems (like execution time, space complexities, functionality issues) rather than on general program understanding. [TR] To be able to understand this complex structure, a number of techniques and tools have evolved over time, a very important technique is visualization. It provides a graphical representation of complex software systems, making it easier to understand, analyse, and modify these systems. The advantages of this are numerous, including security, improved quality and usability.

In this paper we will discuss the advantages of software visualization for developers including the context of computer security. This includes the identification of security risks, the testing of security and its monitoring. To add to this, we will show the usage of visualization on a number of selected examples. Our aim is to provide an overview of how software visualization is done and an evaluation of how valuable visualization can be helpful for developers. The remainder of the paper is structured as following: Section III discusses the methods used to obtain relevant resources. Section IV describes different areas where visualization can enhance the software system development. Section V emphasizes the study results, key aspects, validity of the study and possible future works. Section VI concludes and presents outcomes with scope for future work.

II. Methodology

[SM]This study was conducted through a systemic literature review of relevant research works and publications related to the field of data visualization corresponding to software development and privacy analysis. The search criteria included published articles that are related to software visualization, privacy risk visualization and privacy policy visualization models. The search was conducted using relevant database such as Google scholar, ACM, IEEE, which also comprises papers/presentations related to techniques and ideas presented on SOFTVIS conference. Among the various articles appeared in search results, the abstracts and conclusion were carefully read first to jot down the appropriate articles corresponding to the study.

The articles were selected based on their relevance to the topic of consideration. The articles were evaluated for their contribution to the understanding and evaluation of data visualization methods both in general software development aspects and in specific application towards various stages of development cycle in software systems. Considering general visualization approaches for software development process, priority was given to the research works that performed efficiency, usability evaluations on related techniques and related tools. Such works involved in-depth tool analysis through various methods namely, questionnaire, survey, case study, experiment, interviews, usage scenarios. The methods of analysis involved participants with different exposure towards software development including novice users like students, experienced candidates: junior and senior level software development, software trainers. The papers that merely explained the tools' features and usage examples are out of scope for this study and are not considered for further analysis. Additionally supporting papers to understand certain concepts and applications are referred for information validation purposes. These supporting papers shall not have user evaluations always.

Secondly, journal articles that contribute to the understanding on visualization models of privacy policy, privacy risk, information security are also considered. These works explain models involving diagrammatic and systemic approaches. The analyzed articles concentrate more towards the visualization of

data flow of Personal Identification Information (PII) through the software life cycle and emphasize more towards the analysis of possible risks and data leakage in the development cycle.

III. RESULTS

A. *Improved Code Understanding*

[SM] Code visualization provides insights into the code that may be difficult to acquire through a traditional text editor. Multiple techniques utilize 2D, 3D structures and scenes to improve understandability. Some visualization tools that use these structures are namely SeeSoft, Code Bubbles, CodeCity, CodeFlow, CodePark, CityVR etc... Code Park, CityVR: 3D visualization techniques/tools that organize the source code into 3D game-like environment, making it more readable,interactive and understandable, taking the advantage of human spatial memory [1]. The game-like approach was adapted to improve cognitive concentration of humans in the programming task with elements of enjoyment. Using these tools the source code/program code can be visualized in various forms: park-like structures (figure 1) [1], city based structure (figure 2) [2], enhancing the visual understandability on the scale of the work. Such structures help developers identify patterns and relationships in their code that might not be immediately understandable in traditional code editors. Additionally interactive features that enable developers to manipulate the structures, highlight code elements and search for specific parts of the code are incorporated. Game-like elements, such as score-keeping, achievement badges introduced in CityVR motivated developers to explore and understand code more deeply.

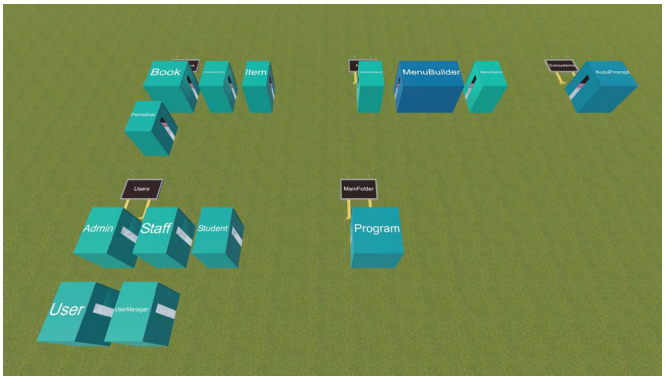

Fig. 1. Top down view Code park codebase [1]

The usability and efficiency study conducted on these tools shows that the tools are intuitive and reduce cognitive load of use [2] [1]. Both the studies involved user study with significant number of experienced developers to work with respective tool. In case of CodePark,the study aimed to discover difference in usability and efficiency of the proposed visualizations technique against standard IDE/text editor tools. The results affirm that increased interactivity and engagement of users towards code was achieved through enhanced code

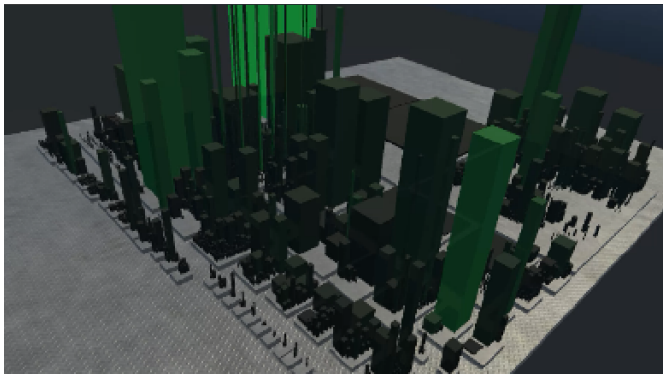

Fig. 2. 3D city structure in CityVR [2]

view technique, where the users were able to visually process the entire code from an exo-centric view. Adaption to new codebase was quicker. Logical loopholes identification were accelerated through better understanding of internal semantics. Additionally it is useful for identifying code issues and providing insights into the structure and organization of code. Though the tools were very effective improvements were needed in areas of selection and inspection of the code snippets/chunks. Participants reported issues in understanding certain features of visualization and needed improved search functions as the traditional search function will be obsolete. Overall, studies reveal that improved interaction is an effective architectural choice [2]. There are variety of studies on techniques proposed by tool developers to enhance the code understandability, where all cannot be analysed for our study. But irrespective of the number of works proposed, it is evident that these techniques/tools serve their purpose.

B. *Enhanced Debugging and Troubleshooting*

[TR] Debugging is a task that usually takes up a lot of time for a developer. The finding of an error can be tedious and expensive. This is a task where the visualization can improve the workflow of a developer. Automated tests oftentimes only provide hard to understand and unclear error messages. This could be improved by using a dashboard that shows the information of tests that include the status of the test, the test that was performed and a log file for more in-depth information.

[SM] Traditional debugging tools incorporated in the IDEs are inflexible and complicated, that the developers spend significant amount of time trying to understand the codebase and identify the issue. It is a common practice for developers to debug codes not written by them exclusively. In such cases significant time loss is experienced. Hence, visualization tools make debugging easier and more efficient, allowing the users to view multiple code fragments/snippets simultaneously and arrange them in a sensible way for analysis. User studies conducted on visualization tools like Code bubbles, HeapViz [3] [4] demonstrated the effectiveness of visualization in debugging and understanding. In the conducted user study with participants, the effectiveness of the tools against regular

IDEs or codebase were tested. The outcomes concur the claimed effects. The participants had good understanding of the code and resolved the issues comparatively faster due to the ability to visualize the code in tree-like structure or bubble like structure respective to the study. The visualization structures are different dependent on the techniques of use described in various studies, but share the same effect on developers. Memory allocations, pointer references, function calls were represented as structures for easy interaction. Tools allow zoom, search and filtering for effective recognition in future. The bubbles or the tree-nodes are re-arrangable to user preference and mental model, improving the customization ability. Hence the debugging experience is elevated along with significant reduction in time spent on these tasks.

[TR] In 3 you can see an example of the visualization in code bubbles. You can see the rearranged bubbles and interconnections of different objects. On the right there are the different method headers. If hovering those you can see the complete method. The whole imagine is floating, that is why some bubbles are not shown completely on the bottom. The user can move to those through dragging the mouse.

C. *Better Collaboration*

[TR] Visualization of software also helps in collaboration of team members. This can be done through a shared understanding of the code by the team. This is because they can see how different parts of a project work together and get an understanding of what the other members did. It reduces the communication time and rate of error through a more precise way of explaining the code. Understanding the code fast and easy is not only valuable for long-time members of the team but especially for new developers as they don't have to take nearly as much time to understand the systems in use. [6]

Another way software visualization improves collaboration is the earlier feedback other developers can give. Usually there needs to be at least a prototype of software before a developer can present their ideas. When using proper visualization, however, the software can be explained and evaluated before a lot of the actual software have to be written.

This, however, needs to be further investigated as there is not a lot of studies revolving this specific topic.

D. *Enhanced Requirements Gathering and Analysis*

When developing software for users there can be a disconnect between what stakeholders wish to have and what the developer thinks they want. This can result in development time used for features that the stakeholder does not need or will not use. Again, the shorter time for visualization rather than development can eradicate this issue because of the improved accuracy of the requirements analysis. [7]

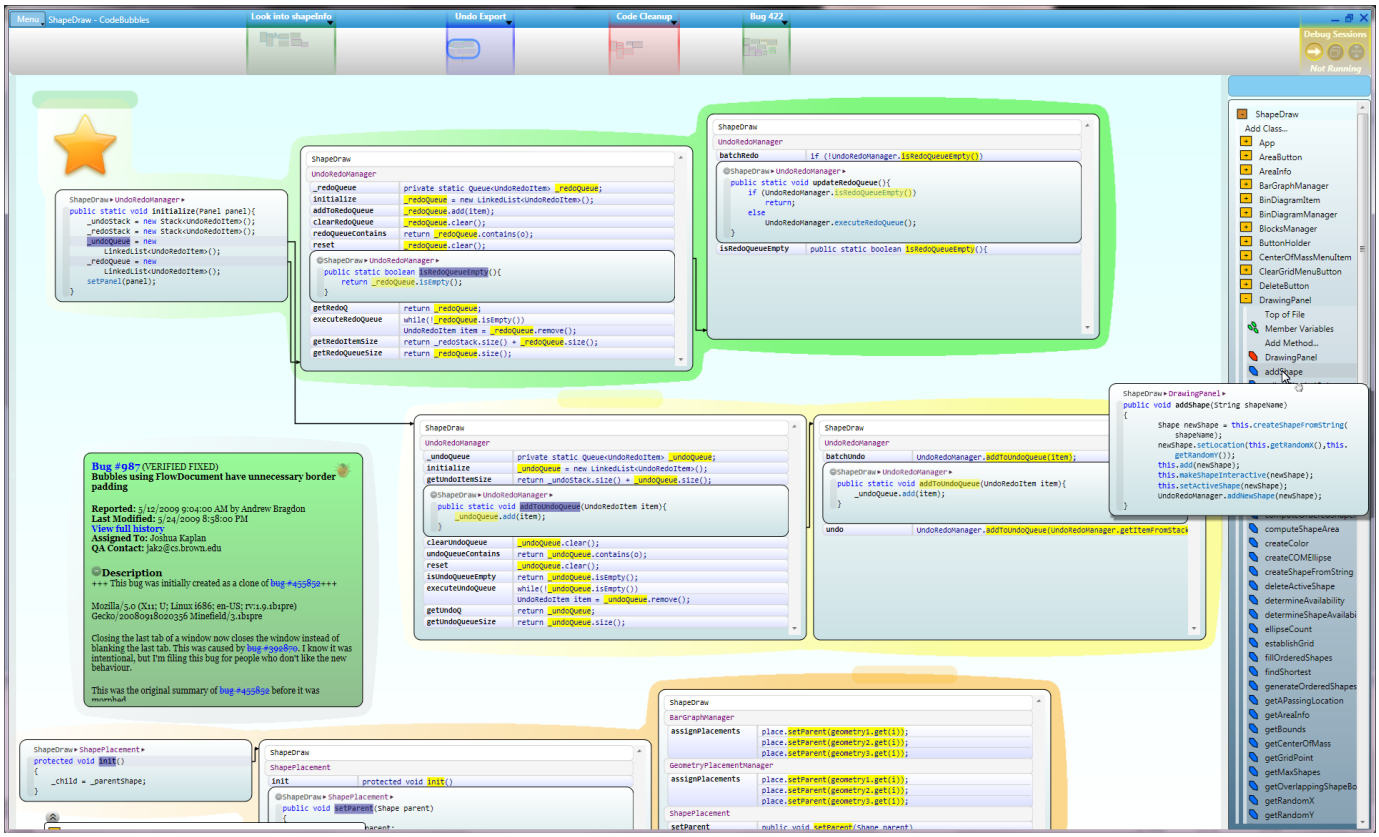

Fig. 3. An example visualization for code bubbles [5]

E. Teaching

[TR] When students start out learning how to code some of their most common problems are self-confidence, lab session approaches and time management [8]. These can be broken down to a lack of understanding. When a student thinks something is too hard for them, they will not feel confident to solve the problem. Time management adds to this, as the time needed to actually understand the task can be too great and not fit into the schedule of the student. Lastly, the lab session approaches can be problematic if the student does not understand what is shown to them. Visualization can help here because as shown in codeUnderstanding software visualization makes it easier for developers to understand and work with code. Visualization can increase the understanding in especially new students that do not already understand programming or specific techniques including object oriented programming [9]. They are more motivated to learn what programming is about as they do not feel like this is too hard for them.

F. Improved Code Quality

[TR] Code quality can be improved by software visualization through numerous ways. Code redundancies and inconsistencies can be identified which leads to an elimination of those. Said things can create software bugs and fixing them in an often used module can remove the problem while for a redundant part the bug has to be found every time the part exists in the code, resulting in greater maintenance and cost.

A way redundancies and inconsistencies can be found is with a tool that visualizes the structure of the code. This could for example be a UML diagram or bubbles from code bubble 3. The class hierarchy and interconnection between different classes can automatically be visualized making it easier to grasp the concept of a project.

To add to this, visualization can help finding bottlenecks in the code as performance can be tracked more easily. As shown in 4 the work time of different functions can be monitored visually and therefor be compared. A small function could use huge amount of processing time and be easily overseen by the developer resulting in worse run time for the entire program.

A commonly used way to visualize function calls is using flame graphs. These are graphs that show when a function was called, how long it was taking and how the tasks were nested. In 4 there is an example of a very simple flame graph. The function 'main' is using up the whole time of the CPU. The main function calls the 'foo1' function which itself calls the 'bar' function. When the bar function is fully computed 'foo1' has to finish another task. This one had to wait for 'bar' to be finished. When it is finished the main function calls 'foo2' which itself calls bar again. When finish this function has some task to finish as well which, however, is shorter than the task of 'foo1'. In the end the 'main' function finishes a task lying in the 'main' function. In the figure this is seen very easily with not a lot of information needed to understand how a flame graph works. If only the code is being looked at, this is much harder to see, as seen in III-F. Even though this is a very easy example of what the code could look like, at first glance a developer would not know how exactly the code folds out. If this were a more complex code they would have no chance to find this out without visualization.

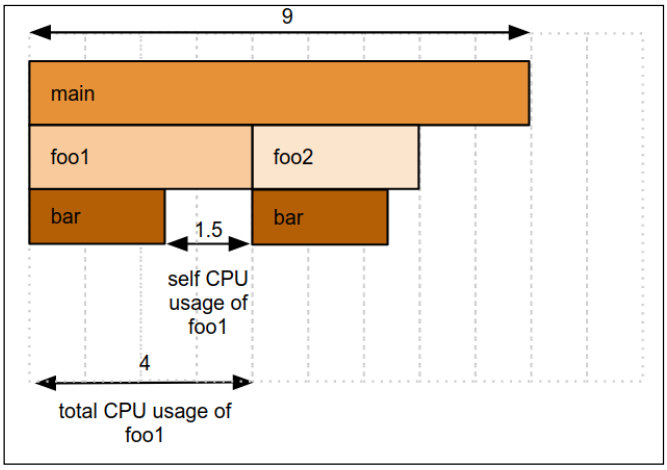

Fig. 4. An example flame graph. [10]

```
# an example code for the flame graph Fig. 3.
def main():
    a = 0
    a = foo1(a)
    a = foo2(a)
    for i in range(2000):
        a += 1

def foo1(a):
    a = bar(a)
    for i in range(1500):
        a += 1

def foo2(a):
    a = bar(a)
    for i in range(500):
        a -= 1
    return a

def bar(a):
    for i in range(2500):
        a += 1
    return a
```

G. Identify and prevent Privacy risks

[SM] Software systems targeting consumers, like banking, shopping, learning, healthcare, and digital government need the personal data of consumers in one form or the other for their seamless operations. Hence there is a need for developers to define and implement better privacy protection methods that are compliant to protect the privacy of customers. The applied privacy protection methods should have profound insights about where and what protection is needed. Every vulnerable aspect of the software/application must be identified and accounted for. The process of identifying such vulnerabilities

is challenging, time involving, and tedious. Visualization aids helping hands in identifying such vulnerabilities.

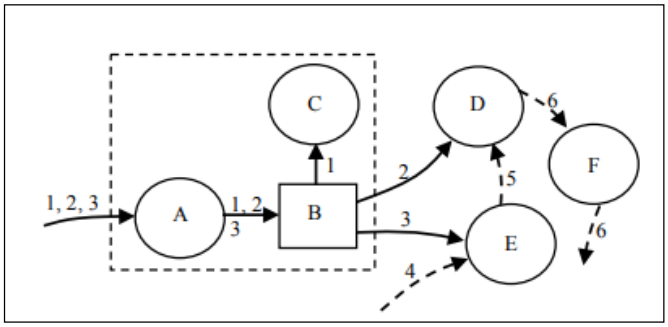

Fig. 5. Basic PIM online streaming platform subscription. [11]

Label	Description
A	Receive user data
B	Database
C	Account creation
D	Selected plan
E	Credit card payment
H	Reflect subscription

TABLE I
DESCRIPTION OF STORAGE LOCATIONS

Label	Description
1	name, email address, password
2	Subscription plan
3	Credit/Debit card details
4	Firm account details
5	Payment status
6	Subscription status

TABLE II
LABELLED PII DESCRIPTION

[SM] Visualization methodology proposed in [11] comprising 2 steps,i)finding possible storage locations of PII(Personal Identification Information) in software system ii)evaluating possible data compromise locations. PII is internally transmitted in a software system for the accomplishment of tasks and services offered. Such transmissions usually happen through different storage locations that shall be within or outside the bound of the software's storage network. Hence the data must be heavily encrypted and safeguarded against data breaches. To identify the data breach locations PIM (Personal Information Map) is used. PIM is a pictorial representation expressing what PII is required, where it is stored, and used, within the software system, as location and usage of PII are the keys. The PIM is not a logic flow diagram of a program as the technical aspects of achieving a feature are not the point of interest here. Importance is given to identifying the critical transmissions and storage locations prone to privacy risk. In PIM a circle represents the personal information usage location, here the information is not stored permanently but is utilized for calculations and task accomplishments. A square represents the data storage location like a database. Directed arrows between the locations imply the transfer of data within locations. A, C, D, E, F of the PIM in figure 1 are PII usage locations and B is PII storage. 1, 2, 3, 4, 5, and 6 are individual PII data. The significance of the PII data and the storage locations are explained in Table II I. Numbered arrows represent data flow that transmits the relevant numbered PII. Inspecting data flow associated with individual locations enumerate privacy violation possibilities, expressed in a Privacy risks table. General privacy risk questions related to the collector, purpose, retention time and owner of PII are considered. The privacy risk table mentions the vulnerable location along with the possible risk. Table III corresponds to the mentioned example in Figure 5. This table can be used to develop securities around the location to prevent PII leakages and misuse.

PIIs/ Locations	Privacy risks
(1,2,3 - path into A) (2 - path into D) (3 - path into E)	Man-in-the-middle attack violates collector, purpose and disclosure. User can be asked other PIIs not related to purpose in path into A
(1,2,3 - A, B) (1 - C) (2 - D) (3 - E)	Trojan horse, hacker, or SQL attack (for B) violates collector, purpose and disclose-to; Information could be kept past the retention time in B

TABLE III
LABELLED PII DESCRIPTION [11]

IV. DISCUSSION

A. *Why is software visualized*

[SM] From the research outcomes presented in the results section, it is obvious that the introduction and use of visualization uplift the result of developer effort on the tasks. Software development is a lengthy and time involving process. Generally, the entire process is subdivided into stages like planning, designing, development, implementation, integration, and quality analysis/testing. Each stage has its own significance and is interrelated as the outcome of one stage is treated as the input for the other. Flaws in previous stages shall affect all the upcoming stage results. Human errors are frequent in such environments and often necessary efforts are considered to reduce such circumstances. Some general causes of errors are the environment, lack of concentration, the complexity of the system, external noises, and lack of understanding. Visualization is one such effort, when implemented properly, to reduce significant human errors. The software development community concurs with this fact, hence frequent improvements and advancements are performed in this technique of visualization. Yearly SOFTVIS / VISSOFT conferences are organized where researchers and developers around the globe present their work. New visualization techniques/tools are presented in the events, accompanied by relevant user studies examining the usability and effectiveness of the proposed techniques are parallel. With proper use of the tools, developers can understand and communicate complex system architecture, and logic. Mental load on debugging and

problem-solving can be reduced by decreasing the need for cognitive data remembrance. Bugs shall be easily diagnosed and fixed through visual representation of code execution. Testing and validation can be benefited from visualization, such as graphical user interfaces, which provide a visual representation of the user experience.

B. *What are the methods that use visualization*

[TR] We showed numerous ways to visualize data. This includes for example the flame graph, a very popular visualization used to understand the flow of a software program and to find irregularities. Finding such irregularities is one of the main points of visualization, hence a lot of other tools focus and displaying such in-frequencies. This decreases work time and therefor cost for a project because errors can be found more easily. Without a visualization a lot of errors in large scope projects could not be pinpointed. For software security this is especially important because bugs could lead to data breaches or other forms of security problems. Software security visualization tools revolve around finding PII storage in the program and places in the software where such information is used. Sometimes functions use information that they do not actually need and if a number of developers work on a project for a long time those places could be easily forgotten and pose a real security threat. A visualization of those usages would show these functions to the developers and make them easier to fix.

C. *Research gaps*

[TR] When searching for studies revolving this topic it becomes obvious that a lot of the studies revolving visualization exist to mostly promote a visualization tool or do a meta analysis of the existing visualization tools and in what direction these go. There are not a lot of research papers doing user studies looking for what software developers would like to have visualized. In the future there could be further studies in this direction to better find out what visualization tools are still needed and which software developers have enough of already.

[SM] To add to this claim, a study of 387 full papers published in the SOFTVIS/VISSOFT conferences claims that more than half of the published articles lack to explicitly disclose the evaluation strategies, methods, data collection procedures involved in the user and usability study [12]. As there are no uniform standards defined for the evaluation, different methods are used by authors, in the wish that the methods shall provide results with a higher degree of efficiency. But these works raise a question on reliability and in most cases, the author and the evaluator of the study are the same. This research gap can be bridged by studies exclusively done on tools/models by external researchers than the proprietary author. ”Moreover abundant studies have shown that visualization is advantageous for software developers, yet adopting visualization during software development is not a common practice due to the large effort involved in finding appropriate visualization” [13].

D. *Threats to Validity*

[TR] This work only had a limited scope of using only a handful of studies revolving this topic. This can threaten the validity because there are many more research papers about this topic that could not be addressed here. Although we tried our best to choose the best sources available for the topic some important ones could have been missed.

[SM] moreover the research works in this field are mostly concentrated on representing new models and tools, rather than considering an existing tool with good performance, usability, and efficiency. Additionally, the analysis conducted on 181 SOFTVIS/VISSOFT papers that documented the evaluation strategy, methods, data collection procedure, and test participants, reveals that the key challenge in user studies is the lack of uniformity in standards to perform the evaluation [12]. Though the authors provide all relevant information, the reliability of the studies conducted is affected in most cases as the author supervises the study. Authors tend to choose participants for the study by their own custom constraints, which shall bias the study results. This may result favourable to the author but may not be always suggested, as the results may vary when the user base is changed. Hence these studies are prone to provide results that can be non-reproducible and subject-dependent. The studies can be generally classified into categories as a Design study, Evaluation, System, and technique based on the quality of analysis [12]. The significance of the classification are mentioned as follows [14]:

- **Evaluations** - describe how tasks in the problem domain are handled using the visualization. Evaluations are generally conducted in laboratory settings with participants.
- **Design studies** - describes how available visualization techniques can be effectively integrated to deal with a particular problem domain. Evaluations are done through case studies and usage scenarios.
- **Systems** - lessons learned from observing the architectural design choices of a tool.
- **Technique** - algorithms that increase the effectiveness of visualization. Benchmarks quantify the evaluation.

This classification is an integral part of understanding the quality of the conducted studies. For our study, the analyzed works were classified into the above-mentioned categories. Among the 9 references mentioned, only 3 [**?**], [4], [8] (i.e., 33%) studies performed experimental studies with active participants and this trend is also experienced in previous researches [12]. The remaining studies can be classified as 3 (33%) Design study [2], [3], [6], 2 (22%) Systems [7], [9] and 1 study that was a combination of Design and Technique [11]. Irrespective of the classification, the research works provide explicit evaluations that support their claim and hence are legit to be considered as reliable background works.

V. Conclusion

[TR] In this study we showed that software visualization has a lot of benefits These include improved code understanding because code is grasped easier and can even be gamified. It can

also enhance debugging as not all the code relations have to be understood by just looking at the code alone. To add to this, we found that collaboration between team members can be enhanced, although there is information lacking in that regard. The communication between stakeholders and developers can also be elevated by visualizing what the developers do. Even for new soon-to-be developers visualization is a powerful tool, increasing the ease of understanding and accessibility. We explained a technique used for improving code quality through displaying the time usage and hierarchy of function calls in the form of a flame graph. Lastly, we presented how visualization helps against security breaches in the form of PII detection.

Afterward, we discussed the reasons behind visualizing software and what methods are used. To finish it we showed the threats to the validity of this work. This includes the scope of literature we used.

A. *Further Work*

[TR] In the future the effect of visualization on a team environment should be looked at further. Our other results show promising features for teams including how to include new team members faster and more easily and reducing time in meetings by making it easier to understand what the other members developed.

To add to this a uniform classifier for different visualization tools should be agreed on to better evaluate if a visualization tool is working well or not.

Lastly, it should be looked at why developers do not use visualization as much as the advantages might suggest.

REFERENCES

[1] P. Khaloo, M. Maghoumi, E. T. II, D. Bettner, and J. L. Jr., "Code Park: A New 3d Code Visualization Tool," *IEEE*, 2017.

[2] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "Cityvr: Gameful Software Visualization," *IEEE*, 2018.

[3] E. E. Aftandilian, S. Kelley, C. Gramazio, S. L. S. Nathan Ricci, and S. Z. Guyer, "Heapviz: interactive heap visualization for program understanding and debugging," *Association for Computer machinery*, pp. 53–62, 2010. [Online]. Available: https://doi.org/10.1145/1879211.1879222

[4] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. L. Jr., "Code bubbles: Rethinking the user interface paradigm of integrated development environments," vol. 1, 2010, pp. 455–464. [Online]. Available: https://doi.org/10.1145/1806799.1806866

[5] "Code Bubbles: An IDE Revolution," https://dzone.com/articles/code-bubbles-ide-revolution, accessed: 24.02.2023.

[6] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," in *Proceedings of the 2007 ACM International Conference on Supporting Group Work*, ser. GROUP '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 147–156. [Online]. Available: https://doi.org/10.1145/1316624.1316646

[7] A. Telea, L. Voinea, and H. Sassenburg, "Visual tools for software architecture understanding: A stakeholder perspective," *IEEE Software*, vol. 27, no. 6, pp. 46–53, 2010.

[8] M. Rahmat, S. Shahrani, R. Latih, N. F. M. Yatim, N. F. A. Zainal, and R. Ab Rahman, "Major problems in basic programming that influence student performance," *Procedia-Social and Behavioral Sciences*, vol. 59, pp. 287–296, 2012.

[9] H. L. Dershem and J. Vanderhyde, "Java class visualization for teaching object-oriented concepts," in *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, 1998, pp. 53–57.

[10] "Flame graphs," https://cloud.google.com/profiler/docs/concepts-flame, accessed: 24.02.2023.

[11] G. O. M. Yee, *Visualization of Privacy Risks in Software Systems*. SECUREWARE, 2016, pp. 289–294.

[12] L. merino, C. Anslow, and O. Nierstrasz, *A Systematic literature review of software visualization evaluation*. ELSIVIER, 2018, vol. 144, pp. 165–180. [Online]. Available: https://doi.org/10.1016/j.jss.2018.06.027

[13] L. Merino, M. Ghafari, and O. Nierstrasz, "Towards actionable visualization for software developers," *Journal of software: evolution and process*, vol. 30, no. 2, p. e1923, 2018.

[14] T. Munzner, "Process and pitfalls in writing information visualization research papers," *Springer*, vol. 30, pp. 134–153, 2008.