Sensorial software evolution comprehension

Subtitle: Reinventing the World

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Dependable Distributed Systems

presented by
Gianlorenzo Occhipinti

under the supervision of
Prof. Michele Lanza
co-supervised by
Prof. Csaba Nagy, Prof. Csaba Nagy

July 2022

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Gianlorenzo Occhipinti
Lugano, Yesterday July 2022

To my beloved

Someone said ...

Someone

Abstract

The comprehension of software evolution is essential for the understandability and maintainability of systems. However, the sheer quantity and complexity of the information generated during systems development make the comprehension process challenging. We present an approach, based on the concept of synesthesia (the production of a sense impression relating to one sense by stimulation of another sense), which represents the evolutionary process through an interactive visual depiction of the evolving software artifacts complemented by an auditive portrayal of the evolution. The approach is exemplified in SYN, a web application, which enables sensorial software evolution comprehension. We applied SYN on real-life systems and presented several insights and reflections.

Acknowledgements

ACK

Contents

Figures

Tables

Chapter 1

Introduction

In 1971 Dijkstra, made an analogy between computer programming an art. It stated that is not important to learn how to compose a software, but instead, it is important to develop its own style and what will be their implications.

Software Understanding

- Section [Challenges of software understanding] - Section [Challenges of software maintainability] - Section [Software comprehension] - Section [Our approach] We present an approach, based on the concept of synesthesia (the production of a sense impression relating to one sense by stimulation of another sense), which represents the evolutionary process through an interactive visual depiction of the evolving software artifacts complemented by an auditive portrayal of the evolution.

Chapter 2

State of the art

2.1 Software visualization

Essential parts of software lifecycles are software maintenance and software evolution. Both activities require the comprehension of the system by the developer. Mayrhauser [24] defined program comprehension as a process that uses knowledge to acquire new knowledge. Generally, programmers possess two types of knowledge: general knowledge and software-specific knowledge, which represent their level of understanding of that software. Software comprehension aims to increase this specific knowledge of the system, and, to do that, it can leverage some software visualization techniques. Software visualization supports the understanding of software systems because it enables the visualization of the system's information (architecture, source code, behavior) with a 2D or 3D representation. Stasko et al.[8] conducted a study in 1998 that shows how visualization arguments human memory since it works as external cognitive aid and thus, improves thinking and analysis capabilities.

History of software visualization

The earliest form of software visualization found in the literature was in the form of 2D diagrams. Haibt [9] in 1959, was one of the first to use them to visualize software systems. He provided a graphical outline of a program and its behavior with flowcharts. As shown in Figure **??**, they were 2D diagrams that described the execution of a program. He wrapped each statement in a box, representing the control flow with arrows. Ten years later, the effectiveness of flowcharts was confirmed by Knuth [14]. Unfortunately, at that time, most of the programs were affected by a lack of readability since they were not well documented. So, it introduced a tool to automatically generate visualizations from the software documentation. Nassi and Schneiderman[18], in 1973, introduced the Nassi–Shneiderman diagram (NSD), able to represent the structure of a program. The diagram was divided into multiple sub-block, each with a given semantic based on its shape and position.

The 80s registered two main directions of software visualization. The first was the source code presentation. Hueras [10], and Waters [25] developed two techniques to format the source code with a prettyprinter. The second direction was the program behavior, used mainly for educational purposes. One of the most prominent visualization systems of that period was

Balsa-II [1]. Around the end of the 80s, Müller et al. [17] released Rigi, a tool able to visualize large programs. It exploited the graph model, argumented with abstraction mechanisms, to represent systems components and their relationships.

The 1990s recorded more interest in the field of software visualization. In 1992 Erik et al [6] introduced a new technique to visualize line-oriented statistics. It was embodied in Seesoft, a software visualization system that allowed to analyze and visualize up to 50,000 lines of code simultaneously. On their visualization, each line was mapped to a thin row. Each row was associated with a color that described a statistic of interest. One year later, De pauw et al. [5] introduced Jinsight, a tool able to provide animated views of the behavior of object-oriented systems.

That period was favorable also for experimenting with novel research directions for visualization, such as 3D visualization and Virtual Reality. In 1998, Chuah and Erick [2] proposed three different techniques to visualize project data. They exploited the concept of glyphs, a graphical object that represents data through visual parameters. The first technique was the Timewhell glyph, used to visualize time-oriented information (number of lines of code, number of errors, number of added lines). The second technique was the 3D wheel glyph; it encoded the same attributes of the time wheel, and additionally, it used the height to encode time. Infobug glyph was the last technique, where each glyph was composed of four parts, each representing essential data of the system (time, code size, number of lines of code added/deleted/modified).

In the same year, Young and Munro [27] explored representations of software for program comprehension in VR. Finally, in 1999, Jacobson et al. [11] introduced what we now know as de facto the standard language to visualize the design on a system: UML.

2.1.1 Software evolution visualization

Before the beginning of the 21st century, visualizing the evolution of a system was an unfeasible task due to the lack of data. However, thanks to the spread of version control systems and of the open-source movement, this information became publicly accessible. As a result, many researchers focused their work on software evolution visualization.

In 2001 Lanza [16] introduced the concept of the Evolution Matrix. It was a way to visualize the evolution of software without dealing with a large amount of complex data. Furthermore, that approach was agnostic to any particular programming language. The Evolution Matrix aimed to display the evolution of classes in object-oriented software systems. Each column represented a version of the software; each row represented a different version of the same class. Cells were filled with boxes whose size depended on evolutionary measurements. Thanks to this approach, he was able to infer some evolution patterns by just looking at the shape of the matrix

Taylor and Munro [23] in 2002 demonstrated that it is possible to use the data contained in a version control repository to visualize the evolution of a system. They also engineered Revision Tower, a tool that showed change information at the file level. In 2005 Pinzger et al. [19] visualized the evolution of a software system through Kivat diagrams. RelVis, their tool, was able to depict a multivariate visualization of the evolution of a system. During the same year, Ratzinger et al. presented EvoLens [20], a visualization approach and tool to explore evolutionary data through structural and temporal views. During the same year, Langelier et al. [15] investigated the interpretation of a city metaphor [13] to add a new level of knowledge to the visual analysis. In 2006, D'Ambros and Lanza [3] introduced Discrete Time Figure, a visualization technique that embeds both historical and structural data in a simple figure. Furthermore, they defined an approach based on the application of this visualization technique to study the evolution of a software system. As a result, they were able to depict relationships between the histories of a system and bugs. They also presented the Evolution Radar [4], a novel approach to visualize module-level and file-level logical coupling information.

In 2010 Steinbrückner and Lewerentz [22] described a tree-staged visualization approach to visualize large software systems. The visualization was supported by a tool called Evo-Streets. Each stage of their approach was responsible represent a different aspect of the system with the city metaphor.

Wettel, in his thesis [26], revised the city metaphor to represent metrics meaningfully. In his work, he represented packages as districts and classes as buildings. This metaphor was applied in different contexts related to reverse engineering (program comprehension, software evolution, software quality) to demonstrate metaphor's versatility. As a result, he found evidence that his approach works. However, he claims that city metaphor brings visual and layout limitations (not all visualization techniques fit well with it). Under those circumstances, he preferred simplicity over the accuracy, so he obtained a simple visual language that facilitates data comprehension. He conducted an experiment of the evidence that the city metaphor enables the creation of efficient software visualizations. His approach was implemented by a software visualization tool called CodeCity that supports the city metaphor.

Ens et al. [7] applied visual analytics methodologies to software repositoies with the goal to helps users comprehend co-evolution informaiton. It enabled to visulize, over time, how source and test fiels were developer together.

Kapec et al. [12] studied if it is possible to ease the graph analysis with argumented reality. They made a prototype of a tool that provided a graph-based visualization of a software and them, they studied some interaction methods to control it with argumented reality.

Schneider et al. [21] presented a tool, CuboidMatrix, that employed a space-time cube metaphor to visualize a software system. Space-time cube is wll-known 3D representation of an evolving dynamic graph.

Bibliography

[1] Marc H. Brown. Exploring algorithms using balsa-ii. *Computer*, 21(5):14–36, may 1988. ISSN 0018-9162. doi: 10.1109/2.56. URL `https://doi.org/10.1109/2.56`.

[2] M.C. Chuah and S.G. Eick. Information rich glyphs for software management data. *IEEE Computer Graphics and Applications*, 18(4):24–29, 1998. doi: 10.1109/38.689658.

[3] M. D'Ambros and M. Lanza. Software bugs and evolution: a visual approach to uncover their relationship. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 10 pp.–238, 2006. doi: 10.1109/CSMR.2006.51.

[4] Marco D'Ambros, Michele Lanza, and Mircea Lungu. The evolution radar: Visualizing integrated logical coupling information. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, page 26–32, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933972. doi: 10.1145/1137983.1137992. URL `https://doi.org/10.1145/1137983.1137992`.

[5] Wim De Pauw, Richard Helm, Doug Kimelman, and John Vlissides. Visualizing the behavior of object-oriented systems. In *Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '93, page 326–337, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897915879. doi: 10.1145/165854.165919. URL `https://doi.org/10.1145/165854.165919`.

[6] Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Trans. Softw. Eng.*, 18(11):957–968, nov 1992. ISSN 0098-5589. doi: 10.1109/32.177365. URL `https://doi.org/10.1109/32.177365`.

[7] Barrett Ens, Daniel Rea, Roiy Shpaner, Hadi Hemmati, James E. Young, and Pourang Irani. Chronotwigger: A visual analytics tool for understanding source and test co-evolution. In *2014 Second IEEE Working Conference on Software Visualization*, pages 117–126, 2014. doi: 10.1109/VISSOFT.2014.28.

[8] Jean-Daniel Fekete, Jarke van Wijk, John Stasko, and Chris North. *The Value of Information Visualization*, volume 4950, pages 1–18. 07 2008. ISBN 978-3-540-70955-8. doi: 10.1007/978-3-540-70956-5_1.

[9] Lois M. Haibt. A program to draw multilevel flow charts. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), page 131–137, New York, NY, USA, 1959. Association for Computing Machinery. ISBN 9781450378659. doi: 10.1145/1457838.1457861. URL `https://doi.org/10.1145/1457838.1457861`.

[10] Jon Hueras and Henry Ledgard. An automatic formatting program for pascal. *SIGPLAN Not.*, 12(7):82–84, jul 1977. ISSN 0362-1340. doi: 10.1145/954639.954645. URL `https://doi.org/10.1145/954639.954645`.

[11] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., USA, 1999. ISBN 0201571692.

[12] P. Kapec, G. Brndiarová, M. Gloger, and J. Marák. Visual analysis of software systems in virtual and augmented reality. In *2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES)*, pages 307–312, 2015. doi: 10.1109/INES.2015.7329727.

[13] C. Knight and M. Munro. Virtual but visible software. In *2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics*, pages 198–205, 2000. doi: 10.1109/IV.2000.859756.

[14] Donald E. Knuth. Computer-drawn flowcharts. *Commun. ACM*, 6(9):555–563, sep 1963. ISSN 0001-0782. doi: 10.1145/367593.367620. URL `https://doi.org/10.1145/367593.367620`.

[15] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. Visualization-based analysis of quality for large-scale software systems. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, page 214–223, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1581139934. doi: 10.1145/1101908.1101941. URL `https://doi.org/10.1145/1101908.1101941`.

[16] Michele Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, page 37–42, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581135084. doi: 10.1145/602461.602467. URL `https://doi.org/10.1145/602461.602467`.

[17] H. A. Müller and K. Klashinsky. Rigi-a system for programming-in-the-large. In *Proceedings of the 10th International Conference on Software Engineering*, ICSE '88, page 80–86, Washington, DC, USA, 1988. IEEE Computer Society Press. ISBN 0897912586.

[18] I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. *SIGPLAN Not.*, 8(8):12–26, aug 1973. ISSN 0362-1340. doi: 10.1145/953349.953350. URL `https://doi.org/10.1145/953349.953350`.

[19] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza. Visualizing multiple evolution metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, page 67–75, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930736. doi: 10.1145/1056018.1056027. URL `https://doi.org/10.1145/1056018.1056027`.

[20] J. Ratzinger, M. Fischer, and H. Gall. Evolens: lens-view visualizations of evolution data. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 103–112, 2005. doi: 10.1109/IWPSE.2005.16.

[21] Teseo Schneider, Yuriy Tymchuk, Ronie Salgado, and Alexandre Bergel. Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube.

In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 116–125, 2016. doi: 10.1109/VISSOFT.2016.17.

[22] Frank Steinbrückner and Claus Lewerentz. Representing development history in software cities. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, page 193–202, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300285. doi: 10.1145/1879211.1879239. URL `https://doi.org/10.1145/1879211.1879239`.

[23] Christopher M. B. Taylor and Malcolm Munro. Revision towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '02, page 43, USA, 2002. IEEE Computer Society. ISBN 0769516629.

[24] A. Von Mayrhauser and A.M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995. doi: 10.1109/2.402076.

[25] Richard C. Waters. User format control in a lisp prettyprinter. *ACM Trans. Program. Lang. Syst.*, 5(4):513–531, oct 1983. ISSN 0164-0925. doi: 10.1145/69575.357225. URL `https://doi.org/10.1145/69575.357225`.

[26] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: a controlled experiment. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 551–560, 2011. doi: 10.1145/1985793.1985868.

[27] P. Young and M. Munro. Visualising software in virtual reality. In *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242)*, pages 19–26, 1998. doi: 10.1109/WPC.1998.693276.