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

Fig. 1: Search shortcuts provide pre-composed combinations of search terms for specific situations.

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 discusses 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
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.

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) 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.

TABLE I: Search Shortcuts

<table>
<thead>
<tr>
<th></th>
<th>Generic Search Shortcuts</th>
<th>Topical Search Shortcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>G1 #problems</td>
<td>G2 #refactoring</td>
</tr>
<tr>
<td></td>
<td>bug fix issue crash</td>
<td>rename typo obsolete</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T1 #security</td>
<td>T2 #sensor</td>
</tr>
<tr>
<td></td>
<td>credential encrypt password</td>
<td>cam gps cam vibration</td>
</tr>
<tr>
<td></td>
<td>secret</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T3 #interaction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>click touch swipe input</td>
<td></td>
</tr>
</tbody>
</table>

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.
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.
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


Sulir [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 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
of a comprehensive system called BLOOM [39]. The GSEE [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 similar 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 GiteProc [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 associated 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 needs 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


