Semi-Automated Refactoring Documentation Bot

Soumaya Rebai
Dr. Marouane Kessentini

Context

Software projects evolve and change continuously which:

- Increases complexity of the project
  - ***60%*** of Developers' time in understanding existing software
- Reduces the productivity
  - ***76%*** of Software Engineers working on maintenance tasks
- Increases fault-proneness
- Increases cost of maintenance
  - ***67%*** of Software Budget allocated to maintenance

Context

Documentation is a recommended practice in software development and maintenance to help developers understand the code quickly and improve their productivity.

GitHub is a well-known collaborative platform to manage software projects as CI process.

➔ Programmers need documentation such as commit messages and PR description to understand the changes.

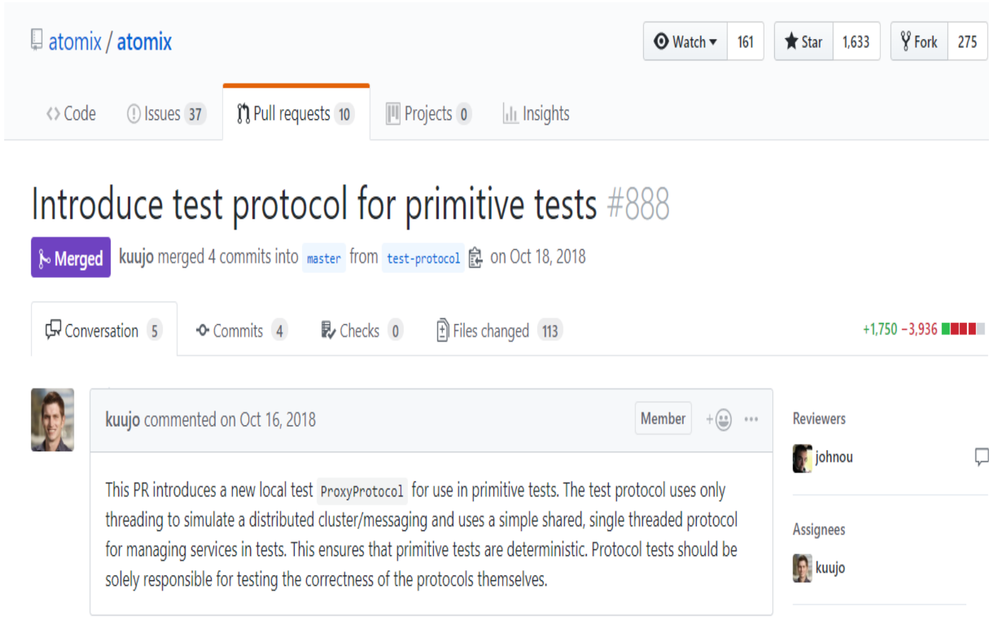
Problem Statement

Motivations

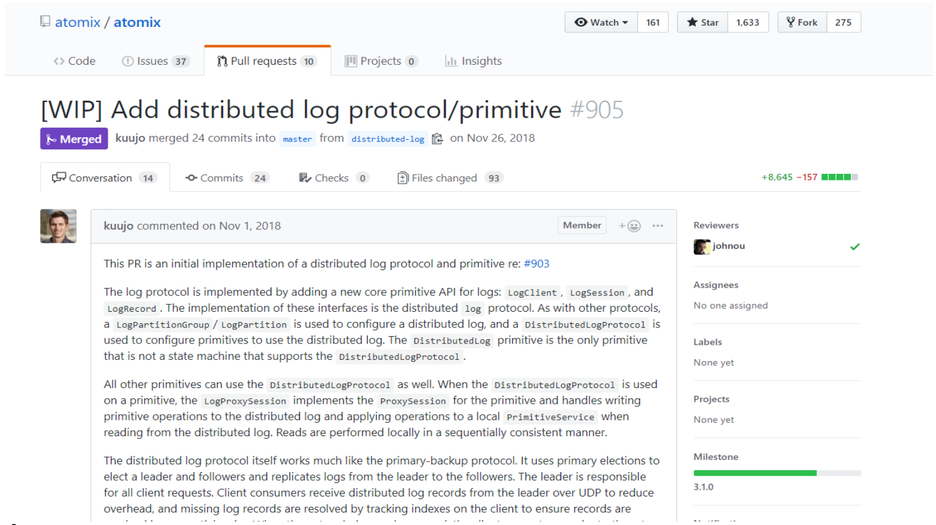
- ***Lack of refactoring documentation:***
  - Developers only/mainly focus on documenting functional changes and bugs fixing.
- ***Due to time and money pressure, developers can't adequatly document their work:***
  - Documenting the changes is time consuming because they have to document what refactorings they applied, theirs locations and what they intended to improve in their code quality.
  - Documenting non functional changes is challenging: It's not straightforward to specify the quality attributes to improve since every developer may use different jargon to describe quality improvements.
- ***Developers perception of refactoring opportunities is not limited to antipatterns/code smells:***
  - (Survey) Average of only 12% of commit messages described applied refactorings for JHotDraw, Xerces and three industrial projects.
  - Only 0.13% of the commit messages from 1,984 popular projects in GitHub contain any antipattern.

Problem Statement

Challenges



An example of a pull request with poor documentation from an open source project



PR with only functional changes are documented

Quality metrics change in the pull request

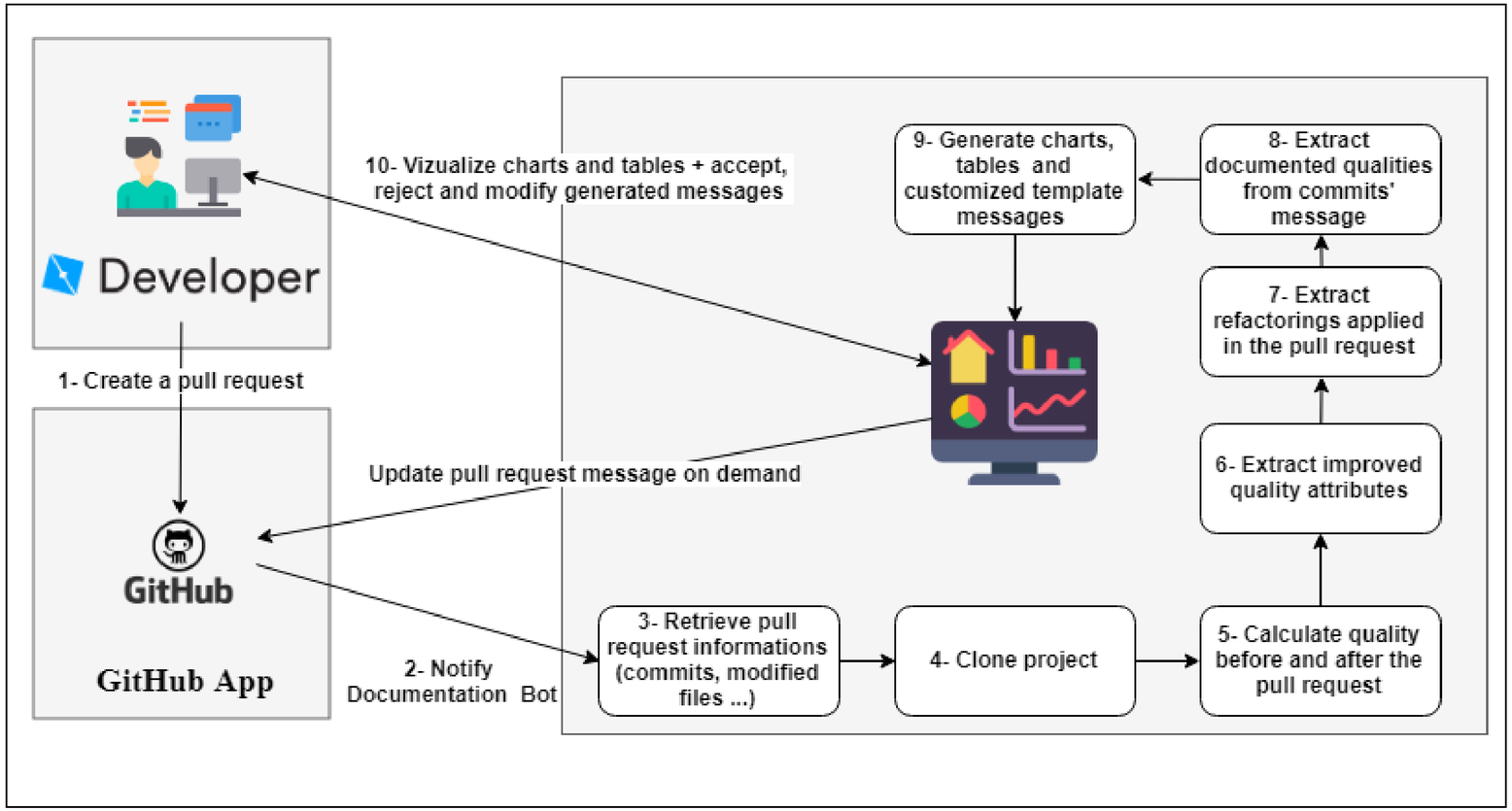
Pull Request #888

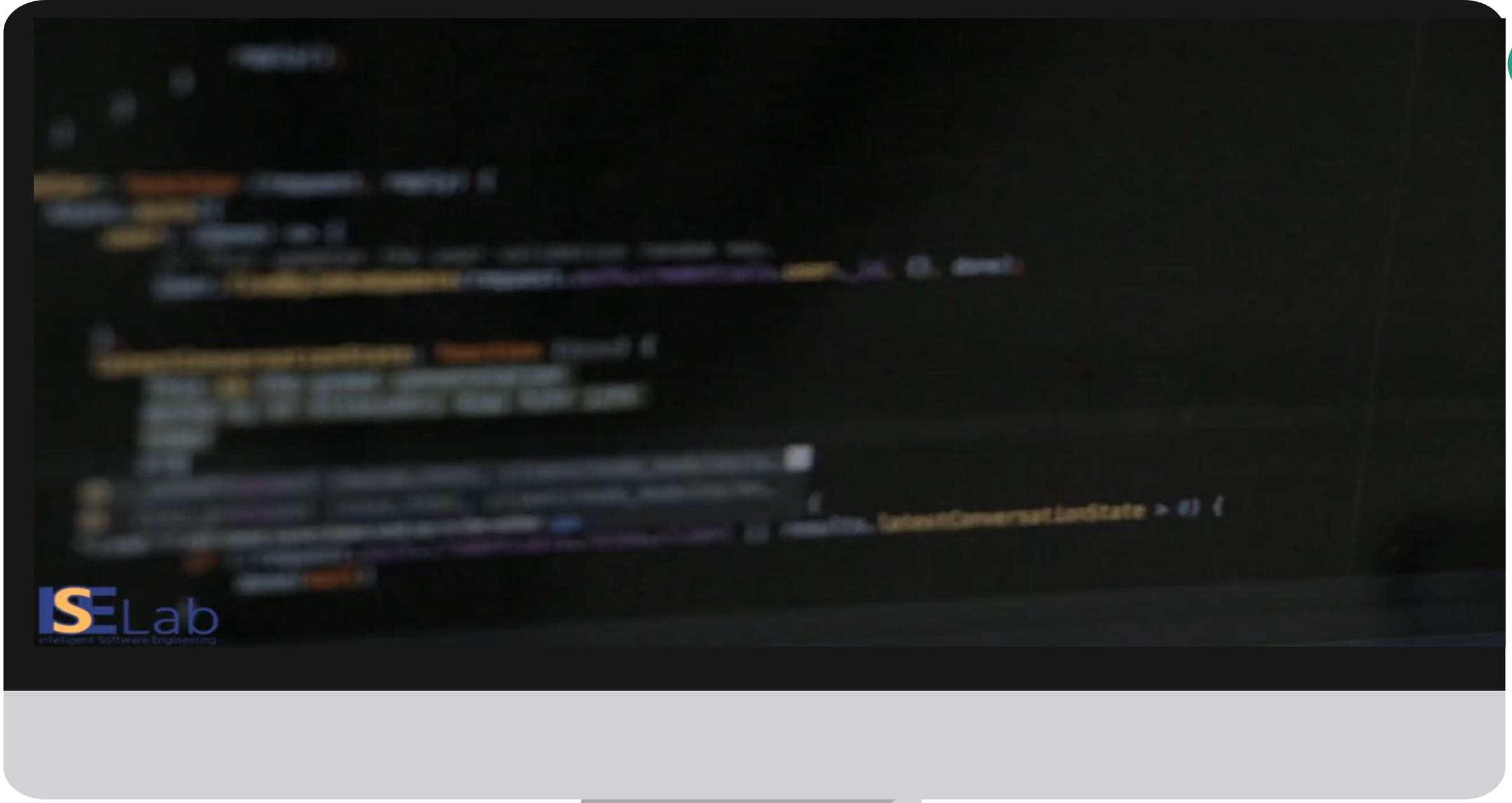
Before PR	***After PR***
DAM : 0.972635293882407	DAM : 0.9717739022881879
ANA : 0.6636042402826855	ANA : 0.6017569546120058
DSC : 1415.0	DSC : 1366.0
DCC : 0.9469964664310954	DCC : 1.0146412884333822
NOH : 101.0	NOH : 74.0
MFA : 0.11277199196527862	MFA : 0.09627351534530657
CIS : 5.567491166077739	CIS : 5.78696925329429
NOM : 6.628975265017668	NOM : 6.855783308931186
CAM : 0.23658681120881084	CAM : 0.24739028632608542
MOA : 0.2939929328621908	MOA : 0.31771595900439237
NOP : 6.015547703180212	NOP : 6.216691068814056Reusability :
Effectiveness : 1.6117104324345548	Effectiveness : 1.6408422800127898
Reusability : 710.1061431692333	Reusability : 685.7016718761204
Functionality : 336.0966589685818	Functionality : 319.470492105223
Understandability : -471.2551475180408	Understandability : -455.2250037826182
Extendibility : 2.9224637344985402	Extendibility : 2.950040125168993
Flexibility : 3.161180024884029	Flexibility : 3.2564866673729256

INTERACTIVE REFACTORING

DOCUMENTATION BOT

Approach Overview




ISELab
Intelligent Software Engineering

Detailed Approach

1. Pull-Request Changes Analysis:

- Bot get notified of a new PR and then clone the repository on GitHub
- Extraction of the commit messages and modified files of the submitted PR
- GitHub API is used to identify the changed files.
- RefactoringMiner to identify the actual refactorings applied.
- Quality assesment at the file level using QMOOD quality metrics.

2. Checking the Current Documentation of the Developer

- Manually defined a large set of keywords that may cover most of the words used to document QA.
- Manually classified our keywords into 6 QMOOD categories (extendibility, reusability, flexibility, understandability, functionality, extendibility, effectiveness).

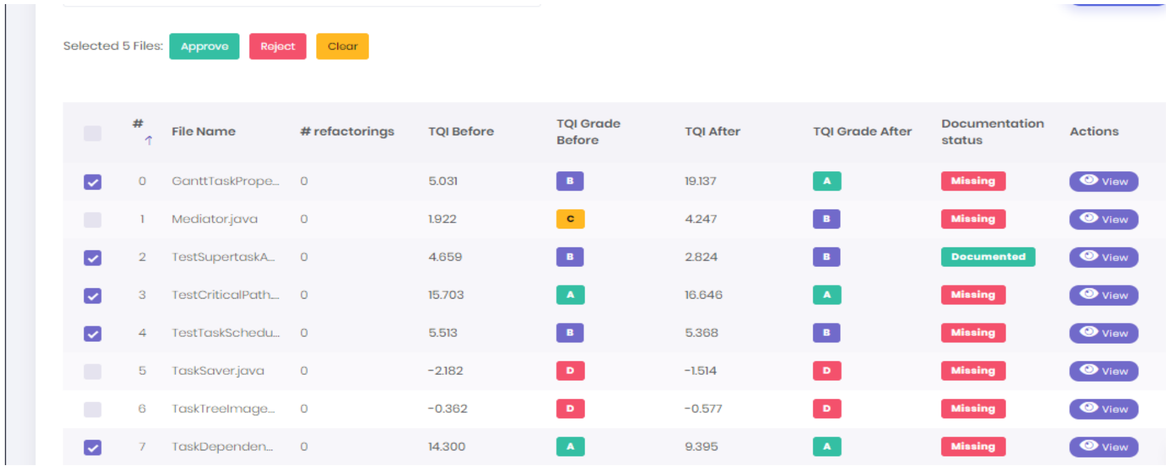
➔ The combination of keywords and the detected refactorings along with the name of the modified files represent a sufficient set of features that help us checking whether the specific quality attributes and refactorings detected in the previous step are documented in any of the commit messages and the developer's pull request description

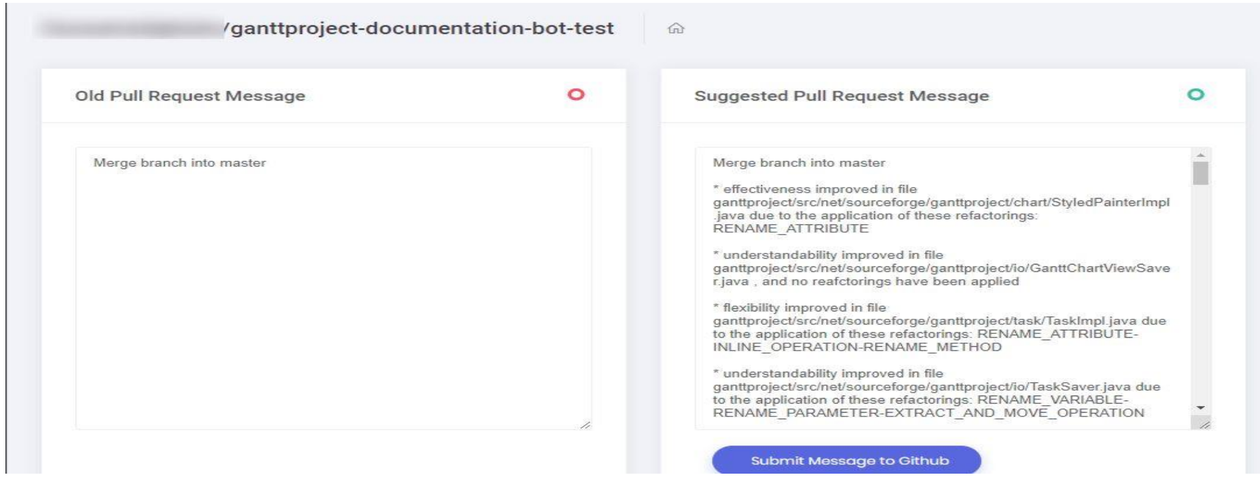
Detailed Approach

3. Generation of the Refactoring Documentation and Interaction with Developers

- Providing a support of the recommended documentation based on the identified refactoring and quality attributes changes
- Enabling developers interaction to accept ,reject or modify the documentation (For ease of interaction : the interaction is done at the file level)
- Our message is composed of : location (file name), the refacotring applied and the missed documented quality attributes.

  ➔ Documenting ***what has been refactored? Why the refactorings were applied? What is the impact of these refactorings on quality***.





Conclusion & Future Work

- An ***interactive semi-automated documentation bot*** to document the developers changes in terms of quality attributes imrpovement and refactorings.

- ***Interaction with the developer*** helped choosing the most adequate pull request description for refactoring-related code changes.

- ***Suvery*** is conducted with 14 software developers and 5 systems shows clear evidence that our bot helped developers documenting the quality improvement of the applied refacotrings;

- Future Work:
  - Extend our experiments on larger set of systems and participants
  - Evaluate different documentation techniques to adopt them for documenting refactorings rather than the use of the rules-based techniques.

Semi-Automated Refactoring Documentation Bot

Soumaya Rebai
Marouane Kessentini