MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Source Code Quality impact on Pull Requests acceptance

Master's Thesis

ONDŘEJ KUHEJDA

Advisor: Assistant professor Bruno Rossi

Department of Computer Systems and Communications

Brno, Spring 2022

MUNI
FI

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ondřej Kuhejda

Advisor: Assistant professor Bruno Rossi

Acknowledgements

I would like to thank my supervisor, Bruno Rossi, for his guidance throughout the whole process.

Computational resources were supplied by the project "e-Infrastruktura CZ" supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Abstract

TODO

Keywords

code quality, pull request, static code analysis

Contents

1 Introduction

1.1 Problem statement

RQ$_1$ Which code issues are typically introduced by the pull requests?

RQ$_2$ Are there some particular issues/code smells that affect the pull request acceptance?

RQ$_3$ Is there a relationship between the source code quality and the pull request acceptance?

RQ$_4$ Does code quality influence the time it takes to close a pull request?

RQ$_5$ Is code quality impact higher in projects that are using some particular programming language?

2 Code quality in pull-based development

The pull-based development model created novel ways how can developers interact between each other. Instead of pushing code changes (patches) into one central repository, developers can work in more decentralized and distributed way. This is mainly done by using distributed version control systems such as Git. Git enables developers to clone repositories and thus to work independently on projects. Furthermore, the Git's branching model helps developers to keep track of repository changes and helps to handle the conflicts between the different changes of the same code base.

To furthermore ease the complicated process of resolving conflicts between different changes (of the same code base) and to provide a more user-friendly environment for developers, platforms such as GitHub was created. These platforms adds new ways how the developers can interact beyond the basic functionality of Git:

- The forks enables to create the server-side copy of the repository.
- Pull requests (on some platforms called merge requests) enables to merge code directly on the platform.
- Users can report issues found in the projects; therefore, platform can also serve as a bug-tracking system.
- The comments can be added to the pull requests and issues in order to build up social interaction between developers.
- Users can star projects and follow other users, projects, pull requests or issues.

In this study, I choose to use GitHub as the main source for data mining. GitHub is one of the leading platforms that enables pull-based collaboration between developers. GitHub hosts huge amount of publicly available repositories and GitHub also provides public REST API that can be easily leveraged for data mining.

The aim of this thesis is to obtain large amount of data about GitHub projects and analyze the pull request in regard of their code quality. How the code quality can be analyzed and how the GitHub

platforms contributes to quality of the code itself is discussed in the following chapters.

1. **TODO** cite: An Exploratory Study of the Pull-based Software Development Model

2.1 Code quality

Code quality is very important aspect of every program — software with high code quality has competitive advantage, is more stable and is also more maintainable then software which is poorly written.

To be able to evaluate the software in regard of its quality, there needs to be some way how can be code quality measured. The testing can be used exactly for this purpose — as a tool for measuring the quality of the source code. There are multiple ways how can be testing performed. Testing techniques can be divided into two categories: static and dynamic testing techniques.

In order to use dynamic testing techniques on large number of programs, there are two large obstacles — the program needs to be executed and there needs to be some inputs (with expected outputs) that can be then used for testing. Program execution can be problematic. Some programs needs to be compiled before they can be executed; others requires special environment for its execution (specific hardware, operating system or shared libraries required by the program). Moreover, the most of the programs does not have sets of input that can be used for testing. There exists some techniques that can be used also without the predefined inputs such as fuzzing, but these techniques are usually time-consuming. Because of that, dynamic testing techniques are not viable option when dealing with the large number of programs.

On the other hand, static testing methods suits the analysis of the large number of programs better. Static techniques include usage formal and informal reviews, walkthroughs and inspections; however, these techniques are performed by humans and therefore are not usable for large datasets. Because of that, in this thesis, the quality of the given source code is evaluated using the tools for automatic static analysis (called linters). Linters are used to find defects and code smells in the source code without the need of source code's execution.

There are several categories of issues which can be detected using linters. Source code can be checked if it follows a conventions of the given programming language. For instance, Python has an official style guide for Python code — PEP 8[1]. This guide defines the conventions that should be followed such as proper indentation of the code blocks, maximum line length or naming conventions.

Furthermore, code can be analyzed against refactoring related checks; for instance linter can detect if some part of the code is redundant and therefore could be omitted. Linters can also detect actual errors such as type mismatches or syntax errors.

However, it is important to note that not all linters have the same capabilities. Number of issues which can be detected by the given linter also heavily depends on the programming language of the studied source code. Which linters were used for the purposes of this thesis is discussed later in the text.

1. **TODO** cite: `https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf`

2.2 TODO GitHub

- GitHub issues and code quality
- Ways to merge code
  - An Exploratory Study of the Pull-based Software Development Model
- PRs and code review
- PRs CI/CD and code quality
  - Wait for It: Determinants of Pull Request Evaluation Latency on GitHub [1]
    * CI and latency

1. `https://www.python.org/dev/peps/pep-0008/`

- Trautsch et al. [2] analyzed several open-source projects in regards to usage of static analysis tools. They found out that incorporating a static analysis tool in a build process reduces the defect density.

3 Pull request acceptance

Pull request acceptance is a problem that has been studied multiple times. Several surveys were performed in order to understand why pull requests are being rejected.

Gousios et al. [3] surveyed hundreds of integrators to find out their reasons behind the PR rejection. Code quality was stated as the main reason by most of the integrators; code style was in the second place. Factors that integrators examine the most when evaluating the code quality are style conformance and test coverage.

Kononenko et al. [4] performed a study of an open-source project called *Shopify*; they manually analyzed PR's and also surveyed *Shopify* developers. They found out that developers associate the quality of PR with the quality of its description and with the revertability and complexity of the PR.

The reasons why contributors abandon their PRs were also studied [5]. The reason number one was the "Lack of answers from integrators."; moreover, the "Lack of time" and the "Pull request is obsolete" was also often stated as the main reason.

Even though the different open-source communities solve the problem of pull request acceptance in a different manner, three main governance styles can be identified — protective, equitable, lenient. Protective governance style values trust in the contributor-maintainer relationship. The equitable governance style tries to be unbiased towards the contributors, and the lenient style prioritizes the growth and openness of the community [6]. Each style focuses on different aspects of PR. Tsay et al. [7] identified the following levels of social and technical factors that influence the acceptance of the PR — *repository level*, *submitter level*, and the *pull request level*.

3.1 Repository level

The *repository level* is interested in the aspects of the repository itself, such as the repository age, number of collaborators, or number of stars on the GitHub.

For instance, the programming language used in the project also influences the acceptance of the PRs. Pull requests containing Java,

JavaScript, or C++ code have a smaller chance to be accepted than PRs containing the code written in Go or Scala [8].

Furthermore, older projects and projects with a large team have a significantly lower acceptance rate [7].

The popularity of the project also influences the acceptance rate — projects with more stars have more rejected PRs [7].

3.2 Submitter level

The *submitter level* is concerned about the submitter's status in the general community and his status in the project itself. There are several parameters that can be considered when evaluating the submitter's status.

PRs of submitters with higher social connection to the project have a higher probability of being accepted [7].

Submitter status in the general community plays an important role in PR acceptance. If the submitter is also a project collaborator, the likelihood that the PR will be accepted increases by 63.3% [7].

Moreover, users that contributed to a larger number of projects have a higher chance that their PR will be accepted [9].

The gender of the submitter is another factor that plays a role in PR acceptance. A study showed that woman's PR are accepted more often, but only when they are not identifiable as a woman [10].

Personality traits also influence PR acceptance. The *IBM Watson Personality Insights* were used to obtain the personality traits of the PR submitters by analyzing the user's comments. These traits were then used to study PR acceptance. It has been shown that conscientiousness, neuroticism, and extroversion are traits that have positive effects on PR acceptance. The chance that PR will be accepted is also higher when the submitter and closer have different personalities [11].

3.3 Pull request level

The *pull request level* is interested in the data that are connected to the PR itself. For instance, on the *PR level*, one can study if there is a correlation between PR acceptance and the number of GitHub

comments in the PR. Another parameter that can be used is "Number of Files Changed" or "Number of Commits".

One of the factors that negatively influence the acceptance rate is the already mentioned number of commits in the pull request. The high number of commits decreases the probability of acceptance. On the other hand, PR's with only one commit are exceptions — they have a smaller chance to be accepted than pull requests which contain two commits [9].

Another observation is that more discussed PR's has a smaller chance to be accepted [7]. Another study did not find a large difference between accepted and rejected PR's based on the number of comments but found that discussions in rejected PR's have a longer duration [12].

Proper testing is the crucial part of every project, and therefore it also influences the pull request acceptance. PR's including more tests have a higher chance to be accepted, and an increasing number of changed lines decreases the likelihood of PR acceptance [7].

Testing plays a significant role in discovering bugs and therefore leads to higher code quality. On the other hand, many test cases do not have to mean that code has a high quality. The code quality is an essential factor on the *pull request level*, therefore, is this study's main interest. Works that are also interested in the code quality and the pull request acceptance are examined in the following chapter.

Another factor that is closely tied to code quality is the code style. This factor has a small (but not negligible) negative effect on acceptance. This means that PRs with larger code style inconsistency (with the codebase) have a smaller chance of being accepted [13].

3.4 TODO Code quality

Although most integrators view code quality as the most important factor regarding PR acceptance, to the best of my knowledge, only one study was performed to discover whether there is a connection between the PR's acceptance and its quality.

- Does code quality affect pull request acceptance? [14]

3.5 TODO Unsorted

- study "Influence of Social and Technical Factors" [7] was replicated [11]
- Replication Can Improve Prior Results: A GitHub Study of Pull Request Acceptance [15]
  - contains interesting table with factors that influences acceptance
- Pull Request Decision Explained: An Empirical Overview [16]
  - also contains interesting table with factors that influences acceptance
- An Exploratory Study of the Pull-Based Software Development Model [17]
- Which Pull Requests Get Accepted and Why? A study of popular NPM Packages [18]
- Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates [19]
- Pull Request Prioritization Algorithm based on Acceptance and Response Probability [20]

3.6 TODO Create table that compares already performed studies with my thesis

4 Data mining

TODO: update graph Information about the pull requests are retrieved

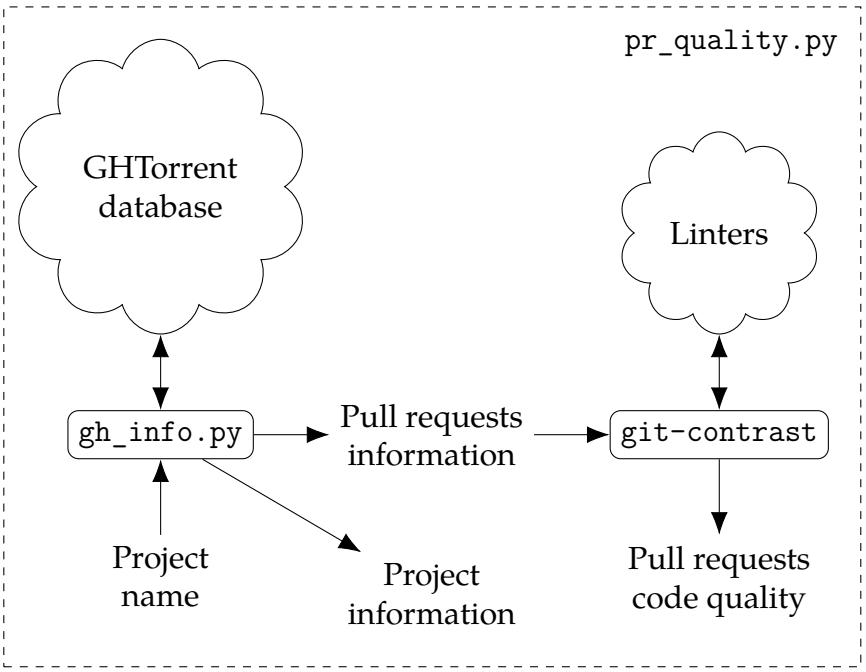

Figure 4.1: The `pr_quality.py` workflow

using the `pr_quality.py` script. This scripts takes names of the projects that will be analyzed as the input and it outputs the JSON files containing the requested data. This script uses internally two other scripts — `gh_info.py` and `git-contrast`.

`gh_info.py` is responsible for querying the GHTorrent database in order to obtain data about the projects. The GHTorrent database is an offline mirror of data offered through the Github REST API. `gh_info.py` returns a JSON file with the information about the project such as number of stars, number of contributors or information about pull requests and their commits.

However, the Github REST API lacks the information about the code quality of the pull requests. This is where the `git-contrast` comes into the play. `git-contrast` is the command-line application which analyzes the code quality of the given pull request using the

external linters. This application is further discussed in the following sections.

TODO: mention that data from REST API are not complete (GH API limit)

4.1 GHTorrent database

As stated before, the script called `gh_info.py` uses the GHTorrent database in order to retrieve GitHub data. GitHub REST API can be leveraged to obtain many interesting factors which can possibly influence the acceptance of pull requests. All the data that are obtained using the `gh_info.py` are listed in the following table:

Table 4.1: Data retrieved from the GHTorrent

Level	Variable	Factor
Repository level	Project name	✗
	Programming language	✓
	Time of creation	✓
	Number of forks	✓
	Number of commits	✓
	Number of project members	✓
	Number of watchers	✓
Submitter level	Username	✗
	Number of followers	✓
	Status in the project	✓
Pull request level	Pull request ID	✗
	Is PR accepted?	✓
	Time opened	✓
	Head repository	✗
	Head commit	✗
	Base commit	✗
	Number of commits	✓
	Number of comments	✓

Variables marked with ticks (✓) are factors that can possibly influence code quality and they can be used for pull request acceptance analysis. Other variables (✗) are not meant to be used as an part of an

data analysis itself, but are kept here for better orientation; and some of them are later used by the `git-contrast` tool (in order to pull the commits which will be subsequently analyzed by linters).

4.2 `git-contrast`

`git-contrast` is the command line application that I implemented in order to be able to analyze the code quality of the given pull request. `git-contrast` expects two commit hashes on the input and returns the information about the change in code quality between these commits on the output. This is done by running the linter on the files in the state of the first commit and then in the state of the second commit. The number of found code quality issues is then written to the standard output.

To measure the change of the quality in the pull request, we simple run the `git-contrast` on the "head commit" and the "base commit" of the given pull request. `git-contrast` supports several linters; which linter will be used is determined by the file extension of the tested file. Linters that are supported by `git-contrast` are listed in the following table:

Table 4.2: Linters supported by the `git-contrast`

Linter	Programming languages	File extensions
OCLint	C/C++	`.c`, `.cpp` and `.h`
HLint	Haskell	`.hs`
ktlint	Kotlin	`.kt` and `.kts`
PMD	Java	`.java`
Pylint	Python	`.py`

The most problematic was to statically analyze the C/C++ source files because some linters also need the information how the source code should be compiled. Luckily, this information can be usually automatically obtained from the makefiles. Another problem is the speed. At first, I was using the Cppcheck linter for the static analysis of C/C++ but I was forced to switch to the OCLint in order to shrink the total execution time of the static analysis.

4.3 TODO Projects selection

Criteria (data from 2019-06-01):

- is in the top 150 most favorite projects written in the given language
- 200+ pull requests and less then 5000
- `https://github.com/EvanLi/Github-Ranking`
- at least 85 % of files are source files written in the given language
- project is a program or program collection (not a book with the script etc.)
- `https://dl.acm.org/doi/abs/10.1145/2597073.2597122`
- `https://dl.acm.org/doi/abs/10.1145/3379597.3387489`
- `https://zenodo.org/record/3858046`
- `https://github.com/XLipcak/rev-rec`
- `https://ghtorrent.org/`
  - `https://github.com/gousiosg/pullreqs`
  - How can I cite this work? (on the web)
- Kalliamvakou et al. noted that data about PR's mined from GitHub are not always reliable, because PR can be also merged using several different approaches.
  - `https://dl.acm.org/doi/10.1145/2597073.2597074`
  - [17]

4.4 TODO Computational resources

5 Data analysis

In order to simplify analysis of retrieved data, I created the script that takes multiple JSON files with the data about each individual project and converts them into the CSV files. Each row in the CSV file represents some pull request. This script also filters the pull requests which are not suitable for the analysis — PRs that do not contains any source code written in the primary language or PRs that contained corrupted files (the linter was unable to analyze those files).

The retrieved data about the pull request were subsequently analyzed in order to answer my research questions. For this analysis was used the Python script[1] provided by Lenurdazzi et al. (this script was used for machine learning classification methods) and also my script written in R. The statistical methods that were used are discussed in the following paragraphs:

Research questions 1 to 4 were analyzed separately for each programming language.

TODO: At first, for **RQ_1**, I summarized the retrieved data for each project — I counted how many suitable pull requests were analyzed and how many of them are accepted/rejected. Then for each issue individually I computed how many accepted/rejected pull request introduced/fixed this issue, how many times this issue occurred in some pull request etc.

- `https://www.scribbr.com/statistics/statistical-tests/`

RQ_2:

- classification (machine learning)
- machine learning while taking into account only introduced issues vs quality change
- correlation matrix

RQ_3:

- PCA scatterplot

1. `https://figshare.com/s/d47b6f238b5c92430dd7`

- contingency matrices
- contingency matrices for PR's that contain only modified source code files vs for all of them
- contingency matrices separately for each project
- ROC curves and AUCs

RQ$_4$:

- regression (machine learning)
- correlation matrix

RQ$_5$:

- Compare results from previous steps.
  - Statistically compare the parameters obtained for each programming language. (check if the pull requests from different languages and retrieved parameters follow the same distributions)
- What is the effect of the programming language on the acceptance and time it takes to close a pull request?
  - ANOVA
  - classification (machine learning)

6 Evaluation

TODO

6.1 Python

In order to analyze the influence of code quality on the pull request acceptance, the following projects from the Python ecosystem were selected:

Table 6.1: Python projects

Project	Stars	Analyzed PRs	Accepted	Rejected	Introduced issues	Fixed issues
pallets/flask	58380	500	76%	24%	2.82	1.88
rg3/youtube-dl	50768	808	49%	51%	5.34	2.11
psf/requests	47100	500	64%	36%	2.76	1.04
nvbn/thefuck	46148	268	47%	53%	6.61	0.98
scrapy/scrapy	43124	500	80%	20%	5.63	5.39
faif/python-patterns	31006	258	90%	10%	4.20	5.30
certbot/certbot	28785	500	81%	19%	3.62	1.60
openai/gym	26986	500	68%	32%	7.12	3.05
soimort/you-get	25437	487	56%	44%	7.77	1.91
explosion/spaCy	23007	500	91%	9%	3.68	3.79
pypa/pipenv	22785	500	86%	14%	5.62	1.78
keon/algorithms	20528	341	81%	19%	11.42	9.82
tornadoweb/tornado	20451	500	80%	20%	3.18	1.36
keras-team/keras	20384	398	53%	47%	4.88	3.49
celery/celery	18850	500	76%	24%	3.60	1.24
locustio/locust	18518	496	74%	26%	9.02	3.90
sanic-org/sanic	15958	500	81%	19%	4.86	1.95
spotify/luigi	15485	500	72%	28%	5.62	2.96
kivy/kivy	14471	500	89%	11%	2.50	1.18
powerline/powerline	13187	396	81%	19%	23.88	4.10

In total, 9452 pull requests were analyzed and 73 % of these PRs was accepted. Pull request were more accepted in less popular projects.

At average, one pull request introduced 5.36 issues and fixed 2.44 issues; accepted pull request introduced 4.62 and fixed 1.99 issues, and rejected pull request introduced 7.86 issues and fixed 4.43 on average. 5 % trimmed mean was used to compute these values.

The Pylint, which was used to lint those projects, classifies issues into the following categories:

Table 6.2: Pylint issue categories

Category	Introduced in total	#PRs which introduced	Fixed in total	#PRs which fixed
warning	48931	3350	36865	1910
error	24657	2540	18841	1310
convention	91770	4683	76324	2447
refactor	16317	2543	14964	1483
info	2	1	2	1

In the analyzed pull requests, Pylint detected 222 different issues.

The list of issues that was fixed/introduced in the largest number of pull request was dominated by the conventions. The convention that was fixed/introduced in the largest number of pull requests is `missing-function-docstring` (in 37 % of PRs); also conventions `invalid-name`, `line-too-long` and `consider-using-f-string` were fixed/introduced in over then 20 % of pull requests. There were 15 issues that were fixed/introduced in more then 10 % of PRs and 72 issues were in over 1 % of PRs (out of the 222 issues which were found in the pull requests). There were 9 issues which was present in the analyzed pull requests but did not influence their quality (number of these issues was not changed by any pull request). 13 issues were introduced/fixed in only one pull request and 10 of them are issues classified as errors. The most common error is `import-error` (24 % of PRs); however, I suspect that there will be a lot of false positives which aroused due to linting in the isolated environment. 60 issues were fixed in more PR's then they were introduced. They are 24 more PRs that fixed the warning `super-init-not-called` than the PR's that introduced it.

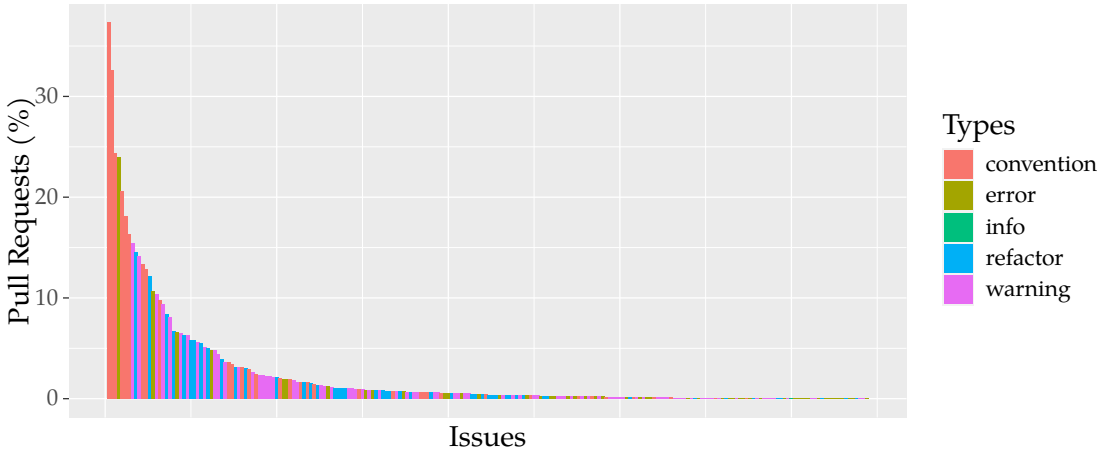

Figure 6.1: Pylint issues and % of PRs which fixed/introduced them

The most important Pylint issue in regards of the PR acceptance is the `syntax-error`. XGBoost classifier gives this error the 1 % importance. However, other classifiers consider this error less important. The average importance of the `syntax-error` is only 0.4 %. **TODO**: investigate the `syntax-error` in more detail

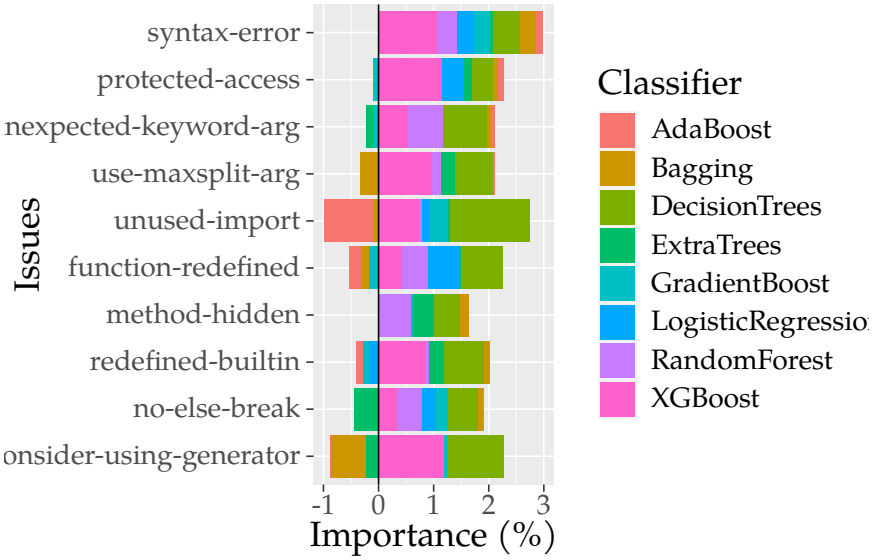

Figure 6.2: Ten most important Pylint issues

6.2 Threads to validity

- PR recognition (rejected PRs can be merged using another way)
- modified files in the PR's that are not written in the primary programming language can influence acceptance
- chosen projects
- PR's filtering
  - linter errors
  - limit of 500 PRs
  - limited execution time for `git-contrast`
  - not available PR's/repositories
- false positives from Linters (`import-error`, `relative-beyond-top-level`)

7 Conclusion

7.1 Future work

8 Appendix

Bibliography

1. YU, Y.; WANG, H.; FILKOV, V.; DEVANBU, P.; VASILESCU, B. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In: *Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 367–371. Available from DOI: 10.1109/MSR.2015.42.
2. TRAUTSCH, A.; HERBOLD, S.; GRABOWSKI, J. A longitudinal study of static analysis warning evolution and the effects of PMD on software quality in Apache open source projects. *Empirical Software Engineering*. 2020, vol. 25, pp. 5137–5192. Available from DOI: 10.1007/s10664-020-09880-1.
3. GOUSIOS, G.; ZAIDMAN, A.; STOREY, M.; DEURSEN, A. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In: *International Conference on Software Engineering*. IEEE, 2015, vol. 1, pp. 358–368. Available from DOI: 10.1109/ICSE.2015.55.
4. KONONENKO, O.; ROSE, T.; BAYSAL, O.; GODFREY, M.; THEISEN, D.; WATER, B. Studying Pull Request Merges: A Case Study of Shopify's Active Merchant. In: *International Conference on Software Engineering*. ACM, 2018, pp. 124–133. Available from DOI: 10.1145/3183519.3183542.
5. LI, Z.; YU, Y.; WANG, T.; YIN, Gang; LI, Shanshan; WANG, Huaimin. Are You Still Working on This An Empirical Study on Pull Request Abandonment. *Transactions on Software Engineering*. 2021, pp. 1–1. Available from DOI: 10.1109/TSE.2021.3053403.
6. ALAMI, A.; COHN, L.; WĄSOWSKI, A. How Do FOSS Communities Decide to Accept Pull Requests? In: *Proceedings of the Evaluation and Assessment in Software Engineering*. ACM, 2020, pp. 220–229. Available from DOI: 10.1145/3383219.3383242.
7. TSAY, J.; DABBISH, L.; HERBSLEB, J. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In: *International Conference on Software Engineering*. ACM, 2014, pp. 356–366. Available from DOI: 10.1145/2568225.2568315.

8. SOARES, D.; DE LIMA, M.; MURTA, L.; PLASTINO, A. Acceptance Factors of Pull Requests in Open-Source Projects. In: *Symposium on Applied Computing*. ACM, 2015, pp. 1541–1546. Available from DOI: 10.1145/2695664.2695856.

9. DEY, T.; MOCKUS, A. Effect of Technical and Social Factors on Pull Request Quality for the NPM Ecosystem. In: *International Symposium on Empirical Software Engineering and Measurement*. ACM, 2020. Available from DOI: 10.1145/3382494.3410685.

10. JOSH, J.; KOFINK, A.; MIDDLETON, J.; RAINEAR, C.; MURPHY-HILL, E.; PARNIN, C.; STALLINGS, J. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*. 2017, vol. 3. Available from DOI: 10.7717/peerj-cs.111.

11. IYER, R.; YUN, A.; NAGAPPAN, M.; HOEY, J. Effects of Personality Traits on Pull Request Acceptance. *Transactions on Software Engineering*. 2019. Available from DOI: 10.1109/TSE.2019.2960357.

12. GOLZADEH, M.; DECAN, A.; MENS, T. On the Effect of Discussions on Pull Request Decisions. In: *Belgium-Netherlands Software Evolution Workshop*. CEUR Workshop Proceedings, 2019. Available also from: http://ceur-ws.org/Vol-2605/16.pdf.

13. ZOU, W.; XUAN, J.; XIE, X.; CHEN, Z.; XU, B. How does code style inconsistency affect pull request integration? an exploratory study on 117 github projects. *Empirical Software Engineering*. 2019, vol. 24, pp. 3871–3903. Available from DOI: 10.1007/s10664-019-09720-x.

14. LENARDUZZI, V.; NIKKOLA, V.; SAARIMÄKI, N.; TAIBI, D. Does code quality affect pull request acceptance? An empirical study. *Journal of Systems and Software*. 2021, vol. 171, pp. 110806. Available from DOI: 10.1016/j.jss.2020.110806.

15. CHEN, D.; STOLEE, K.; MENZIES, T. Replication Can Improve Prior Results: A GitHub Study of Pull Request Acceptance. In: *International Conference on Program Comprehension*. IEEE, 2019, pp. 179–190. Available from DOI: 10.1109/ICPC.2019.00037.

16. ZHANG, X.; YU, Y.; GOUSIOS, G.; RASTOGI, A. Pull Request Decision Explained: An Empirical Overview. *Computing Research Repository*. 2021. Available from arXiv: 2105.13970.

17. GOUSIOS, G.; PINZGER, M.; DEURSEN, A. An Exploratory Study of the Pull-Based Software Development Model. In: *International Conference on Software Engineering*. ACM, 2014, pp. 345–355. Available from DOI: 10.1145/2568225.2568260.

18. DEY, T.; MOCKUS, A. Which Pull Requests Get Accepted and Why? A study of popular NPM Packages. *Computing Research Repository*. 2020. Available from arXiv: 2003.01153.

19. SOARES, D.; DE LIMA, M.; MURTA, L.; PLASTINO, A. Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates. In: *International Conference on Machine Learning and Applications*. IEEE, 2015, pp. 960–965. Available from DOI: 10.1109/ICMLA.2015.41.

20. AZEEM, I.; PENG, Q.; WANG, Q. Pull Request Prioritization Algorithm based on Acceptance and Response Probability. In: *International Conference on Software Quality*. IEEE, 2020, pp. 231–242. Available from DOI: 10.1109/QRS51102.2020.00041.