Pharo Git Thermite

A Visual Tool for Deciding to Weld a Pull Request

Ronie Salgado
Pleiad Lab, DCC, University of Chile

Alexandre Bergel
Pleiad Lab, DCC, University of Chile

Abstract

Collaborative software development platforms such as GitHub simplify the process of contributing into open source projects by the use of a pull request. The decision of accepting or rejecting a pull request has to be made by an integrator. Because reviewing a pull request can be time consuming, social factors are known to have an important effect on the acceptation of a pull request. This effect can be especially important for large and complicated pull request.

In this paper we present *Git Thermite*, a tool to assess the internal structure of a pull request and simplifying the job of the integrator. Git Thermite details the structural changes made on the source code. In Git Thermite we use a pull request business card visual metaphor for describing a pull request. In this business card, we present the pull request metadata and describe the modified files, and the structural changes in the modified source code.

ACM Reference Format:
Ronie Salgado and Alexandre Bergel. 2017. Pharo Git Thermite: A Visual Tool for Deciding to Weld a Pull Request. In *Proceedings of IWST '17*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3139903.3139916

1 Introduction

Git Workflow Git is a distributed version control system. The usual workflow for working with git consists on *cloning* an upstream version of a repository into a local working copy of the repository. Then the user performs some changes in his local copy which are *commited* into the cloned repository. After making some commits, the user can decide to *push* his commits into the original upstream repository.

GitHub Platform A popular collaborative platform for working on open source projects using git is GitHub. GitHub augments the capabilities of git by adding software maintenance tools such as an issue tracker and a wiki for each project. An important feature that expands the workflow of working with *git* repositories, which is added by GitHub and some other collaborative platforms, is the support for doing a *pull request*.

Pull Request Description A pull request is a formal request to the maintainers of a project to integrate some commits made by a contributor. Any developer registered in GitHub can perform a pull request to any open source project that is also present in GitHub. A pull request is made in GitHub by forking the main branch of the project where the contribution is going to be made. Once a developer has forked the project, he can make and push all of the commits that he wants into a branch of his forked version of the project. When the contributors has the commits on his personal branch in GitHub, he can generate a pull request to the main version of the project, by just clicking a button.

Once a pull request is made, it can be discussed before being accepted or rejected by an integrator. If more commits are made to the git branch that was used for creating the pull request before the pull request is closed, then these new commits are added automatically to the pull request.

Pull Request Acceptance Factors The acceptance or rejection of a pull request can be influenced by several factors. The factors of acceptance can be divided into two main categories: technical factors, and social factors [1, 2]. Technical factors are related to the changes themselves, meanwhile that social factors are related to the social relationships between the core developers of the project, the contributor and the users of the project. The following are some important pull request acceptance factors:

- Pull request size.
- Unit test inclusion.
- Number of files modified.
- Developer closeness with the core project developers.

Small pull requests that can be easily understood and reviewed tend to be integrated more easily. As for a larger pull request, the social factors can have an important impact on the acceptance of a pull request [1] [2].

Pull Request Visualization To increase the acceptance of pull requests, we propose using a visualization tool for assessing a pull request. Our visualization contains the following elements:

- Metadata gathered from GitHub pull request.
- Metrics from the files modified in the pull request.

IWST '17, September 4–8, 2017, Maribor, Slovenia

ACM ISBN 978-1-4503-5554-4/17/09…$15.00
https://doi.org/10.1145/3139903.3139916

- Static source code analysis of the changes.

For the static analysis of source code, we are basing our work on Torch [3][4]. Torch is a dashboard for software evolution analysis. The visualizations displayed by Torch compare arbitrary versions of Smalltalk packages that are stored in Monticello. From Torch we are taking some of its visualizations and concepts like having a *contextual diff*.

Unlike Torch, our tool can be used with arbitrary GitHub pull requests for repositories whose projects are in any language. At that current stage, we are able to comfortably analyze source code written in Pharo (*e.g.,* Figure 3, Figure 4) and Python (*e.g.,* Figure 1). As a consequence, our tool display source code structural changes for projects written in any of these two languages. For files that do not contain source code written on any of these two languages, we limit ourselves to standard text based diffs and metrics.

2 Description

Our interactive visualization (*e.g.,* Figure 1) for pull requests is designed around a business card metaphor. The idea of this metaphor is having a summary of a pull request in a compact visualization that displays the big picture of the changes present in the pull request. This business card shows the relevant entities that are changed in a pull request with additional information and metrics that can help an integrator. An integrator may be interested in metrics such as the size of the pull request, and whether the pull request contains unit tests or not [1].

Business Card Elements Our business card card is composed of a title bar (Figure 1, part A) followed by sections arranged in a vertical layout: below the pull-request title, the list of modified files is given (Figure 1, part B), and below the list of structural source code changes are given (Figure 1, part C). By highlighting an element with the mouse pointer, it is possible to obtain a tooltip that describes the highlighted element. Visual elements may be dragged with the mouse to rearrange them, and the elements can be clicked to get additional details.

Title Bar The title bar of the business card (Figure 1, part A) contains the title of the pull request, followed by the type of the pull request, and a colored square that indicates whether the pull request can be merged automatically or not. We try to detect the type of a pull request by looking at the presence of particular keywords in the title and in the comments of the pull request. If a pull request type cannot be inferred, no type is displayed. Because our simple heuristic is far from perfect, and we try to choose only one type, our heuristic can give a false positive or erroneous answers about the pull request type. The pull request types inferred with this heuristic are the followings:

- Bug fix (*Bug* keyword).
- New feature (*Feature* keyword).

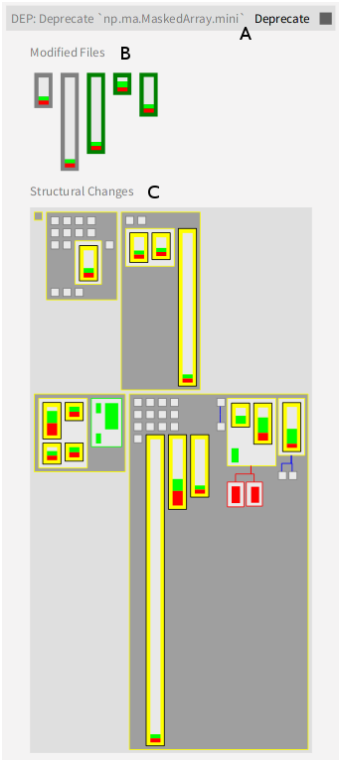

Figure 1. The business card of pull request number 8939 made to *numpy*, a Python project. In the first section (Modified Files), unlike Pharo, in Python we represent files that can have unit tests with a dark green color. This contains five files, three of them could have unit test because they have the *test* keyword on their name. In the second section (Structural Changes), the internal structure of the same files is represented. In this last section, the files contains classes and functions. The classes contains methods. Added elements are in green, removed elements are in red and modified elements are in yellow.

- Enhancement (*Enhance* keyword).
- Feature deprecation (*Deprecate* keyword). This can be seen in Figure 1.

The color of the square in the title bar indicates the difficulty to integrate the pull-request:

- *Green* – It is possible to merge automatically the pull request by clicking a button in GitHub. GitHub determines this by creating a temporary merge commit. We get this data via the GitHub API.
- *Orange* – It is not possible to merge automatically the pull request. Conflicts have to be solved manually.
- *Gray* – Automatic merge status is not reported by the GitHub API. This can be seen in Figure 1. The pull cannot be merged automatically. One of the potential reason for having gray status is looking at a pull request that is closed, which could be already merged.

Title Bar Interaction By highlighting with the mouse the square that indicates the possibility of doing an automatic merge it is possible to get a tooltip explaining the meaning of its color. By clicking on the title bar it is possible to see the commit tree of the commits involved in the pull request.

Commit Tree In the commit tree visualization (See Figure 2), we represent metrics about the commits that compose the pull request. In this visualization, the older commits are located at the top, the newer commits are below. The edges are joining a commit to its parent commit. A git merge commit can have more than one parent commit. Each element representing a commit contains in its interior a bar chart representing the number of lines added in green, and the number of lines removed in red. It is possible to interact with the commit meta-model by clicking on a commit. We obtain this visualization by retrieving the list of the commits that are composing a pull request by using the GitHub API.

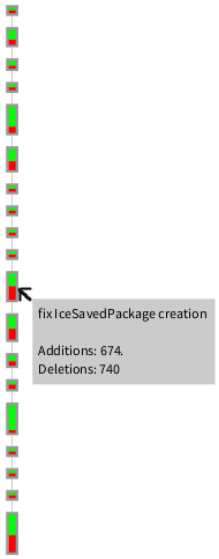

Figure 2. Commit Tree for pull request number 297 in *Iceberg*, a git front-end for Pharo. Each gray rectangles represents one commit that is part of the pull request. The chart in the interior of a commit represents the lines added (red), and the lines removed (green). Commits that are above are older than commits that are below. Edges are joining newer commits with their older parent commits.

Modified Files Panel The file panel of the business card (Figure 1, part B) contains the files that are changed in the pull request. The chart in the interior of the files represents the number of unchanged lines in gray, the number of added lines in green, and the number of removed lines in red. If the path of a file contains the word *test*, then its color is green to indicate the possibility of a file containing unit tests. In the case of Pharo source code, we filter the files that belongs to Smalltalk source code that is stored in a *filetree* package. We do this filtering because the *filetree* format stores each Smalltalk method in a different file, which introduces redundant information and lot of cluttering. We only present the changes to the Pharo source code in the pane with the structural changes, which provides an isomorphism between the files present in the *filetree* package.

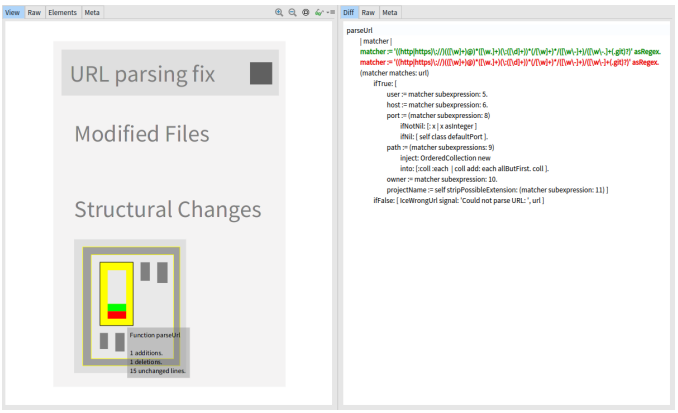

Figure 3. Structural changes for pull request number 363 in *Iceberg*, with only the modified classes. In this pull request a single method (yellow outline) was modified. By clicking on this method, we can see the textual diff of the method

Structural Changes Panel The structural changes panel (Figure 1, part C) displays a blueprint with the structural changes. We have the option of displaying in this view only the modified classes in the modified packages (See Figure 3), or displaying all the classes in the modified package (See Figure 4). A developer may be interested on watching all the classes to see how the modified classes are related to the unmodified classes in terms of inheritance. We can also filter the methods to display only the modified methods. The filtering of classes and methods is optional, and can be selected by the user before building the visualization.

Structure Changes Visual Encoding Elements that are completely added or removed are represented with a solid color, green for added elements, and red for removed elements. The size of these elements is proportional to the number of lines that were added or removed. The elements that are changed have a yellow outline, with a chart in its interior representing the number of changes made to the element. The bars in these charts represents the number of unchanged lines with a gray color, the number of added lines with green, and the removed lines in red. Inheritance relationships are represented using edges between classes whose color indicates whether an inheritance relationship was removed (red), added (green) or unchanged (blue).

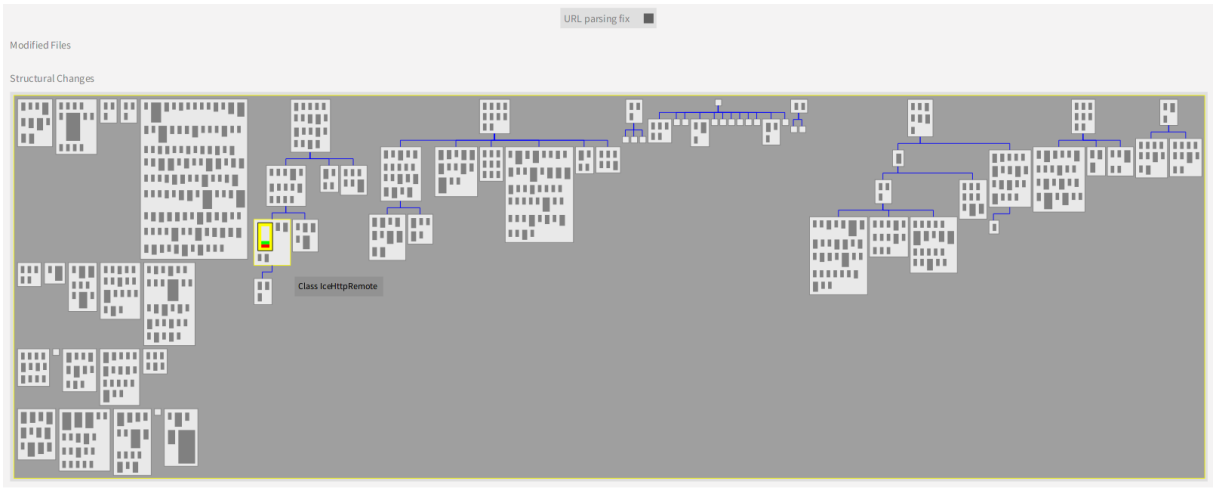

Figure 4. Structural changes for the same pull request presented in Figure 3, but with all the classes in the modified package.

Structure Interactions The elements of this blueprint can be dragged or moved. This interaction is useful when having a class hierarchy and the layout that is used by the visualization lay out elements in an overlapping position, or in a position where the subclass hierarchy gets hidden. By clicking on one of the elements on this panel, it is possible to see the textual diff of the node (See Figure 5)

Figure 5. Tooltip with metrics and diff

3 Usage Example

Changes Visibles on the Visualization Gomez *et al.* [3][4] describes a list of change scenarios that are identified and characterized by using Torch. On Figure 6 we can see pull request number 297 in *Iceberg*, a git front-end for Pharo. This is a large pull request where it is possible to characterize some of the same change scenario. These change scenarios are the following:

- *Adding features.* We see that many classes, full packages and class hierarchies are being added in the pull request.
- *Removing features.* We see that two classes are removed, which means that some features could have been removed. We do not see another class with a similar shape being added into a different package, so we discard the possibility that these classes were simply moved.
- *Refactoring methods into a new and common superclass.* In one part, we see that a superclass relationship is removed, and replaced by a new class. This new class is in the middle between the old subclass and its old superclass. We also see that some of its methods are removed, which means that they could be moved into the common super class.
- *Modifying methods.* We see many yellow elements, which means that an important number of methods have been modified.

4 Limitations

Unlike Torch, currently we are not able to detect and display whether an element is moved instead of simply being removed and added into a different place (*e.g.*, A class moved into a different package). This limitation prevents us from displaying some of the change scenarios that are displayed in Torch.

Another limitation is related to the very same nature of pull requests. Because pull requests are composed of multiple commits, for computing our structural diff we have to choose two commits in the pull request: the last commit in the pull request, and a base commit to compare with. We are choosing the parent commit of the first commit present in the pull request as a base commit. The problem with this is that along the lifetime of a pull request the main branch changes, and from time to time the code in the main branch can be merged back into the pull request. This can introduce lot of false positives, which probably can be seen in Figure 6. The apparent solution of comparing with the current version of the main branch can introduce other false positive related to the changes made in the main branch. We need to explore a way for reducing these false positives in our visualization.

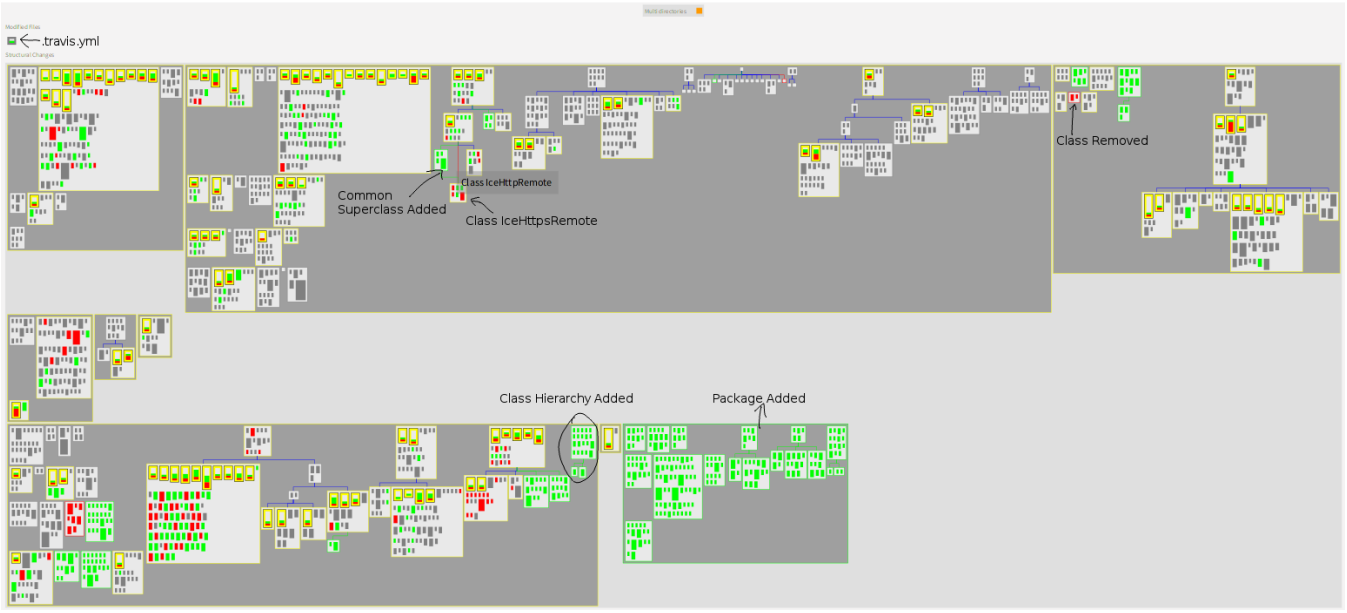

Figure 6. Structural diff for pull request number 297 in *Iceberg*. The non-Pharo file *.travis.yml* for continuous integration settings was modified in this pull request. We see that many elements were added (green), removed (red) and modified (yellow). In the low right part of the business card we see that a full package is added. We can see that some class hierarchies are added, and that two classes were removed. There is one part where a class is splitted in two classes, by adding a common intermediate class.

Torch does also have this problem related to history noise, because it can visualize the changes of multiples commits at once, which can have unrelated changes [4].

5 Related Work

Torch [3] [4] is a Dashboard for visually supporting code changes. Torch can visualize multiple arbitraries commits made in Pharo source code using Monticello. The Torch dashboard contains the following elements: a visualization for changes, metrics, a change lists, a symbolic cloud for visualizing a change in source code vocabulary, and omni-present contextual diff. For this work we are taking inspiration on Torch. Unlike Torch, this work is designed to work with Git and GitHub pull requests. From Torch we are reusing Ring [5] for doing the static analysis of Smalltalk code.

Motive [6] is a tool for visualizing software change sets for Java. The visualizations generated by Motive are based around Entity-Relationship diagrams and UML diagrams, for classes. The main difference between Motive and Thermite is that Motive is about visualizing high-level architectural changes, and Thermite is about visualizing source code AST structural changes.

6 Conclusions and Future Work

We want to visualize pull requests with code written in different languages. Currently the user has to select the language used in a pull request before building the visualization. Git repositories can have source code for more than one language. A typical use case is having scripts for continuous integration and deployment in addition to pure Pharo code.

Currently we are only focusing on visualizing the code present in pull request. A GitHub pull request can also have comments, and they can have some additional information that is valuable to the integrators. We need to add the pull request comments into our visualization, or add a way to interact with the pull request comments.

We need to perform a controlled experiment with a baseline tool to validate whether our visualization does actually assist the job of the integrator, and whether it can help on reducing the effect of the social factors on the integration of a pull request.

In this paper we presented Git Thermite, a visualization for GitHub pull request based on Torch. With Git Thermite we are now able to visualize pull requests made in a git based workflow. In this on going work, we still need to validate whether this visualization is actually useful and can help the integrators on their daily job.

References

[1] J. Tsay, L. Dabbish, J. Herbsleb, Influence of social and technical factors for evaluating contribution in github, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, ACM, New York, NY, USA, 2014, pp. 356–366. doi:10.1145/2568225.2568315. URL http://doi.acm.org/10.1145/2568225.2568315

[2] D. M. Soares, M. L. de Lima Júnior, L. Murta, A. Plastino, Acceptance factors of pull requests in open-source projects, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15, ACM, New York, NY, USA, 2015, pp. 1541–1546. doi:10.1145/2695664.2695856.
URL http://doi.acm.org/10.1145/2695664.2695856

[3] V. U. Gomez, S. Ducasse, T. D'hondt, Visually supporting source code changes integration: the torch dashboard, in: Reverse Engineering (WCRE), 2010 17th Working Conference on, IEEE, 2010, pp. 55–64.

[4] V. U. Gómez, S. Ducasse, T. DâĂŹHondt, Visually characterizing source code changes, Science of Computer Programming 98 (2015) 376–393.

[5] V. U. Gómez, S. Ducasse, T. D'Hondt, Ring: a unifying meta-model and infrastructure for smalltalk source code analysis tools, Computer Languages, Systems & Structures 38 (1) (2012) 44–60.

[6] A. McNair, D. M. German, J. Weber-Jahnke, Visualizing software architecture evolution using change-sets, in: Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on, IEEE, 2007, pp. 130–139.