https://doi.org/10.2352/ISSN.2470-1173.2020.1.VDA-387
© 2020, Society for Imaging Science and Technology

A Visualization Tool for Analyzing the Suitability of Software Libraries via Their Code Repositories

Casey Haber; Chartio; San Francisco, CA, USA; haber.casey@gmail.com
Robert Gove; Two Six Labs; Arlington, VA, USA; robert.gove@twosixlabs.com

Abstract

Code repositories are a common way to archive software source code files. Understanding code repository content and history is important but can be difficult due to the complexity of code repositories. Most available tools are designed for users who are actively maintaining a code repository. In contrast, external developers need to assess the suitability of using a software library, e.g. whether its code repository has a healthy level of maintenance, and how much risk the external developers face if they depend on that code in their own project. In this paper, we identify six risks associated with using a software library, we derive seven requirements for tools to assess these risks, and we contribute two dashboard designs derived from these requirements. The first dashboard is designed to assess a software library's usage suitability via its code repository, and the second dashboard visually compares usage suitability information about multiple software libraries' code repositories. Using four popular libraries' code repositories, we show that these dashboards are effective for understanding and comparing key aspects of software library usage suitability. We further compare our dashboard to a typical code repository user interface and show that our dashboard is more succinct and requires less work.

Introduction

Software developers often need to assess software libraries to find one that is suitable to use in their projects. For example, a team of software developers may look for a software library to perform statistical calculations in their urban planning tool. This team of developers could be termed *external developers*, in contrast with *internal developers* who are the ones developing the statistical software library. Many tools help internal developers understand the content and history of their own code repositories; however, we believe that external developers' needs largely have not been met.

A primary concern for external developers is whether a project is dependable and well-maintained and will continue to be so. They must assess software libraries, packages, and tools to determine which ones are suitable to be used as dependencies in their own projects. For example, a developer writing a project in Python may need to add a machine learning component, and therefore needs to identify a Python machine learning library that will be maintained for as long as required. Some of the data to analyze a software library's usage suitability exists in its code repository—a common way to archive software source code files and the changes made to them over time—but it is difficult to understand because the data is disaggregrated and high-dimensional. To illustrate that software usage suitability analysis is not well supported by existing tools, we present related work in the Background section and an example usage suitability analysis using an existing tool in the Motivating Example section.

In this paper, we identify six risks associated with using a software library, and we derive seven user requirements for tools to analyze software library usage suitability. Although these risks and requirements are not exhaustive for completely understanding usage suitability, they are important and we believe they are not adequately met by current tools. We contribute two dashboard designs based on these requirements: (1) a usage suitability analysis dashboard developed as an extension for the Chrome web browser that displays inside GitHub[1], and (2) a standalone tool for comparing usage suitability across multiple software libraries' code repositories. Furthermore, we propose to simplify the process of software library usage suitability analysis by automatically processing the code repository's commit and issues data. Then our dashboards, which are directly integrated into the code repository user interface, display this information for the user.

In two example usage scenarios we demonstrate that our dashboards can show detailed information about key indicators of usage suitability. The first example uses our dashboard with the code repository for Theano, a well-known machine learning library. Theano was recently deprecated and serves as validation of our analysis, which clearly shows that development has essentially stopped. We compare this analysis to analysis conducted using only GitHub's code repository user interface and show that using our dashboard we are able to develop richer insights more easily. The second example uses the usage suitability comparison dashboard to identify the least risky software libraries out of several related libraries. These usage scenarios demonstrate that the dashboards are effective for understanding important aspects of software library usage suitability.

Background

In version control systems, a *code repository* is an archive of source code files and directories with metadata for the purpose of tracking changes and coordinating work. Git is a common example of a version control system, and GitHub is a popular host for Git code repositories. For our research prototype, we use Git and GitHub due to the popularity of the latter within the open source community, but we acknowledge that a widely available tool would need to support more platforms. Git tracks commits, in which files, directories, and content can be added or removed. Git also tracks which contributor authored a commit, and therefore who added or removed files and content. In addition to Git's functionality, GitHub also has functionality to open or close issues for a repository. Each issue notes a bug, enhancement, idea,

[1] `https://github.com`

or task related to the code in the repository.

SonarQube[2] is a tool for analyzing source code quality, code repository branches, and code repository pull requests. Plugins for SonarQube add additional functionality, such as alternate analysis engines [6] or external analysis tools [7]. Other static analysis tools such as FindBugs [1] or linters [12] identify a variety of source code problems. Although these types of analyses are useful, we argue they do not fully address the problem at hand. First, they may not provide much insight into the maintenance and activity of a project, which are important considerations for development teams who are considering using the code as a library in their own project. Second, these tools typically do not provide an overview, but instead show a list or summary of identified problems. In contrast, our goal is to analyze the source code's repository to gain additional insight into characteristics such as maintenance and activity, and to present this analysis in an overview dashboard visualization.

Past research indicates that a repository's data can reveal important information about its status and longevity [2, 5, 10, 14]. There are many papers on visually analyzing contributor networks [9, 11] and code dependencies [15–17] within code repositories. Storey et al. [13] survey visualization tools to analyze code repositories. These tools are typically designed to support tasks related to code exploration, project management, version control, module management, artifact comparison, architectural comparison, team coordination, project release exploration, and code evolution analysis. As summarized by Storey et al., the intended users tend to be those directly involved in creating and maintaining the code repository, i.e. *people internal to the project*. These are people such as developers, managers, and testers. As a consequence, the tools are designed to support the tasks and requirements of those users, such as code comparison, code evolution analysis, and team coordination. These tools are not designed to help external developers decide whether the code is suitable to use in their own project.

In contrast, *external developers* are those who are not directly involved with creating or maintaining a code repository. These are people who have different tasks and requirements than internal developers. For example, external developers would be considering using the software library in their own project. In that case, they would wish to analyze the risks associated with adding that software library to their own project, such as if the level of activity of the code contributors is too low, if the library's repository is not well maintained, or if there is a single point of failure among the contributors. In this paper we call this *usage suitability analysis*. External developers will often have additional questions about code quality and the quantity of documentation, but these questions are relatively easy to answer using existing tools like those described above. However, we are not aware of tools developed specifically to answer software library usage suitability questions such as those relating to maintenance and contributor dependability.

GitHub[3] provides a few visualizations to support tasks related to understanding usage suitability. For example, GitHub has a chart showing the number of commits to the code repository per day and additional charts showing the number of commits per day for each contributor. Although this can give an indication of overall activity on the project, the large number of charts on many projects can make it difficult to understand the distribution of work, the contributors' responsiveness to issues, and which contributors are responsible for different semantic elements of the project (e.g. the database or UI code). Furthermore, there is no built-in mechanism for comparing multiple repositories.

Motivating Example: Analyzing Software Library Usage Suitability Using GitHub

This section describes a real scenario that helps illustrate the need for specialized dashboards to analyze software library usage suitability. In this scenario, we wish to analyze the usage suitability of Theano[4], a well known Python machine learning library. Although we would be concerned with the quality of documentation and the number of available examples, judging these characteristics is outside the scope of this paper, and we believe those tasks are relatively well supported by existing tools. In this example, we focus on analyzing the usage suitability of Theano with respect to risks related to maintenance and contributor dependability. Technology suitability is also important and will be discussed, but this can vary depending on the individuals and teams who are considering adopting the software library.

First, we would like to understand whether Theano's developers actively contribute to the project. On Theano's GitHub homepage (Fig. 1) we see the most recent commit was pushed to the repository 14 days from the time of writing (i.e. September 13, 2019). After clicking on the commit to see its details, we see that this was actually a pull request and that the changes in the pull request were pushed 20 days ago. We also see a list of the repository's files and directories. The list of files and directories is too long to be visible all at once. After scrolling through them, we found that most files were last changed one year ago or more, two files in the root directory were changed in the past eight months, and three directories have files that were changed in the past 8 months. Clicking to see the list of commits, we count only 10 commits that were made in the last four months, and that the commits were made from three developers. It is difficult to see the trend in commit rates from the list, so we click the Insights button to see the overall commit activity along with the commit activity from each contributor (Fig. 2). We see that the activity ranged from 2008 until the present (2019), but there was a sharp decline in overall commits near the end of 2017. GitHub orders the list of contributors by their number of commits. Scrolling through the list of 100 contributors, we see that some of the top contributors continue to push a small number of commits after 2017 (such as the users nouiz and abergeron), but many other contributors completely stopped pushing commits before 2018, and in some cases considerably earlier (e.g. the user goodfeli had little activity since 2012).

We might additionally wish to know whether the project relies on a small number of contributors to do most of the work or maintain a critical component. Since it appears that the developers are no longer actively changing the code, this is a moot question. However, if we were to try to answer the question of whether the project relies on a small number of contributors, we would need to filter the list of contributors to show only recently active con-

[2] https://www.sonarqube.org/

[3] https://github.com/

[4] https://github.com/Theano/Theano

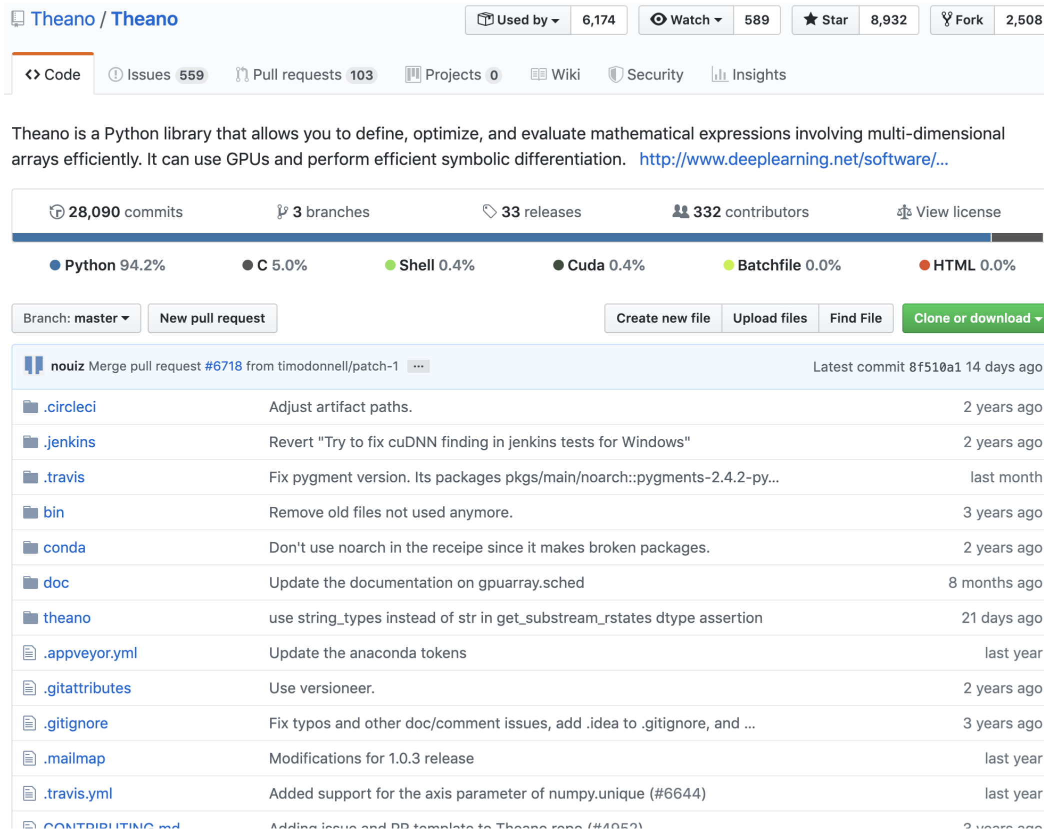


Figure 1. *The Theano software library's GitHub repository homepage.*

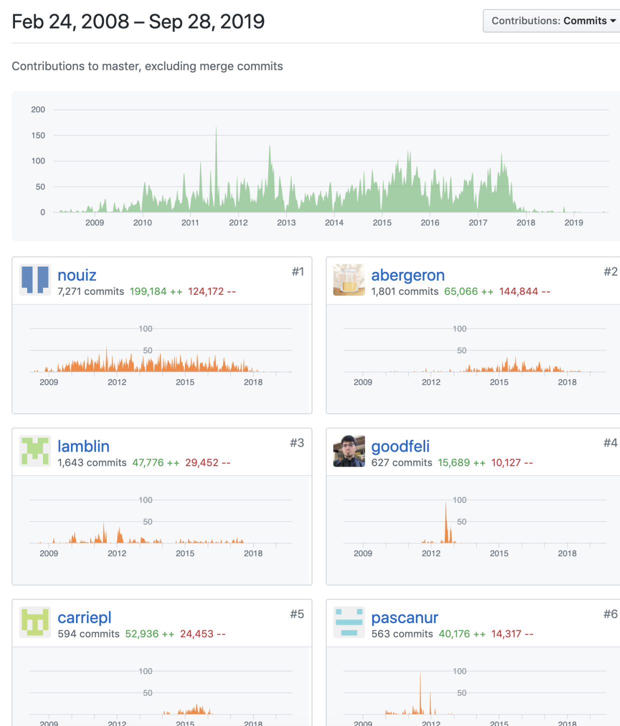


Figure 2. *The GitHub page showing Theano's contributors.*

tributors and then manually compare the number of their recent commits to estimate how many contributors are doing most of the work. Determining whether a small number of contributors maintain a critical component is even more labor intensive: we would need to click on each contributor in the list to see their commits, and then examine each commit individually to try to see which technologies and components those contributors are working on.

We would also like to know whether developers are still responding to issues from the community even if they are no longer actively changing code. To analyze this, we open the Issues page. We see that there are 559 open issues and 2,066 closed issues. Manually counting, we see that 26 open issues were opened this year, and after clicking to see the list of closed issues we count eight issues that were closed in 2019. It is difficult to understand the long-term trend, but it appears that issues are still being addressed, although more issues are being opened than being closed. Assessing the long-term trend would require additional manual effort to page through the list of open and closed issues.

Finally, to understand which technologies are used by the project, GitHub provides a breakdown of programming languages used in the project. These languages are inferred from file names (e.g. `__init__.py` is a Python file). This means GitHub provides some information about programming languages used, but not much additional information about technologies used (e.g. whether the Python code connects to a database or makes statistical calculations). Theano's code is primarily Python (94.2%), with some code in C (5.0%), shell (0.4%), CUDA (0.4%), and a small amount of code in other languages.

By searching the open web we find that in September 2017 it was publicly stated that Theano is deprecated and new development will stop, but that minimal maintenance will continue for some time.[5] The timing of this announcement comes shortly after the large spike in commit activity near the end of 2017, which is right before the last major release of Theano in November 2017. This is congruent with our analysis and corroborates our findings that this project is not actively maintained.

We would also like to compare Theano to other Python machine learning libraries to see if they have less risk than Theano. Making this comparison would largely require the same work we performed to analyze Theano's usage suitability, but for each library we wanted to compare to Theano. Doing this work would require additional cognitive load to remember and compare all of the acquired knowledge of each library [4], or we would need to take notes using some external tool such as a notepad.

Design Requirements

From our own experience and conversations with five developer and manager collaborators (5–15 years of software engineering experience, primarily working in small teams), we found that external developers and managers have an unmet need for understanding the usage suitability of a software library and comparing the usage suitability of multiple software libraries. Some tools already support aspects of software library usage suitability analysis, such as showing the number of commits over time or the programming languages used by the software. However, we found that several other aspects are not well supported. Specifically, we found that assessing the following risks of using a software library are not well supported by currently available tools:

1. **Single Point of Failure.** A software library may have a small number of primary contributors for the project, or a small number of primary contributors for important functionality. This can result in a single point of failure where the project or the important functionality is at risk of not being well maintained if those contributors leave the project.
2. **Unresponsive to Community.** If the software library developers are unresponsive to feature requests or bug reports from the community, then the software library users may have to perform extra work to patch the library for their own needs.
3. **Low Ratio of Contributors to Project Size.** Large projects with a small number of contributors may be more likely to have components or functionality that is not being actively maintained, or the software library developers may be less responsive to feature requests or bug reports from community members.
4. **Inactive Developers.** Projects with inactive developers are less likely to be maintained, resulting in less responsiveness to community requests and fewer releases of new software library versions.
5. **Project Decline.** Due to technological advancement and changes in the software library developers (such as a new job or changing interests), older projects are more likely to become outdated, deprecated, or inactive.
6. **Unsuitable Technology.** The software library could use a technology, such as a programming language or database, that is unfamiliar to the external developers who wish to use the software library. This can increase the amount of effort to use the software library and increase the risk that the external developers will not be able to use the software library to its full extent. Alternatively, the software library could use another library or technology that is at risk of becoming outdated, deprecated, or inactive, thereby posing a risk to the software library under consideration.

Although assessing some of these risks is partially supported by existing tools, we believe current support is inadequate. For example, as discussed in Motivating Example section, GitHub will show some of the programming languages used in a project, but this is primarily determined by file extensions. This approach limits its accuracy—for example, `.html` files can contain a mixture of HTML, CSS, and JavaScript code. Furthermore, the programming language itself might not reveal fine grained information about the technology used—for example, JavaScript code can be frontend website code, backend website code, even code completely unrelated to websites.

We propose designing dashboards to help external developers assess these risks when analyzing the usage suitability of software libraries. To do so, we translate these risks into the following set of requirements that our dashboards should support:

1. **Requirement:** Survey Code Change Quantity.
   Description: View the amount of code changes over time from the top authors to understand the current level of productivity, how productivity has changed over time, whether primary contributors have become inactive, or whether the project is old and could become outdated, deprecated, or inactive because of age.
   Risks addressed: Inactive Developers, Project Decline
2. **Requirement:** Changes in Code Content.
   Description: Understand the changes in code content over time to determine whether important parts of the repository are still maintained.
   Risks addressed: Project Decline
3. **Requirement:** Required Skills.
   Description: Assess the technology used in the project to determine whether the potential adopters have the required skills to use the repository's code.
   Risks addressed: Unsuitable Technology
4. **Requirement:** Age of Project.
   Description: Assess the age of the project to understand the risk that development and support may end while the adopter still needs the repository's code.
   Risks addressed: Project Decline
5. **Requirement:** Issues Backlog.
   Description: Understand the issues backlog to get a high-level understanding of the activity level of the contributors and whether the repository contributors are responsive to bug reports or feature requests from the community.
   Risks addressed: Unresponsive to Community, Inactive Developers, Project Decline
6. **Requirement:** Bus Factor.
   Description: Assess the number of core developers over

[5] `https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ`

time to determine the number of active core contributors, whether there is a single point of failure among contributors, and whether primary development has slowed or ended.
Risks addressed: Single Point of Failure, Inactive Developers, Project Decline

7. **Requirement:** Activity Relationships.
Description: View the number of core contributors and the types of contributor activity to gauge whether there is a small number of core contributors relative to the size of the project.
Risks addressed: Single Point of Failure, Low Ratio of Contributors to Project Size

We further identify two tasks performed by external developers who are looking for suitable software libraries to use:

1. **Analyze an Individual Software Library.** After external developers identify a potential software library, they need to assess it for the risks described above.
2. **Compare Multiple Software Libraries.** In some cases, there are multiple potential software libraries that external developers can use. In this case, the external developers need to be able to compare multiple software libraries simultaneously to assess the relative risks of each.

System Design

We designed a visualization system to support the two use cases described above: gain insight into the usage suitability risks of a single software library, and to compare multiple software libraries to assess their relative risks. We considered these as separate use cases, and therefore we designed one dashboard for each task. These dashboards show semantic tags for code, commits over time, total vs. closed issues, and changes in the core contributors over time.

We implemented these dashboards in a client-server web application. To support the dashboards, the web application has three main components: a data pipeline that ingests, processes, and stores data in the database for each repository; a backend server that interfaces with the database, processes the data further, and sends data to the frontend; and two frontend user interfaces that visualize the code repository data. All data is stored in a Postgres database with tables to store data for the tags, files, commits, and issues. The following sections describe each component in more detail.

Data Pipeline

The data pipeline fetches data from the given code repository's master branch using git commands, and the data pipeline fetches the GitHub issues using the GitHub API (Application Programming Interface).

To support requirements 2 and 3 (Changes in Code Content and Required Skills), the data pipeline uses Gelman *et al.*'s system [8] to generate a set of tags for each code file. These are semantic tags learned from Stack Overflow, such as `c++`, `multithreading`, or `machine-learning`. This system only generates tags on a per-file basis. Because our tool focuses on project-level analysis, we aggregate tags at the project level. To do this we remove tags if the tag is not assigned to at least 5% of all code files in the project.

Backend Server

The backend is a REST server run in Node.js. Upon receiving a request from a client for a repository's data, the server queries the database, processes the data, and then returns the resulting data to the client. The data processing step performs two primary operations. The first is to aggregate the commit data by author, month, and commit identifier. The second is to calculate the bus factor, which measures the risk to the project in terms of the number of core contributors. Bus factor does not have a clear mathematical definition, so this system uses the Augmented Pony Factor[6], which is a related concept that finds the fewest committers doing at least half of the work in a given month. It is defined as $\sum_{n=1}^{P} C_n \geq V/2$ where P is the pony factor (which we call the bus factor in this paper), C_n is the number of commits from committer n in the given month (sorted decreasing by number of commits), and V is the total number of commits in the given month.

The repository's issues data is passed directly to the frontend to calculate running totals of the total number of issues and the number of closed issues.

Dashboards

The frontend is displayed as one of two dashboards. The first is a dashboard showing usage suitability metrics of a single code repository (Fig. 3), which addresses the Analyze an Individual Software Library task. The second is a repository comparison dashboard comparing multiple code repositories (Fig. 4), which addresses the Compare Multiple Software Libraries task. The components of each dashboard are built with React[7] and D3 [3].

The *usage suitability summary dashboard* is built as an extension for the Chrome web browser. When users visit the webpage for a GitHub repository, the extension checks if the repository has already been processed and stored in the Postgres database. If it has, the extension then requests the repository's data from the backend, as described in the Backend Server section. Finally, the extension draws the data in the visualization components, and the components are inserted in the GitHub webpage underneath the project statistics (see Fig. 3). We designed this to address Requirement 7 (Activity Relationships) by making these visualizations all visible at the same time to aid comparison of different types of activities. The dashboard has four main components.

First, the Tags component (Fig. 3A) is a visual representation of the tags generated for the repository's files. It presents the most common tags as well as the number of code files per tag. This provides insight to two major questions: "What is in this repository?" and "What technologies are used in this project?" Additionally, the tags inform users of the number of code files with different content in this project. This addresses Requirement 3 (Required Skills). This component serves as context and background knowledge for the other components and it also serves as a legend for the Total Commits area plot.

Second, the Total Commits component (Fig. 3B) provides useful information about the number of commits and the commit content over time. This component shows two area charts of monthly commits, where each layer is colored by tag, showing

[6]`https://ke4qqq.wordpress.com/2015/02/08/pony-factor-math/`
[7]`https://reactjs.org/`

Theano / Theano
Used by 5,876
Watch 590
Star 8,857
Fork 2,495
Code
Issues 555
Pull requests 102
Projects 0
Wiki
Security
Insights

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It can use GPUs and perform efficient symbolic differentiation. http://www.deeplearning.net/software/...

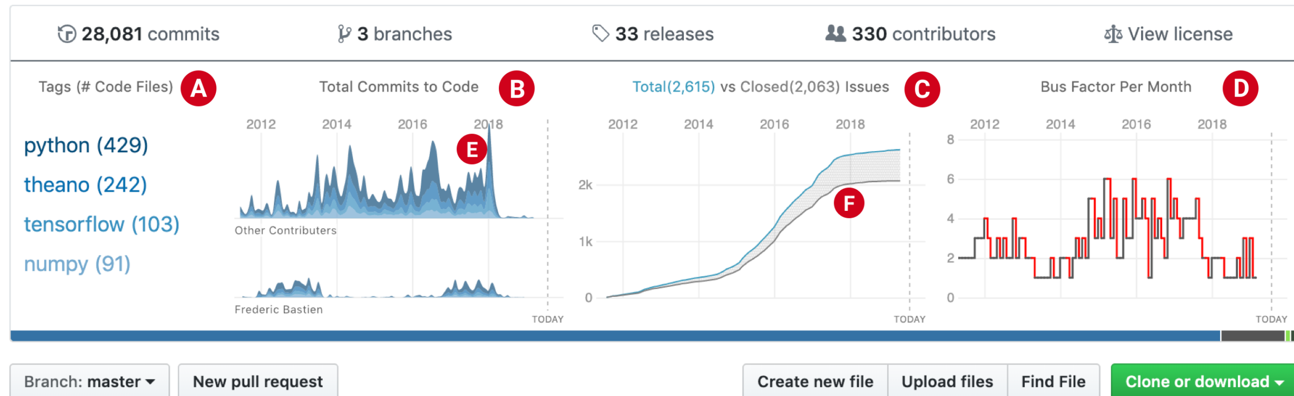


Figure 3. *The software library usage suitability dashboard is inserted into the GitHub UI for the given repository, which makes the information immediately accessible to users viewing the repository. The dashboard contains: (A) a list of tags indicating the most common automatically identified content, (B) an area chart showing commits over time for the top contributor and one for all other contributors broken down by tag, (C) a burn down chart showing the total number of issues and the number of closed issues over time, and (D) the monthly bus factor. Components include tooltips (not shown) to provide additional context and definitions (e.g. explaining how to read the issues burn down chart, or explaining what "bus factor" means).*

how the content of commits has changed over time. This addresses Requirements 1, 2, and 4 (Survey Code Change Quantity, Changes in Code Content, and Age of Project). One of the two area charts is the contributor with the most total commits. The second area chart is an aggregation of all other contributors combined. This shows the relationship between the most prolific contributor, who is often the lead developer, and the rest of the contributors. Additionally, the relationship between top and other contributors can indicate how much growth the project has taken on since the first commit. Specifically, if the most prolific contributor is committing much less than the other contributors combined, this indicates the project has exited an initial stage of development and now has a healthy number of contributors. On the other hand, if the top contributor is the only one working on an important section of the code (e.g. the database), then the project would be at risk if the top contributor left the project.

Third, the Total vs. Closed Issues component (Fig. 3C) is a line chart that shows the number of total issues and closed issues at monthly intervals. The cross-hatching between the lines is the focus of this visualization. This indicates whether or not a project's maintainers are keeping up with development and community requests. This is designed to address Requirement 4 (Issues Backlog).

Fourth, the Bus Factor component (Fig. 3D) is presented as a line chart showing the number of contributors responsible for at least half of the commits on monthly intervals. This is calculated using the Bus Factor as described above. Any drops in the bus factor from the previous month are represented in red while a continuing or increasing factor is in gray. This is designed to satisfy Requirement 5 (Bus Factor). While the Total Commits chart indicates the total amount of work performed, as measured by the number of commits, the Bus Factor chart shows us how concentrated this work is among the contributors. An increasing bus factor indicates the project is gaining core developers; declines tend to indicate the project is losing core developers. A bus factor with many dips and spikes may indicate many irregular contributors; a smooth line may indicate more consistency among the top contributors.

The *usage suitability comparison dashboard* (Fig. 4) is a standalone web page and does not require any browser extensions. It is configurable to show usage suitability summary dashboards for selected groups of repositories. This is designed to compare a group of similar or related repositories, such as Python machine-learning libraries, to compare their content and contributor activity. This enables users to answer questions such as which repository has more active developers, which repository's developers are most responsive to issues, and which repository is most sensitive to core developers leaving the project.

We developed these dashboards using an iterative process where we gathered informal usability feedback from four programmer collaborators over several months. We incorporated their feedback to improve the usability of the dashboard designs.

Example Usage Scenarios

To demonstrate the ability of these dashboards to gain helpful insight into the usage suitability of software libraries, we present two example analyses. The first demonstrates software library usage suitability analysis of a single library, and the second demonstrates using the usage suitability comparison dashboard to compare the health of four libraries.

Analyzing the Usage Suitability of a Software Library

To demonstrate analysis using the usage suitability summary dashboard, we select Theano, a popular machine learning library for Python (see Fig. 3).

The Total Commits component shows an increasing number of commits up until the end of 2017, when there is a sharp decrease in commits by all the other contributors. Importantly, the number of commits is at its highest rate in the ending stage of this project (Fig. 3E). The top author comes back to the project after a long period of low development. Therefore, it appears that development has stopped, but there was a strong effort to leave this project in a finished state.

The Total vs. Closed Issues component indicates the project's lack of ongoing development. Changes in the number of total issues and closed issues rapidly slow down in late 2017, and we see a large gap in the number of total issues and closed issues (the cross-hatched portion Fig. 3F). This occurs around the final stage of development seen in the Total Commits graph. This visualization also shows the long term growth of this project. The number of total issues and closed issues increases, but the rate sharply declines at the same time as commits decline.

The Bus Factor component highlights a weakness of this project. The other visualizations indicate that this is a large project with a history of growth, but the bus factor was around two from 2008 to 2015. That is a span of seven years where there were only two main contributors. That is not enough core developers for any sizable project, especially one that was intended to be used in production scenarios by companies. Even during its peak levels of development, the bus factor was only around five. A bus factor of five could be considered decent in smaller scale projects but is not a good sign on a project that has 2,484 forks and over 28,000 commits. Since late 2017 the bus factor dropped and oscillated between one and three, indicating a decline in core developers that corresponds with the decline in commit and issues activity.

Our analysis indicates that Theano is not in a healthy state: changes to the code have almost stopped, few issues are being opened or closed, and the number of primary developers is very small.

Comparison to the GitHub Example Analysis

Comparing the analysis using our dashboard to the analysis using the GitHub UI in the Motivating Example section, we are able not only to develop the same insights, but also additional insights, and with less work.

The code tags give us insight into the technology of the project. The "python" tag indicates that a lot of files are in the Python language, which mirrors what we see in GitHub's programming language details. The "theano" tag is self evident, but the "tensorflow" tag reflects that Theano's API is similar to Tensorflow's. Similarly, the "numpy" tag indicates that NumPy is a technology often used with Theano. This are insights we could not see from GitHub's UI alone.

The dashboard's Total Commits to Code component shows similar information as GitHub's list of contributors, but it shows more fine-grained information about the number of commits related to each of the four tags. Furthermore, this information is shown on the repository's homepage, making the information faster and easier to access than the list of contributors that is on a separate page.

The Total vs. Closed Issues component also provides more nuance than we saw using GitHub's UI. In GitHub, we had to navigate from the repository's homepage to the Issues page, and from there we had to manually read the dates of each issue in the list to see how many issues were opened and closed this year. However, in the Total vs. Closed Issues component in our dashboard, we were able to see more nuanced information about the rate of change in the number of total vs. closed issues as well as the gap between them–all without leaving the repository's homepage.

Finally, the Bus Factor component enabled us to much more easily understand whether the project relies on a small number of developers. With GitHub's UI, we had to scroll through the long list of 100 contributors to try to see which are active during any given period of time. But with the Bus Factor component we see right away that the number of core developers decreased substantially in late 2017, and has frequently dipped down to one since then.

Having all of these components visible together also allows us to visually correlate data we see in one component with data we see in another.

Comparison to Ground Truth

By searching the open web we find that in September 2017 it was publicly stated that Theano is deprecated and new development will stop.[8] The timing of this announcement comes shortly before the large spike in commit activity near the end of 2017, which corresponds with the last major release of Theano. This announcement also comes around the time we see a sharp decline in the bus factor and the number of new issues. Therefore our analysis reflects the reality that this project is deprecated and not actively maintained, and in fact with our dashboard it is possible to predict this outcome before the announcement because some signs (e.g. the low bus factor and the widening gap between total and closed issues) are visible in the dashboard before the announcement was made.

It has been publicly stated that Theano is deprecated and new development will stop, but even without the public announcement it is easy to see that this is the case.

Using only GitHub's default UI and visualizations we see some of this trend, but with less nuance and less predictive signal. With GitHub's default UI and visualization it was possible to see that commit activity had sharply declined around the time of Theano's deprecation announcement. However, GitHub lacks an ability to see trends in created and closed issues, so we are not able to see that the number of new issues has declined and that an increasing number of them remain open. This weakens our ability to see the reduction in overall project activity by making it nearly impossible to see this trend.

Comparing the Usage Suitability of Machine Learning Libraries

For the usage suitability comparison dashboard, consider a scenario where we need to identify a Python machine learning library that is actively maintained and shows signs of longevity. We

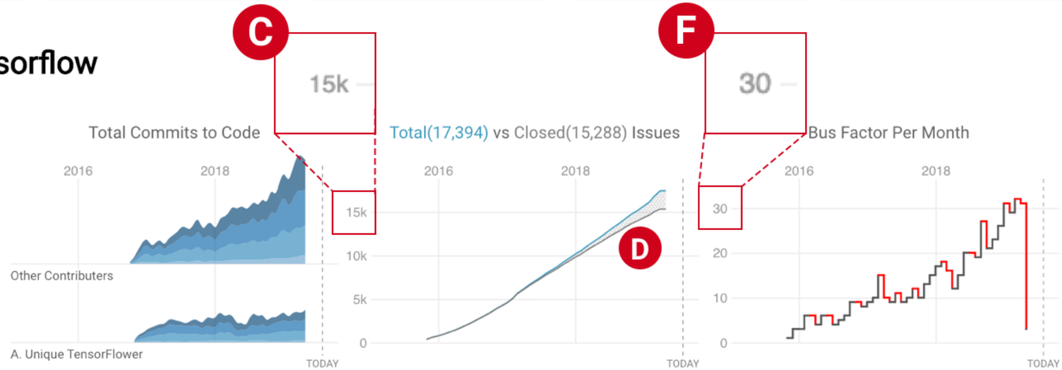
[8] `https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ`

Python Statistical Packages | JS Frontend Frameworks | Python Machine Learning Libraries | Python Visualization Libraries | JS Machine Learning Libraries

tensorflow/tensorflow

Tags (# Code Files)

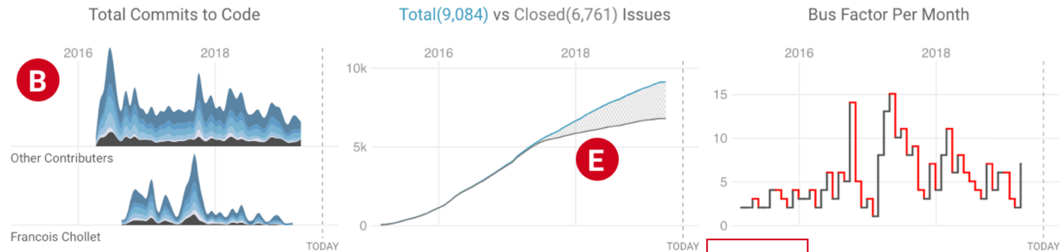
c++ (6355)
tensorflow (4736)
python (3574)
cuda (1646)



keras-team/keras

Tags (# Code Files)

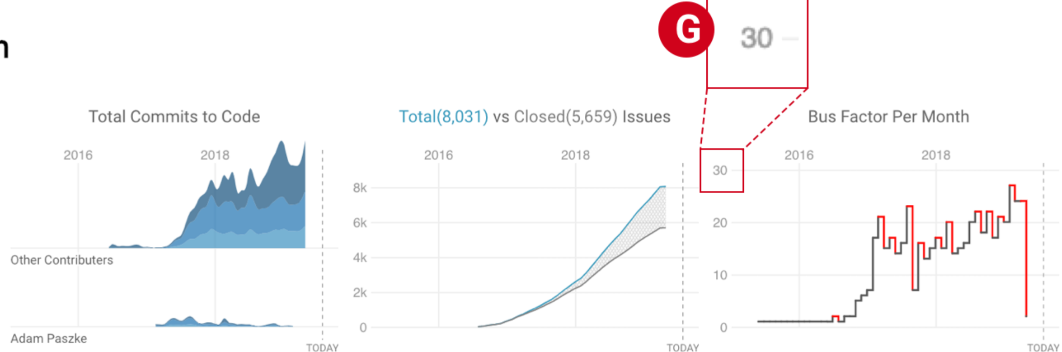
python (192)
theano (86)
tensorflow (78)
keras (73)
django (28)
other tags (65)



pytorch/pytorch

Tags (# Code Files)

c++ (2701)
python (1205)
tensorflow (446)



Theano/Theano

Tags (# Code Files)

python (429)
theano (242)
tensorflow (103)
numpy (91)

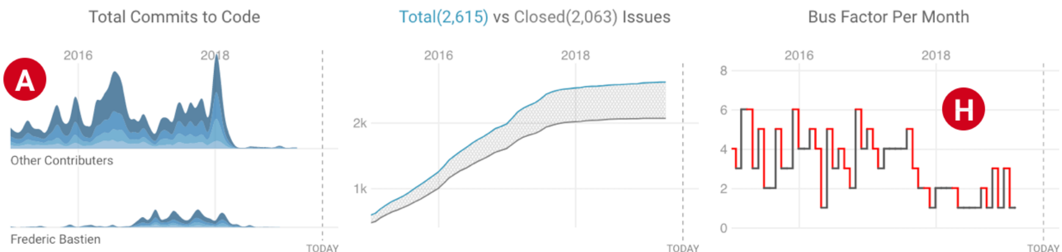


Figure 4. *The usage suitability comparison dashboard is composed of one instance of the usage suitability summary dashboard for each code repository being compared. This example compares the usage suitability of four Python machine learning libraries. See the section "Comparing the Usage Suitability of Machine Learning Libraries" for information on the callouts.*

investigate four popular Python machine learning libraries: Tensorflow, Keras, PyTorch, and Theano. We load their code repositories into the usage suitability comparison dashboard (see Fig. 4) to compare them.

From the Total Commits charts, we can see that Theano is the oldest project (Fig. 4A), and Keras is the second oldest (Fig. 4B). TensorFlow and PyTorch are much newer and begin ramping up in late 2016 and early 2017. The second thing that attracts our attention is the status of issues across all projects. TensorFlow has almost as many total issues as all the other repositories combined (Fig. 4C). TensorFlow appears to do a better job of closing issues than the other code repositories, which is demonstrated by the narrow difference between the number of total issues and closed issues (Fig. 4D). In 2017 we also see a sharp decrease in the rate that Keras closes issues (Fig. 4E). This makes sense because in late 2016 it was announced that Keras would become the default API in TensorFlow[9]; correspondingly, we see a notable decline in commits from the lead author, Francois Chollet, as he began to shift his focus to TensorFlow.

Bus factor provides additional information. TensorFlow and PyTorch have a similar bus factor growth rate and a large number of core contributors (Fig. 4F and G). In contrast, Keras and Theano's bus factor tends to be declining, and Theano's bus factor is particularly low (Fig. 4H).

Based on this quick comparison, we would not select Theano. Keras appears healthier than Theano but lacks signs of longevity, which is consistent with what we know about Francois Chollet's shift in focus. Tensorflow or PyTorch both appear substantially healthier with no signs of a decline in activity, and we would select both of them for a more detailed comparison (e.g. functionality and API quality) to ensure compatibility with our project.

Conclusion

In this paper, we identify six risks and derive seven requirements for external developers who wish to assess the usage suitability of a software library. From these requirements, we developed two dashboards. The first dashboard visualizes three code repository metrics related to software library usage suitability: trends in commits to the repository, trends in the number of total issues and closes issues, and trends in the bus factor. The second dashboard visualizes these metrics across multiple code repositories so that external developers can easily compare multiple software libraries and identify the most suitable ones. We demonstrated, with two example usage scenarios, that these dashboards are effective for analyzing a library's usage suitability and comparing the usage suitability of multiple libraries. In these scenarios, the Total Commits, Total vs. Closed Issues, and Bus Factor visualizations proved to be the most useful; in these examples we did not find major insights into the repositories' usage suitability by examining the semantic tags. We did find that this analysis required substantially less manual counting, comparison, and clicking compared to conducting this same analysis using a typical code repository's user interface.

Future work could expand these dashboards. For example, clicking the Total vs. Closed chart could show information about which contributors are responding to issues. The dashboards could also integrate other information, such as the number of core contributors who work at the same organization. This prototype dashboard could be directly integrated into a code repository's UI to avoid needing to install a browser plugin. Future work should also evaluate these dashboards with users to better understand the dashboards' utility in practice.

[9] `https://www.fast.ai/2017/01/03/keras/`

Acknowledgment

We thank Ben Gelman, Michael Lack, Jessica Moore, Banjo Obayomi, and David Slater for advice and feedback on this work.

This project was sponsored by the Air Force Research Laboratory (AFRL) as part of the DARPA MUSE program.

References

[1] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. Using static analysis to find bugs. *IEEE software*, 25(5):22–29, 2008.

[2] T. Ball and S. G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[3] M. Bostock, V. Ogievetsky, and J. Heer. Data-Driven Documents. *IEEE TVCG*, 17(12):2301–2309, 2011.

[4] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[5] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE TSE*, 27(1):1–12, 2001.

[6] R. Ferenc, L. Langó, I. Siket, T. Gyimóthy, and T. Bakota. Source meter sonar qube plug-in. In *International Working Conference on Source Code Analysis and Manipulation*, pp. 77–82, 2014.

[7] J. García-Munoz, M. García-Valls, and J. Escribano-Barreno. Improved metrics handling in sonarqube for software quality monitoring. In *International Conference on Distributed Computing and Artificial Intelligence*, pp. 463–470, 2016.

[8] B. Gelman, B. Hoyle, J. Moore, J. Saxe, and D. Slater. A language-agnostic model for semantic source code labeling. In *Proc. 1st Int. Workshop on Machine Learning and Software Engineering in Symbiosis*, pp. 36–44, 2018.

[9] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer. Visualizing collaboration and influence in the open-source software community. In *Proc. MSR*, pp. 223–226, 2011.

[10] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proc. MSR*, pp. 99–108, 2008.

[11] A. Jermakovics, A. Sillitti, and G. Succi. Mining and visualizing developer networks from version control systems. In *Proc. 4th CHASE*, pp. 24–31, 2011.

[12] S. C. Johnson. Lint, a c program checker. Technical report, Bell Laboratories, 1978.

[13] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proc. ACM SOFTVIS*, pp. 193–202, 2005.

[14] F. van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proc. 20th IEEE Int. Conf. on Software Maintenance*, pp. 328–337, 2004.

[15] L. Šubelj and M. Bajec. Community structure of complex software systems: Analysis and applications. *Physica A Stat. Mech. Appl.*, 390(16):2968–2975, 2011. doi: 10.1016/j.physa.2011.03.036

[16] L. Šubelj and M. Bajec. Software systems through complex networks science: Review, analysis and applications. In *Proc. KDD Workshop on Software Mining*, pp. 9–16, 2012.

[17] L. Šubelj, S. Žitnik, N. Blagus, and M. Bajec. Node mixing and group structure of complex software networks. *Advances in Complex Systems*, 17(7):1450022, 2014.

Author Biography

Casey Haber received a BS in Computer Science with a minor in Design from the University of San Francisco (2018). Recently, he worked at Two Six Labs in Arlington, VA where he worked as a designer, full-stack developer, and visualization specialist on government R&D programs. He is currently a Visualization Engineer at Chartio.

Robert Gove received dual BS degrees in Computer Science and Applied Mathematics from the University of North Carolina at Greensboro (2009) and his MS in Computer Science from the University of Maryland (2011). He has worked at Two Six Labs (formerly Invincea Labs) in Arlington, VA since 2014. He has focused on applying data visualization techniques to cybersecurity problems and building scalable graph visualization tools.

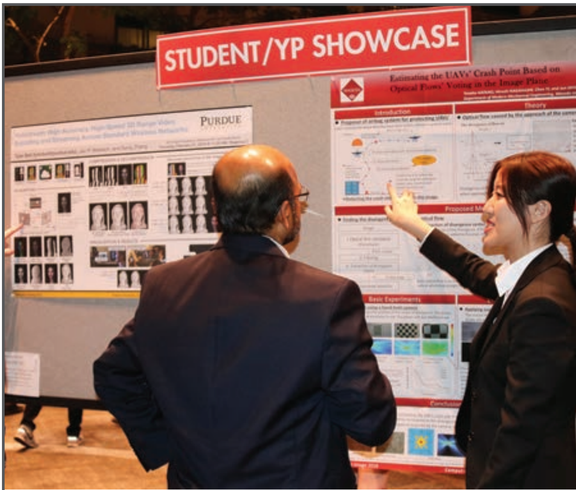
JOIN US AT THE NEXT EI!

IS&T International Symposium on

Electronic Imaging

SCIENCE AND TECHNOLOGY

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org