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. Student's Advisor
co-supervised by
Prof. Student's Co-Advisor

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

Essential parts of software lifecycles are software maintenance and software evolution. Both activities require the comprehension of the system by the developer. Mayrhauser [7] 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.[2] 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.

2.1 Software visualization

The earliest form of software visualization found in the literature was in the form of 2D diagrams. Haibt [3] in 1959, was one of the first to use diagrams to visualize software systems. He provided a graphical outline of a program and its behavior with flowcharts. As shown in Figure 2.1 they were 2D diagrams that describes the execution of a program. Ten years later, the effectiveness of flowcharts was confirmed by Knuth [4]. It also created a tool to automatically generate visualizations from the software documentation. Nassi and Schneiderman[6], in 1973, introduced the Nassi–Shneiderman diagram (NSD), that represents the structure of a program. This diagram is divided into multiple sub-block, each one with a given semantic based on their shape and position.

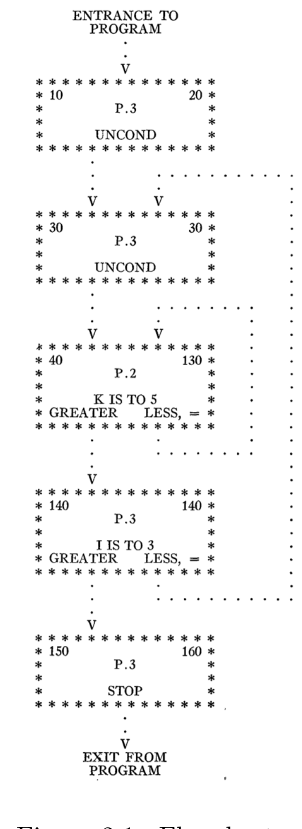


Figure 2.1. Flowchart presented by Haibt in 1959

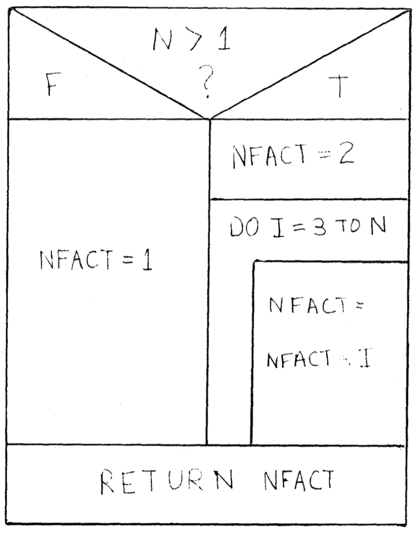


Figure 2.2. NSD of the factorial function.

In the same year, Chuah and Erick [1] 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).

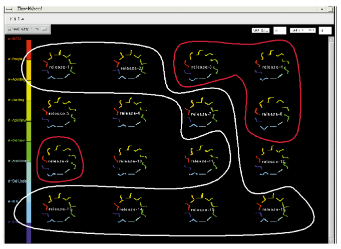


Figure 2.3. Timewhell

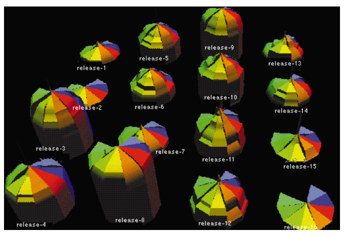


Figure 2.4. 3D wheel

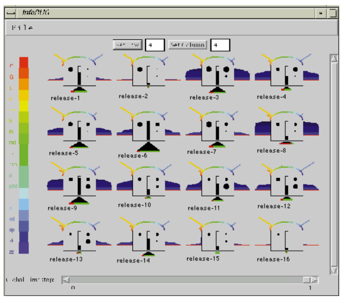


Figure 2.5. Infobug

In 2001 Lanza [5] introduced the concept of the Evolution

Matrix. It was a way to visualize the evolution of software with-
out 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, and each row represented a different version of the same class. The cells were filled with boxes whose size depended on two different metrics. Thanks to this approach, he was able to infer some evolution information by just looking at the shape of the matrix

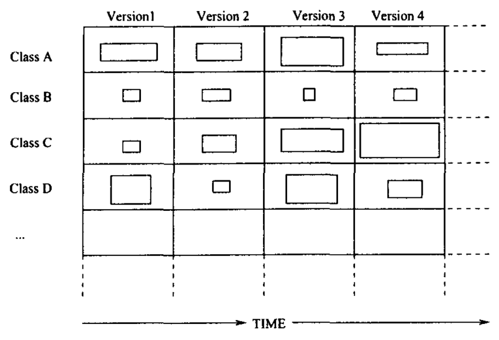


Figure 2.6. A schematic display of the Evolution Matrix

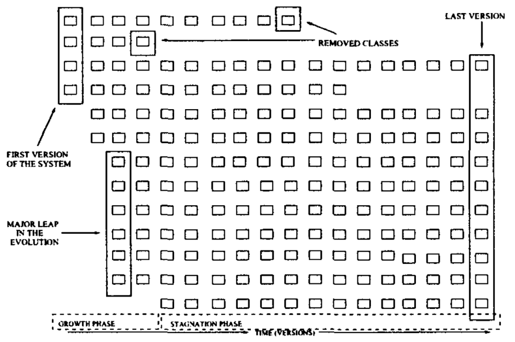
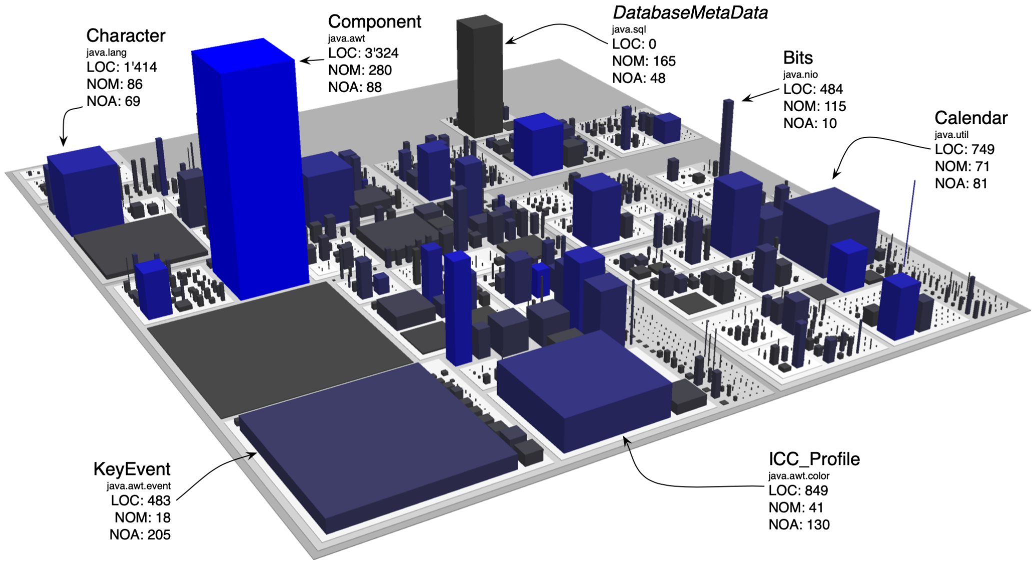


Figure 2.7. Some characteristics of the Evolution Matrix

Wettel, in his thesis [8], defined a city metaphor for software visualization that represents software as cities. His work represents 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 comprehension of data. 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.



Appendix A

Some retarded material

A.1 It's over…

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Bibliography

[1] 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.

[2] 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.

[3] 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`.

[4] 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`.

[5] 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`.

[6] 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`.

[7] 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.

[8] 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.