Jordan Richards

ENC 1102

22 October 2019

The methods described in the research proposal describe generally the process of scraping, coding, and analyzing a repository, but several additional details are required to actually employ such a method. In order to simplify the scraping and coding process, a script was written to generate a summary document including a random sample of a few features from a given repository. While such a summary might not be complete, the key elements we pick up on will help to guide how additional data will be collected to supplement the data collected in the sample.

The choice of features to extract in the script, therefore, should provide an effective overview of a repository, but some sacrifices could be made to limit the scope of pages to scrape. For instance, the script does not identify and scrape external documentation sites as there was no consistent methods to do so. In the end, the set of features chosen for the initial sampling of each repository included markdown documents, inline comments, issues, and pull requests. These categories are present in all of the top repositories, play a key role in the contribution and usage of an open source project, and allow us to automate the scraping process.

Access to the data collected in this study is available via a Github repository [See the Appendix]. The repository includes the script that was used to scrape the repository, the reasoning behind the selection of the top 10 repositories, and the summary documents for each of the sampled repositories. This provides the tools for reproducing this study with a different sample of repositories. The documents with codes in the margins will also be made available once the analysis is complete.

(Disclaimer: Data analysis is not yet complete, so the results below might not accurately reflect all collected data)

The first set of documents fall under the markdown category. These rich text documents are the most flexible of any scraped category of documents, serving a variety of purposes from *README* files to API documentation. This flexibility might be attributed to the wide range of features Github markdown (and other markdown variants) provide, allowing developers to present complicated ideas effectively at minimal cost.

Markdown documents in a repository can be split into a few subcategories based on their usage. These documents are often ordered in some form of hierarchy, rooted at a *README* file for the repository as a whole, then branching into a series of other files providing more detailed descriptions of different parts of the project. There are several elements common across the subcategories. For instance, most categories begin with a description of the topic the markdown file is meant to present, often as one of the first lines. markdown files also often provide examples, consisting of code snippets or diagrams, helping to better illustrate key ideas. Markdown files also take advantage of links to external pages, such as a Wiki, forum, or external documentation, perhaps to present more information without sacrificing readability, or to present info in other formats not supported by the markdown document. Other common elements include API references and developer tutorials.

A key subcategory of markdown documentation is the contribution guideline. These guidelines might provide information to a developer about how to format other forms of documentation, such as pull request, commit messages, and issues. This information is often supported by templates, providing a specific format to ensure consistency of code and documentation across the repository.

Inline comments provided another source of documentation. The scope covered by inline comments was far more limited than the markdown documents, often falling into one of two categories with significant overlap between them: comments explaining flow, and comments explaining technical details. Comments that explained flow allow developers already familiar at a high level with the code's function to more easily follow along during a read through. While design paradigms such as functions and abstraction already provide some level of logical organization, flow comments are often needed to supplement in longer functions where the purpose of some lines might be unclear. The comments that explained technical details covered other cases when low level technical details were needed to provide clarification for why code was implemented in a certain way. Both types of comments are often sparse, often spanning only a line or so, with few words, perhaps to avoid clutter.

Issues and pull requests are the only types of documentation generated without modifying the source of the repository. While the other forms of documentation involve modifying the "ground truth" of the repository, issues and pull requests provide a history of the changes to the repository, new information is only intended to be added to the record, rather than modifying and updating existing records.

Focusing on issues, most issues follow a set schema. Users that start an issue often begin with a general description of the bug they are experiencing, followed by formatted steps for reproducing the bug. This format is standardized through the use of issue templates, and regulated through the use of an automated system that flags and closes issues that don't adhere to the template. Developers or maintainers contributing to the repository might then join the conversation, offering solutions or closing the issue if it is off topic. Often bugs are false positives, resulting in a developer clarifying a feature. As a whole, issues provide documentation for the project, providing other developers more information about unclear features, commonly asked questions, as well as information about any ongoing issues that they might contribute back to.

<TODO: Genre analysis, Appendix, etc.>