GITS 2.0 - IRIS

Ashok Kumar Selvam
aselvam@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Rithik Jain
rjain25@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Sri Athithya Kruth Babu
sbabu@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Subramanian Venkataraman
svenka25@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Zunaid Hasan Sorathiya
zhsorath@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

We see that for a number of projects on Github, there is a lack of a proper file structure, a contributing file, a code of conduct document and even Readme files at times. These factors take away from the world the opportunity to easily learn, clone and reuse these projects, In our project, we identify criteria that make a repository "good" and have created software that we believe can work to nudge users towards making their repositories "good" - maintainable, reusable and understandable.

KEYWORDS

GitHub, Good Repository, Readability and Reusability

ACM Reference Format:
Ashok Kumar Selvam, Rithik Jain, Sri Athithya Kruth Babu, Subramanian Venkataraman, and Zunaid Hasan Sorathiya. 2018. GITS 2.0 - IRIS. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Github is a provider of software hosting and version control that is widely used among developers in the community. There are thousands of repositories on Github that contain a variety of software projects. However, it may be noted that several of these projects, while containing useful content, are rather hard, cumbersome or tedious to make contributions for or reuse effectively.

This is because these repositories lack features and items which make understanding these aspects possible. Features like contributing guidelines, codes of conduct that explain how interaction with the project can be done will strongly enhance the reuse-ability and understanding of the project by new users.

In this project, we extended a Command Line Interface tool that allows users to extend and simplify functionality, GITS, which stands for Git Simplified. We extended GITS due to its extend-ability, excellent documentation and large selection of features which made it a valuable basis for further work.

Basing our workflow on the Linux Kernel Best Practices, we have developed a tool that will allow users both create and get information on how to work towards, a "good" repository. We have designed and implemented functionality to accomplish creation of a basic repository structure with files that can be used to nudge the contributors towards writing code.

We plan to implement functionality to allow users pushing code to get information about whether the overall "goodness" of their repository has decreased, and the factors that cause this decrease. With this, we believe that new users and even long-time programmers will be nudged to effectively build repositories that can be easily reused, extended further and work with efficacy.

2 CONSENSUS-ORIENTED MODEL

In our work on this project, we have followed a consensus-driven approach to the decisions taken, the features to work on in the current phase, and to determine how the work is to be approached in terms of design and implementation.

Wherever possible, we have tried to follow a logical, democratic approach to make our decisions. We believe that this has led to improvements in overall clarity, given us a strong framework for approaching the addition and execution of tasks and sub-tasks in the project which we has been through copious use of Github's projects feature.

Using this also allowed us to determine which methodology of change suited us most effectively as a team. This was a gitflow-based methodology, with developers contributing their new features to different branches, and allowed asynchronous and self-driven work, improving our velocity and agility as a team.

Discussion on strategies for issues and implementation have been taken up in all our meetings which happen twice a week on fixed days. We have enforced communication through a chat channel where team members can provide updates on any open or pending issues and challenges so the entire team is aware and can suggest solutions.

3 NO INTERNAL BOUNDARIES

We have maintained openness and clarity on the methods and implementation in the project, and any approach, implementation approach followed by a developer could be improved upon by any other regardless of ownership of the task, with the correct justification.

Our choice of language was determined to be Python, since this was the language used by GITS, which we have extended. The

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

packages, libraries and tooling have been maintained across the team members to maintain consistency and continuity with the same adopted by GITS, allowing for minor improvements wherever feasible.

This afforded us convenience in sharing tasks, and reduced confusion in code reviews where we were able to debug and understand the code and the notations used easily. Changes and fixes could be introduced with justification in a straightforward manner after discussion.

4 DISTRIBUTED, HIERARCHICAL DEVELOPMENT MODEL

We enforced a model of democratic, open discussion to allow the free flow of creativity. Group meetings always had moderators who were rotated among the members of the group, and were present to ensure that all members were presenting and contributing to the project's workflow. We also ensured regular updates from every member on progress and any standing issues, to which group members would suggest solutions.

The tasks, after being decided by the group in meetings, would need to happen asynchronously i.e without dependencies on other tasks or other sections of code as much as possible. This distributed model of development improved our speed and velocity as a group. The tasks would be added as issues to the corresponding project on the Github Repository for the project, and assigned to the member who would be working on them. The status of the same would be moved to "In Progress" and then "Done" as progress was made on the task in question.

To ensure that changes across the project were made without impacting existing functionality, we introduced a system of two-fold reviews where changes to the main project had to be approved by two other members in the team. This ensured that changes went through with no effects to the previously working functionality, and changes were made primarily to the new code being added to the code base.

For work to occur without delays due to reviews from different developers involved, there is no restriction on any single developer to act as the reviewer. All developers in the team can and are expected to contribute to reviews in this manner to ensure that code being pushed to branches is reviewed.

We have added documentation for all of the new features, and reviewed the previously written documentation as well for both understanding and to explore possible improvements or augmentations. We have also added guides on the steps that can be followed for new features that can be added in the future.

5 NO REGRESSIONS RULE

In our work extending GITS, we have taken care to ensure that none of the existing functionalities and commands from the original GITS have been affected, degraded or deprecated. By structuring our goals, methodology and implementation strategy in this fashion, we have achieved extension of the original project with no regressions to the existing features.

In choosing the language to extend GITS, we decided that no other alternative would make sense to allow us to seamlessly integrate with the existing code base apart from Python.

To avoid any effect to existing features in the GITS code base, we have taken care to ensure the changes made have been done in a separate files which are separate from yet seamlessly integrate with the already present code. In addition to that, we have maintained existing file naming and code-specific naming conventions followed by GITS.

6 TOOLS MATTER

Tools enable effective contribution and greatly improve the pace of development, ensuring consistency and convenience without affecting productivity. In our project, the tools we used have certainly done so.

GitHub has been the backbone on which the previous GITS project we extended, and our current project, reside. Using Git allowed for contribution to take place from different developers without conflict. This has enabled distributed development without dependencies.

Projects and issues on Github have been our primary choice for organizing and maintaining the workflow of different features. We also reviewed the existing requirements files for any changes that might be necessary for the new features.

We have ensured that for every new command or functionality added, that there are ample test cases to validate the working of the same. These have been routinely executed to maintain the working of the existing components while new features were added. We used Travis CI to monitor build quality and make sure testing occurred regularly on code being pushed to the repository.