PROTOCOL OF NON-FUNCTIONAL REQUIREMENTS IDENTIFICATION

Goal

Manually analyze pull requests and identify the presence (or absence) of non-functional requirements. This identification will serve as our dataset to answer our RQs.

Non-Functional Requirements and Software Evolution

Non-Functional Requirements (NFRs) specify desired qualities or attributes that a system must have (*e.g.*, security, maintainability, and performance). Due to the central role that NFRs play in software design (Nuseibeh B., 2001), they also act as guides on how systems should be developed and evolved. Therefore, NFRs should be defined as early as possible, ideally before the development process begins.

Identification of NFRs

In an ideal scenario, given their importance, NFRs should be described in software requirements documents. However, this type of document generally does not have the description of the NFRs for the system (Cleland-Huang J. et al. 2007). In addition, discussions related to NFRs may be related to various system artifacts, such as the so-called repository artifacts.

Using repository artifacts for qualitative analysis

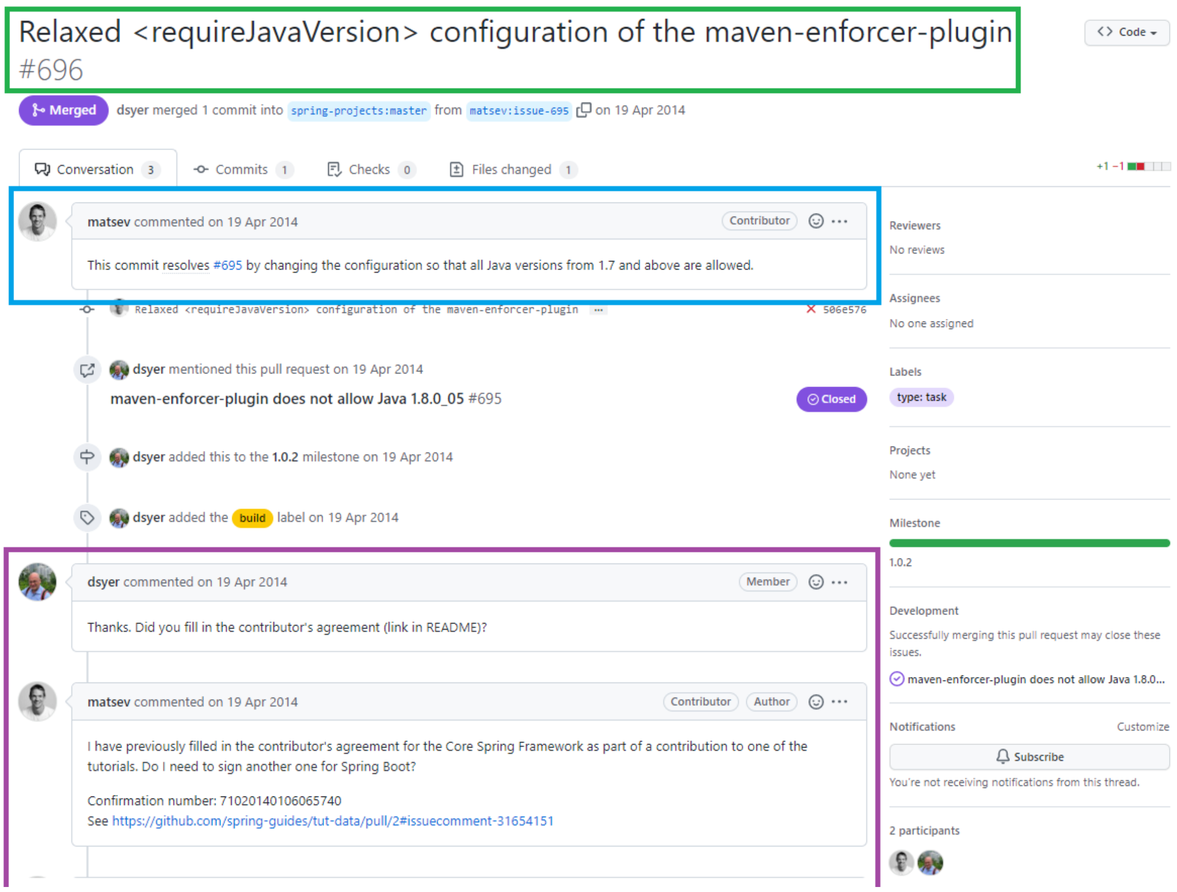
In our context, repository artifacts encompass pull request messages. The motivation for using this type of artifact comes from the fact that developers tend to discuss topics related to NFRs in pull requests.

Description of the type of data to be analyzed

The analyzed data are pull request messages, mined on the GitHub platform. For each pull request, there is an associated title, description (first comment), comments, and related commits.

Structure of the Data to be Analyzed

Each pull request will be presented as follows:
The area marked by the green square represents the pull request title. In blue is the PR description. The PR comments are highlighted in purple.



Relaxed <requireJavaVersion> configuration of the maven-enforcer-plugin #696

Merged dyser merged 1 commit into `spring-projects:master` from `matsev:issue-695` on 19 Apr 2014

Conversation 3 Commits 1 Checks 0 Files changed 1 +1 −1

matsev commented on 19 Apr 2014 Contributor

This commit resolves #695 by changing the configuration so that all Java versions from 1.7 and above are allowed.

`Relaxed <requireJavaVersion> configuration of the maven-enforcer-plugin` ✗ 506e576

dsyer mentioned this pull request on 19 Apr 2014
maven-enforcer-plugin does not allow Java 1.8.0_05 #695 Closed

dsyer added this to the 1.0.2 milestone on 19 Apr 2014

dsyer added the build label on 19 Apr 2014

dsyer commented on 19 Apr 2014 Member

Thanks. Did you fill in the contributor's agreement (link in README)?

matsev commented on 19 Apr 2014 Contributor Author

I have previously filled in the contributor's agreement for the Core Spring Framework as part of a contribution to one of the tutorials. Do I need to sign another one for Spring Boot?

Confirmation number: 71020140106065740
See https://github.com/spring-guides/tut-data/pull/2#issuecomment-31654151

Reviewers
No reviews

Assignees
No one assigned

Labels
type: task

Projects
None yet

Milestone
1.0.2

Development
Successfully merging this pull request may close these issues.
maven-enforcer-plugin does not allow Java 1.8.0...

Notifications Customize
Subscribe
You're not receiving notifications from this thread.

2 participants

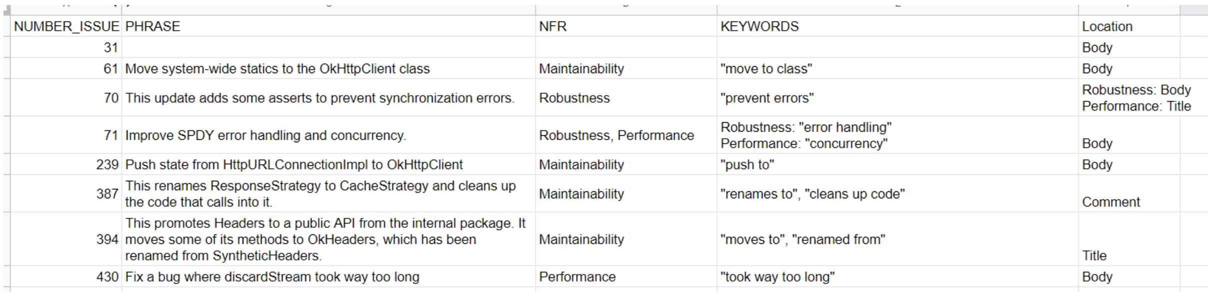
Analysis Process

Each participant must perform their assessment individually. For each pull request, the following fields must be filled in.

- **NUMBER_ISSUE:** the issue number evaluated
- **PHRASE:** The phrase (or phrases) that led her to identify the NFR
- **NFR:** For each instance of NFR, which NFR was identified. We are focusing on Security, Maintainability, Robustness and Performance. Note that an issue can have more than one NFR type.
- **Keywords:** The keywords that made her identify the NFR. It can be any set of keywords, either single words or short sentences.
- **Location:** Location of the NFR keyword:

    PR title
    PR description
    PR comments

Identification Example



NUMBER_ISSUE	PHRASE	NFR	KEYWORDS	Location
31				Body
61	Move system-wide statics to the OkHttpClient class	Maintainability	"move to class"	Body
70	This update adds some asserts to prevent synchronization errors.	Robustness	"prevent errors"	Robustness: Body Performance: Title
71	Improve SPDY error handling and concurrency.	Robustness, Performance	Robustness: "error handling" Performance: "concurrency"	Body
239	Push state from HttpURLConnectionImpl to OkHttpClient	Maintainability	"push to"	Body
387	This renames ResponseStrategy to CacheStrategy and cleans up the code that calls into it.	Maintainability	"renames to", "cleans up code"	Comment
394	This promotes Headers to a public API from the internal package. It moves some of its methods to OkHeaders, which has been renamed from SyntheticHeaders.	Maintainability	"moves to", "renamed from"	Title
430	Fix a bug where discardStream took way too long	Performance	"took way too long"	Body

Sample composition for qualitative analysis

Each reviewer must analyze a sample of X artifacts

These artifacts are composed of the pull request information. This pull request consists of: (i) title, (ii) description, (iii) comments

Systems to identify

Spring Security, Spring Boot, Spring Framework

NFR Complementary Material

Maintainability*: Refers to how well a product or system can be modified to improve, correct, or adapt to changes in the environment as well as requirements

Example

> WIP: enhanced errors
>
> this thing works and adds it to the CouchbaseResponse but its behind XERROR
> hello - need to check if thats intentional but shouldn't be tied to that
> change anyways.
>
> needs java-client *integration* too, and I think we may need to adapt CouchbaseException as well by giving it somehow more info? not too sure...

Robustness*: Defines the system's ability to recover from failures, stress or invalid system inputs

Example:

> The new generic streaming query parser is more *reliable* and makes it possible to avoid one-off response parser that are fragile.

Security*: Defines how the system should deal with parts of the system that can be attacked by malware or unauthorized access.

Example (Also includes discussion related to Robustness)

> JCBC-1096: Explicitly *handle auth errors* on observe.
>
> Motivation
> ----------
> With Server 5.0 and RBAC it is now possible that a user only has write access but not read access, leading to the situation where a PersistTo or ReplicateTo *might fail after* correctly storing the doc but then the user is not allowed to use the observe command.
>
> Modifications
> -------------
> This change explicitly *checks for authentication errors* and supplies a helpful message, including the enhanced error messages back to the user.
>
> Result
> ------
> If the user is performing an op and it now fails because of an auth issue, the error message is now clear.

Performance*: Defines how the system responds to certain user actions given a certain applied workload. It helps to explain, for example, the time a user will wait until the performed operation takes place (transaction, page rendering), when several users are accessing the system at the same time.

Example:

> JVMCBC-465: Harden AbstractEndpoint for concurrent reconnects.
>
> "Under certain circumstances (like failed auth connect), it is possible that multiple codepaths end up forcing the endpoint to reconnect. It is important that *race conditions* are avoided which might lead to too *many reconnects* or the endpoint ends up in a state which it should not belong in."

*Definitions based on ISO 25010