RepoVis: Visual Overviews and Full-Text Search in Software Repositories

Johannes Feiner
FH JOANNEUM University of Applied Sciences, Austria
Email: johannes.feiner@fh-joanneum.at

Keith Andrews
Graz University of Technology, Austria
Email: kandrews@tugraz.at

***Abstract*—Project managers and software developers often have difficulty maintaining an overview of the structure, evolution, and status of collaborative software projects. Some tools are available for typical source code management systems, which provide summary statistics or simple visual representations of merge-branch graphs. However, comprehensive visual overview and search facilities for such repositories are lacking.**

RepoVis is a new tool which provides comprehensive visual overviews and full-text search for projects maintained in Git repositories. The overview shows folders, files, and lines of code colour-coded according to last modification, developer, file type, or associated issues. Full-text searches can be performed for terms of interest within source code files, commit messages, or any associated metadata or usability findings, with matches displayed visually in the overview.

The utility of the RepoVis approach is illustrated with three use cases of real-world software inspection. Insights are presented into the utility of full-text search and visual presentation of matches for program comprehension.

***Index Terms*—Software visualisation, program comprehension, usability, metrics, visual overview, full-text search, git repositories.**

I. Introduction

Program comprehension [1, 2] is relevant to most software development teams in a variety of situations. When new developers join a project, they must rapidly acquire a solid understanding of the software development so far. Project managers must maintain an overview of the entire project. Some of the decisive factors about the software under inspection include the project team (main contributors), code structure (folders, files, and lines of code), code quality (code smells, security, bugs and issues, usability findings, documentation, test artefacts), and code evolution (commit messages, age of code fragments, frequency of changes).

Visual overviews and full-text search can be used to help facilitate rapid program comprehension. RepoVis is a new tool which provides comprehensive visual overviews for projects maintained in Git repositories. The overview visualises folders, files, and lines of code colour-coded according to last modification, developer, file type, and associated issues in a manner similar to Seesoft [3, 4, 5].

In addition, RepoVis provides full-text search for terms of interest within source code files, commit messages, or any metadata or usability findings associated with a software project. Text matches are displayed visually in the overview and can be combined with other visual filters, as can be seen in Fig. 1. Search shortcuts provide pre-composed combinations of search terms for specific situations.

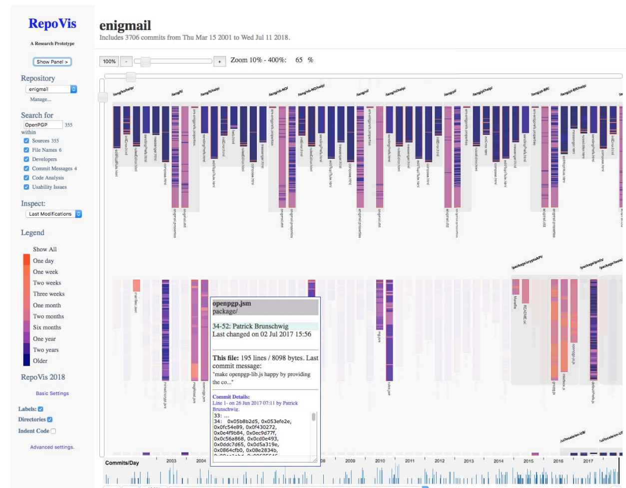


Fig. 1: RepoVis provides a comprehensive visual overview of a software project with overlaid full-text search results.

The remainder of this paper is structured as follows: Section II describes the RepoVis system. Section III describes the full-text search functionality. Section V discusses use cases illustrating the utility of the system for analysing real-world software projects. Related work is discussed in Section VI. Finally, Section VII dicusses some of the current limitations of RepoVis and possible future work.

II. The RepoVis System

RepoVis is designed as a client-server web application. The RepoVis frontend (client) communicates with the RepoVis backend (server) via a RESTful web API [7].

A. Architecture

The RepoVis architecture is shown in Fig. 2. The RepoVis frontend, at the top, runs as a web application inside a web browser. The frontend is written in JavaScript and uses the PixiJS [6] library for 2D graphics rendering with WebGL [8] hardware acceleration. No page reloading is required, since data is fetched asynchronously. At any one time, the frontend shows a single version of the project, corresponding to a

978-1-5386-8292-0/18/$31.00 ©2018 IEEE
DOI 10.1109/VISSOFT.2018.00009

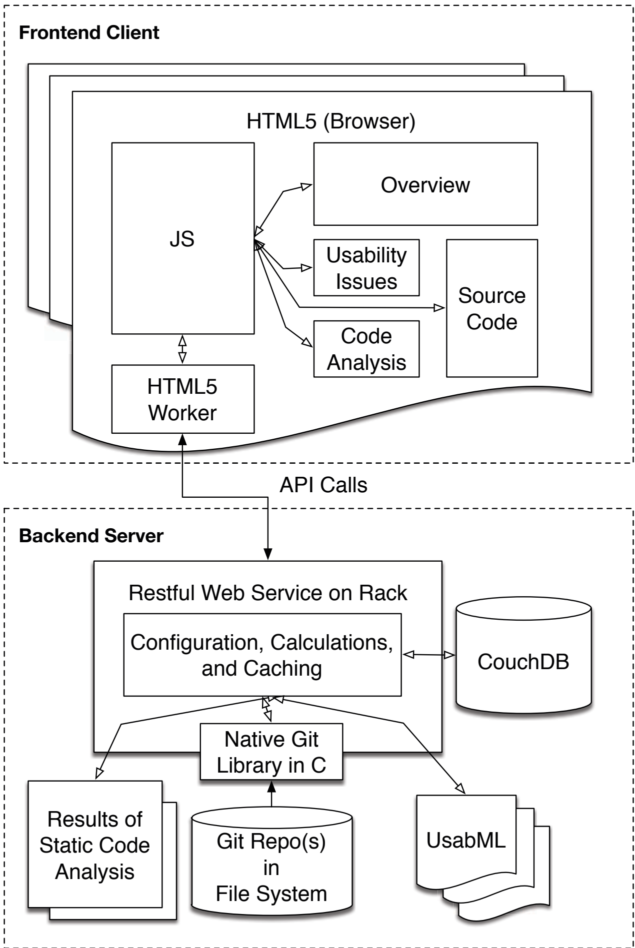


Fig. 2: The RepoVis backend extracts source code, calculates metrics and integrates usability reports. The frontend is a web application built with the PixiJS library [6], which supports WebGL rendering.

```xml
<negative-finding heuristic-id="heu13" rank="1"
    is-main-negative="true" id="neg1">
  <title>Manual Mode in Setup Wizard</title>
  <description><![CDATA[
     The most severe problem was discovered
     in the manual mode for experts, where the
     setup wizard simply does not work correctly.
     After going through the ...]]>
  </description>
  <reproduce>Install Enigmail -> manual setup
      wizard.
  </reproduce>
  <found-by evaluator-id="eval_vk"/>
  <severity evaluator-id="eval_kb">
     <value>4</value>
  </severity>
  ...
  <document type="image">
    <description>Selecting the manual mode for
      experts in the setup wizard window.
    </description>
    <key>krnjic-000258</key>
    <filename>krnjic-000258.png</filename>
  </document
  <document type="video">
    <description>After the unsuccessful manual
        mode of the setup wizard,...
      </description>
  </document>
  <code-reference project-id="plugin/enigmail"
    version-id="commit-eef54326rfe8843ffee23"
    class-id="ui/content/enigmailSetupWizard.js"
    package-id="ui.content"
    method-id="configuration"
    line-number="14" />
</negative-finding>
```

Listing 1: Usability findings in structured UsabML format can be integrated into RepoVis.

particular commit ID. To navigate through time, any commit can be selected on a time slider and the view is refreshed.

The RepoVis backend, at the bottom, is implemented as a Rack [9] web service using the Ruby Sinatra [10] framework. This way, the backend server can optionally be hosted on a cloud platform for better scalability. Access to Git repositories is implemented using the `libgit2` [11] library with the `rugged` wrapper [12] for Ruby. Custom logic is used to integrate source code metrics and usability findings.

A software project under inspection is cloned locally to the file system of the RepoVis backend. The source code, commit messages, and any metadata of interest are then extracted from the local clone on demand. Metadata such as the age of the project, the list of developers, and the timestamp when each line of code was (last) changed by which developer are also extracted from the commit messages.

Static source code analysis can be triggered on demand for a particular software project to generate source code quality reports and associated metrics, for example Pylint [13] for Python or JSLint [14] for JavaScript. This information augments the metadata extracted from the local Git repository.

Usability issues can be integrated into RepoVis by providing them in a structured electronic format like UsabML [15]; an example is provided in Listing 1. The association between a usability issue and a block of code must currently be entered manually for a particular snapshot (commit) of the project. If such findings are available, they will be visualised and displayed as shown in Fig. 3.

The RESTful web service endpoint provides a simple API with JSON-encoded responses. The backend stores data requested by the client as JSON-encoded chunks in a persistent `couchdb` [16] document database. This way, time-consuming operations, such as parsing commit information to determine the last modification of each line of code at a particular point in time, can be calculated once (possibly in advance) and stored to improve performance.

B. *Visualisation*

RepoVis was inspired by Seesoft [3, 4, 5] and adopts its metaphor of looking at source code listings hanging on a wall from far away. The centrepiece is a comprehensive visual

Fig. 3: Usability issues loaded from UsabML are shown within RepoVis.

Fig. 4: RepoVis provides four different colour mappings for the overview visualisation.

overview of the structure (folders, files, and lines of code) of a software project at a particular point in time. Source code files are rendered as boxes, with coloured rows representing one or more lines of code.

Four predefined mappings are available for the colour-coding of each row: the age of a line of code ("Last Modifications"), the developer who last modified it ("Developers"), file type ("File Types"), and usability findings ("Issues"). These are shown in Fig. 4. Categorical data such as "Developers", "File Types", and "Issues" is encoded using a palette of distinct colours chosen for maximum contrast, similar to those suggested by Kelly [17] and Trubetskoy [18]. For "Last Modifications", the age of a line of code is mapped to a colour scale using the `chroma.js` JavaScript library [19].

Selecting one or more facets (categories or bins) in the legend, such as a particular age range or developer, filters the display to shown only files matching that selection, greying out any others. This can be seen in Fig. 5.

The RepoVis timeline at the bottom of the screen allows the user browse through and select any particular commit of the master branch, causing the corresponding state (snapshot)

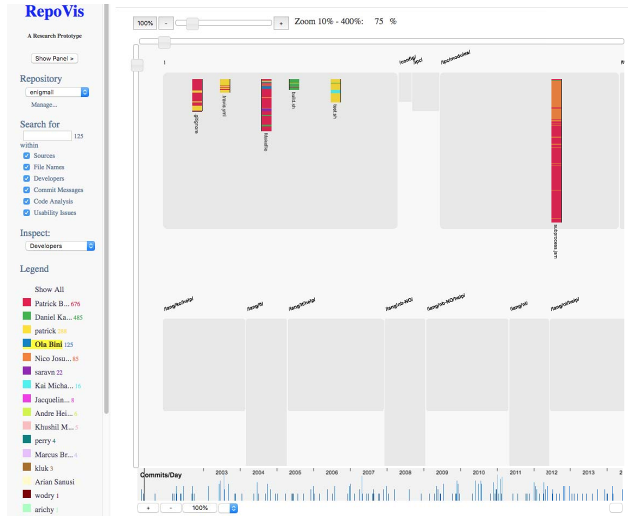


Fig. 5: One or more facets can be selected in the legend to restrict both the display and search scope accordingly. Here, the files worked on by the developer Ola Bini can be seen.

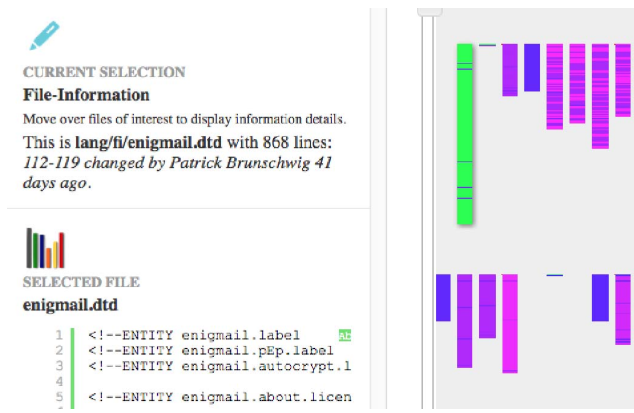


Fig. 6: Linked views provide details on demand.

of the repository at that point in time to be visualised.

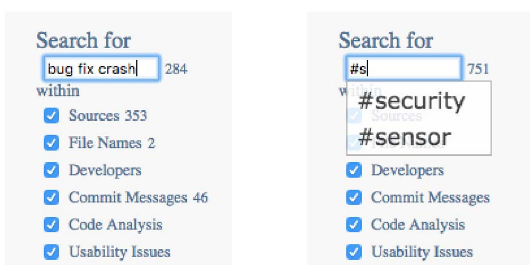
Panning and zooming are provided down to individual lines of code using the mouse and mouse wheel. The display of labels and folder outlines can be toggled to reduce visual clutter. Hovering over the box representing a file reveals its commit details. Clicking on a file shows its source code in a linked view, as shown in Fig. 6.

III. Visual Search Results

RepoVis supports full-text search within a software repository. The search results are displayed in context by highlighting them in the overview visualisation, as shown in Fig. 7. The scope of the search can be restricted to one or more selected facets (categories or bins). It is also possible to define and use search shortcuts.

The search feature is implemented directly in the web client using simple but effective JavaScript regular expressions. No

Fig. 7: Searching for one or more terms (here `thunderbird`) within a software repository highlights the files containing those term(s).



(a) Multiple search terms. (b) Search shortcuts.

Fig. 8: Multiple search terms can be used. Search shortcuts are auto-suggested and implicitly represent multiple search terms.

request/response cycle is necessary when refining or modifying a search.

A. Search Queries

RepoVis provides a single input field for entering search terms, as shown in Fig. 8a. The scope of the search defaults to search within file and path names, developers (committers), commit messages, and any available code analysis reports and usability issues. The search can be extended to source code files on demand, but this option is currently disabled by default, since it requires the RepoVis client to download every source code file in the repository, which may take some considerable time.

Single or multiple search terms can be entered into the search box. Exact phrases can be entered inside double quotation marks. Incremental search is provided by triggering the search on every keystroke to immediately update the visualisation.

TABLE I: Search Shortcuts

Generic Search Shortcuts		
G1	`#problems`	bug fix issue crash
G2	`#refactoring`	rename typo obsolete
Topical Search Shortcuts		
T1	`#security`	credential encrypt password permission secret
T2	`#sensor`	cam gps cam vibration
T3	`#interaction`	click touch swipe input

B. Search Shortcuts

Search shortcuts are available which prefill the search box with a set of specific search terms. Some examples are listed in Table I. Predefined generic search shortcuts include `#problems` and `#refactoring`. These are appropriate to many software projects in a variety of application fields. In addition, topical search shortcuts can be defined for specific application areas.

Search shortcuts are prefixed with a hash # character. They are automatically suggested when the user starts their input with a hash # character, as can be seen in Fig. 8b. For example, `#s` would bring up the suggestions `#security` and `#sensor`. A partially typed shortcut is sufficient, once it can be uniquely identified.

C. Result Highlighting

Search results are currently shown by highlighting the files which contain the corresponding search terms, as shown in Fig. 7. In future, individual matches within files will be shown. The combination of filtering and search provides additional possibilities. When files are filtered to certain facets (categories or bins), the search scope is also restricted to those facets.

IV. Typical Scenario

A typical software inspection with RepoVis is shown in Fig. 9 and might proceed as follows:

1) A repository of choice is cloned to the backend for analysis.
2) Search terms are entered into the search box and the search scope is defined.
3) Different colour mappings can be chosen corresponding to typical inspection tasks.
4) Individual facets can be included or filtered out.
5) A comprehensive overview shows the locations of matching search results in context.
6) Details are shown on demand for any particular line of code.
7) Additional information, such as corresponding issues from usability findings or reports from static code analysis are displayed for that line of code.

V. Use Cases

The following three use cases have been selected as explanatory examples where search and filter with visual overviews can assist in program comprehension.

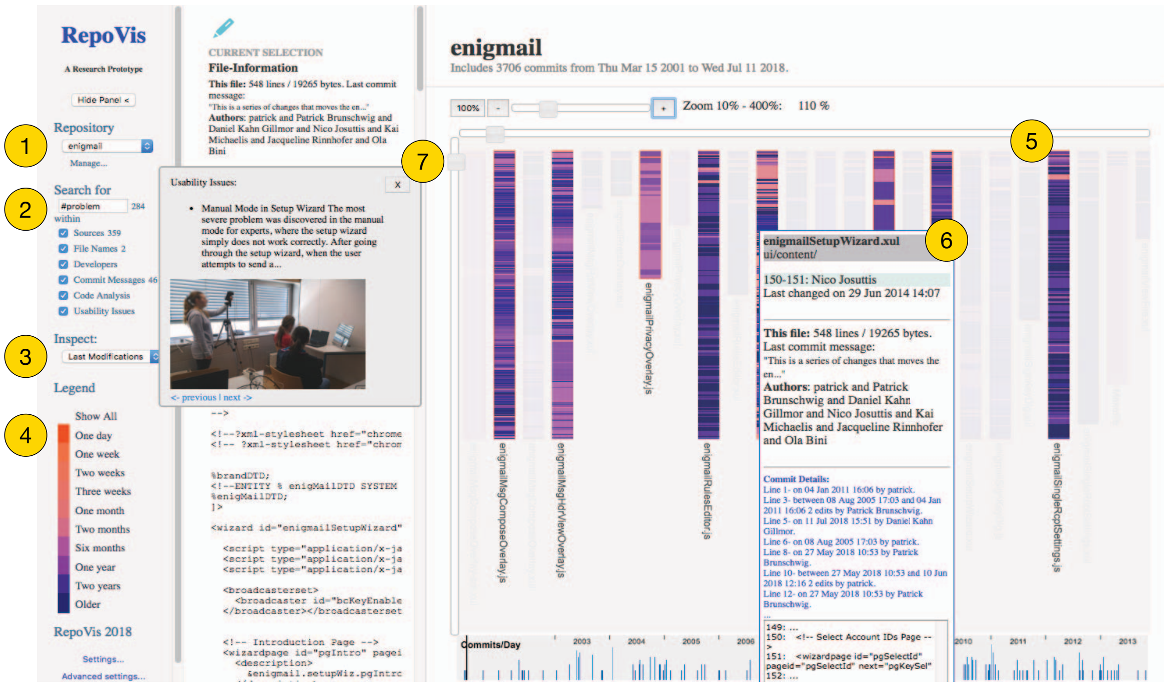


Fig. 9: A typical software inspection with RepoVis. ① A Git repository is cloned to the RepoVis backend. ② A full-text search is issued within all available search scopes. ③ A specific colour mapping is chosen under Inspect. ④ The Legend allows filtering to particular facets. ⑤ Search matches are visualised in the overview. ⑥ Details are shown on demand for a particular line of code. ⑦ Usability issues or code analysis reports are shown for that line of code.

Use Case 1: Code Quality Inspection

Description: To explore potential technical debt and refactoring activities, commit messages can be searched for terms indicating edits or hints about unfinished or not yet implemented features. It is also possible to find out which developer(s) performed refactoring activities such as renaming files or fixing typos.

Example: A search query was performed with the generic search shortcut `#refactor` on the open source project Enigmail [20], an extension to Thunderbird for encrypting emails.

Implications: Four files are highlighted, as shown in Fig. 10.

Use Case 2: Security Inspection

Description: To locate areas of code related to security, the topical search shortcut `#security` can be used. Code related to the use credentials for login, storing passwords or performing encryption might be found.

Example: The Enigmail project was inspected to find out which part of the code deals with security related aspects. A number of pre-existing usability reports for Enigmail, see EnigUsab by Andrews and Wozelka [21], were converted to UsabML [15] in advance and the findings were imported into RepoVis.

Implications: As shown in Fig. 11, several files are concerned with security. This worked well for Enigmail, but a limiting factor for other projects was that the search term `#permission` delivered many files with the Apache License 2.0 in their header. In general, more rarely used words, such as `#credentials`, yield more focused search results. In a further step of this use case, it is possible to restrict the search scope to Usability, to reveal matches within usability findings associated with particular blocks of code.

Use Case 3: Domain-Specific Inspection

Description: For domain-specific inspection, specialised search shortcuts can be defined. For example, for the analysis of mobile apps, topical search shortcuts like `#sensor` or `#interaction` might be of interest.

Example: The open source library ZXing [22] supports reading QR codes within Android apps. A search using the topical search shortcut `#sensor` should quickly bring to light which areas of the project handle the logic for taking photos.

Implications: As can be seen in Fig. 12, several files are related to camera usage and that some lines of specific files have been updated recently. Another term implicitly included in the query for the shortcut `#sensor` was `gps`. However, for the given project this term was not relevant at all.

Discussion

For the selected use cases, full-text searches for multiple terms worked well in many cases and enabled users of RepoVis to quickly gain insights. The interactive and visual approach made it easy to try out various combinations of search terms.

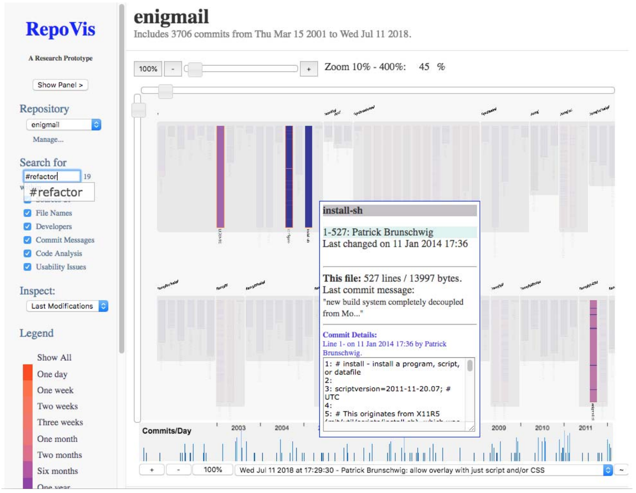


Fig. 10: *Use Case 1:* To inspect code quality, using the shortcut `#refactoring` reveals which developer(s) performed activities on which file(s), such as renaming or fixing typos.

In some cases, far too many results were generated and too many files highlighted. For domain-specific terms, or for project-specific terms, expert knowledge is beneficial. With know-how about typical terms used in the project at hand, developers can find files of interest even faster. Searching within the scope of the source code benefits from sources which are well documented. Sparsely commented files will contain fewer text passages and hence fewer potential matches.

RepoVis requires little instruction to be used for program comprehension tasks. The immediate feedback of the search feature allows users to experiment with search terms and search shortcuts during the inspection of software systems.

VI. Related Work

RepoVis builds on previous research in many related fields, including program comprehension, software visualisation, source code analysis, repository mining, and usability reporting and issue tracking.

A. Program Comprehension

Program comprehension deals with how software developers and software project managers comprehend, understand, and manage software source code. In 1978, Brooks [23] proposed a behavioural theory for program comprehension in software engineering. Pennington [24] looked at how expert programmers comprehend code. Dasgupta [25] discussed the influence of the authorship of code on its comprehension.

Sulír [1] presents a short overview of approaches and tools for program comprehension. and Cornelissen *et al.* [26] review techniques and tools for program comprehension through dynamic analysis of software at run-time. A topology of human reading techniques for software is presented in Shull *et al.* [27]. Different procedures and questions for the analysis of object-oriented source code are discussed. Techniques which

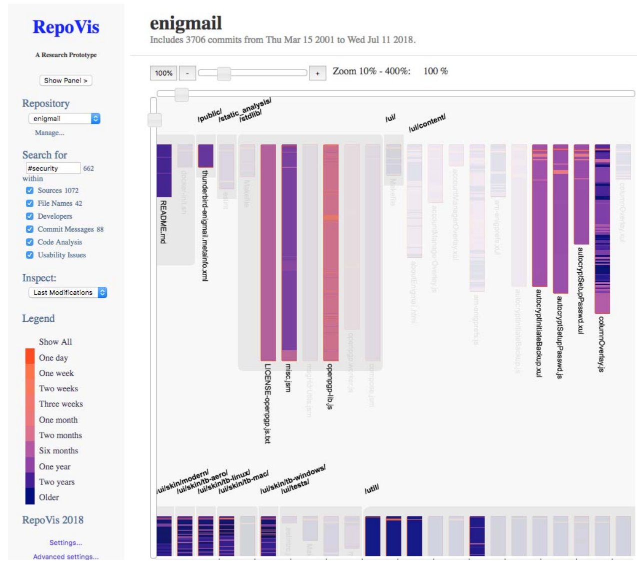


Fig. 11: *Use Case 2*: When inspecting security aspects, the search shortcut `#security` reveals a series of files related to encryption, permission, and secret.

focused more on semantics than on syntax showed faster detection of bugs within the given code.

In education, Nelson *et al.* [28] promote a *comprehension first* approach for young software developers. The tool ARCC [29] was created to assist with the detection and comprehension of recurring code snippets. Siegmund *et al.* [30] investigated how beacons (semantic cues such as method signatures) are used by programmers to comprehend large software programs. Further research in the domain of software maintenance is surveyed by Koschke [31].

RepoVis builds on the research done in program comprehension through its visual overviews and full-text search. Program comprehension tasks such as spotting problematic areas of code or identifying areas of code recently edited or last edited by particular programmers can quickly be performed with no up-front instructions.

B. Software Visualisation

Source code consists of many artefacts and internal structures, which can be visualised in various ways. Myers [32], Price *et al.* [33], and Maletic *et al.* [34] all present taxonomies for software visualisation systems.

Seesoft [3, 4, 5] provides a colour-coded overview visualisation of software source code with multiple linked views and drill-down features. Information murals [35, 36] can be used to provide a grand overview of large software projects. MosaiCode [37] draws on a similar metaphor to visualise entities of large software systems as coloured tiles according to various attributes or metrics.

Reiss [38] discusses a software visualisation backend called Bee/Hive including the retrieval of trace, analysis, and semantic data from different sources. The framework is part

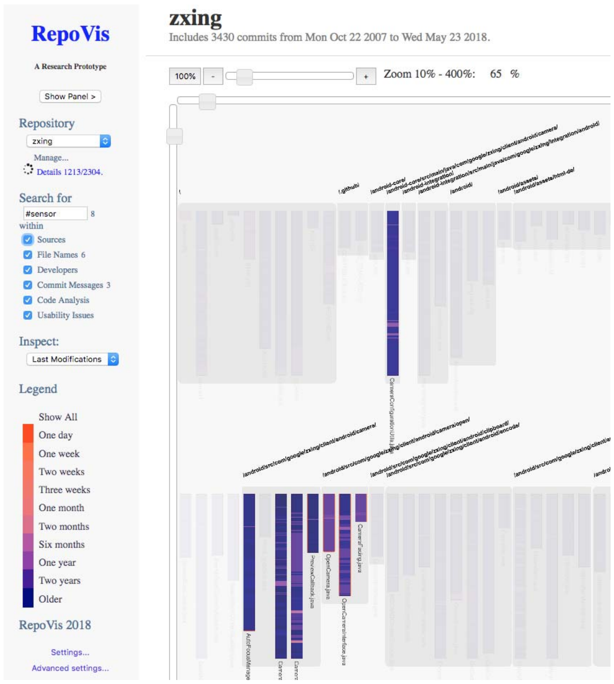


Fig. 12: *Use Case 3*: Search is useful to inspect domain-specific topics. The topical search term `#sensor` identifies those parts of the source code of a mobile application dealing with the camera.

of a comprehensive system called BLOOM [39]. The G^{SEE} [40] software visualisation framework uses a spreadsheet-like approach to access "software facts" from a variety of sources. Voinea and Telea [41] developed the Solid* [42] toolset with SolidSX [43] for visual exploration of program structure, dependencies, and metrics.

Software as cities was introduced by Wettel *et al.* [44] where classes are arranged in a virtual town corresponding to the package structure. Nunes *et al.* [45] used the same approach to detect problems in the CodeCity system [46]. ExplorViz [47] visualises software as landscapes. Fittkau [48] further suggest tracing the program flow when using the city metaphor. The collaborative aspect is implemented in the web-based tool TeamWATCH [49], allowing cooperation and source code visualisation in 3D. Teyseyre and Campo [50] provide an overview of 3D software visualisation tools.

SAMOA [51] provides views onto the source code of mobile applications with a focus on structural and historical information. Microprints [52] are coloured graphics (a coloured box is generated for each character of the corresponding source code) which characterise the source code according to its semantics.

Kuhn *et al.* [53] discuss visualising software artefacts on a thematic map. CodeSurveyor [54] produces a simlar map-like visualisation based on an underlying code dependency graph generated with Frappé [55]. Griswold *et al.* [56] describe a software evolution tool called AspectBrowser which uses a map metaphor for tracking global changes in large systems. The tool BugMap [57] visualises the distribution of bugs on a topographic map. D'Ambros *et al.* [58] discuss the visualisation of bugs in a bug database.

RepoVis employs the 2D listings hanging on wall visualisation metaphor introduced by Seesoft, in combination with full-text search, source code metrics, and usability findings.

C. Static Source Code Analysis

Software bugs can become extremely expensive to fix once the software is in production [59]. Finding bugs early is financially advantageous. To judge software in terms of quality, for example to track deterioration over time, it is necessary to objectively measure code. Many classic software quality metrics defined many decades ago are still used today, for example by Halstead [60] and McCabe and Butler [61], or for object-oriented software by Martin [62]. Some metrics are specialised for finding patterns as discussed by Hovemeyer and Pugh [63], others for security [64, 65]. Garbervetsky *et al.* [66] propose a distributed system for static code analysis of large systems with live updates.

Hindle *et al.* [67] argue that many facets of source code are similar to natural language processing. A thesaurus of different terms frequently used by developers is helpful to categorise and classify artefacts, such as comments in code snippets or commit messages. To identify topics in source code, Kuhn *et al.* [68] use semantic clustering techniques.

Code inspection techniques are supported by RepoVis, but are currently limited to the integration of static analysis reports. Arbitrary tools can be configured to be run against the code on the backend server. RepoVis does not yet incorporate semantic analysis techniques.

D. Software Repository Mining

Software repository mining utilises not only source code, but also additional metadata such as commits, version information, and issue tracking. Tools such as GitcProc [69] use regular expressions to extract relevant changes within the codebase.

Software evolution tools make it possible to inspect changes over time in the source code of a software system, and to extract differences between commit snapshots. North *et al.* [70] explored approaches of understanding Git history. Servant and Jones [71] created the tool CHRONOS to show slices of history.

RepoVis augments textual querying of source code and associated metadata with an overview visualisation.

E. Usability Reporting and Issue Tracking

Reporting and fixing usability findings is essential to improve the user experience (UX). There has been some discussion as to whether and how usability issues should be integrated into classic bug-tracking systems [72].

UsabML [73, 15] was designed to provide a structured report format allowing hand-over of usability findings to other systems (such as bug trackers) and automated re-use. UseApp [74] supports usability evaluation and reporting when inspecting or testing mobile apps. RepoVis is able to integrate usability findings from UsabML reports and then display them with their associted blocks of source code. Feiner *et al.* [75] describe a pathway for the collection and integration of such findings.

Thung *et al.* [76] describe an information retrieval approach to analyse bug reports and attempt to localise the associated bugs in the source code, which was implemented as an extension to Bugzilla. However, they did not visualise the results.

VII. Limitations and Future Work

RepoVis is intended for small to medium-sized software repositories. More work needs to be done on scaling up the system to larger repositories.

The full-text search features are currently implemented in JavaScript on the client, which imposes some limits on scalability. Possibilities for server-side search are currently under investigation. Furthermore, the overview visualisation currently highlights the files in which search matches are located, not the individual locations within each file. Of course, this would be highly desirable and has high priority in terms of future work.

The RepoVis timeline still needs to be perfected and there are definitely more possibilities to visualise project evolution and merge-branch graphs.

The visualisation of metrics is limited in RepoVis insofar as metrics are visualised on a per file basis. For example, dependencies between files cannot currently be shown.

Usability findings can be imported from UsabML into RepoVis, but the association of each finding to a particular block of code in a particular commit currently has to be done manually by editing the UsabML file. At some point, it would be desirable to have a user interface for project managers or developers to graphically associate area(s) of code with a usability finding. A related issue is how to "migrate" findings from commit to commit, if associated blocks of code are moved, edited, or deleted.

Three use cases have been described illustrating the utility of RepoVis for specific scenarios, but much more evaluation need to be done. Firstly, some formative usability testing needs to be done to find and fix usability issues. Secondly, it is intended to test RepoVis for one or two trial projects over a longer period of time.

VIII. Concluding Remarks

RepoVis is a new tool which provides both comprehensive visual overviews and full-text search for projects maintained in Git repositories. Its visual overview shows folders, files, and lines of code colour-coded according to last modification, developer, file type, or associated issues. Full-text searches can be performed for terms of interest within source code files, commit messages, or any associated metadata or usability findings, with matches displayed visually in the overview.

Much work remains to be done, but RepoVis has the potential to support both software developers and project managers maintain an overview of the structure, evolution, and status of their software projects.

References

[1] M. Sulír, "Program comprehension: A short literature review," in *Proc. 15th Scientific Conference of Young Researchers (SCYR)*, FEI TU of Košice, May 19, 2015, pp. 282–286, ISBN: 8055321302. [Online]. Available: http://mat.us.to/papers/Sulir15program.pdf.

[2] A. R. Santos, I. do Carmo Machado, and E. S. de Almeida, "Aspects influencing feature-oriented software comprehension: Observations from a focus group," in *Proc. 11th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS 2017)*, (Fortaleza, Ceará, Brazil), ACM, Sep. 18, 2017, 10:1–10:10, ISBN: 1450353258. DOI: 10.1145/3132498.3133838.

[3] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr., "Seesoft — a tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 957–968, Nov. 1992. DOI: 10.1109/32.177365. [Online]. Available: http://www.cs.kent.edu/~jmaletic/softvis/papers/eick1992.pdf.

[4] S. G. Eick, "Graphically displaying text," *Journal of Computational Graphics and Statistics*, vol. 3, no. 2, pp. 127–142, 1994. DOI: 10.1080/10618600.1994.10474635. [Online]. Available: http://www.cs.kent.edu/~jmaletic/softvis/papers/eick1994.pdf.

[5] T. Ball and S. G. Eick, "Software visualization in the large," *IEEE Computer*, vol. 29, no. 4, pp. 33–43, Apr. 1996. DOI: 10.1109/2.488299. [Online]. Available: http://www.cs.kent.edu/~jmaletic/softvis/papers/BallEick1996.pdf.

[6] M. Groves. (Jan. 20, 2013). PixiJS, [Online]. Available: http://pixijs.com/ (visited on 03/27/2018).

[7] R. T. Fielding, "Architectural styles and the design of network-based software architectures," PhD thesis, University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[8] WebGL. (Mar. 3, 2011). WebGL – OpenGL ES for the web, [Online]. Available: https://khronos.org/webgl/ (visited on 05/21/2018).

[9] C. Neukirchen. (May 17, 2007). Rack: A ruby webserver interface, [Online]. Available: http://rack.github.io (visited on 03/29/2018).

[10] B. Mizerany. (Sep. 9, 2007). Sinatra, [Online]. Available: http://sinatrarb.com (visited on 04/23/2018).

[11] libgit2. (Oct. 26, 2008). Libgit2 – the git linkable library, [Online]. Available: https://github.com/libgit2/libgit2 (visited on 03/27/2018).

[12] V. Marti and S. C. A. Schreiber. (May 2, 2010). Rugged, [Online]. Available: https://github.com/libgit2/rugged (visited on 03/27/2018).

[13] S. Thenault. (May 19, 2003). Pylint, [Online]. Available: https://pylint.org/ (visited on 03/27/2018).

[14] D. Crockford. (Nov. 7, 2010). JSLint, [Online]. Available: https://www.jslint.com (visited on 03/27/2018).

[15] J. Feiner, K. Andrews, and E. Krajnc, "UsabML – the usability report markup language: Formalising the exchange of usability findings," in *Proc. 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2010)*, (Berlin, Germany), ACM, Jun. 19, 2010, pp. 297–302, ISBN: 1450300839. DOI: 10.1145/1822018.1822065.

[16] couchdb. (Mar. 23, 2008). Couchdb, [Online]. Available: http://couchdb.apache.org/ (visited on 07/06/2018).

[17] Kelly, *Twenty-two colors of maximum contrast*. Color Eng., 1965, pp. 26–27.

[18] S. Trubetskoy. (Jan. 11, 2017). List of 20 simple, distinct colors, [Online]. Available: https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/ (visited on 06/11/2018).

[19] G. Aisch. (Apr. 30, 2018). Chroma.js, [Online]. Available: http://gka.github.io/chroma.js/ (visited on 07/13/2018).

[20] P. Brunschwig. (Jul. 30, 2012). Enigmail, [Online]. Available: https://enigmail.net/ (visited on 03/27/2018).

[21] K. Andrews and R. Wozelka. (2015). Enigusab, [Online]. Available: https://projects.isds.tugraz.at/enigusab/ (visited on 03/27/2018).

[22] S. Owen and D. Switkin. (Mar. 1, 2008). ZXing barcode scanning library for java, android, [Online]. Available: https://github.com/zxing/zxing (visited on 05/20/2018).

[23] R. Brooks, "Using a behavioral theory of program comprehension in software engineering," in *Proc. 3rd International Conference on Software Engineering (ICSE 1978)*, (Atlanta, Georgia, USA), IEEE Press, 1978, pp. 196–201. [Online]. Available: https://dl.acm.org/citation.cfm?id=803210.

[24] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987. [Online]. Available: https://pdfs.semanticscholar.org/3036/e10df2e4855c56bf174fc1e00f6dd4100f55.pdf.

[25] C. Dasgupta, "That is not my program: Investigating the relation between program comprehension and program authorship," in *Proc. 48th Annual Southeast Regional Conference (ACM SE 2010)*, (Oxford, Mississippi), ACM, 2010, 103:1–103:4, ISBN: 1450300642. DOI: 10.1145/1900008.1900142.

[26] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, Apr. 3, 2009. DOI: 10.1109/TSE.2009.28.

[27] F. Shull, G. H. Travassos, J. Carver, and V. R. Basili, "Evolving a set of techniques for OO inspections," University of Maryland, Tech. Rep., 1999. [Online]. Available: http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/CS-TR-4070.pdf.

[28] G. L. Nelson, B. Xie, and A. J. Ko, "Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1," in *Proc. ACM Conference on International Computing Education Research (ICER 2017)*, (Tacoma, Washington, USA), ACM, Aug. 18, 2017, pp. 2–11, ISBN: 1450349684. DOI: 10.1145/3105726.3106178.

[29] W. Z. Nunez, V. J. Marin, and C. R. Rivero, "Arcc: Assistant for repetitive code comprehension," in *Proc. 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*, (Paderborn, Germany), ACM, Sep. 4, 2017, pp. 999–1003, ISBN: 1450351050. DOI: 10.1145/3106237.3122824.

[30] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann, "Measuring neural efficiency of program comprehension," in *Proc. 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*, (Paderborn, Germany), ACM, Sep. 4, 2017, pp. 140–150, ISBN: 1450351050. DOI: 10.1145/3106237.3106268.

[31] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003. DOI: 10.1002/smr.270.

[32] B. A. Myers, "Taxonomies of visual programming and program visualization," *Journal of Visual Languages & Computing*, vol. 1, no. 1, pp. 97–123, Mar. 1, 1990. DOI: 10.1016/S1045-926X(05)80036-9. [Online]. Available: http://cs.cmu.edu/~bam/papers/vltax2.pdf.

[33] B. A. Price, I. S. Small, and R. M. Baecker, "A taxonomy of software visualization," in *Proc. 25th Hawaii International Conference on System Sciences (HICSS 1992)*, vol. 2, Jan. 1992, pp. 597–606. DOI: 10.1109/HICSS.1992.183311.

[34] J. I. Maletic, A. Marcus, and M. L. Collard, "A task oriented view of software visualization," in *Proc. 1st IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002)*, (Paris, France), Jun. 26, 2002, pp. 32–40. DOI: 10.1109/VISSOF.2002.1019792.

[35] D. F. Jerding and J. T. Stasko, "The information mural: A technique for displaying and navigating large information spaces," in *Proc. 1995 IEEE Symposium on Information Visualization (InfoVis 1995)*, (Atlanta, Georgia, USA), IEEE Computer Society, Oct. 1995, pp. 257–271, ISBN: 0818672013. DOI: 10.1109/2945.722299. [Online]. Available: https://cc.gatech.edu/~stasko/papers/infovis95.pdf.

[36] D. F. Jerding and J. T. Stasko, "Using information murals in visualization applications," in *Proc. 8th Annual ACM Symposium on User Interface and Software Technology (UIST 1995)*, (Pittsburgh, Pennsylvania, USA),

ACM, Nov. 15, 1995, pp. 73–74, ISBN: 089791709X. DOI: 10.1145/215585.215660.

[37] J. I. Maletic, D. J. Mosora, C. D. Newman, M. L. Collard, A. M. Sutton, and B. P. Robinson, "MosaiCode: visualizing large scale software: A tool demonstration," in *Proc. 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2011)*, (Williamsburg, Virginia, USA), Sep. 29, 2011, pp. 1–4, ISBN: 1457708221. DOI: 10.1109/VISSOF.2011.6069457.

[38] S. P. Reiss, "Bee/Hive: A software visualization back end," Apr. 20, 2001.

[39] S. P. Reiss, "An overview of bloom," in *Proc. 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2001)*, (Snowbird, Utah, USA), ACM, 2001, pp. 2–5, ISBN: 1581134134. DOI: 10.1145/379605.379629.

[40] J.-M. Favre, "A flexible approach to visualize large software products," in *Proc. Workshop on Software Visualization (ICSE'01)*, (Totonto, Canada), May 12, 2001. [Online]. Available: http://www-adele.imag.fr/Les.Publications/intConferences/WSV2001Fav.pdf.

[41] L. Voinea and A. C. Telea, "Visual clone analysis with solidsdd," in *Proc. 2nd IEEE Working Conference on Software Visualization (VISSOFT 2014)*, (Victoria, British Columbia, Canada), Sep. 29, 2014, pp. 79–82. DOI: 10.1109/VISSOFT.2014.22.

[42] D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Sci. Comput. Program.*, vol. 79, pp. 224–240, Jan. 2014. DOI: 10.1016/j.scico.2012.05.002.

[43] D. Reniers, L. Voinea, and A. Telea, "Visual exploration of program structure, dependencies and metrics with solidsx," in *Proc. 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2011)*, (Williamsburg, Virginia, USA), Sep. 29, 2011, pp. 1–4. DOI: 10.1109/VISSOF.2011.6069461.

[44] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proc. 33rd International Conference on Software Engineering (ICSE 2011)*, (Waikiki, Honolulu, Hawaii, USA), ACM, May 21, 2011, pp. 551–560, ISBN: 1450304451. DOI: 10.1145/1985793.1985868.

[45] R. Nunes, M. Rebouças, F. Soares-Neto, and F. Castor, "Visualizing swift projects as cities: Poster," in *Proc. 39th International Conference on Software Engineering Companion (ICSE-C 2017)*, (Buenos Aires, Argentina), IEEE Press, May 20, 2017, pp. 368–370, ISBN: 1538615894. DOI: 10.1109/ICSE-C.2017.115.

[46] R. Wettel and M. Lanza, "Visually localizing design problems with disharmony maps," in *Proc. 4th ACM Symposium on Software Visualization (SoftVis 2008)*, (Ammersee, Germany), ACM, Sep. 16, 2008, pp. 155–164, ISBN: 1605581127. DOI: 10.1145/1409720.1409745.

[47] F. Fittkau, A. Krause, and W. Hasselbring, "Software landscape and application visualization for system comprehension with explorviz," *Inf. Softw. Technol.*, vol. 87, no. C, pp. 259–277, Jul. 4, 2016. DOI: 10.1016/j.infsof.2016.07.004.

[48] F. Fittkau, "Live trace visualization for system and program comprehension in large software landscapes," Dissertation, Dissertation, Department of Computer Science, Faculty of Engineering, Kiel University, Dec. 2015, ISBN: 3739207167.

[49] M. Gao and C. Liu, "Teamwatch demonstration: A web-based 3d software source code visualization for education," in *Proc. 1st International Workshop on Code Hunt Workshop on Educational Software Engineering (CHESE 2015)*, (Baltimore, MD, USA), ACM, Jul. 14, 2015, pp. 12–15, ISBN: 9781450337113. DOI: 10.1145/2792404.2792408.

[50] A. R. Teyseyre and M. R. Campo, "An overview of 3D software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 1, pp. 87–105, 2009. DOI: 10.1109/TVCG.2008.86.

[51] R. Minelli and M. Lanza, "Software analytics for mobile applications – insights & lessons learned," in *Proc. 17th European Conference on Software Maintenance and Reengineering (CSMR 2013)*, Mar. 5, 2013, pp. 144–153. DOI: 10.1109/CSMR.2013.24.

[52] S. Ducasse, M. Lanza, and R. Robbes, "Multi-level method understanding using microprints," in *Proc. 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005)*, (Budapest, Hungary), Sep. 25, 2005, pp. 1–6. DOI: 10.1109/VISSOF.2005.1684300.

[53] A. Kuhn, P. Loretan, and O. Nierstrasz, "Consistent layout for thematic software maps," in *Proc. 15th Working Conference on Reverse Engineering*, 2008, pp. 209–218. DOI: 10.1109/WCRE.2008.45.

[54] N. Hawes, S. Marshall, and C. Anslow, "Codesurveyor: Mapping large-scale software to aid in code comprehension," in *Proc. 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, (Bremen, Germany), Sep. 27, 2015, pp. 96–105. DOI: 10.1109/VISSOFT.2015.7332419.

[55] N. Hawes, B. Barham, and C. Cifuentes, "Frappé: Querying the linux kernel dependency graph," in *Proc. Graph Data-management Experiences & Systems (GRADES 2015)*, (Melbourne, Australia), ACM, Jan. 2015, 4:1–4:6, ISBN: 1450336116. DOI: 10.1145/2764947.2764951.

[56] W. G. Griswold, J. J. Yuan, and Y. Kato, "Exploiting the map metaphor in a tool for software evolution," in *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, (Toronto, Ontario, Canada), IEEE Computer Society, May 12, 2001, pp. 265–274, ISBN: 0769510507.

[57] J. Gong and H. Zhang, "Bugmap: A topographic map of bugs," in *Proc. 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*, (Saint Petersburg, Russia), ACM, Aug. 18, 2013, pp. 647–650, ISBN: 1450322379. DOI: 10.1145/2491411.2494582.

[58] M. D'Ambros, M. Lanza, and M. Pinzger, ""A Bug's Life" visualizing a bug database," in *Proc. 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, (Banff, Canada), Jun. 24, 2007, pp. 113–120, ISBN: 1424405998. DOI: 10.1109/VISSOF.2007.4290709.

[59] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *Strategic Planning*, May 2002.

[60] M. H. Halstead, *Elements of Software Science.* Elsevier, May 1977, ISBN: 0444002057.

[61] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, Dec. 1989. DOI: 10.1145/76380.76382.

[62] R. Martin, *Oo design quality metrics - an analysis of dependencies*, 1994. [Online]. Available: http://www.objectmentor.com/resources/articles/oodmetrc.pdf (visited on 12/19/2009).

[63] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Notices*, vol. 39, no. 12, pp. 92–106, Dec. 2004. DOI: 10.1145/1052883.1052895. [Online]. Available: http://cs.nyu.edu/~lharris/papers/findbugsPaper.pdf.

[64] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," in *Proc. 4th International Workshop on Software Engineering for Secure Systems (SESS 2008)*, (Leipzig, Germany), ACM, May 17, 2008, pp. 57–64, ISBN: 1605580422. DOI: 10.1145/1370905.1370913.

[65] L. Krautsevich, F. Martinelli, and A. Yautsiukhin, "Formal approach to security metrics – what does "more secure" mean for you?" In *Proc. 4th European Conference on Software Architecture: Companion Volume (ECSA 2010)*, (Copenhagen, Denmark), ACM, Aug. 23, 2010, pp. 162–169, ISBN: 1450301797. DOI: 10.1145/1842752.1842787.

[66] D. Garbervetsky, E. Zoppi, and B. Livshits, "Toward full elasticity in distributed static analysis: The case of callgraph analysis," in *Proc. 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*, (Paderborn, Germany), ACM, Sep. 4, 2017, pp. 442–453, ISBN: 9781450351058. DOI: 10.1145/3106237.3106261.

[67] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, "On the naturalness of software," *Commun. ACM*, vol. 59, no. 5, pp. 122–131, May 2016. DOI: 10.1145/2902362.

[68] A. Kuhn, S. Ducasse, and T. Girba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, pp. 230–243, Jan. 4, 2007. DOI: 10.1016/j.infsof.2006.10.017.

[69] C. Casalnuovo, Y. Suchak, B. Ray, and C. Rubio-Gonzàlez, "Gitcproc: A tool for processing and classifying github commits," in *Proc. 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*, (Santa Barbara, CA, USA), Jul. 2017, pp. 396–399, ISBN: 1450350763. DOI: 10.1145/3092703.3098230.

[70] K. J. North, A. Sarma, and M. B. Cohen, "Understanding git history: A multi-sense view," in *Proc. 8th International Workshop on Social Software Engineering (SSE 2016)*, (Seattle, Washington, USA), ACM, 2016, pp. 1–7, ISBN: 145034397X. DOI: 10.1145/2993283.2993285.

[71] F. Servant and J. A. Jones, "Chronos: Visualizing slices of source-code history," in *Proc. 1st IEEE Working Conference on Software Visualization (VISSOFT 2013)*, (Eindhoven, Netherlands), Sep. 27, 2013, pp. 1–4. DOI: 10.1109/VISSOFT.2013.6650547. [Online]. Available: http://spideruci.org/papers/servant13sep.pdf.

[72] C. E. Wilson and K. P. Coyne, "The whiteboard: Tracking usability issues: To bug or not to bug?" *Interactions*, vol. 8, no. 3, pp. 15–19, May 2001. DOI: 10.1145/369825.369828.

[73] J. Feiner and K. Andrews, "Usability reporting with UsabML," in *Proc. 4th International Conference on Human-Centered Software Engineering (HCSE 2012)*, M. Winckler, P. Forbrig, and R. Bernhaupt, Eds., vol. 7623, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Oct. 29, 2012, pp. 342–351, ISBN: 3642343465. DOI: 10.1007/978-3-642-34347-6_26.

[74] J. Feiner, K. Andrews, and E. Krainz, "Convenient mobile usability reporting with UseApp," in *Proc. 9th Forum Media Technology (FMT 2016)*, (St. Pölten, Austria), W. Aigner, G. Schmiedl, K. Blumenstein, M. Zeppelzauer, and M. Iber, Eds., CEUR Workshop Proceedings, Nov. 24, 2016, pp. 41–46. [Online]. Available: http://ceur-ws.org/Vol-1734/fmt-proceedings-2016-paper5.pdf.

[75] J. Feiner, E. Krainz, and K. Andrews, "A new approach to visualise accessibility problems of mobile apps in source code," in *Proc. 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, S. Hammoudi, M. Smialek, O. Camp, and J. Filipe, Eds., vol. 2, SciTePress, Feb. 22, 2018, pp. 519–526, ISBN: 9897582983. DOI: 10.5220/0006704405190526. [Online]. Available: https://ftp.isds.tugraz.at/pub/papers/feiner-iceis2018-qac.pdf.

[76] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo, "Buglocalizer: Integrated tool support for bug localization," in *Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, (Hong Kong, China), ACM, Nov. 16, 2014, pp. 767–770, ISBN: 1450330568. DOI: 10.1145/2635868.2661678.