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

Software maintenance and evolution are essential parts of the software development lifecycle. Both require that developers deeply understand their system. Mayrhauser and Vans defined *program comprehension* as a process that "*knowledge to acquire new knowledge*" [37]. Generally, programmers possess two types of knowledge: general knowledge and software-specific knowledge. Software comprehension aims to increase this specific knowledge of the system, and, it can leverage some software visualization techniques for this purpose. Software visualization supports the understanding a software systems by visually presenting various information about it, e.g., its architecture, source code, or behavior. Stasko et al.[11] 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.

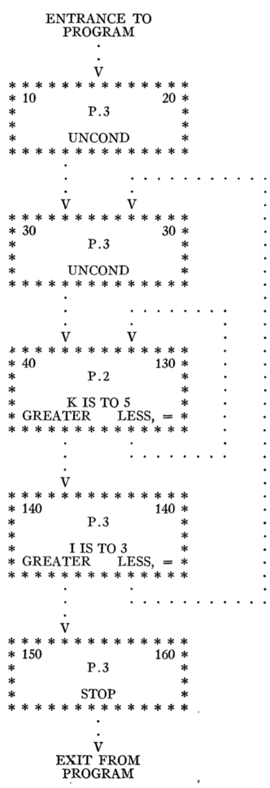

Figure 2.1. Flowchart presented by Haibt in 1959

The earliest software visualization techniques in the literature used 2D diagrams. For example, Haibt, the first to use them in 1959, provided a graphical outline of a program and its behavior with flowcharts [12]. As shown in Figure 2.1, 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, Knuth also confirmed the effectiveness of flowcharts [20].

Unfortunately, around that time, programs were affected by a lack of readability. Therefore, it introduced a tool to generate visualizations from the software documentation automatically. Nassi and Schneiderman[27], 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. For example, Hueras and Ledgard [13] then Waters [38] developed techniques to format the source code with a prettyprinter. The second direction was the program behavior, used mainly for educational purposes. One of that period's most prominent visualization systems was Balsa-II [3].

Balsa-II was a visualization system that, through animations, displayed the execution of an algorithm. Programmers were able to customize the view and the control execution of the algorithm, to understand them with a modest amount of effort. The program was domain-independent, and users could use it with any algorithm.

Around the end of the 80s, Müller et al. [26] released Rigi, a tool used to visualize large programs. It exploited the graph model, augmented with abstraction mechanisms, to represent systems components and relationships.

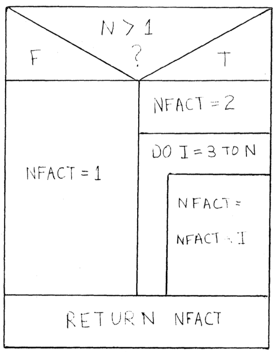

Figure 2.2. NSD of the factorial function.

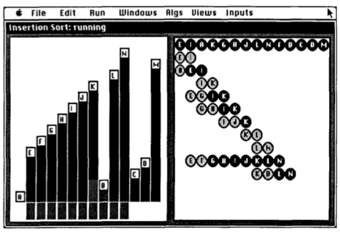

Figure 2.3. Balsa-II

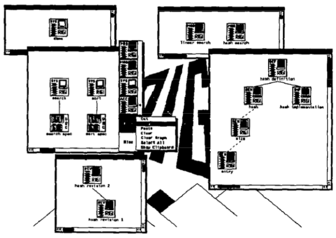

Figure 2.4. Rigi

The 1990s recorded more interest in the field of software visualization. In 1992, Erik et al. introduced a new technique to visualize line-oriented statistics [9]. It was embodied in Seesoft, a software visualization system 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. [7] introduced Jinsight, a tool able to provide animated views of object-oriented systems's behavior.

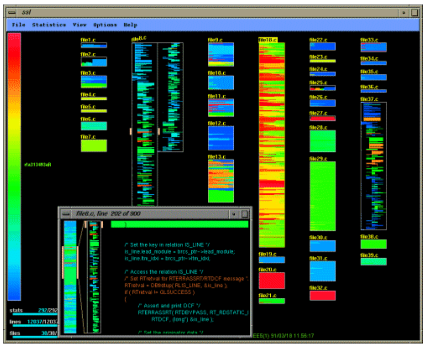

Figure 2.5. Seesoft

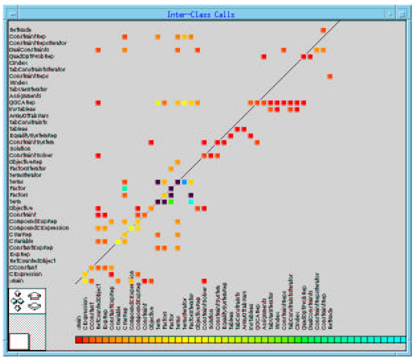

Figure 2.6. Jinsight

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 [4] 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, such as time, code size and the number of added, deleted, or modified code lines.

Figure 2.7. Timewhell

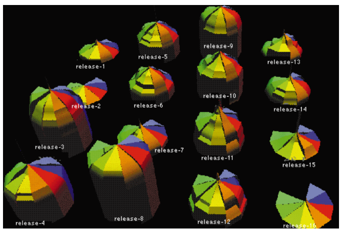

Figure 2.8. 3D wheel

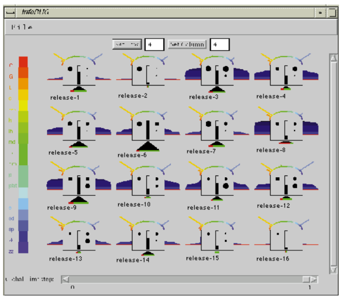

Figure 2.9. Infobug

Also in 1998, Young and Munro [40] explored representations of software for program comprehension in VR. Finally, in 1999, Jacobson et al. [14] introduced what we now know as de facto the standard language to visualize the design of a system: UML.

Before the beginning of the 21st century, thanks to the spread of version control systems and the open-source movement, visualizing the evolution of a system became a more feasible activity since there was more publicly accessible system information. As a result, many researchers focused their work on software evolution visualization.

Lanza [22] 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, this 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. The shape of the matrix could also be used to infer various evolutionary patterns.

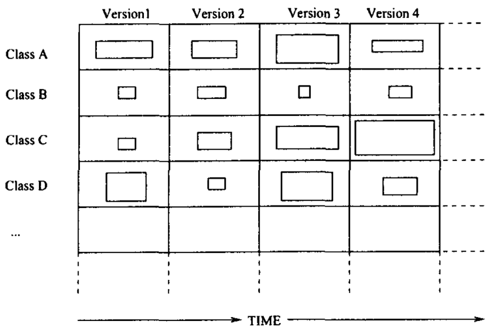

Figure 2.10. A schematic display of the Evolution Matrix

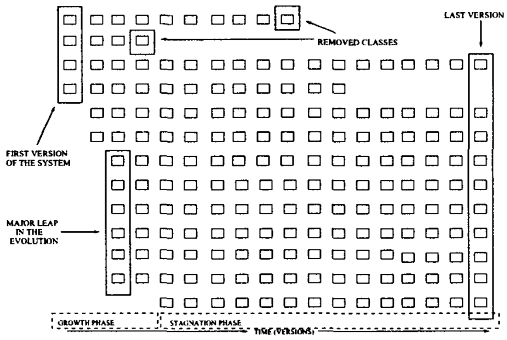

Figure 2.11. Some characteristics of the Evolution Matrix

Taylor and Munro [34], demonstrated that it was possible to use the data contained in a version control repository to visualize the evolution of a system. They developed Revision Tower, a tool that showed change information at the file level. Pinzger et al. [28] visualized the evolution of a software system through Kivat diagrams. RelVis, their tool, was able to depicted a multivariate visualization of the evolution of a system. During the same year, Ratzinger et al. presented EvoLens [29], a visualization approach and tool to explore evolutionary data through structural and temporal views. During the same year, Langelier et al. [21] investigated the interpretation of a city metaphor [19] to add a new level of knowledge to the visual analysis.

D'Ambros and Lanza [5] introduced the concept of Discrete Time Figure concept. It was a visualization technique that embedded both historical and structural data in a simple figure. Their approach depicted relationships between the histories of a system and bugs. They also presented the Evolution Radar [6], a novel approach to visualize module-level and file-level logical coupling information.

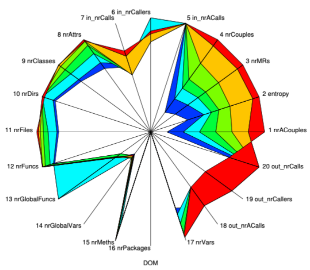

Figure 2.12. RelVis

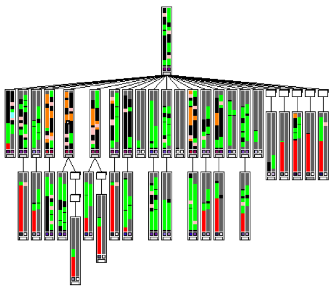

Figure 2.13. Tree of Discrete Time Figures

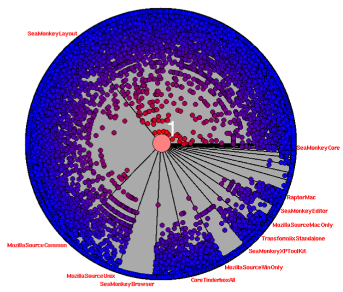

Figure 2.14. Evolution Radar

Steinbrückner and Lewerentz [33] described a three-staged visualization approach to visualize large software systems. Thir visualization was supported by a tool called Evo-Streets. Each stage of their approach was responsible for representing a different aspect of the system with the city metaphor.

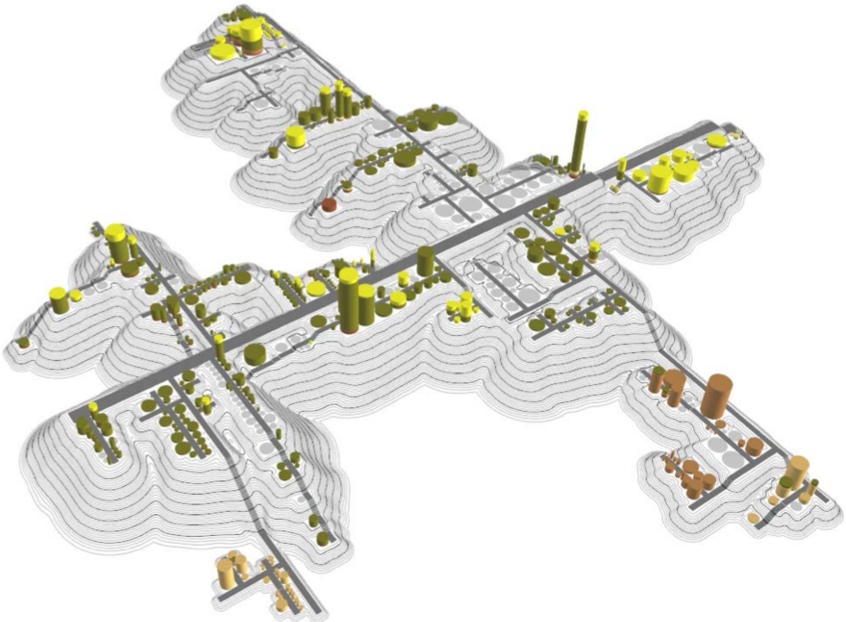

Figure 2.15. Evo-Streets

Wettel revised the city metaphor to represent metrics meaningfully [39]. In his thesis, he represented packages as districts and classes as buildings. The metaphor was used for various purposes, e.g., reverse engineering, program comprehension, software evolution or software quality analysis. He claimed that the city metaphor brought visual and layout limitations; for example, not all visualization techniques fit well. Under those circumstances, he preferred simplicity over accuracy, so he obtained a simple visual language that facilitated data comprehension. His approach was implemented as software visualization tool called CodeCity.

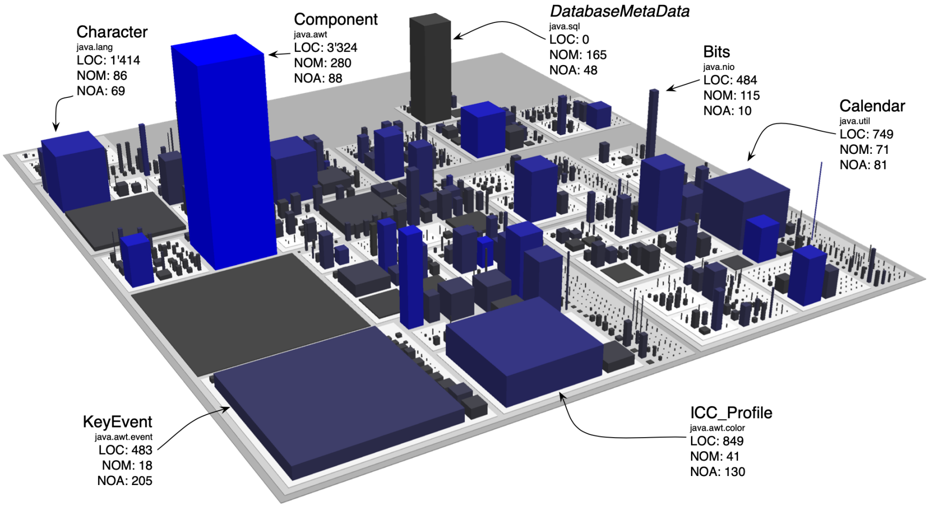

Figure 2.16. CodeCity

Ens et al. [10] applied visual analytics methods to software repositories. His approach helped users comprehend co-evolution information by visualizing how source and test files were developed together.

Kapec et al. [17] proposed a graph analysis approach with augmented reality. They made a prototype of a tool that provided a graph-based visualization of software, and then they studied some interaction methods to control it with augmented reality.

Schneider et al. [30] presented a tool, CuboidMatrix, that employed a space-time cube metaphor to visualize a software system. A space-time cube is a well-known 3D representation of an evolving dynamic graph.

Merino et al. [25] aimed to augment software visualization with gamification. They introduced CityVR, a tool that displays a software system through the city metaphor with a 3D environment. Working with virtual reality, they scaled the city visualization to the physically available space in the room. Therefore, developers needed to walk to navigate the system.

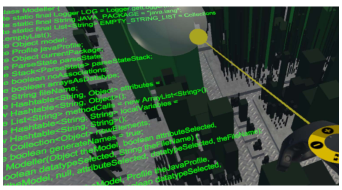

Figure 2.17. CityVR

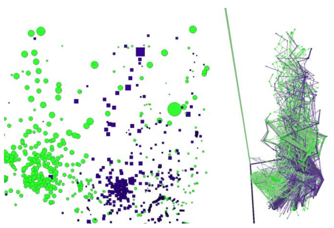

Figure 2.18. ChronoTwigger

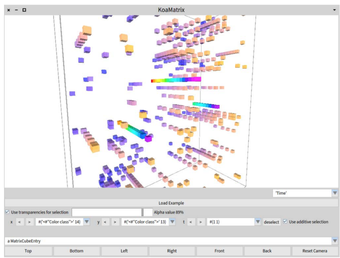

Figure 2.19. CuboidMatrix

Khaloo et al. [18] revised the idea of gamification with a 3D park-like environment. They mapped each class in the codebase with a facility. The wall structure depended on constituent parts of the class e.g., methods and signatues.

Finally, we mention Alexandru et al., who proposed a method to visualize software structure and evolution, with reduced accuracy and a fine-grained highlighting of changes in individual components [1].

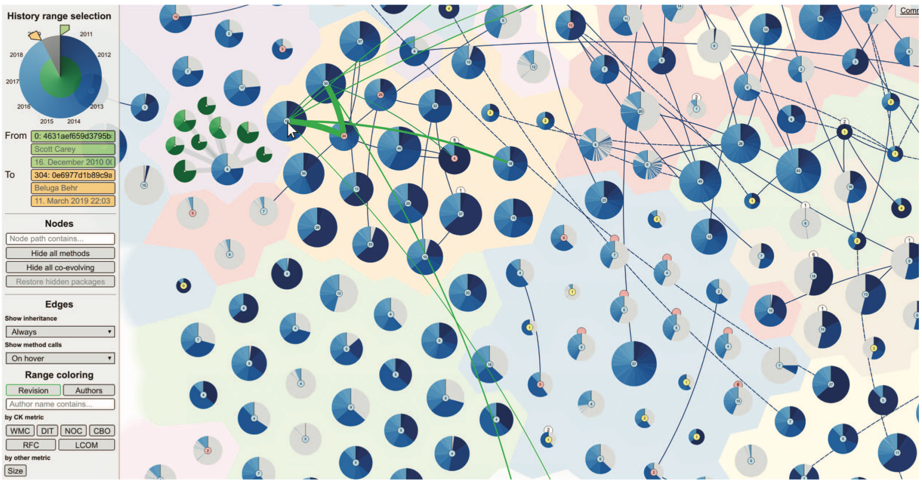

Figure 2.20. Evo-Clock

2.2 Mining software repositories

Software repositories contain historical data about the evolution of a software system. Thanks to the spread of the git protocol, and consequently of GitHub, the largest code hosting website, Mining software repositories (MSR) became a popular research field.

In 2022, the number of GitHub repositories lays around 200 million. Even if it seems a promising source of data, Kalliamvakou et al. [16] raised some issues with its mining. They evidenced that a repository doens't always match with a project. A reason of this can be found in the fact that most repositories had had very few commits before becoming inactive. When they made their research, over the 70 percent of the GitHub projects were personal, and some of them weren't used for software development. Finally, the last perils that they raise, were related to GitHub features that are not properly used by software developers. To find actively developed repositories, they considered only projects with a good balance between number of commits, number of pull requests and number of contributors.

Spadini et al. [32] developed a Python framework, called PyDriller, that enables users to mine software repositories. They showed that their tool can be used to extract information about the evolution of a software system from any git repository.

2.3 Data sonification

External auditory representations of programs (known as program auralisation) is a researching field that is getting even more interest in the recent years.

One of the first attempts at program was described by Sonnenwald et al [31] They tried tp enhance the comprehension of complex applications by plating music and special sound effects. The prototype system that supported this approach was called InfoSound. This first approach, seen its adoption for the understanding of the program's behavior.

This approach was followed by many other researchers. To cite some of them, DiGiano and Baecker [8] made LogoMedia, a tool to associate non-speech audio with program events while the code is being developed. Jameson [15]] developed Sonnet, an audio-enhanced monitoring and debugging tool. Alty and Vickers [36] had a similar idea: using a structured musical framework, they was able to map the execution behavior of a program to locate and diagnose software errors.

Despite the usefulness of these tools, the adoption of a simple mapping lead them to have a limited musicality representations. Vickerts [35] evidenced the necessity of a multi-threaded environment to enhance the comprehension given by the musical representation. He proposed the adoption of an orchestral model of families of timbres, to enable programmers to distinguish between different activities of different threads.

The size and the complexity of systems can be a problem for the effectiveness of a visual representation of a software system. Having a large number of visual information, observers might find difficulty to focus only on the relevant aspects. Boccuzzo and Gall [2] supported software visualization with sonification. They used audio melodies to improve navigation and comprehension of their tool, CocoViz. Their ambient audio software exploration approach, exploited audio to intuitively describe the position of an entity in the space. Thanks to the adoption of surround sound techniques, the observers perceived the origin of an audio source so, it could adjust his navigation in the visualization. Each kind of entity played a different sound, based on a mapping criteria.

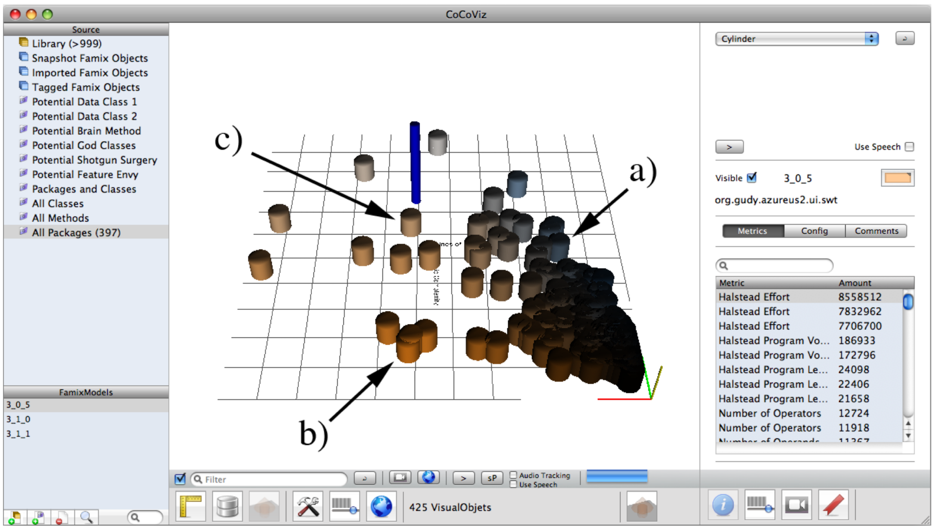

Figure 2.21. CocoViz

McIntosh et al. [24] explored the use of a parameter-based sonification, to produce a musical interpretation of the evolution of a software system. They technique mapped musical rests to inactive period of development and consonance and dissonance to interersting phenomena (like co-changing of components).

Mancino and Scanniello [23] presented an approach to transform source code metrics in a musical score that can be both visualized and played.

Chapter 3

Approach

In this chapter we will present an approach that produce a visual and auditive depiction of the evolution of a system. Our approach consists of two parts: In the first part we modelled the evolution of the software artifacts, and we enginereed a tool that implement it. In the second part, we used the concept of synesthesia (the production of a sense impression relating to one sense by stimulation of another sense) to enhance the effectiveness of the visualization.

3.1 Evolutionary model

Analyzing the evolution of a systems requires to consider numerous aspects. In our approach we focus on systems that resides on git repositories. We made this choiche because git is the most common repository management system and it also track all the changes made to the system. As a consequence we can use it to reconstruct the history of a system.

In our approach, we model the evolution of repository files, therefore we model its history. To do that, we considered all the information that can be extracted from a git repository: files and commits.

Git uses commit to record changes on a repository. So, each commit is a snapshot of the repository at a particular point in time. Git has the possibility to inspect every commit of a repository by using the command `git checkout`. In this way, we can navigate through the history of a repository, track all the files and their metrics.

A commit operation contains also other meta-information such as the author, the date, the message and the list of files that were updated. In our approach a commit is represented with a Project Version, and a file is represented with a FileHistory. Both a FileHistory and a ProjectVersion contains a list of FileVersions, wich represents a file at a particular point in time.

For example, if we have a file A.java that is being modified in commit c1 and c3, we will have a FileHistory A.java with 2 FileVersions: the fileVersion of c1 and the fileVersion of c2. With the same fashion, ProjectVersions c1 and c3 will have the same FileVersions referred to the FileHistory A.java.

Chapter 4

Implementation

This chapter details sensorial SYN, a tool that implements the software evolution comprehension approach defined in chaper X.

4.1 Platform overview

SYN is a platform tools that allows developers to have a visual and auditive deption of an evolving system. This section aims to describe the tools and modules that are part of SYN.

4.1.1 SYN CLI

SYN CLI is a command line interface that allows developers to interact with SYN. It gives to developers full control over the system. For example, with the command `syn project create` it is possible add a project and then, analyze it with the command `syn analyze`.

* LIST OF AVAIL COMMANDS - APPENDIX? *

4.1.2 SYN Analyzer

SYN works with evolutionary metrics that represents the history of a system. To this aim, we developed SYN Analyzer, a Java tool on top of jGit. Having a languange-agnostic implementation, it can analyze every git repository written in any programming language.

Four steps compose the analysis process:

1. The repository is cloned.
2. The source code of the HEAD revision is obtained.
3. All the files are analyzed and the metrics are obtained.
4. If the revision has a parent, the step 2 is repeated with the parent revision.

As a result, starting from the HEAD revition it will go back in time until the first revision. All the collected metric will be stored in a object, called analysis result, that can be serialized in a JavaScript Object Notation (JSON) object. We chose JSON because is a lightweight, easy to use and human readable format. It is capable to analyze large repositories, as it uses a join algorithm to merge these analysis results if they are computed in parallel.

4.1.3 SYN Server

SYN Server is responsable for providing the elaborated information, given by the analysis results, in an intermediate languange between the front-end (SYN Debugger) and the back-end. We chose to spin up a GraphQL web server, that uses JSON as exchange language between the front-end and the back-end. In this way, the front-end can ask exactly for the information it needs, and the back-end can send it back once they are computed.

The computation made by the back-end, is responsable to create the view that will be shown in the front-end. To do that, the server process a *view specification, that must be given by the front-end, and then provides to the front-end JSON objects representing only the object that must be depicted. Although the information provided by the server are limitated to the view itself, it also provides debugging information if requested.*

4.1.4 SYN Debugger

SYN Debugger is a web application that allows developers to interact with SYN. It is written with React.js, a popular JavaScript framework. The aim of this application is to have a visual depition of the view generated by the server, plus some additional information. For example, it allows you to ckick on a entity and see the information that is related to it. The visualization is based on Babylon.js, a popular 3D library. SYN Debugger provides different kind of customizations to the view, such as the shape and the colors of the entities. All theese customizations are sent to the back-end server, through a view specification file.

The main purpose of this application is to debug the view and explore all the possible visualization combinations of a system.

4.1.5 SYN Sonic

....

Bibliography

[1] *Carol V. Alexandru, Sebastian Proksch, Pooyan Behnamghader, and Harald C. Gall. Evo-clocks: Software evolution at a glance. In* 2019 Working Conference on Software Visualization (VISSOFT)*, pages 12–22, 2019. doi: 10.1109/VISSOFT.2019.00010.*

[2] *Sandro Boccuzzo and Harald C. Gall. Cocoviz with ambient audio software exploration. In* 2009 IEEE 31st International Conference on Software Engineering*, pages 571–574, 2009. doi: 10.1109/ICSE.2009.5070558.*

[3] *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`.

[4] *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.*

[5] *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.*

[6] *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`.

[7] *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`.

[8] *Christopher J. DiGiano, Ronald M. Baecker, and Russell N. Owen. Logomedia: A sound-enhanced programming environment for monitoring program behavior. In* Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, page 301–302, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897915755. doi: 10.1145/169059.169229. URL* `https://doi.org/10.1145/169059.169229`.

[9] *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`.

[10] *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.*

[11] *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.*

[12] *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`.

[13] *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`.

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

[15] *David H Jameson. Sonnet: Audio-enhanced monitoring and debugging. In* SANTA FE INSTITUTE STUDIES IN THE SCIENCES OF COMPLEXITY-PROCEEDINGS VOLUME-*, volume 18, pages 253–253. ADDISON-WESLEY PUBLISHING CO, 1994.*

[16] *Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In* Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 92–101, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328630. doi: 10.1145/2597073.2597074. URL* `https://doi.org/10.1145/2597073.2597074`.

[17] *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.*

[18] *Pooya Khaloo, Mehran Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. Code park: A new 3d code visualization tool. In* 2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 43–53, 2017. doi: 10.1109/VISSOFT.2017.10.*

[19] *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.*

[20] *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`.

[21] *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`.

[22] *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`.

[23] *Andrea Mancino and Giuseppe Scanniello. Software musification. In* 2017 21st International Conference Information Visualisation (IV)*, pages 127–132, 2017. doi: 10.1109/iV.2017.28.*

[24] *Shane McIntosh, Katie Legere, and Ahmed E. Hassan. Orchestrating change: An artistic representation of software evolution. In* 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 348–352, 2014. doi: 10.1109/CSMR-WCRE.2014.6747192.*

[25] *Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. Cityvr: Gameful software visualization. In* 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 633–637, 2017. doi: 10.1109/ICSME.2017.70.*

[26] *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.*

[27] *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`.

[28] *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`.

[29] *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.*

[30] *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.*

[31] *D.H. Sonnenwald, B. Gopinath, G.O. Haberman, W.M. Keese, and J.S. Myers. Infosound: an audio aid to program comprehension. In* Twenty-Third Annual Hawaii International Conference on System Sciences*, volume 2, pages 541–546 vol.2, 1990. doi: 10.1109/HICSS.1990.205229.*

[32] *Davide Spadini, Maurício Aniche, and Alberto Bacchelli. Pydriller: Python framework for mining software repositories. In* Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 908–911, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355735. doi: 10.1145/3236024.3264598. URL* `https://doi.org/10.1145/3236024.3264598`.

[33] *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`.

[34] *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.*

[35] *Paul Vickers. External auditory representations of programs: Past, present, and future an aesthetic perspective. 01 2004.*

[36] *Paul Vickers and James L. Alty. Siren songs and swan songs debugging with music.* Commun. ACM*, 46(7):86–93, jul 2003. ISSN 0001-0782. doi: 10.1145/792704.792734. URL* `https://doi.org/10.1145/792704.792734`.

[37] *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.*

[38] *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`.

[39] *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.*

[40] *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.*