Visualizing GitHub Issues

Aron Fiechter, Roberto Minelli, Csaba Nagy, Michele Lanza

REVEAL @ Software Institute — USI, Lugano, Switzerland

***Abstract*—The rise of distributed version control systems, such as git, and platforms built on top of it, such as GitHub, has triggered a change in how software is developed. Most notably, state-of-the-art practice foresees the use of pull requests and issues, enriched by means to enable discussions among the involved people. Platforms like GitHub and GitLab have thus turned into comprehensive and cohesive modern software development environments, also offering additional mechanisms, such as code review tools and a transversal support for continuous integration and deployment. However, the plethora of concepts, mechanisms, and their interconnections are stored and presented in textual form, which makes the understanding of the underlying evolutionary processes difficult.**

We introduce the notion of an *issue tale*, a visual narrative of the events and actors revolving around any GitHub issue, and present an approach, implemented as an interactive visual analytics tool, to depict and analyze the relevant information pertaining to issue tales. We illustrate our approach and its implementation on several open-source software systems.

***Index Terms*—software evolution visualization, visual analytics, github issues**

I. Introduction

Modern software development has been shaped by the use of distributed version control systems like Git, and enriched by collaboration platforms like GitHub. These platforms provide a cohesive and comprehensive experience for developing and evolving software through the use of specific mechanisms, such as issues and pull requests (PRs).

Issues on GitHub can be used to report bugs, or also to request new features. Many teams also use GitHub Issues to guide their software development by associating issues with PRs where bugs are fixed or new features are implemented. Issues have become the central element that drives development, and more and more open-source projects are pointing potential new contributors to good first issues to facilitate onboarding, as encouraged by GitHub itself [1].

Both issues and PRs support open discussions via comments and can be categorised and tagged using customizable labels. Issues also show whenever they are mentioned by other issues or by commit messages. It is apparent that all these concepts and mechanisms are deeply interconnected. However, GitHub shows all this information in textual form and shows the interconnections through hyperlinks, which hinders a comprehensive understanding of the evolution of an issue.

We introduce the notion of an *issue tale*, a visual narrative centered around a GitHub issue that encompasses all events and actors related to it. We present our implementation of issue tales as an interactive visual analytics tool and illustrate it on open-source software systems.

II. Visualizing GitHub Issues

A. Modeling a Software Project

We model a software project as a graph with nodes and edges. Fig. 1 shows a simplified diagram of the model.

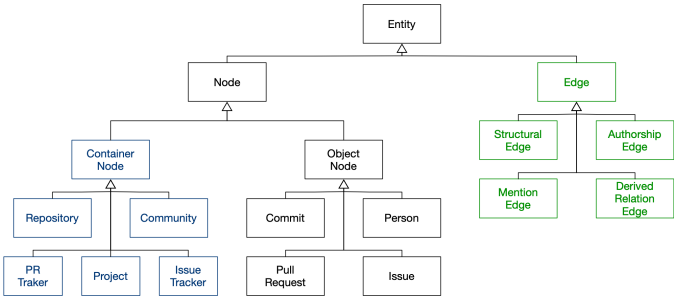

Fig. 1. The Model of a Software Project

The nodes are divided into two main categories: Container nodes and Object nodes. The root node, Project, contains a Repository, an Issue Tracker, a Pull Request Tracker, and a Community. All of these are container nodes. For a Repository node, we retrieve commits and compute diffs in the implementation using the git command-line tool. We retrieve the issues and pull requests using the GitHub API.

The Object nodes are the following: Commit, Issue, Pull Request, and Person. These entities also include secondary nodes, such as comments, labels, and events (*e.g., labeled* or *closed*).

We consider four edge types to describe relationships: Structural, Authorship, Mention, and Derived. Structural edges are static, and exist between a node and another that contains it, *e.g.,* a commit and its repository. Authorship edges represent the relation between a node and the person who created it, *e.g.,* a comment and its author. Mention edges exist between two nodes, where one mentions the other, *e.g.,* a commit message that mentions an issue, or a comment that mentions a person. Derived relation edges are generated from other edges in the last step of the linking process.

B. The Linking Process

The linking process happens once all nodes of a project have been retrieved, and consists of three phases: (1) creation of structural and authorship edges; (2) creation of mention edges, in which we search textual mentions between object nodes (*e.g.,* an issue key mentioned in a commit message); and (3) creation of derived relation edges, which are built on top of existing edges and support the creation of issue tales.

Once the linking is finished, the graph representing the project is completed. In Fig. 2, we show an example structure of a project. Container nodes are shown in blue, object nodes in black. Edge classes are in green except for structural edges that can be seen as entities containing subentities.

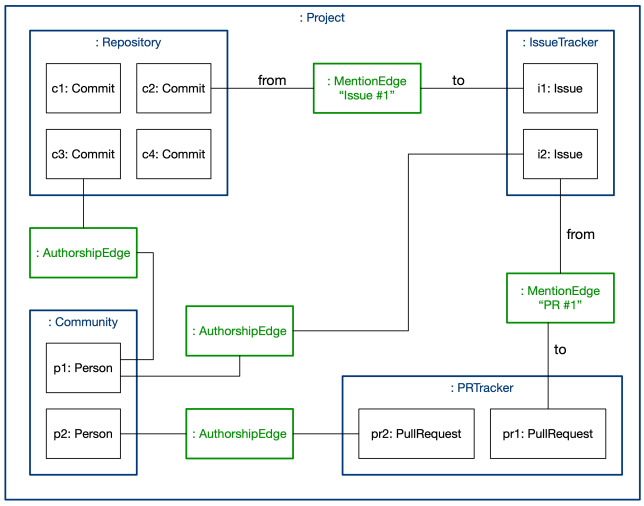

Fig. 2. Object Diagram of a Simple Project

C. Creating Issue Tales

We define an *"issue tale"* as the collection of all nodes related to an issue. Starting from the issue for which we want to build the tale (*i.e.,* the subject issue), we collect all related issues, commits, pull requests, and people by traversing derived relation edges. We also include in the issue tale all comments and events of the subject issue, and of course, the subject issue itself.

From this creation process, it is already apparent that issue tales can heavily overlap, especially for closely related issues.

III. Coarse-Grained Visualization

Fig. 3 shows a coarse-grained view of all issue tales in JetUML.[1] The view shows each issue tale as a striped rectangle, where stripes represent the related model entities, which are issues, pull requests, commits, people, and comments. The width represents the duration and a black border means the subject issue of the tale is still open.

The color-coding helps to get a quick overview of the issue tales in a project. In the case of JetUML, it is interesting to see that most issue tales contain a large amount of commits (in blue), and that only a few issues are still open.

There are other projects, like Braintree Android SDK[2] (depicted in Fig. 4), that have a prevalence of comments in issue tales (in orange), and others, like Zerocode[3] (depicted in Fig. 5), that are more varied. The number of open issues also varies significantly across projects.

To better understand specific issue tales, we implemented a fine-grained visualization, detailed in the next section.

[1]See `https://github.com/prmr/JetUML`
[2]See `https://github.com/braintree/braintree_android`
[3]See `https://github.com/authorjapps/zerocode`

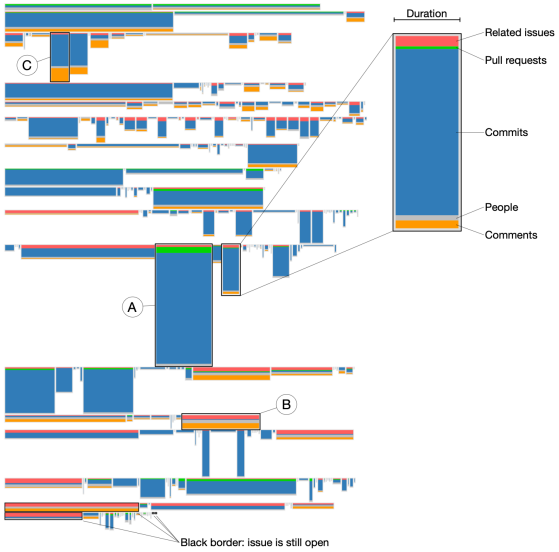

Fig. 3. Coarse-Grained View of the Issue Tales in JetUML

Fig. 4. Coarse-Grained View of the Issue Tales in Braintree

Fig. 5. Coarse-Grained View of the Issue Tales in Zerocode

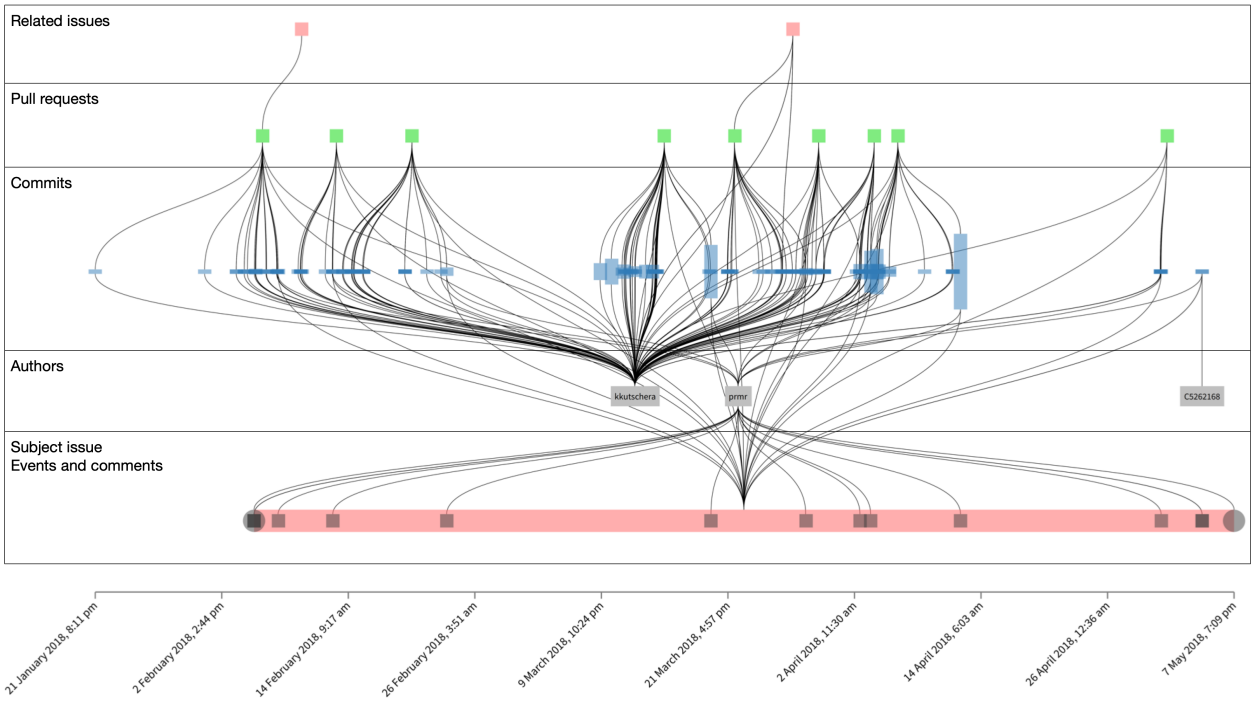

Fig. 6. Tale of Issue JetUML #231: Port UI to JavaFX

IV. Fine-grained timeline

Given the time-based nature of an issue tale, we chose to adopt a timeline for the fine-grained visualization, with time flowing from left to right. We maintain a structure similar to the coarse-grained striped rectangle, with different nodes being on separate rows. Fig. 7 shows a template for the fine-grained visualization of a single issue tale. The bottom row shows the central issue as a large rectangle with all its events and comments as small squares.

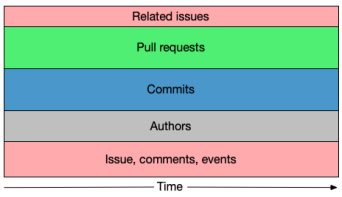

Fig. 7. Structure of an Issue Tale Timeline

All nodes except for authors have a timestamp, and are thus shown at the corresponding horizontal position. The nodes have various metrics mapped to visual properties such as position and width. Table I shows which metrics are mapped to which properties.

In this view, we also show the relationship between the various nodes as edges between them. Since the authors are not time-based, they are positioned horizontally by averaging the positions of their connected nodes.

At the bottom, we show a time scale. The top row, where we display related issues, acts as a navigation bar, allowing the user to open the timeline views of the tales of other issues.

TABLE I
Metrics mapped to attributes in the fine-grained view.

Node Type	Metric	Attribute
All except Person	Timestamp	Horizontal Position
Main issue	Duration	Width
Commit	Number of changed files	Height

A. Two Tales from JetUML

In Fig. 3, we annotated three issue tales with the letters A, B, and C, which correspond to issues #231, #300, and #12.

1) Tale A: A Major Architectural Change: We start by showing issue tale A in Fig. 6. We repeat the layering explained in Fig. 7 for convenience.

The timeline view featured a related issue that was placed much earlier than all other nodes in the tale, but since the view is interactive, we removed it to allow all other nodes to be shown clearly on the timeline.

Tale A is very rich: there are over 150 commits, 9 pull requests, and the issue spans three months. There are only three people involved with the issue: `kkutschera`, `prmr`, and `C5262168`. The first two people are the most important in the issue tale: `kkutschera` authored the vast majority of the commits, while `prmr` seems to be mostly concerned with orchestrating the status of the issue, adding milestones, labels, and closing the issue. `C5262168` has only one commit. Further inspection reveals a duplicate of a commit by `prmr`.

This specific issue tale is about development, porting part of the codebase to a new UI platform.

2) Tale B: A Nasty Bug: In the fine-grained timeline view of tale B, shown in Fig. 8, we see no pull requests and only one single commit, which is related to only one of the issues.

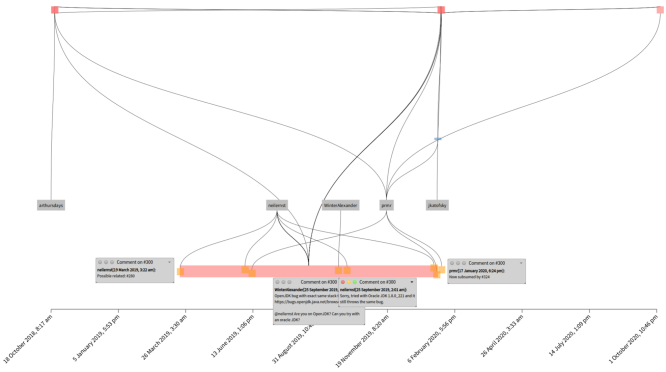

Fig. 8. Tale of Issue JetUML #300: Opening second diagram crashes on Mac

We interacted with the view to remove all events and show some of the comments posted on the issue. After opening the issue, `neilernst` immediately suggested that it could be related to issue #280. After that, the tale only contains a few other interactions in more than a year, such as suggestions for potential fixes. On January 17, 2020, `prmr` closed the issue with the comment "*Now subsumed by #324*". Notice that #324 is one of the related issues shown on the top of the view. By inspecting its timeline, we see a similar situation. Now it is issue #385 that subsumed #324. The new issue, #385, is still open and does not feature commits other than a minor fix. To conclude, this issue tale is telling a story of a persistent bug.

B. Tangled Issue Tales

In Fig. 9, we show tale C of issue #12, as well as the one of issue #13 in Fig. 10. We show both because the issues are related and share the same commits in their tales.

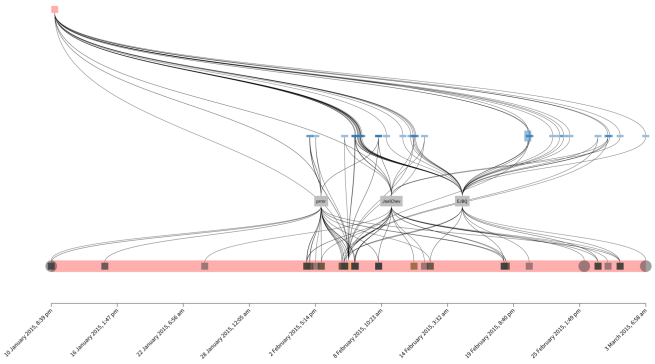

Fig. 9. Fine-grained tale timeline of issue #12 in JetUML

The two issues relate to two features that need to be added: copy-and-paste and undo. The fine-grained views of the tales and the near-simultaneousness of the two issues tell us that they are tightly related.

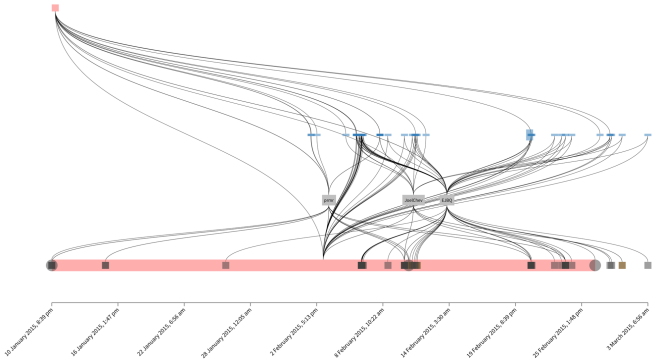

Fig. 10. Fine-grained tale timeline of issue #13 in JetUML

C. Sample Tales from Other Projects

1) A Quick Contribution: In Fig. 11, we present a very short tale timeline for Issue #285 in the project Zerocode. We opened two comments and a word cloud of the file changes of the three commits in the tale. The font size of the file names depends on the total number of changed lines, while the color shows whether there is a majority of deletions or additions (in red or blue).

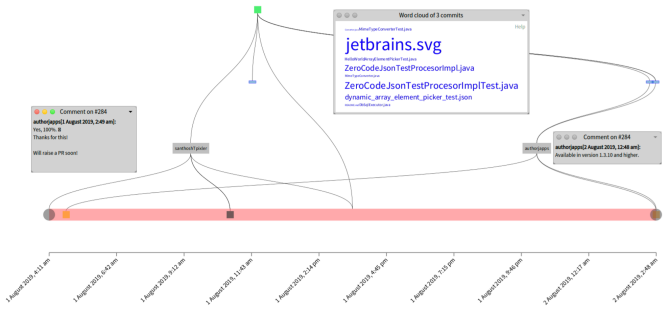

Fig. 11. Tale of Issue Zerocode #284: Mention Jetbrains support in Readme

We see that the biggest file change is `jetbrains.svg`, which is the JetBrains logo. There are also other seemingly unrelated changes to code files. After inspection, we understand that these are part of an integration merge of the main branch before merging the PR. There are also changes to the README, as the issue title suggests.

2) Adopting a Fix Takes Time: Fig. 12 shows the tale of an issue in the project Braintree Android SDK. We made hidden on the view all event nodes, opened two comments, and created a word cloud of the last seven comments on the issue, which all happened after the issue was closed.

The word cloud suggests that people commenting after the issue was closed were having problems with the Google Play Store related to removal and build rejection.

This demonstrates how an issue tale is not finished when the issue is closed, but continues until people stop interacting with the issue. An issue tale can also start a long time before the issue is even created, for example, in old related issues or pull requests.

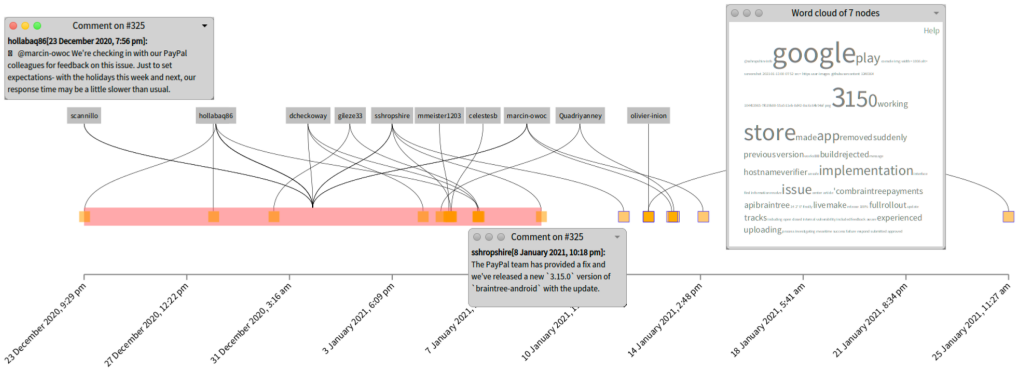

Fig. 12. Tale of Issue Braintree Android SDK #325: Unsafe implementation of the HostnameVerifier interface - Google policy violation

V. Related Work

Software Evolution Visualization. Much research has been done to visualize software evolution, from single systems, as in CodeCity [2], to entire ecosystems in Complicity [3]. Burch *et al.* explored ways to visualize work processes in a software system through the concept of Developer rivers [4].

Visualization of Issues. D'Ambros *et al.* visualized an entire bug database, as well as single issues, focusing on the properties of the bug report, such as the status, description, and involved people [5]. Knab *et al.* presented an approach to visualize problem reports that helps in uncovering hidden patterns and supports analysis by allowing the combination of various visualizations flexibly [6]. Hora *et al.* created BugMaps, a tool that visualizes issues together with links to other artifacts of the software system such as source code entities [7]. Dal Sasso and Lanza created a web-based visual analytics tool to explore bug reports, with a coarse-grained view of all bug reports and a fine-grained view that shows various properties of a single bug report [8]. More recently, Liao *et al.* devise various aggregated views of GitHub issues [9].

Techniques. We used an approach similar to the one explained by Bird *et al.* to disambiguate users [10]. Our implementation creates different views using the concept of intensional views [11]. Every view is built using a complete graph and a view specification, which defines its representation and the metrics mapped on visual attributes for each node or edge kind. The fine-grained timeline visualization of an issue tale is a time-based view of a complex graph [12] [13].

VI. Conclusions

We introduced the concept of issue tales and presented our approach and implementation as an interactive visual analytics tool. We have shown two visualizations of issue tales: a coarse-grained view that encodes the size and duration of tales and offers a global view of all issue tales in a project, and a fine-grained timeline view which explains the internal structure of a single issue tale while enabling a deeper understanding of the entities that compose it and their relations.

Acknowledgements. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "INSTINCT" (SNF Project No. 190113).

References

[1] X. Tan, M. Zhou, and Z. Sun, "A First Look at Good First Issues on GitHub," in *Proceedings of ESEC/FSE 2020 (28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM Press, 2020, p. 398–409.

[2] R. Wettel and M. Lanza, "CodeCity: 3D Visualization of Large-Scale Software," in *Proceedings of ICSE 2008 (Companion of the 30th International Conference on Software Engineering)*. ACM Press, 2008, p. 921–922.

[3] S. Neu, M. Lanza, L. Hattori, and M. D'Ambros, "Telling stories about GNOME with Complicity," in *Proceedings of VISSOFT 2011 (6th International Workshop on Visualizing Software For Understanding and Analysis)*. IEEE CS Press, 2011, pp. 14–21.

[4] M. Burch, T. Munz, F. Beck, and D. Weiskopf, "Visualizing Work Processes in Software Engineering with Developer Rivers," in *Proceedings of VISSOFT 2015 (3rd Working Conference on Software Visualization)*. IEEE CS Press, 2015, pp. 116–124.

[5] M. D'Ambros, M. Lanza, and M. Pinzger, ""A Bug's Life" Visualizing a Bug Database," in *Proceedings of VISSOFT 2007 (4th International Workshop on Visualizing Software for Understanding and Analysis)*. IEEE CS Press, 2007, pp. 113–120.

[6] P. Knab, B. Fluri, H. C. Gall, and M. Pinzger, "Interactive Views for Analyzing Problem Reports," in *Proceedings of ICSM 2009 (IEEE International Conference on Software Maintenance)*. IEEE CS Press, 2009, pp. 527–530.

[7] A. Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M. T. Valente, and J. Martins, "Bug Maps: A Tool for the Visual Exploration and Analysis of Bugs," in *Proceedings of CSMR 2021 (16th European Conference on Software Maintenance and Reengineering)*. IEEE CS Press, 2012, pp. 523–526.

[8] T. Dal Sasso and M. Lanza, "A Closer Look at Bugs," in *Proceedings of VISSOFT 2013 (First IEEE Working Conference on Software Visualization)*. IEEE CS Press, 2013, pp. 1–4.

[9] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[10] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining Email Social Networks," in *Proceedings of MSR 2006 (International Workshop on Mining Software Repositories)*. ACM Press, 2006, p. 137–143.

[11] K. Mens, A. Kellens, F. Pluquet, and W. Roel, "The Intensional View Environment," in *Proceedings of ICSM 2005 (21st IEEE International Conference on Software Maintenance)*. IEEE CS Press, 2005, pp. 81–84.

[12] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*. Springer London, 2011.

[13] T. Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. Wijk, J.-D. Fekete, and D. Fellner, "Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges," *Computer Graphics Forum*, vol. 30, pp. 1719–1749, 2011.