Enabling efficient and simple analysis of software repositories

[Project Proposal]

Muhammad Shayan
mdshayan@cs.ubc.ca
University of British Columbia

ABSTRACT

The availability of open-source repository hosting services like Github has led to immense amount of data available for SE researchers to understand and analyze the evolution of source code over time. This has created an active field of software engineering research called Mining Software Repositories (MSR). Researches in MSR are heavily involved in performing different kinds of analysis on the vast amount of source code available in these repositories to understand the software evolution process. The source code is accessed through the Github API which offers no analysis capabilities to researchers. Therefore, I propose a software system that can provide a different interface to MSR researchers, such that they can more easily and efficiently perform different kinds of static analysis on source code in git repositories.

1. MOTIVATION

Mining software repositories (MSR) is a recent and growing research field that aims to extract information from repositories to understand and control the evolution of software systems. A popular information source for mining software repositories is Github. The most important artifact available from source code repositories is the source code because its evolution largely contributes to the overall software evolution. In order to study this evolution, a very popular technique in MSR approaches use static program analysis to extract facts and other information from versions of a system. These approaches span across a wide range of available techniques for parsing, processing, and extracting facts from source code. These analysis have proved to be a vital part of Unfortunately, there is no interface available to MSR researchers to perform this type of analysis directly on a certain github repository. MSR researchers usually have to code the interface themselves to bridge between the data provided by the github API and the analysis features offered by various program analysis tools. This process is extremely time-consuming and is common among many MSR approaches for bug finding and fixing, successful software use, detecting function usage patterns etc. Therefore, exposing a general purpose API that allows users to easily perform these tasks would allow the creation of complicated tools dependant on these analysis that were previously difficult to build.

2. ARCHITECTURE/SYSTEM DESIGN

The system proposed will be divided into a four-tiered architecture described as follows:

- Repository: The GHTorrent dataset will act as the database of git software repositories which we will use to query and access the required source code to perform our analysis on.
- **Analytics Engine:** This engine will build the ASTs, control flow and data flow graphs for the set of source code files specified by the miner.
- Query engine: This engine will allow the miner to query the ASTâĂŹs, Control Flow Graphs to generate meaningful insights about the source code evolution e.g. get the commit id when a certain variable was introduced.
- **REST API:** The web service exposed to the miner to enable him to perform static analysis on a given github repositories.

3. EVALUATION

The following research question should be answered at the end of the course project:

1. Does our system help researchers perform static analysis on source code on Github quickly and efficiently?

To answer this question, a set of 5 static analysis tasks will be identified that are common in MSR papers over the last 5 years. We will analyze the authors source code to compare the LOC it took for them to implement a certain static analysis task and whether it could have been implemented using less LOCs with our API.

4. MILESTONES

- Expose a REST API to MSR researchers that performs common static analysis tasks on a github repo such as AST generation, computing control flows and data flow graphs for files
- Enable MSR researchers to perform queries on the Intermediate Representations
- Evaluate the effectiveness of the API by comparing lines of code to evaluate its effectiveness in providing ease and comfort to the programmer.