CodePanorama: a language agnostic tool for visual code inspection

Marc Etter
OST Eastern Switzerland University of Applied Sciences
Department of Computer Science
Switzerland
marc.etter@ost.ch

Farhad Mehta
OST Eastern Switzerland University of Applied Sciences
Department of Computer Science
Switzerland
farhad.mehta@ost.ch

ABSTRACT

Software projects change hands frequently. Oftentimes, developers are interested in the quality of the code before taking over responsibility on a project. This quality is commonly assessed using various code metrics, reducing the code into a handful of numbers. While useful, these numerical reductions quickly become detached from the real code. CodePanorama uses an alternative approach to summarize code not into numbers, but into images. By generating zoomed-out images of the code-base, the human eye can quickly spot anomalies without the need to rely on numerical metrics and statistics. This paper describes the tool CodePanorama, the images it generates, and the insights that can be gained from these images. We finally invite the software engineering community to start using it.

KEYWORDS

Software Visualization, Code Review, Evolution and Maintenance, Software Quality, Software Engineering

ACM Reference Format:
Marc Etter and Farhad Mehta. 2022. CodePanorama: a language agnostic tool for visual code inspection. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3524610.3527874

1 INTRODUCTION

There exist a number of metrics for the estimation of software quality [5]. By their very nature, these metrics are reductionistic: they aim to express software quality with a handful of numbers. In practice, anyone who wants to assess the quality of a piece of software for themselves, does not solely depend on these sets of numbers. Instead, they typically also scroll through the source code to build their own impression of its quality. Such an impression can be important when deciding to maintain or develop a project further, or to grade a project as an instructor.

We present *CodePanorama*: a language agnostic tool for visual code inspection. In contrast to reductionistic metrics, *CodePanorama* generates zoomed-out images (so-called code panoramas) of the entire selected code-base of a software project, thereby allowing the reviewer to take advantage of their innate image processing skills [3] to instantly get a good first impression of thousands of lines of code. Furthermore, the tool allows quickly honing in on the code's more potentially problematic parts directly within the tool.

In addition to just displaying a zoomed-out version of the code-base, a user can additionally enrich a code panorama with colors representing important project-relevant information such as change frequency, author, or custom text searches. They are thereby able to reach a more conclusive judgement of code quality.

We claim that the use of such a tool complements existing quality estimation metrics: Using it, an experienced developer is able to have a similar estimate of code quality as reported by more involved code analysis tools. Furthermore, its use allows an experienced developer to spot problems in the code that are not detected by standard code quality metrics.

We have made the *CodePanorama* tool freely available for public use as a web application[1].

2 RELATED WORK

The idea behind this visualization originated out of a need to evaluate the code quality of large student projects. Our initial search for such a tool did not reveal anything that we could use. After developing the tool, we were made aware that similar visualizations had been attempted before. *SeeSoft* [4], developed in the 1990s, is often cited as the first implementation of this so-called "code-map metaphor" [1]. Bacher et al. have published a review paper, comparing 21 different implementations of the code-map metaphor [1]. Their study concludes that such a visualization has great benefits in the software engineering process. Nevertheless, we observe that its use is virtually nonexistent in the standard software engineering process. We believe that the *CodePanorama* tool will allow this form of visualization to be used more frequently in modern software engineering projects. The reasons for our belief are that it fills useful and important gaps not (or only partially) covered by the other tools, namely:

Convenience *CodePanorama* is publicly available as a web application. Therefore, there is no need to install anything locally to analyze a project.

Applicability *CodePanorama* was designed to be language-agnostic, therefore it is able to visualize any repository, regardless of which programming language is used. It can even be used for projects that contain just text, but no code, such as technical documentation. For these projects, information such as authorship or change frequency are still relevant.

[1] https://codepanorama.io

ICPC '22, May 16–17, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9298-3/22/05.
https://doi.org/10.1145/3524610.3527874

Availability *CodePanorama* is free to use. There are no licensing requirements.

Privacy In case privacy is an issue, *CodePanorama* can also be run as a self-hosted application, where it can analyze private repositories.

Scalability *CodePanorama* can display more lines per image than its predecessors. This is achieved through a more compact arrangement of the visualization.

Extensibility *CodePanorama* supports visualizing arbitrary user-supplied data as a color-overlay, enabling flexibility for whatever metrics are desired.

3 CODEPANORAMA IN DETAIL

CodePanorama takes code from a Git repository and produces interactive images by generating a line of pixels for each line of code. This abstraction generates a "zoomed-out" view of the code. This view can also be thought of as the silhouette of the code, as indentation and whitespace are preserved. *CodePanorama* is built with a server component and a user interface component. The backend server is implemented using Haskell[2]. The web interface is implemented using Elm[3].

The following section describe the basic features of *CodePanorama* in the sequence of a typical user workflow.

3.1 Repository Selection

To start, *CodePanorama* must be told which repository to analyze. On the public instance, this is done by providing an HTTPS git URL to a public repository. On self-hosted instances, repositories requiring credentials can be analyzed as well. A self-hosted instance can also be used to analyze repositories from a local directory, as well as non-git projects.

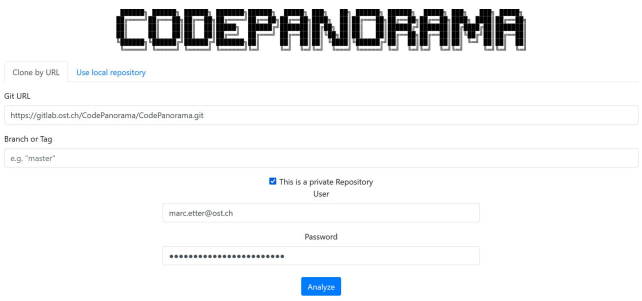

Figure 1: Starting page of *CodePanorama*

3.2 Filters

After *CodePanorama* has cloned the supplied repository and gathered some initial information, the user is presented with the filter view. Most software repositories contain many non-source-code files. Such files include documentation, licensing and copyright information, and binary files. These files are usually not interesting or not even possible to view in an image representation. Even amongst source code files, there might be uninteresting files. *CodePanorama* provides the user with various filter options to tailor the generated images to the user's needs. Using these filter options, the user can specify exactly which files of the repository the user would like to have included or excluded in the generated images.

In the same vein, *CodePanorama* allows for configuration of the size and zoom-level of the generated images. This configuration controls how many lines can fit into a single image. A preview bar shows whether the current selection fits into the configured panorama images. Only upon confirmation by the user are the actual images generated.

We noticed that coming up with an appropriate filter is a time-consuming process. For users that prefer a text-based approach, we offer an alternative filter method based on globs[4]. Another advantage of the text-based filter is that it can be persisted in the repository and will automatically be loaded by *CodePanorama*.

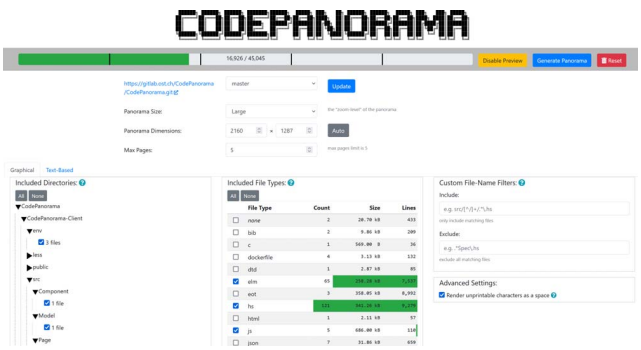

Figure 2: Filter page of *CodePanorama*

3.3 Overlays

CodePanorama can also enrich the images with color-coded information. Color overlays provide a quick and easy way to combine the code panorama with arbitrary information on a per-file or even per-line basis. The following overlays are currently supported:

Change Frequency Displays how often a file was changed relative to other files, as recorded by Git.

Participation Shows how often an author has contributed to each file.

Blame Shows the current author of each line.

Search Highlights all lines matching a given regular expression.

File Type Colors each file based on its extension.

Custom Users can provide a custom JSON-file to visualize arbitrary data. An example of this is providing test coverage data as an overlay to be visualized in *CodePanorama*.

The colors for most overlays can be specified by the user. This not only allows for a more customized visual presentation, but also improves usability for people with color vision deficiencies.

[2]https://www.haskell.org/
[3]https://elm-lang.org/
[4]https://en.wikipedia.org/wiki/Glob_(programming)

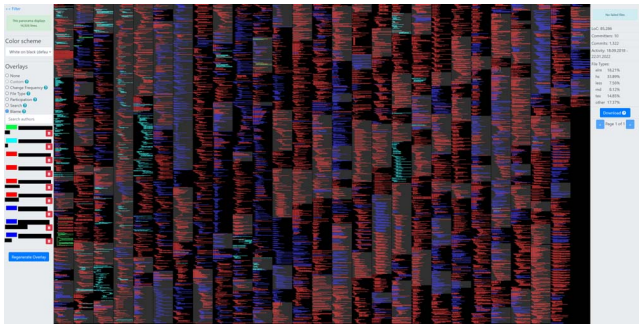

Figure 3: Code panorama of *CodePanorama* with blame overlay (~17k lines)

3.4 Drilldown View

When the user finds a particularly interesting section in the code panorama, a single click is sufficient to drill down into the code, displaying the file containing the selected section. Using this two-step approach, the user can easily switch between looking at the broad picture presented by the code panorama, and investigating the related source code. Allowing a user to view the code directly within *CodePanorama* reduces context-switching and helps maintain the user's focus.

CodePanorama-Server\core\src\Panorama\Panorama.hs

```
65      , colCounter = 0
66      , rowCounter = 0
67      }
68
69  assemblePanorama :: PanoramaConfig -> [FileLines] -> (CodePanorama, [(FilePath, String)])
70  assemblePanorama config lines =
71      let state = splitLinesIntoColumns config lines
72          panorama = assembledFiles state
73              |> sortOn fst3
74              |> groupBy ((==) `on` fst3)
75              |> map (\case
76                  groupedFiles@((pageNr, _, _):_) -> Just
77                      ( pageNr
78                      , map (\(_, file, (rects, panoramaLines)) ->
79                              (file, (rects, panoramaLines))
80                          ) groupedFiles
81                      )
82                  [] -> Nothing)
83              |> catMaybes
84      in
85          (panorama, failedFiles state)
```

Figure 4: Drilldown view of a source file in *CodePanorama*

4 POSSIBLE USE-CASES

While *CodePanorama* can be used for many different scenarios, we have identified some promising use-cases, which will be described in this section.

In our opinion, the main use-case for *CodePanorama* is getting a first intuitive impression of a new project. Using our tool, a developer can quickly visualize an overview of the code base and get a "gut-feeling" about the size, structure, and quality of that project. The tool was originally intended as an aid for an instructor to grade student projects. Instructors often have multiple student projects to grade. Having a tool to more quickly identify key locations of the code to assess, drastically cuts down on the work required.

We realized early on that the utility of this tool is not limited to an educational context. When taking over a new software project, developers face the tough decision to estimate the effort this new project will cause. To do this, they usually look at the project's source code. In this scenario, a developer can also use *CodePanorama* to make educated guesses as to where to start with an in-depth review. Any anomalies in the code's structure, or any of the overlaid metrics, can be a sign of something worth investigating.

Initial qualitative studies suggest that the intuitive judgment based on the usage of the tool has some correlation with common software metrics. In particular, technical debt and, to a lesser degree, complexity seem to be represented well in a code panorama. We believe that *CodePanorama* can be effectively used to complement existing code reviews and metrics in a typical software engineering project.

Lastly, through custom overlays, *CodePanorama* can visualize nearly any metric on top of the code. While some effort is required to generate such a custom overlay, this allows users to combine the code-map metaphor with information we could never have anticipated. Such a custom overlay is shown in figure 5.

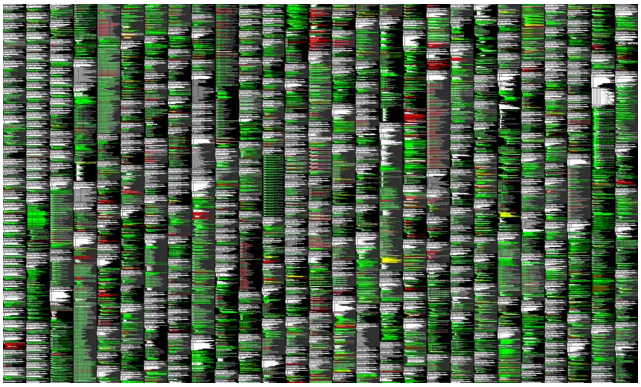

Figure 5: A custom overlay displaying code coverage as reported by JaCoCo

5 EXAMPLES

A handful of example code panoramas with interesting findings are shown in figures 6 through 8. We invite the reader to identify these findings before looking at their descriptions in the captions.

6 FURTHER IDEAS

Based on our current experiences with *CodePanorama* and early feedback from other users, we believe that *CodePanorama* has potential to become a standard tool in a developer's toolbox. Here are some ideas to further improve this tool:

Machine Learning Applying machine learning to analyze software code is a trending topic [2]. Since machine learning has produced exceptional results in image recognition [6], it could be interesting to apply machine learning techniques to the images generated by *CodePanorama*.

More Overlays While the generated code panorama itself is what visualizes the code, the overlays provide the necessary

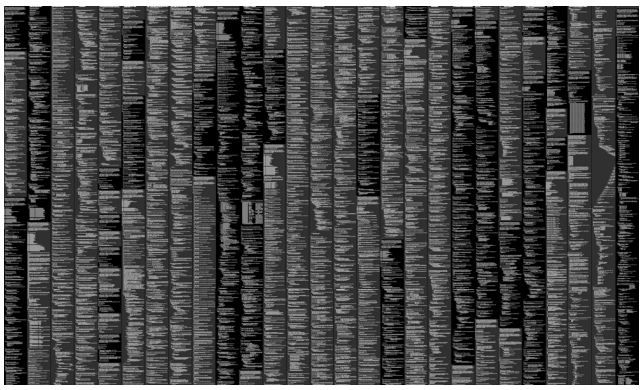

Figure 6: A code panorama containing unusually deep indentation (second column from the right)

Figure 7: A code panorama showing repetitive code structures (most noticable in the two right-most columns)

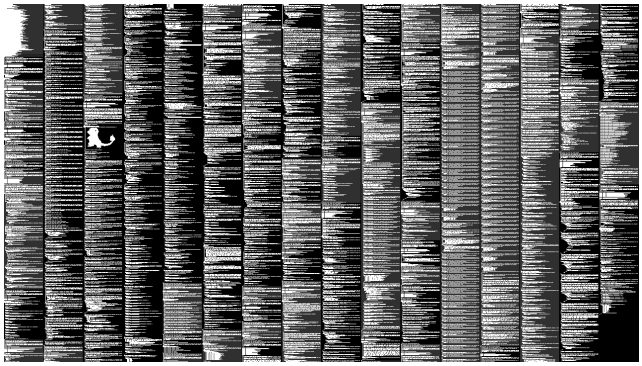

Figure 8: An ASCII-art Easter egg revealed by *CodePanorama* (third column from the left)

breadth of information. In our experience, having the ability to switch between multiple different overlays strongly improves the intuitive judgment about the code. Having more built-in overlays would certainly improve this even further, as implementing a custom overlay requires some effort.

Improved Drilldown View Currently, the drilldown view simply displays the selected text file with a best-effort on syntax highlighting. For very large files, however, it can be difficult finding whatever anomaly was spotted on the zoomed-out image. It would be helpful if the drilldown view automatically scrolled to the clicked position and displayed the same color overlays.

Improved Filtering For large projects, tailoring the panorama image to contain exactly what the user wants to see is vital, but time-consuming. Using more sophisticated configuration, or even machine learning, better defaults could be provided. That way, a user would have to spend less time refining the filters and has more time to analyze the code.

7 CONCLUSION

Code reviews and software metrics are the state-of-the-art techniques to assess software quality. However, we believe there is room for tooling support in between. *CodePanorama* eases the transition from metrics (represented as overlays) to the code, by presenting said code in a visual zoomed-out representation. With the use of *CodePanorama*, we believe that code reviews will become more efficient, as less time is spent scrolling through uninteresting code. As a next step, we plan to investigate to what extent this claim is valid.

CodePanorama started out as an aid to evaluate student projects and as a party trick. However, it ended up having wider applications in the area of code visualization and review.

We invite the software engineering community to start using *CodePanorama* and send us their comments and feedback to info@codepanorama.io.

REFERENCES

[1] Bacher, I., Mac Namee, B., and Kelleher, J. D. The code-map metaphor - a review of its use within software visualisations. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP, (VISIGRAPP 2017)* (2017), INSTICC, SciTePress, pp. 17–28.

[2] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

[3] Dicarlo, J., Zoccolan, D., and Rust, N. How does the brain solve visual object recognition? *Neuron 73* (02 2012), 415–34.

[4] Eick, S. C., Steffen, J. L., and Sumner, E. E. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering 18*, 11 (11 1992), 957–968.

[5] Fenton, N., and Bieman, J. *Software Metrics: A Rigorous and Practical Approach, Third Edition*, 3rd ed. CRC Press, Inc., USA, 2014.

[6] Pak, M., and Kim, S. A review of deep learning in image recognition. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)* (2017), pp. 1–3.