



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



REFACTORING

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.

DOCUMENTATION

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 adequately document their work:***

- Documenting the changes is time consuming because they have to document what refactorings they applied, their 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

atomix / atomix

Watch 161 Star 1,633 Fork 275

Code Issues 37 Pull requests 10 Projects 0 Insights

Introduce test protocol for primitive tests #888

Merged kuujo merged 4 commits into master from test-protocol on Oct 18, 2018

Conversation 5 Commits 4 Checks 0 Files changed 113 +1,750 -3,936

kuujo commented on Oct 16, 2018

This PR introduces a new local test `ProxyProtocol` for use in primitive tests. The test protocol uses only threading to simulate a distributed cluster/messaging and uses a simple shared, single threaded protocol for managing services in tests. This ensures that primitive tests are deterministic. Protocol tests should be solely responsible for testing the correctness of the protocols themselves.

Reviewers: johnou

Assignees: kuujo

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

atomix / atomix

Watch 161 Star 1,633 Fork 275

Code Issues 37 Pull requests 10 Projects 0 Insights

[WIP] Add distributed log protocol/primitive #905

Merged kuujo merged 24 commits into master from distributed-log on Nov 26, 2018

Conversation 14 Commits 24 Checks 0 Files changed 93 +8,645 -157

kuujo commented on Nov 1, 2018

This PR is an initial implementation of a distributed log protocol and primitive re: #903

The log protocol is implemented by adding a new core primitive API for logs: `LogClient`, `LogSession`, and `LogRecord`. The implementation of these interfaces is the distributed `Log` protocol. As with other protocols, a `LogPartitionGroup / LogPartition` is used to configure a distributed log, and a `DistributedLogProtocol` is used to configure primitives to use the distributed log. The `DistributedLog` primitive is the only primitive that is not a state machine that supports the `DistributedLogProtocol`.

All other primitives can use the `DistributedLogProtocol` as well. When the `DistributedLogProtocol` is used on a primitive, the `LogProxySession` implements the `ProxySession` for the primitive and handles writing primitive operations to the distributed log and applying operations to a local `PrimitiveService` when reading from the distributed log. Reads are performed locally in a sequentially consistent manner.

The distributed log protocol itself works much like the primary-backup protocol. It uses primary elections to elect a leader and followers and replicates logs from the leader to the followers. The leader is responsible for all client requests. Client consumers receive distributed log records from the leader over UDP to reduce overhead, and missing log records are resolved by tracking indexes on the client to ensure records are

Reviewers: Johnou

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: 3.1.0

Quality metrics change in the pull request

PR with only functional changes are documented

Before PR

Pull Request #888

After PR

DAM : 0.972635293882407
 ANA : 0.6636042402826855
 DSC : 1415.0
 DCC : 0.9469964664310954
 NOH : 101.0
 MFA : 0.11277199196527862
 CIS : 5.567491166077739
 NOM : 6.628975265017668
 CAM : 0.23658681120881084
 MOA : 0.2939929328621908
 NOP : 6.015547703180212
 Effectiveness : 1.6117104324345548
 Reusability : 710.1061431692333
 Functionality : 336.0966589685818
 Understandability : -471.2551475180408
 Extendibility : 2.9224637344985402
 Flexibility : 3.161180024884029

DAM : 0.9717739022881879
 ANA : 0.6017569546120058
 DSC : 1366.0
 DCC : 1.0146412884333822
 NOH : 74.0
 MFA : 0.09627351534530657
 CIS : 5.78696925329429
 NOM : 6.855783308931186
 CAM : 0.24739028632608542
 MOA : 0.31771595900439237
 NOP : 6.216691068814056
 Reusability : 685.7016718761204
 Effectiveness : 1.6408422800127898
 Functionality : 319.470492105223
 Understandability : -455.2250037826182
 Extendibility : 2.950040125168993
 Flexibility : 3.2564866673729256



INTERACTIVE REFACTORING

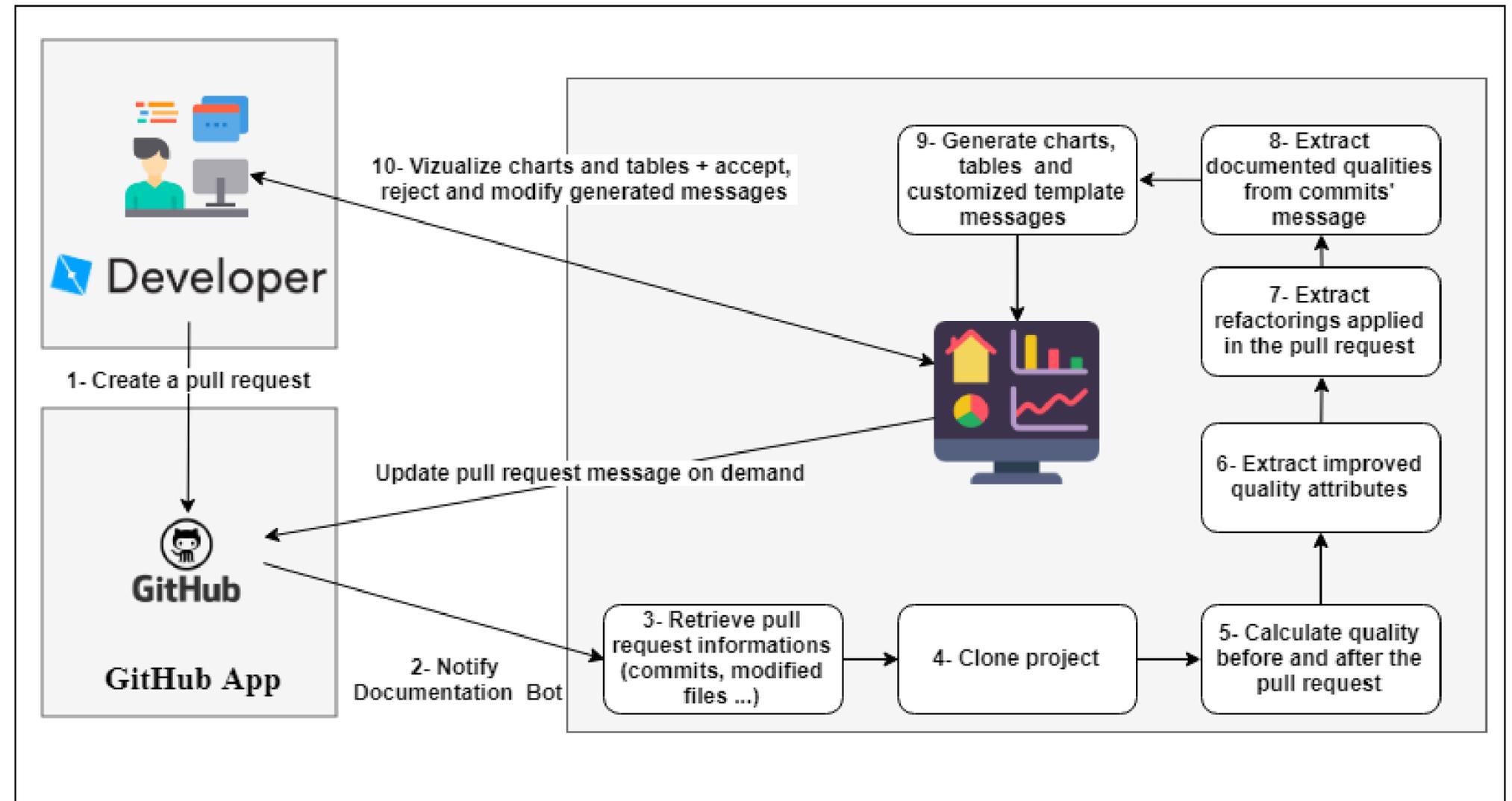
DOCUMENTATION BOT

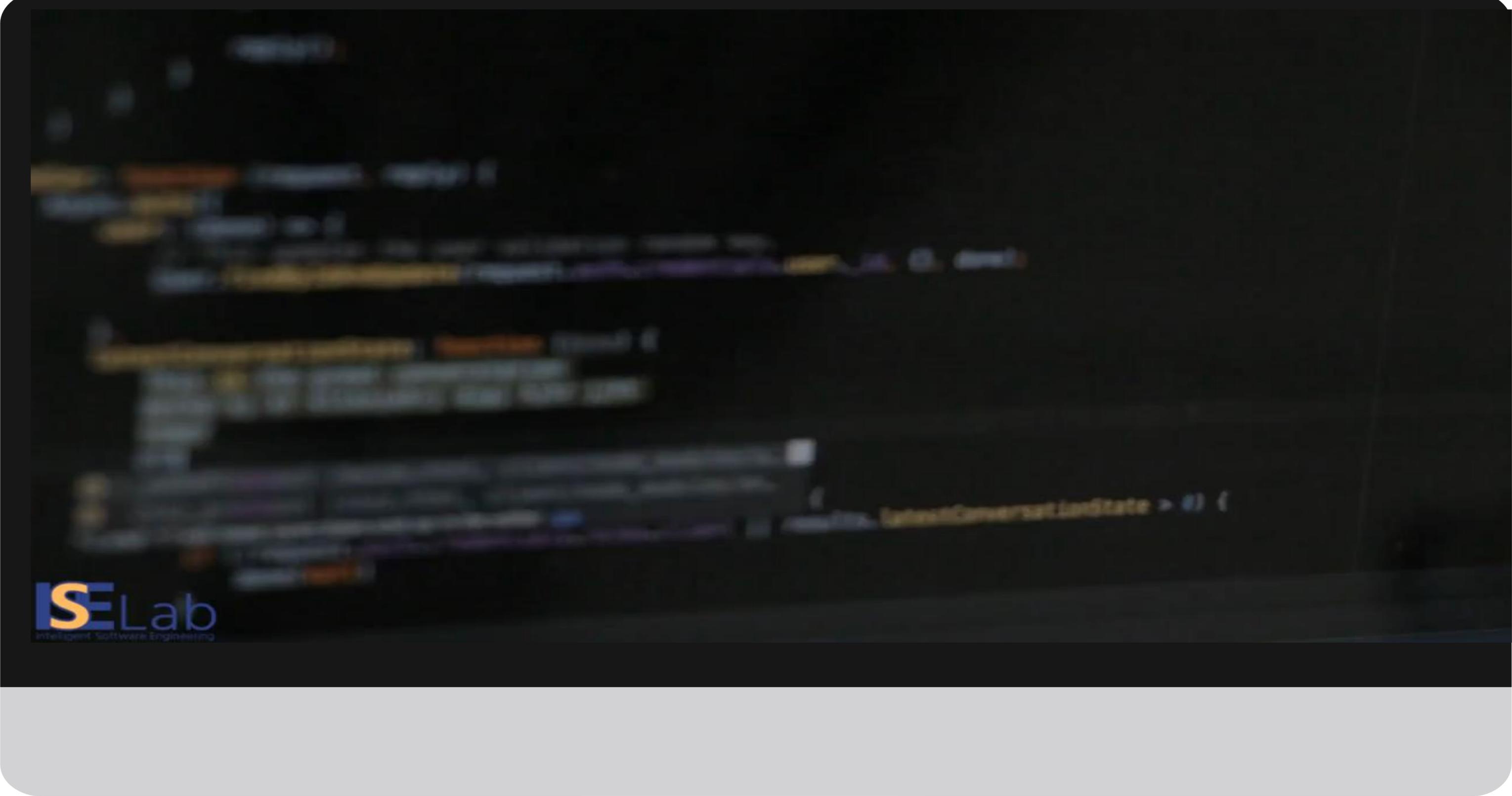


Approach Overview



Documentation Bot





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.**

Selected 5 Files: Approve Reject Clear

#	File Name	# refactorings	TQI Before	TQI Grade Before	TQI After	TQI Grade After	Documentation status	Actions
<input checked="" type="checkbox"/>	GanttTaskPrope...	0	5.031	B	19.137	A	Missing	View
<input type="checkbox"/>	Mediator.java	0	1.922	C	4.247	B	Missing	View
<input checked="" type="checkbox"/>	TestSupertaskA...	0	4.659	B	2.824	B	Documented	View
<input checked="" type="checkbox"/>	TestCriticalPath...	0	15.703	A	16.646	A	Missing	View
<input checked="" type="checkbox"/>	TestTaskSchedu...	0	5.513	B	5.368	B	Missing	View
<input type="checkbox"/>	TaskSaver.java	0	-2.182	D	-1.514	D	Missing	View
<input type="checkbox"/>	TaskTreeImage...	0	-0.362	D	-0.577	D	Missing	View
<input checked="" type="checkbox"/>	TaskDependen...	0	14.300	A	9.395	A	Missing	View

/ganttproject-documentation-bot-test

Old Pull Request Message

Merge branch into master

Suggested Pull Request Message

Merge branch into master

- * effectiveness improved in file ganttproject/src/net/sourceforge/ganttproject/chart/StyledPainterImpl.java due to the application of these refactorings: RENAME_ATTRIBUTE
- * understandability improved in file ganttproject/src/net/sourceforge/ganttproject/io/GanttChartViewSaver.java , and no refactorings have been applied
- * flexibility improved in file ganttproject/src/net/sourceforge/ganttproject/task/TaskImpl.java due to the application of these refactorings: RENAME_ATTRIBUTE-INLINE_OPERATION-RENAME_METHOD
- * understandability improved in file ganttproject/src/net/sourceforge/ganttproject/io/TaskSaver.java due to the application of these refactorings: RENAME_VARIABLE-RENAME_PARAMETER-EXTRACT_AND_MOVE_OPERATION

[Submit Message to Github](#)

Conclusion & Future Work



- An ***interactive semi-automated documentation bot*** to document the developers changes in terms of quality attributes improvement 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 refactorings;
- 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

