Can Git Repository Visualization Support Educators in Assessing Group Projects?

Mircea Lungu[1], Rolf-Helge Pfeiffer[1], Marco D'Ambros[2], Michele Lanza[2] Jesper Findahl[2]
1: University of Copenhagen, Denmark
2: Software Institute - Università della Svizzera italiana, Switzerland

***Abstract*—In the past years numerous software visualization tools have been introduced to support the analysis of software systems and their evolution as captured in the versioning systems. Usually the target audience of such tools comprises software engineering professionals. In this paper we argue that such tools are also beneficial for educators who need to evaluate the quality of software systems developed by students. However, since the needs of educators are different than those of the software engineering professionals, we discuss several educator needs first. We report several usage examples that we believe are useful for educators when using repository visualization tools. We illustrate them with examples from several student projects from different courses in two universities. We conclude with a series of considerations that should be heeded by both educators and future tool-builders.**

I. Introduction

Visualization approaches have been proposed and adopted for a long time to support software engineers in understanding the structure and evolution of software systems [1]–[5]. With the ultimate goal of easing the complexity of software system development, researchers have also investigated how the usage of visualization approaches can improve software maintenance tasks, by performing experiments [6]–[8] and controlled studies [9], [10]. However, with the exception of algorithm animation [11], [12], very little work has been done about the usage of such tools in education.

We argue that a special class of educators that can benefit from visualization tools are those who must evaluate projects in software engineering, human-computer interaction, and other group-based projects. Nowadays, many such group projects use git and GitHub for supporting collaborative programming work [13], [14]. Such projects can be of significant size and assessing the individual contributions of the group members can be challenging [14]. Indeed, educators evaluating large projects in a short time, have a very different context than other stakeholders supported by visualization tools:

- Reverse engineers aim specifically at making sense of the code and have usually plenty of time for understanding a subject system.
- Developers that are "onboarded" have ample time and also have access to the expertise of senior colleagues.
- Solo-developers who visualize their own software to identify improvement or refactoring candidates have deep expertise of their own code.
- Technology or domain experts have resources to learn and adopt specialized visualization tools over time.

II. Are educators a special kind of user for repository visualization tools?

We believe the following requirements and constraints are special to educators who have to assess group projects that include a programming component:

1) The need for assessing **projects of considerable size**. Often, the result of a software engineering, human-computer interaction, or thesis project is a multi-person multi-month software system that can easily reach tens of thousands of lines. This is much larger than for projects in algorithms and other programming courses, where educators usually can read a complete solution.
2) The need for **expedite assessment** since often the source code of the project is only one of multiple deliverables and an educator's time is limited. Indeed, often requirements, problem statement, user evaluation, experimental design, etc. are just as essential for educators to assess and evaluate.
3) The need to assess also **the individual contributions** and not only the final result. Often, when several students submit a software repository for a group project, their contributions are not the same. Some students might focus on particular parts of the project, and some may not contribute any code at all to the code base. Being able to assess individual contributions can help educators to steer conversations with student groups.
4) The need to evaluate **many technologies and many languages**. Often, educators teach more than one course at a time and supervise multiple thesis projects simultaneously. Each of these courses is likely to be using different languages, frameworks, and technologies. Consequently, to help assess all these heterogeneous projects, educators would benefit from a tool that would be as technology-independent as possible.
5) The need for **privacy for the analyzed code**. Educators must often assess private or institutional repositories, which are usually not shared publicly via software forges like GitHub, GitLab, Bitbucket, etc. They also need to visualize projects for which the source code might be protected by an NDA.

Assumptions. To summarize and clarify our assumptions about educators, students, and student projects that delineate the scope of our work, we list the following assumptions that frame the context for this paper:

Educators need to evaluate the code of student projects as part of the final assessment but they have limited time for the task, e.g., half an hour per group. If complete evaluation is impossible, they want to have a starting point for discussing with students at the final assessment.

Projects being assessed are multi-month multi-person projects of sufficient complexity that make them impossible to exhaustively assess within constrained time, i.e. no time to read complete sources.

Students work in groups and use file-based version control systems that track changes and their authors (e.g., git).

III. Using Git-Truck for Repository Visualization

To investigate the use cases presented in this paper, we used a visualization tool named `git-truck` [15]. which visualizes the structure of a `git` repository using hierarchical metric-enhanced layouts, such as, *circle packing visualizations* [16] or *tree maps* [17]. The tool presents containment structures of directories and files in such a way that the visual size of files is proportional to their size in bytes[1]

On top of the repository structure `git-truck` uses color maps to highlight evolution derived metrics that can be either continuous (e.g., color intensity proportional to the number of commits to a given file), or discrete (e.g., highlighting only those files that have a single author). Several such visualizations are presented in the next Section and explained in-situ.

All visualizations are highly interactive with support for filtering, zooming, and presentation of details on demand.

For inspection, `git-truck` supports interactive author-unification, i.e., multiple authors can be grouped into single *logical* authors and co-author attribution, i.e. commit co-authors that are identified via the `Co-authored-by` tag in commit messages are extracted. Such a feature is especially relevant in pair programming.

Against the increasingly popular trend for online software-as-a-service, `git-truck` is meant to be executed directly on personal computers from a local clone of a git repository. The installation instructions and source code for the tool are available online at: https://github.com/git-truck/git-truck. For the figures generated in this paper we used version 1.0.3 of the tool.

IV. Usage Examples

The following usage examples of a software repository visualization tool are generated by educators from IT University of Copenhagen, Denmark (ITU) and Università della Svizzera italiana, Switzerland (USI). They are based on observations derived from four different courses at these two institutions and we illustrate each with at least one example.

A. Discovering Single-authored Components of Systems

The *Single Author View* in `git-truck` highlights in red single-authored files with the aim of supporting fast assessments of the degree of actual group work or the lack thereof. It allows educators to quickly detect components or projects without collaboration amongst students.

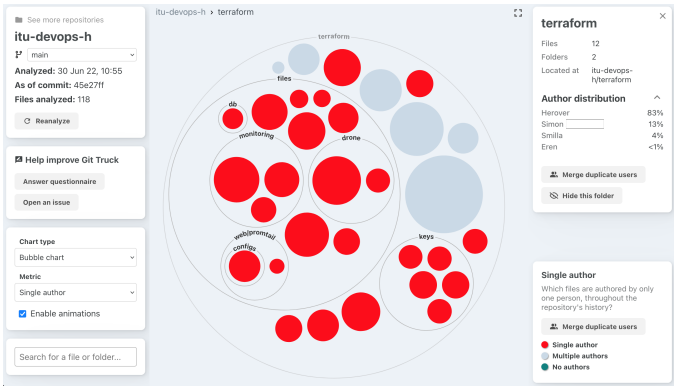

Figure 1. Component of a system which is mostly created by a single author.

Figure 1 shows the infrastructure-as-code specifications (with `terraform`) submitted as a component of a group project in the *DevOps, Maintenance, and Software Evolution* [2] course at ITU. This is a zoom-in on the *terraform* folder from a bigger project. The visualization suggests that although the group has five members almost all the work on this component has a single author. During the oral exam we give the other group members the chance to discuss infrastructure-as-code concepts but only one can do in any meaningful way.

B. Investigating Responsibility Distribution in a Project

The *Top Contributor View* colors each file in a repository according to the author who changed (both added and removed) most lines of code in the file throughout the history. The goal is to support gauging how work is distributed between members in the project or in specific areas of it.

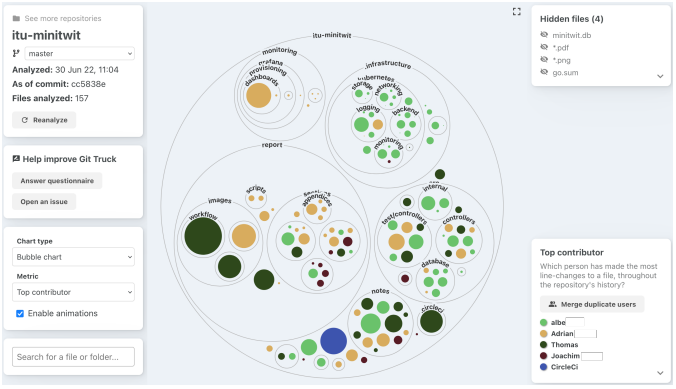

Figure 2. A system with a good balance of contributor responsibilities

Figure 2 shows the Top Contributors view in another repository from the *DevOps, Software Evolution and Software Maintenance* course at ITU. Visual inspection suggests that all the members (*albe*–green, *Adrian*–ochre, *Thomas*–dark-green,

[1]Size in bytes is used as a metric since it is uniformly available for plain text as well as binary files, which are both present in any large projects [18].

[2]https://github.com/itu-devops/lecture_notes

Joachim–scarlet) worked on all parts of a backend system, though the scarlet author is top contributor for a lower amount of files[3] Compare this with Figure 3 which presents a project that creates a data analysis platform in the course *Visual Analytics Atelier*[4] at USI. Here a single author (*Student 1*) is top contributor to most of the code files in the project. Since the project goal is that students practically experience composition of a larger application from independently taught components, the pattern shown in Figure 3 represents a starting point for a discussion with students about individual contributions.

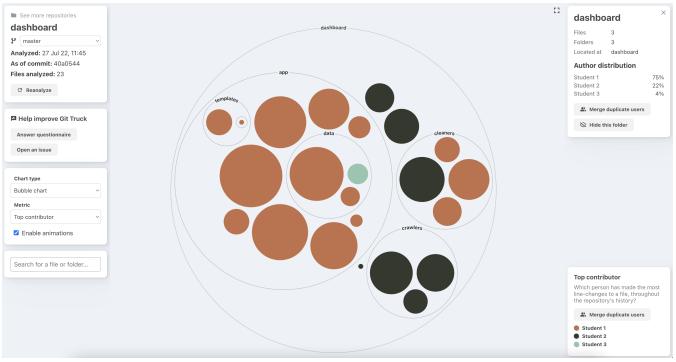

Figure 3. A system with one author dominating the contribution

Note. In one case, `git-truck` visualized a single author as *Top Contributor* with all the other project group members missing. Discussions of that visualizations with the students revealed that the "top contributor" is a novice `git` user who accidentally deleted the entire repository and subsequently added all contents again. This is a reminder that, educators should not blindly act based on the visualizations but rather discuss them with students.

`git-truck` provides a rudimentary feature for selecting the last commit up to which the analysis can be done. With the help of this feature, an educator can spot projects where collaboration first becomes an issue after a certain commit.

C. Gaining High-Level Architectural Insights

The *File Extension View* colors files according to their extensions besides visualizing them proportional to their size and folder containment hierarchy. That view is particularly useful to gain insights into the structure of multi-lingual systems. For example, in the *Technical Interaction Design* course at ITU – an introductory web-development course in which multiple groups implemented the same front-end with `React` – we observed two extremes of code organization: (1) some groups decided to rely on one big CSS file and many JavaScript files, see Figure 4, while others decided to (2) distribute CSS files across the system, see Figure 5 (yellow files are JavaScript and violet files are CSS).

[3]However, after discussing with students and after inspecting a second front-end repository of the same group (not illustrated), the perceived and assessed contributions are more or less equal

[4]https://search.usi.ch/en/courses/35263637/sde-atelier-visual-analytics

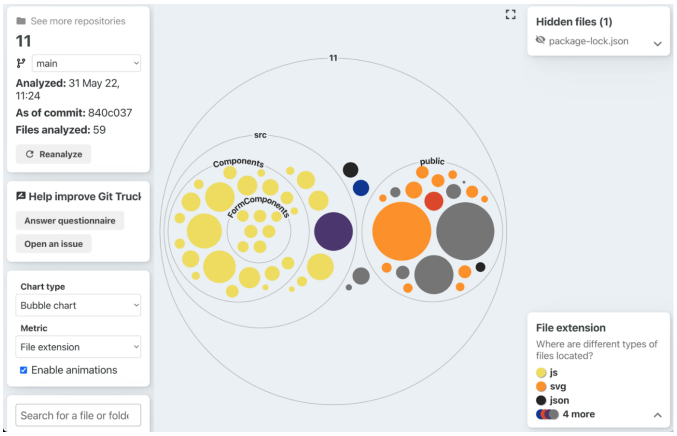

Figure 4. A system in which students create a single CSS file (violet)

After observing these architectural extremes, we realized the importance of including discussions of file organization in the future iterations of the course.

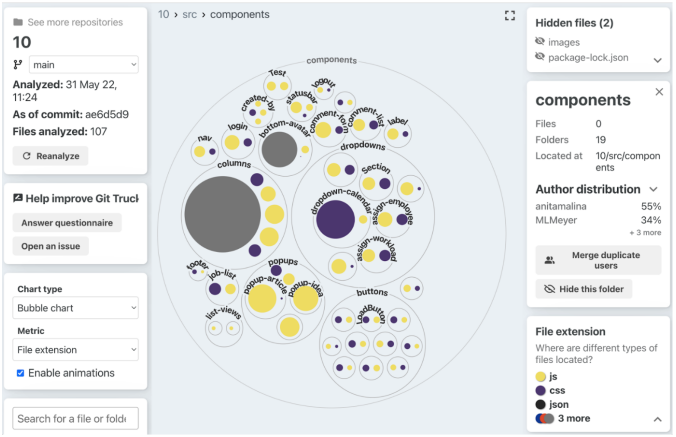

Figure 5. A system in which stylesheet files (violet) are near their components

D. Uncovering Critical Components of a System

`git-truck` provides a *Number of Commits* view, which highlights a repository's files that are most changed. Files are colored with a gradient proportional to the number of commits that modify them. Under the assumption that frequently changed files are most relevant [19], educators can direct their assessment to most relevant components.

Figure 6 shows the *Number of Commits* view for `Maths Camp` (https://github.com/MathsCamp/MathsCamp), a system for adaptive maths practice developed by two master students at ITU. The view highlights that one single file is disproportionately changed during the project evolution. This Javascript file is also the largest file in the system. Closer assessment of this file reveals a *God Class* which is responsible for everything from user interface interactions over database querying to scheduling.

Before `git-truck` was available, one of the authors of this paper browsed the repository to assess the project but failed to

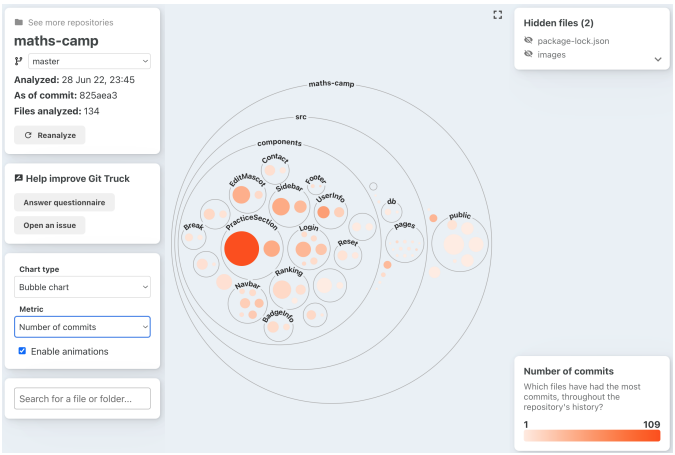

Figure 6. A system where most change effort goes into a single file

observe the bad design since the respective file is hidden quite deeply in the directory tree and has an inconspicuous name.

Note. Files that can be relevant for group project work and that change often, might not contain source code: e.g., `README`, `CHANGELOG`, `package-lock.json`, etc. A tool should support easily filtering out such files in order to let the critical parts of the system stand out.

V. Discussion

A. Limitations Of Metrics-Based Repository Visualization

Metrics based views like the ones presented in the previous sections can tempt an educator to jump too easily to conclusions. However, educators should not rely solely on the visualizations without confirming hypotheses with students or triangulating with other sources. Two examples of situations that can throw off contribution metrics are: (1) automatically generated code and (2) files being committed that should not be versioned. Besides this, as the example in subsection IV-B in which a student removes and adds all contents of a repository, there will always be unexpected usage patterns. A repository visualization tool can highlight these unusual patterns but it is the duty of the educator to investigate them.

B. Educator-specific Tool Support

We believe that educator-specific functionality increases usefulness of a software repository visualization tool for assessment of group projects. For tool-builders that aim to support educators, we list a series of functionalities that we believe are essential to increase the adoption of such tools.

1. Interactivity. In our experience, interactivity, e.g. filtering of which kinds of files are to be visualized or zooming in on a particular folder, is critical for assessment. Before `git-truck`, we relied on `git` command line tools like `git log`, `git shortlog`, `git blame` and various scripts that wrap these tools to assess student repositories. Even though more versatile, their generality means that some use cases (e.g. focusing the analysis on a given subfolder) are not easy to achieve.

2. Author unification. `git-truck`'s author unification feature is essential when assessing student repositories. Often students commit using multiple user names when their `git` configuration differs across computers, e.g., lab computers, private laptops, or home computers. Author-unification could rely completely on `git` *mailmap* files (https://git-scm.com/docs/gitmailmap), but students configure them rarely.

3. Configurable thresholds. The *Single Author View* has a hard-coded threshold of 100% authorship embedded in its definition. Arguably, a threshold of 98% could still qualify as a very useful one, but the tool we use does not support the changing of the threshold. Increased configurability would empower educators but has to be balanced against ease of use.

4. Automatic filtering of files. Currently, `git-truck` supports manual filtering of the kinds of files that are visualized. However, manual filtering is laborious and repetitive especially when assessing multiple similar projects. Future research could aim to detect – based on language independent heuristics – files of low relevance, such as, automatically generated code.

5. Integrating commit messages. Even though contributions – in terms of number of commits or size of applied changes – may be equally distributed amongst student group members, the kind of contributed work may vary. Currently, we inspect commit messages with `git shortlog -n` in combination with `git-truck`'s visualizations. Ideally, commit messages would be integrated and browse-able directly inside of the tool.

6. Supporting multi-repositories. Depending on projects and depending on course design, student groups use mono- or multi-repositories to organize their work. A tool like `git-truck` (and most of the tools we are aware of, including the GitHub statistics) only work with mono-repositories. An educator has to visualize a multi-repository as a series of multiple individual mono-repositories comparison among which is difficult.

C. Generalizability

`git-truck` is designed with usability in mind. However, none of the visualizations it implements are unique or difficult to implement (the interaction requires more effort though). Thus, we argue the usage examples presented in section IV are likely to be useful for other educators using similar tools.

VI. Related Work

Both Wattenberger [20] and Tornhill [19] use circle packing to highlight metrics on top of repository structure. However, their work is not targeted at educators, but rather developers and business responsibles. Furthermore, although both provide online systems that could possibly visualize *non-private* GitHub repositories, none of the two services has the interactive nature that we argue for in this paper. Raclet and Silvestre proposed Git4School, an analytics dashboard [21] that enables a lecturer to follow the work of the students, commit by commit, to identify students experiencing difficulties. Their work is intended for a kind of educator and student in a context where the educator needs to guide individual student closely.

Kim et al. presented an interactive `git` repository visualization tool called *Githru* [22]. They aim to support developers

and domain experts in understanding a projects development history focusing on properties of the `git` commit graph. That is different to the goal presented in this paper.

Specialized tools focus on certain aspects of `git` repositories only. For example, Cosentino et al.'s *Gitana* [23] infers those authors that are most crucial for a software project (it computes *truck factors*), *Gource* (https://gource.io/) animates authors and their contributions over time, the *Git Timeline Generator* (https://www.preceden.com/git) visualizes contribution frequencies over time, `git-of-theseus` creates static visualizations of repository growth over time, or GitHub's built-in repository visualizations present activity statistics, like commit frequencies, number of contributors, etc.

Other tools target other interaction mechanisms. For example Ciani et al.'s *UrbanIt* [24], relies on a city metaphor (a tree map with an added third dimension for file size) to visualize logical structure of software repositories on mobile device with touch-screen, or Scott-Hill et al.'s *DashVis* [25] aims to support teams in tracking progress using large touch displays and visualization techniques.

VII. Conclusions and Future Work

We argued that educators are in some cases a unique kind of user for software visualization tools. We have shown with multiple case studies from multiple courses and two universities that git repository visualization can be an aid for the educator in assessing group projects.

However, the case studies we presented are based on the experience of a handful of educators who used a specific tool in group project assessment. In the future we plan to extend the study to more educators to arrive at stronger conclusions regarding educator needs and guidelines for both educators and tool-builders. Moreover we plan to investigate in which way such tools can be used during the course of the semester and not only as support for the final evaluation.

Acknowledgements. D'Ambros gratefully acknowledges the financial support of the Swiss National Science Foundation through the NRP-77 project 187353.

References

[1] M.-A. Storey, K. Wong, F. D. Fracchia, and H. A. Muller, "On integrating visualization techniques for effective software exploration," in *Proceedings of VIZ'97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*. IEEE, 1997, pp. 38–45.

[2] J. I. Maletic, A. Marcus, and M. L. Collard, "A task oriented view of software visualization," in *Proceedings first international workshop on visualizing software for understanding and analysis*. IEEE, 2002, pp. 32–40.

[3] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, and W. Chen, "A systematic literature review of modern software visualization," *Journal of Visualization*, vol. 23, no. 4, pp. 539–558, 2020.

[4] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonça, and N. Zazworka, "Software evolution visualization: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 11, pp. 1860–1883, 2013.

[5] H. B. Salameh, A. Ahmad, and A. Aljammal, "Software evolution visualization techniques and methods-a systematic review," in *2016 7th International Conference on Computer Science and Information Technology (CSIT)*. IEEE, 2016, pp. 1–6.

[6] M. Sensalire, P. Ogao, and A. Telea, "Evaluation of software visualization tools: Lessons learned," in *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2009, pp. 19–26.

[7] P. Khaloo, M. Maghoumi, E. Taranta, D. Bettner, and J. Laviola, "Code park: A new 3d code visualization tool," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, 2017, pp. 43–53.

[8] T. Schneider, Y. Tymchuk, R. Salgado, and A. Bergel, "Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, 2016, pp. 116–125.

[9] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: a controlled experiment," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 551–560.

[10] B. Sharif and J. I. Maletic, "The effect of layout on the comprehension of uml class diagrams: A controlled experiment," in *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2009, pp. 11–18.

[11] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[12] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, pp. 1–64, 2013.

[13] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang, "The emergence of github as a collaborative platform for education," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1906–1917. [Online]. Available: https://doi.org/10.1145/2675133.2675284

[14] M. Tushev, G. Williams, and A. Mahmoud, "Using github in large software engineering classes. an exploratory case study," *Computer Science Education*, vol. 30, pp. 155 – 186, 2020.

[15] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino, and M. Lungu, "Git truck: Hierarchical metric-enriched file-focused git project visualizations for truck factor analysis," in *2022 IEEE Working Conference on Software Visualization (VISSOFT)*, 2022, pp. 1–11.

[16] W. Wang, H. Wang, G. Dai, and H. Wang, "Visualization of large hierarchical data by circle packing," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 517–520.

[17] B. Johnson, "Treeviz: treemap visualization of hierarchically structured information," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1992, pp. 369–370.

[18] R.-H. Pfeiffer, "What constitutes software? an empirical, descriptive study of artifacts," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 481–491.

[19] A. Tornhill, "Your code as a crime scene: use forensic techniques to arrest defects, bottlenecks, and bad design in your programs," *Your Code as a Crime Scene*, pp. 1–218, 2015.

[20] A. Wattenberger. (2021) Visualizing a codebase. [Online]. Available: https://githubnext.com/projects/repo-visualization

[21] J.-B. Raclet and F. Silvestre, "Git4school: A dashboard for supporting teacher interventions in software engineering courses," in *European Conference on Technology Enhanced Learning*. Springer, 2020, pp. 392–397.

[22] Y. Kim, J. Kim, H. Jeon, Y.-H. Kim, H. Song, B. Kim, and J. Seo, "Githru: visual analytics for understanding software development history through git metadata analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 656–666, 2020.

[23] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 499–503.

[24] A. Ciani, R. Minelli, A. Mocci, and M. Lanza, "Urbanit: Visualizing repositories everywhere," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 324–326.

[25] B. Scott-Hill, C. Anslow, J. Ferreira, M. Kropp, M. Mateescu, and A. Meier, "Visualizing progress tracking for software teams on large collaborative touch displays," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–5.