Using Animations to Understand Commits

Carmen Armenti*, Michele Lanza*

*REVEAL @ Software Institute – USI, Lugano, Switzerland

Abstract—**Commits, which log the changes that have been performed by developers, are the central mechanism to drive the evolution of software systems. Understanding the intricacies of commits can be a non-trivial endeavour. Firstly, this is due to the diff-based textual nature of how versioning systems record the changes. Moreover, a commit can involve several files and pertain to various, overlapping tasks that the developer was tackling, which can lead to difficult to understand "tangled commits". Furthermore, often commit messages lack quality. The only mechanism to really understand the changes performed in a commit is given by text-based "diff" representations, which are cumbersome to use.**

We present an approach, based on interactive animated visualizations, to facilitate the comprehension of the changes tracked by commits. To validate the approach, we implemented an interactive visual analytics tool which allows developers to dissect a commit in its constituent parts and observe, through the animations supported by our tool, the specifics of each change. We illustrate our approach with examples, and report on our findings and insights.

Index Terms—**software animation, program comprehension, software evolution**

I. INTRODUCTION

Program comprehension is a complex mental activity which can take up to 70% [1] of the time of developers: Indeed, developers spend substantial amount of time reading and understanding source code, progressively piecing together a coherent mental model [2] which is necessary to be able to perform the required changes [3], [4]. The same scenario applies to software evolution where, indeed, *"people are still reviewing changes by looking at static diffs"*[1]. When analyzing commits, *e.g.,*, during pull request reviews or code review activities, people are faced with walls of (diff) text that pertain to the parts of the system that have been changed.

The understanding is not only required at a finer-grained level, but can also reside at the level of grasping "who did what, when and why". Commits, and the information that they contain, are the starting point for that.

Our work is thus not directly related to code review [5], where proposed changes to source code are reviewed by a small number of developers before being integrated. While we also believe that code review tools are in desperate need of ameliorations, the focus of this work is on understanding a commit that has been performed in the past. One might argue that a commit is appropriately explained in the comment that accompanies a commit. Research has however proven that commit comments, if they even exist, are often of negligible quality [6]–[9], which has generated a slew of research on analyzing and even automatically generating them [10]–[12].

[1]🌐 See: https://x.com/girba/status/1748071471820263894?s=20

Visualization is a well-known program comprehension technique, and indeed there have been approaches that used it both to statically depict commits [6] and dynamically display the history of repositories [13], [14]. However, a commit is not a static entity, it transposes (part of) a software system from one version to the next. Hence, we believe that while visualization has potential, it needs to be augmented with a technique that adequately represents the transitional nature of a commit. A pioneer of software visualization, John Stasko, claimed that "Programmers can more easily understand programs displayed in an animated graphical way" [15]. Taking inspiration from Stasko's work we present a novel approach to aid the understanding of commits by leveraging the potential of animations. We use interactive visualization techniques to represent the commit context *before* the commit, and a set of animations to depict how the context changes due to the commit, resulting in the system's new state *after* the commit.

We implemented a tool exemplifying our approach, which provides also interaction capabilities. What we believe is the main strength of our approach is the interactivity of our animations which offers the ability to investigate and inspect the files pertaining to the commit, which goes beyond the basic features offered by animations tools, such as pausing, re-winding or replaying the animation steps: At any moment in time our tool allows one to inspect every entity partaking in a commit. We describe our approach, present the tool supporting the approach, and illustrate both on selected examples, reflecting on the findings and discussing the future potential of the idea.

II. RELATED WORK

Software visualization seeks to ease program comprehension [16], aiding the user by providing insights and understanding [17], [18]. Yet, most of the views depict static information thus struggling to represent dynamic aspects of software systems, *e.g.,* software behaviour. Often, a demonstration of the states through which the system passes is needed, especially when the scope of the representations is software systems, which are dynamic by nature [19].

Stasko proposed an approach to convey the meaning and purpose of programs, based on animated graphical views [15]. He stated that the meaning, methodology and purpose of a program are better explained by algorithm animations than program's textual representation, and that program animations help with: (a) program understanding, also useful in computer science education; (b) evaluating existing programs, helping in monitoring system performance; (c) developing new programs, illustrating behaviors not evident during their initial design and serving as a "graphical debugger".

The use of animations was employed in other disciplines other than software engineering [19]–[21]. Algorithm animations have been used primarily for instructional purposes, but also for industrial prototyping and simulation [22].

Salomon reports that animations, by dynamically displaying a process or a procedure, compensate for a student's scarce aptitude or skill to imagine motions [23]. Dynamic visualizations help in visualizing a process reducing cognitive load compared to a situation in which the process or the procedure has to be reconstructed from a series of static pictures [24]. Hoffler *et al.* conducted a meta-analysis of 26 primary studies, yielding 76 pair-wise comparisons of dynamic and static visualizations, which revealed a medium-sized overall advantage of instructional animations over static pictures [25].

The rationale for the effectiveness of animations is based on the *cognitive load* theory, which provides a theoretical foundation to explain the superiority of educational animations over static graphics [26]–[28]. It reports that by viewing animations, learners do not exert cognitive effort to mentally construct dynamic representations, releasing cognitive resources, which could be used for learning-related activities and deep processing. Learning with static visual representations requires information integration and inferential reasoning, imposing mental load on learners. Animations are a sequence of visual representations composed by sequential frames, and one of the purposes they are used for is to ease the understanding of the functioning of dynamic systems that change over time [29].

Commits, atomic units of change, by tracking information about the evolution of systems are a valuable resource to support software evolution research. Researchers aimed at *understanding commits* while analysing history logs of software systems to comprehend their evolution. Hattori *et al.* [30] proposed a size segmentation of commits based on the number of files they contain. Alali *et al.* [31] also aimed at characterising commits, including file and hunk diffs. Commits adhere to some rationale [32]. Tao *et al.* [33] found that the most important, the most frequent according to Codoban *et al.* [34], information needed to understand commits is logic. The logic of a piece of software, documented by commits, is conveyed by textual code-diffs. However, reading and understanding text, and thus code-diffs, is a demanding task [15], [18], [35].

Commits analysis and understanding is key in the context of program comprehension. The available tools either rely on overwhelming textual representations of code-diffs or on more effective, although highly coarse-grained, animated displays. Tools such as GitLens and GitGraph allow the exploration of commits histories. However, the systematic way they offer to control different versions of files as they evolve (file diffs), is text-based. Similarly, the vast majority of code reviews tools, such as Gerrit or the GitHub web interface, present the changes to review as a list of textual diff-hunks.

Visualization tools such as Gource allow to visualize the activity of a git repository in an animated fashion, but often follow a "play and stay still" approach: The final movie does not allow for interaction and inspection of the changes being displayed, and the granularity depicted is fairly coarse-grained.

Even though visual interfaces to explore commits history have been provided, developers broadly rely on code-diffs to understand what happened in a commit. Relying on textual information is useful to some extent, but generates issues pertaining to comprehension – an exemplifying evidence of this was presented by Fregnan *et al.* [36]: Code-diffs are the first information every developer looks at when a commit message is not well written. On the other hand, the reviewed literature shows that animations are useful for representing and explaining concepts that are difficult to be comprehended at an abstract level. Similarly, text processing is more difficult than visual processing for human brains.

Our vision of commits understanding is based on a "play, watch, interact" rather than a "read-only" or "play and stay still" approach. Therefore, we base our idea on intuitive and interactive animated visualization, where the change between two sequential commits is represented in its dynamic nature and where interaction to further inspect and analyze the content of each file version displayed is provided.

III. APPROACH

Commits are depicted as text-based "diff" representations by all modern tools. This respects the way that VCS track system versions, but neglects the real dynamics of how files mutate from one version to another. It also limits one to consider individual files, while in fact the files pertaining to a commit often change in concert. With respect to *time*, diff-based changes logging has limitations: It squashes the time period during which changes have occurred discarding relevant details [6], [30], [37]; it hinders program comprehension, leading developers to spend most of their time understanding *"why code is implemented the way it is"* [3]. The central idea of our approach (see Figure 1) is to reveal the evolutive nature of file versions changes through the means of *animations*.

A. Data Preparation

We use PyDriller [38] to mine a repository, with each commit undergoing further analysis using Cloc [39]. For merge commits, we run additional custom analysis scripts to retrieve the modifications specific to such commits, which PyDriller is limited in doing. The data is stored in .csv files.

Commits are our unit of observation: A single commit is decomposed into its constituent parts, namely all the files that have been added, deleted, renamed, and modified.

B. Animation Composition

The central part of our approach deals with composing the animation, split into 5 stages sequential in time, as follows:

1) **Pre-Commit.** Each file participating to the commit is visualized, together with its name and a bar, whose length encodes the lines of code of the file version *before* the commit. Files that will be removed during the commit are colored red; files boxes that will be added during the commit are not depicted at this stage, only their names are; files that are modified are color-coded: They are light green if they grow during the commit and light red if they shrink.
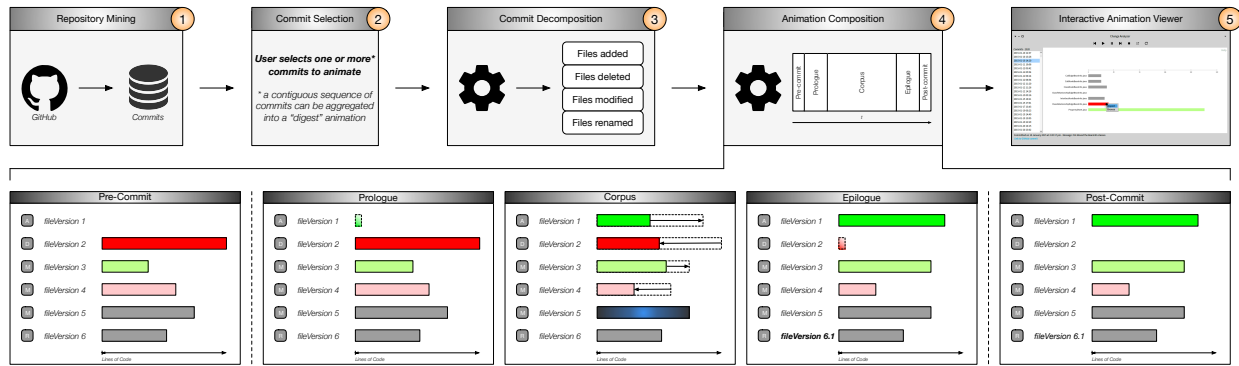
Fig. 1. Overview of the approach.

2) **Prologue.** This stage is dedicated to showing the appearance of newly added files during the commit, using for each one of them a fade-in effect. This allows users to understand at one glance which files are new. The newly added files are color-coded as green bars, their length is kept to a minimum value, as their growth will be depicted during the main *corpus* stage.

3) **Corpus.** This is the main stage of the animation, and takes up most of the time. Files that grow or shrink because of the commit, grow/shrink correspondingly. Files which are changed but do not grow or shrink (i.e., lines are substituted) are depicted as "activated" bars, following a fading color scheme going from grey (only 1 line is substituted) to deep blue (all lines are substituted), depending on how many lines within the files have been touched.

4) **Epilogue.** This stage is dedicated to showing the disappearance of files deleted during the commit, using for each one of them a fade-out effect. This allows users to understand at one glance which files are gone. For renamed files, this stage will fade out the old and fade in the new name.

5) **Post-Commit.** This stage visualizes the situation of the files participating to the commit after the commit has been performed. Only the names of deleted files are visible; the length of each file bar represents now the lines they possess after the commit, the colors still denote what type of change the files have experienced due to the commit.

The duration of an animation and its phases can be set by the user. Experience values are 5-10 seconds for the whole animation, most of which is taken up by the corpus.

C. Interactive Animation Viewer

The animation is fed to a tool we implemented, which allows users to pick the commit they want to animate, and offers several means to interact with the animation, such as pausing, rewinding, stepping, etc. Moreover, it provides a one-click direct access to the GitHub diff viewer pertaining to a particular file version. The Viewer has additional features, such as exporting the animation as a movie file or as a set of stills (sets of pictures). Lastly, the viewer offers the modification of animation-specific parameters, such as frame rate and animation duration.

IV. USING ANIMATIONS TO UNDERSTAND COMMITS

Using a concrete example taken from JetUML[2], we illustrate how we use animations to understand commits. Our tool depicts an animation as an interactive "movie", which can be paused, rewound, etc., and also each entity can be inspected at any time. This is difficult to illustrate in a paper, but can be viewed as YouTube video (link included in the caption of Figure 2), while here we decompose the animation into a set of frames[3], which we use for the discussion.

A. Commit Animation – An Example

Figure 2 pictures the commit 2529943[4] and the commit message is "Move methods from ViewerUtils. Methods in ViewerUtils belongs either to ViewUtils or to DiagramViewer. This commit moves the method to their better location and deletes class ViewerUtils".

Since no file was added, pre-commit and prologue phases are identical. While 2 files grow (*DiagramViewer.java*, *ViewUtils.java*), one gets deleted (*ViewerUtils.java*) shrinking to its minimum size and fading out at the end of the epilogue phase (frames 3-6). As the commit message claims, the content of *ViewerUtil.java* was moved into the files that grow. This lead to resolving dependencies in all the other files, whose bars change color according to the magnitude of the refactoring (yellow and blue).

V. PRELIMINARY EVALUATION

We performed a preliminary evaluation, also to collect feedback, by exposing single commit animations to several people (developers and PhD students). Our goal was to observe and discuss with them what they could understand about the commits under investigation simply by glancing at the animations. The commit references are as follows:

- Commit 4612af4 – December 12, 2017 – "Add basic code/decode to JSON";

---

[2] See: https://github.com/prmr/JetUML

[3] We only show main frames; the animation is composed of many more frames, whose number depends on the settable animation duration and the desired frame rate. For example, a 5 seconds animation at 30 FPS will generate a total of 150 frames.
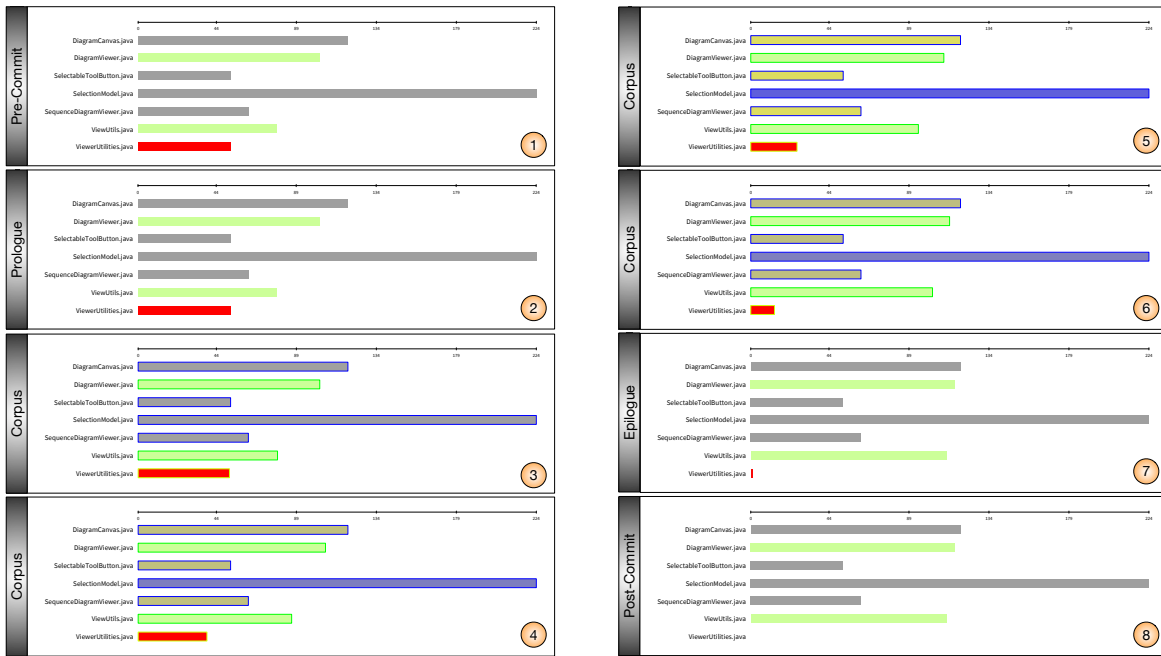
[4] See: https://github.com/prmr/JetUML/commit/2529943

Fig. 2. JetUML – Feb 6, 2022. ▶ Commit animation video: https://youtu.be/5-7ZNMNCDAI

- Commit `2529943` – February 6, 2022 – "Move methods from ViewerUtils. [...]".

The feedbacks we collected indicated that the commit animations, if compared to the corresponding textual diff, are intuitive and allow for a quick summary of the changes of each file version tracked by the commit.

The length of the bars was readily matched to the lines of code of file versions – the ruler at the top of the visualizations aided in this regard. Also, the metaphor of growing and shrinking bars was effortlessly identified with files where lines of code were added or removed.

The color mapping was simple to comprehend: It mimics colors used by Git and the GitHub web interface to display diffs. Although the color(s) of files that changed but did not grow or shrink (whose bars follow a fading color scheme) differ from those used in other interfaces/tools, most of the people could still identify them as just modified files.

The objective behind the interactive animations we propose is to give a means to quickly summarize all the changes that occurred in a commit and interact with each of them: Interaction is essential, indeed. While the animations are intuitive, they lack fine-grained information that is contained in textual code-diffs. In light of this, our visualizations allow to inspect the raw content of the file versions and browse the textual diff on the GitHub web interface. The latter can be reached through a click on each bar.

The feedbacks indicate that the ability to browse and interact with each file version was deemed valuable, as it links the intuition of the nature of the change to the detailed and logical change provided by the text-based diff. Also, some of the people observed that animations can be beneficial to identify relationships between file versions, *e.g.,* when pieces of code are removed from one file and added to another – as it happens in the example commit (Section IV-A).

Overall, the feedbacks showed that the visual metaphor augmented with animation, albeit simple, makes the comprehension of the file versions animations intuitive. Also, what struck most of the people was that they were capable of understanding the essence of commits without needing to look at source code and/or textual diff representations.

VI. REFLECTIONS ON THE IDEA

The idea we presented is in its early stages. The purpose of our approach was to supply a means to understand commits using animated and interactive visualizations: We aimed at providing a file-based granularity summary of the overall changes using a simple yet intuitive visual metaphor. In line with this purpose, the feedback we have collected has been generally positive. Yet, much future work lies ahead of us.

(Comparative) Evaluation. The anecdotal evidence that the approach shows promise is not enough to make any claims of substance. In the near future we plan on running a controlled experiment to perform a comparative evaluation. The baseline is the *de facto* standard diff representation offered by GitHub. The variables we want to control are *time* (can people save time understanding a commit with the help of animations?) and *accuracy* (does the use of animations lead to a better understanding of the intricacies of a commit, or is there an upper bound of what can be represented with the help of animations?).

Color Schemes. The colors we have used to encode the properties of the files involved in a commit are currently hard-coded. As there is no semantics in a particular color, we will implement the possibility of user-settable color schemes.
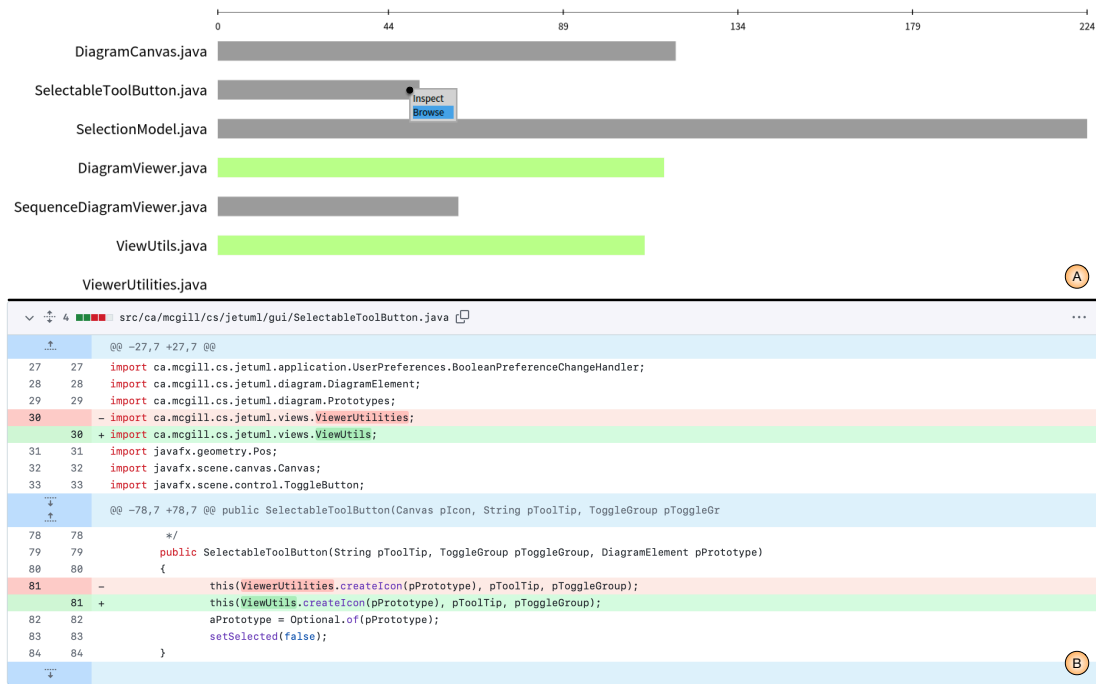
Fig. 3. The visual and animated representation is interactive (A). The textual diff representation provided by the GitHub interface (B) is accessible through a click on the bars at any step of the animation. The details of the diffs can be inspected by clicking on the glyphs. At the top the last state of the animation of the commit `2529943` (A); at the bottom the textual counterpart (B). ▶ See how to interact with the animations: https://youtu.be/0-SMccwmbAQ

Refining the Visualizations. The visual representation of a file is currently overly simple, namely a bar. We plan on extending the visual metaphor in several ways:

- Taking inspirations from the SeeSoft tool [40] and the Microprints presented by Ducasse *et al.* [41], the idea is to represent the single line changes within the files [42]. For example, this would allow to understand where in a file lines are added, deleted, or substituted.
- In their current state, the bars represent an overview of the changes occurred, *i.e.,* when files change in more than one way our visualization represents a summary of the changes. For instance, when files that both increased/decreased in size and modified lines of code, only the change of size is represented. Similarly, when lines of code are both added and deleted, the delta between insertions and deletions is depicted. We will include the possibility to watch at more comprehensive (than summary-based) animated diff.

Commit Digests. One promising direction we are currently exploring is the possibility to animate a sequence of commits and commits related to each other, for example those part of the same pull request. The first scenario, involving the sequence of commits, would allow the answering of questions such as "what happened yesterday/last week?", while the second scenario might provide a more comprehensive means to understand a pull request, which is a cumbersome activity. The same scenario could also be interesting for supporting code review activities, which naturally feature a comprehension part.

Enabling Selective Analysis. The visualization we offer depicts all the file versions in a commit and animates them from their prior state to the one tracked by the commit under observation. Another interesting avenue is the possibility to select a subset of file versions, *e.g.,* focusing exclusively on modified files. Otherwise, given all the file versions in a commit, it may be useful to select all commits that impacted those file versions, to watch and interact with their animated histories. Similarly, developers may benefit from selecting specific packages or components and interacting with the animated histories of the file versions included in the selection.

VII. CONCLUSION

The comprehension of evolving software systems remains a complicated problem, as it deals with understanding complex changes which are often difficult to unravel. Not only is the data heterogeneous and difficult to handle, but it is also imprecisely documented, *e.g.,* with poor quality commit messages [43] or pull request that appear not merged despite being merged [44]. The current state of the art is to look at the changes using diff-based textual representations, which is rather reductive, since source code is more than just text [35].

Our approach takes a fresh perspective on how software changes can be seen and understood. The approach needs certainly more refinement, as we presented in the previous section, but the initial results based on the idea we proposed show considerable promise.

▶ Playlist of referenced videos: https://www.youtube.com/playlist?list=PL9LiNpHT0QqhHPnZUp1pTiy7LGEeUabiA

REFERENCES

[1] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer-An investigation of how developers spend their time," in *Proceedings of ICPC 2015*. IEEE, 2015, pp. 25–35.

[2] M. D. Storey, F. D. Fracchia, and H. A. Müller, "Cognitive design elements to support the construction of a mental model during software exploration," *Journal of Systems and Software*, vol. 44, no. 3, pp. 171–185, 1999.

[3] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proceedings of ICSE 2006*. ACM, 2006, pp. 492–501.

[4] V. Singh, L. L. Pollock, W. Snipes, and N. A. Kraft, "A case study of program comprehension effort and technical debt estimations," in *Proceedings of ICPC 2016*. IEEE, 2016, pp. 1–9.

[5] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of ICSE 2013*. IEEE, 2013, pp. 712–721.

[6] M. D'Ambros, M. Lanza, and R. Robbes, "Commit 2.0," in *Proceedings of Web2SE 2010*. ACM, 2010, pp. 14–19.

[7] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? On the relation between source code and comment changes," in *Proceedings of WCRE 2007*. IEEE, 2007, pp. 70–79.

[8] M. Kajko-Mattsson, "A survey of documentation practice within corrective maintenance," *Empirical Software Engineering*, vol. 10, pp. 31–55, 2005.

[9] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *Proceedings of ICSE 2012*. IEEE, 2012, pp. 255–265.

[10] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *Proceedings of ASE 2017*. IEEE, 2017, pp. 135–146.

[11] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "A neural architecture for generating natural language descriptions from source code changes," in *Proceedings of ACL 2017*. ACL, 2017, pp. 287–292.

[12] Y. Huang, Q. Zheng, X. Chen, Y. Xiong, Z. Liu, and X. Luo, "Mining version control system for automatically generating commit comment," in *Proceedings of ESEM 2017*. IEEE, 2017, pp. 414–423.

[13] C. M. Taylor and M. Munro, "Revision towers," in *Proceedings of VISSOFT 2002*. IEEE, 2002, pp. 43–50.

[14] G. Occhipinti, C. Nagy, R. Minelli, and M. Lanza, "Syn: Ultra-scale software evolution comprehension," in *Proceedings of ICPC 2023*. IEEE, 2023, pp. 69–73.

[15] J. T. Stasko, "Tango: A framework and system for algorithm animation," *ACM SIGCHI Bulletin*, vol. 21, no. 3, pp. 59–60, 1990.

[16] S. Diehl, "Software visualization," in *Proceedings of ICSE 2005*. ACM, 2005, pp. 718–719.

[17] S. Benford, C. Brown, G. Reynard, and C. Greenhalgh, "Shared spaces: Transportation, artificiality, and spatiality," in *Proceedings of CSCW 1996*. ACM, 1996, pp. 77–86.

[18] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[19] D. L. Sonnier and S. L. Hutton, "Enhancing visual aids through the use of animation," in *Proceedings of MSCCC 2004*. MSCCC, 2004, p. 155–164.

[20] L. P. Rieber, "Using computer animated graphics in science instruction with children." *Journal of Educational Psychology*, vol. 82, no. 1, p. 135, 1990.

[21] E.-M. Yang, T. Andre, T. J. Greenbowe, and L. Tibell, "Spatial ability and the impact of visualization/animation on learning electrochemistry," *International Journal of Science Education*, vol. 25, no. 3, pp. 329–349, 2003.

[22] R. L. London and R. A. Duisberg, "Animating programs using Smalltalk," *Computer*, vol. 18, no. 08, pp. 61–71, 1985.

[23] G. Salomon, *Interaction of Media, Cognition, and Learning: An Exploration of How Symbolic Forms Cultivate Mental Skills and Affect Knowledge Acquisition*, 1st ed. Routledge, 1994.

[24] M. Bétrancourt and B. Tversky, "Effect of computer animation on users' performance: A review," *Le travail humain*, vol. 63, no. 4, p. 311, 2000.

[25] T. N. Höffler and D. Leutner, "Instructional animation versus static pictures: A meta-analysis," *Learning and instruction*, vol. 17, no. 6, pp. 722–738, 2007.

[26] F. Paas, A. Renkl, and J. Sweller, "Cognitive load theory and instructional design: Recent developments," *Educational psychologist*, vol. 38, no. 1, pp. 1–4, 2003.

[27] W. Schnotz and C. Kürschner, "A reconsideration of cognitive load theory," *Educational Psychology Review*, vol. 19, pp. 469–508, 2007.

[28] J. Sweller, J. J. Van Merrienboer, and F. G. Paas, "Cognitive architecture and instructional design," *Educational Psychology Review*, vol. 10, pp. 251–296, 1998.

[29] S. Berney and M. Bétrancourt, "Does animation enhance learning? A meta-analysis," *Computers & Education*, vol. 101, pp. 150–167, 2016.

[30] L. P. Hattori and M. Lanza, "On the nature of commits," in *Proceedings of ASE 2008*. IEEE, 2008, pp. 63–71.

[31] A. Alali, H. Kagdi, and J. I. Maletic, "What's a typical commit? A characterization of open source software repositories," in *Proceedings of ICPC 2008*. IEEE, 2008, pp. 182–191.

[32] K. A. Safwan and F. Servant, "Decomposing the rationale of code commits: The software developer's perspective," in *Proceedings of ESEC/FSE 2019*. ACM, 2019, pp. 397–408.

[33] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes? An exploratory study in industry," in *Proceedings of FSE 2012*. ACM, 2012, pp. 1–11.

[34] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens: A study on why and how developers examine it," in *Proceedings of ICSME 2015*. IEEE, 2015, pp. 1–10.

[35] G. Weinberg, *The Psychology of Computer Programming*, Silver Anniversary ed. Dorset House, 1998.

[36] E. Fregnan, L. Braz, M. D'Ambros, G. Çalıklı, and A. Bacchelli, "First come first served: The impact of file position on code review," in *Proceedings of ESEC/FSE 2022*. ACM, 2022, pp. 483–494.

[37] N. Tsantalis, A. Ketkar, and D. Dig, "RefactoringMiner 2.0," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 930–950, 2020.

[38] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python framework for mining software repositories," in *Proceedings of ESEC/FSE 2018*. ACM, 2018, pp. 908–911.

[39] A. Danial, "cloc: v1.92," Dec. 2021.

[40] S. Eick, J. Steffen, and E. Sumner, "Seesoft-A tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 957–968, 1992.

[41] S. Ducasse, M. Lanza, and R. Robbes, "Multi-level method understanding using microprints," in *Proceedings of VISSOFT 2005*. IEEE, 2005, pp. 33–38.

[42] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and M. D. Penta, "LHDiff: A language-independent hybrid approach for tracking source code lines," in *Proceedings of ICSM 2013*. IEEE, 2013, pp. 230–239.

[43] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of ICSE 2022*. ACM, 2022, pp. 2389–2401.

[44] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of MSR 2014*. ACM, 2014, p. 92–101.