

1 Usage and structure of continuous integra-
2 tion as configuration?

3 Joseph Ling
j1653@kent.ac.uk



School of Computing
University of Kent
United Kingdom

Word Count: 6,100

4 March 10, 2020

6 This paper describes a simple heuristic approach to solving large-scale con-
7 straint satisfaction and scheduling problems. In this approach one starts
8 with an inconsistent assignment for a set of variables and searches through
9 the space of possible repairs. The search can be guided by a value-ordering
10 heuristic, the *min-conflicts heuristic*, that attempts to minimize the num-
11 ber of constraint violations after each step. The heuristic can be used with
12 a variety of different search strategies. We demonstrate empirically that on
13 the n -queens problem, a technique based on this approach performs orders of
14 magnitude better than traditional backtracking techniques. We also describe
15 a scheduling application where the approach has been used successfully. A
16 theoretical analysis is presented both to explain why this method works well
17 on certain types of problems and to predict when it is likely to be most
18 effective.

1 Introduction

<https://arxiv.org/ftp/arxiv/papers/1703/1703.07019.pdf>

Continuous integration (CI) is becoming more popular over the last few years. This can be seen by how major version control hosting services Github, Bitbucket and Gitlab have all started to or have been improving their CI product. In terms of research, configuration as code Rahman, Mahdavi-Hezaveh and Williams (2019) and continuous integration Copeland (2010) with Shahin, Ali Babar and Zhu (2017) demonstrating breadth of the research.

Continuous integration is a process of automatically running compiling, running tests and checking that the product works. This can be combined with Continuous Delivery where the product is deployed or released after it has gone through CI.

This can get complicated quickly therefore configuration as code (or infrastructure as code) is used to configure it. The main kind of configuration format used for this is yaml (reference to what it is??) followed by xml and java based scripting formats.

In terms of looking at usage we are going to do a similar look at the data as did Michael Hilton, Marinov and Dig (2016). The important aspect will be looking at how usage has changed over the last 5 years along with looking more closely at which repositories are more likely to use CI/CD. For this we are going to focus on the following research questions:

- What percentage of open-source projects use CI?
- multiple CI used
- what is the breakdown of usage of different services?
- Do certain types of projects use CI more than others?

This should give us a better understanding of the sample of repositories from Github. From there we look at the structure of the configuration files to understand how certain aspects of it are used.

- configuratizon errors when loading the config (just yaml parsing errors atm)
- how are comments used in the configuration?
- how scripts with the configuration files? (need to elloborate more on this one)

A key aspect is that these questions do not look too deeply into the individual implementation of each CI system. This is because there are already some good papers looking Gallaba and McIntosh (2018) at this but in order to be able to compare the different configuration types it is important to compare similar attributes (there is also a time factor in here as well).

2 Previous Works

2.1 Continous integration

Continous integration is the frequent submission of work normally tied into a feedback loop. For example using version control daily committing changes. That then a server builds and tests the changes informing you of status of those cahnges. The generally agree upon detailed definition is Fowler (2010).

2.2 Usage of continous integration

The actual usage of continous integration as configuration was looked at by Michael Hilton, Marinov and Dig (2016). In this they use three source of information github repositories, travis builds and a survery. In order to be do a more systematic study of CI usage than Vasilescu et al. (2015). In analysing that data they found that "The trends that we discovered point to an expected growth of CI. In the future, CI will have an even greater influence than it has today.". As we are looking at the same question we will use four of the same research questions out of the fourteen. In order to see what difference four years has made to the growth of usage of CI.

74 2.3 Config as code

75 Configuration as code or infrastructure as code has been an increasing area
76 of research over the last few years. There seems to be slightly more research
77 in infrastructure as code Rahman, Mahdavi-Hezaveh and Williams (2019).
78 There has been a focus on Puppet and Chef, for example in Sharma, Frangkoulis
79 and Spinellis (2016) looks at code quality by the measure of "code smell" of
80 Puppet code. This tackles the problem by defining by best practices and
81 analyzing the code against that. In the case of Cito et al. (2017) it uses
82 the docker linter in order to be able to analyse the files. For the continuous
83 integration systems we pick we will look into the tooling around that to aid
84 the analysis.

85 3 Methodology

86 In order to get repositories with CI/CD configuration from Github we have a
87 number of approaches. The first is to use the search for particular files but
88 this is limited to only 1000 results. The alternative is to search for repositories
89 and we bypass the 1000 result limit to an extent by getting results for every
90 'star' count (stars are used to like or upvote a repository). Although this will
91 be giving us a lot of results it will still only be a sample of the population but
92 will give us a wider range of results. As there is rate limiting multiple github
93 api keys can be used to speed up the scraping of data (gitter could also
94 be used to speed up the process I think).

95 After we have got a repository we need to get the CI/CD files from it.
96 This is fairly easy as the CI/CD systems normally require a strict naming
97 convention and location within the repository. However as most of them are
98 yaml based you can have ".yaml" and ".yml" and users can use all sorts of
99 mixtures of upper and lower case. We try to account for this but won't get
100 every scenario. This combined with the fact that we are only looking for
101 top configuration files based on github (2017) along with github actions and
102 azure pipelines. Is why we also check repositories for their README.md file

103 to check if it has a build tag.

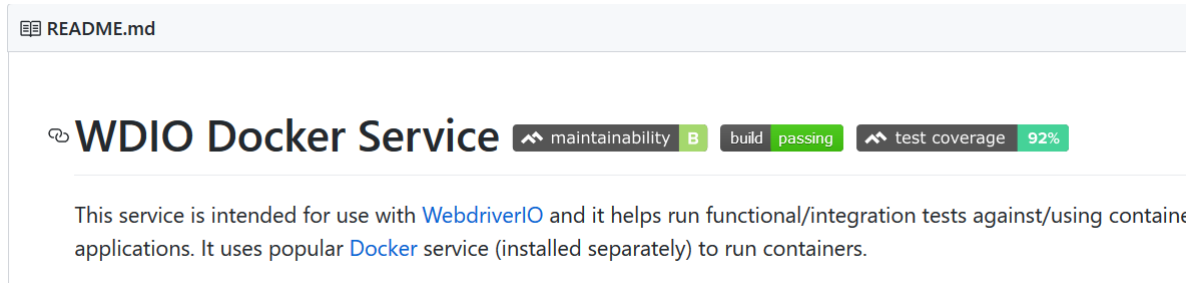


Figure 1: Example of CI tag for Github ReadMe

where did this image come from??
reference it man

104
105 In doing so it should give a wider net when sampling and help to under-
106 stand when a CI system is either not using configuration as code or using a
107 different CI system.

108 There are dangers in scraping data off github in terms of assumptions to
109 do with the population as found in Kalliamvakou et al. (2014). Our dataset
110 does not contain any forked repositories. But due to time constraints number
111 of commits and frequency of recent commits has not been looked at. This
112 would be an interesting area of further research in order to improve the
113 quality of the sample but also to look at how that affects the frequency of
114 CI usage.

115 Additionally the assumption that all repositories are of programming
116 projects with code in them is wrong. A number of repositories can be used
117 for storage, experimental, academic and other things. However they to all
118 some extent can use CI/CD for their work as a number of books were found
119 when looking through the dataset could use CI/CD.

120 Tooling for the configuration files, I looked into Travis, Github Actions
121 and Jenkins to work out whether or not it could aid in the research or not.
122 As a key part of understanding the first relies on knowing whether or not it is
123 valid. In terms for travis there is currently two parsers to validate the config-
124 uration. One which is deprecated since 2017 travis (2017) the other which is
125 currently in development travis (2020). Both didn't provided the necessary
126 results with the most recent one not being able to handle default fields. For

127 Github Actions as it's still a new tooling for it hasn't been developed out-
128 side of the Github editor web page ([https://github.community/t5/GitHub-](https://github.community/t5/GitHub-Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557)
129 [Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557](https://github.community/t5/GitHub-Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557)).
130 For Jenkins which is older solution allows validation through http/ssh request
131 to the Jenkins server (Gitlab follows this style as well) Jenkins (2020) Gitlab
132 (2020). This could work well although would require setting up a server for
133 each configuration type and might not validate if variables from the config
134 aren't defined on the server. As well as it would be best to be able to validate
135 them all or none of them in terms of being able to compare results easily.

136 4 Usage of CI

137 4.1 What percentage of open-source projects use CI?

138 Based a search for configuration as configuration files for the following CI
139 systems: Travis, Gitlab, Azure, App Veyor, Drone, Jenkins, Github, Circleci,
140 Semaphore, Teamcity and buildkite. Wrecker got bought by Oracle and from
141 doing a search on Github for what I think based on the docs (docs: Wrecker
142 and Oracle (2018) and search: GitHub (2020)) for their config file naming
143 convention. I was only able to find 20 results so did not include in the scraping
144 script to speed up the process of searching for the other configuration file
145 formats.

CI/CD	count	repos with config	no. multiple	multiple percent
config file(s)	12128	38.51%	1675	13.81%
found in ReadMe	873	2.77%		
none found	18493	58.72%		

Table 1: Percentage of CI used for projects

146 Our sample of repositories is 31,494 in comparison to Michael Hilton,
147 Marinov and Dig (2016) which had a sample of 34,544. The percentage of
148 CI projects they had was 40.27%. As if you combined the "config file(s)"

149 and "found in ReadMe". However in order to work out if a project might be
150 using CI but the config file wasn't picked a search string is used. Therefore
151 it is not as accurate as finding a config file as their could be false postives.

152 However that doesn't give us too much insight into the dataset. Here is a
153 graph showing the subscribers plotted against the number of stars. The key
154 here to understand is not potentially any correlation but to see the spread
155 of data that the table is showing.

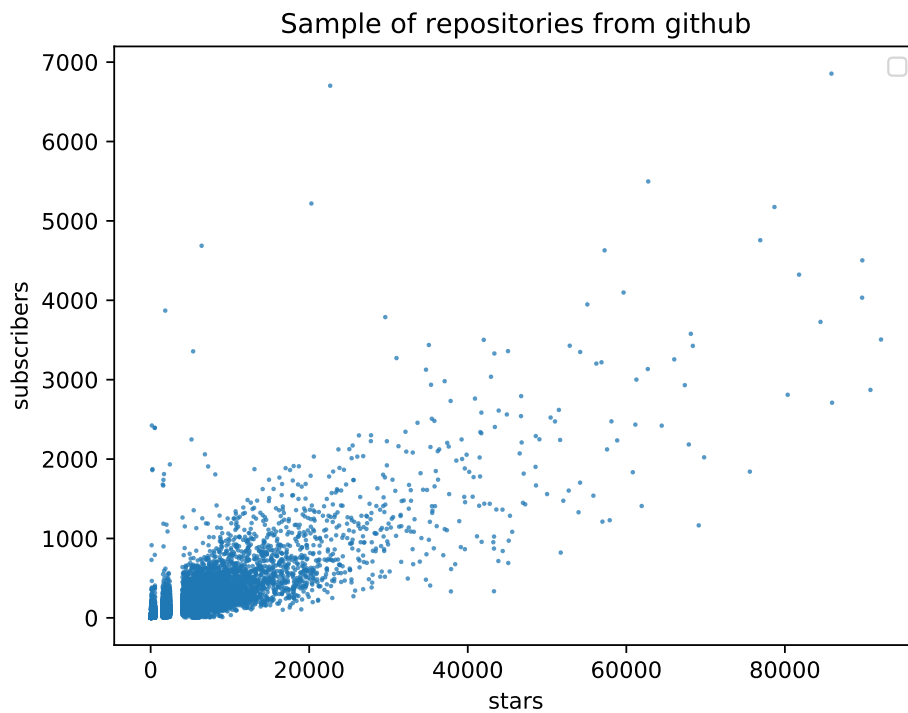


Figure 2: Scatter graph of Github stars against subscribers

156 Figure 2 helps give a understanding to the give a depth of the data for
157 where the graph is just blue. This is because on Github you get more repos-
158 itories with smaller star counts than large ones.

159 Figure 3 provides insight into the density of the data for between 0 to
160 25000.

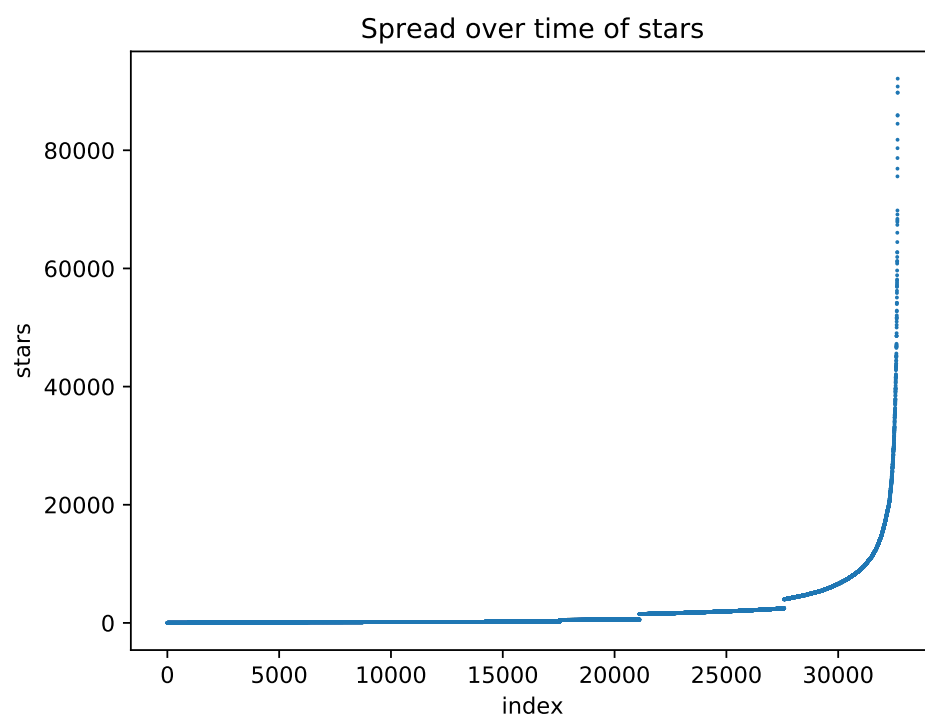


Figure 3: Stars graph

161 4.2 What CI systems are projects using?

162 Like all other research travis is the most popular CI system in use. However
163 over the last 4 years since the github (2017) Circleci has lost out on it's rough
164 quarter that it owned. In particular the rise of github actions seems to have
165 taken second place even though it is still very young in comparison (DATES).
166 However this might not be down to the Circleci loosing out on their existing
167 share. But potentially as the rise in CI usage goes up on github. Projects
are more likely to pick in the built in solutions to github.

Table 2: Configuration types spread

	config	percentage
travis	10607	74%
github	2301	16%
circleci	1109	8%
jenkinsPipeline	161	1%
drone	84	1%
buildkite	32	0%
teamcity	4	0%
semaphore	2	0%
azure	1	0%

4.3 Do certain types of projects use CI more than others?

Below shows all the CI projects sorted then grouped together per 540 projects. Then in this case we choose to categories via star count for each project.

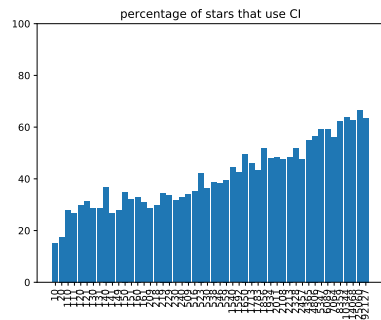


Figure 4: 2020 dataset

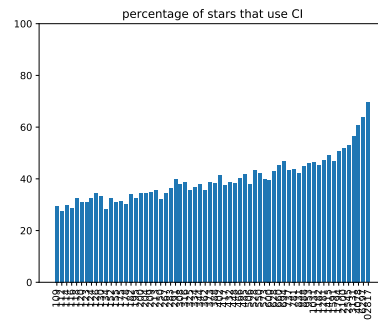


Figure 5: 2016 dataset

In Figure 4 is the results from this research and in Figure 5 is the results from Michael Hilton, Marinov and Dig (2016).

Here we are comparing whether or not in the last 4 years the number of stars increases the CI being used. Their seems to a steeper gradient in the more recent datasets. However as 4 starts at zero stars and 5 starts at 100 stars their is signifacant dip at the start of the first graph.

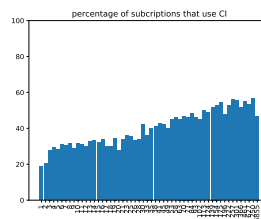


Figure 7: Subs graph

Figure 7 uses the same method as Figure TODO SORT except is does it based the number of subscribers. Subscribers are used on github to keep

179 update on the changes on the project. This ranges from core team memem-
180 bers working on the project to people that want to be notified about a new
181 release. In looking at this metric the hypothesis was that it would have a
182 sharper rise in percentage of projects using CI per subscriber. However that
183 was not the case overall the gradient is not as strong. There is no compar-
184 isson to Michael Hilton, Marinov and Dig (2016) because their final corpus
185 does not contain subscriber count for each project.

186 5 Config file results

187 5.1 configuration errors when loading the config (just 188 yaml parsing errors atm)

Composer error In the example it has two steps that are using an yaml anchor. This allows for the yaml below it to be referenced somewhere else. However if you define the anchor twice it causes a composer error. As you have two references for the samething so it won't know which one to use.

```
definitions:
  steps:
    - step: &build-test
      name: Build and test
      script:
        - mvn package
    - step: &build-test
      name: deploy
      script:
        - ./deploy.sh target/my-app.jar
```

Scanner error The first step of loading the yaml is to scan it to create the tokens. However invalid characters such as "\t" are invalid.

```
definitions: \t
```

189 As can be seen in the table their our configuration files with yaml errors
190 meaning that the CI for that project will not load. Yet it seems that a
191 very small percentage of projects that have them. For example the two

Parse error In this example it has scanned the file and created tokens for the syntax. Now it parses the syntax and works out if each token is valid given it's current context. In this case a closing] without an opening [is invalid.

```
definitions: ]
```

Table 3: yaml configuration errors

config	composer error	constructor error	parse error	scanner error	no. config
circleci	1	0	0	1	1109
drone	31	0	0	0	84
github	0	1	0	3	2301
travis	6	0	10	21	10607
buildkite	0	0	0	0	32
semaphore	0	0	0	0	2
azure	0	0	0	0	1

highest configuration types with errors are drone (36.90%) followed by travis (0.348%).

In the case for drone all the errors are for the same type of error. Potentially this could be because of how anchors are a lot more common in drone.

For travis it is the most common form of CI found therefore it is more likely to contain more errors. Yet with such a small amount it seems like yaml errors aren't a major problem in CI. Although as they are required to be fixed in order for the CI to run the chances of it working are higher and a more detailed study would need to be done.... ah

5.2 How are comments used in configuration?

202 The assumption was the as continuous integration setups can be compli-
203 cated and have edge cases. Therefore comments would be used to describe
204 and handle that complexity.

205 An example configuration file below for Github actions using the default
206 template slightly altered. Shows two examples of comment usage, the first
207 being including useful information about why a particular version of the
208 programming language was chosen. The second is that the tests have been
209 disabled by commenting them out.

In order to pick up on all these different types of comments. All the CI files were parsed and then regular expressions were used to pick on up key factors such as "note:". Along with multiple single line comments which made up a block/multi-line comment.

For example in to the left there is an example Github Action yaml file. If were it would be parsed we would get: one multi line comment, 15 lines of code, 1 single line comment, a total of 5 comments and 20 lines in the file. Therefore their is a their is a raito of 4:1 for code in this config file.

```
name: Python package
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v1
        # note: only works with python 3
        with:
          python-version: 3.8
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      #
      # - name: Test with pytest
      #   run: |
      #     pip install pytest
      #     pytest ./src
```

210 line count box plots for each config type

211 5.3 How are script tags used?

212 - how scripts with the configuration files? (need to elloborate more on this
213 one)

Analysis of yaml continous integration file formats

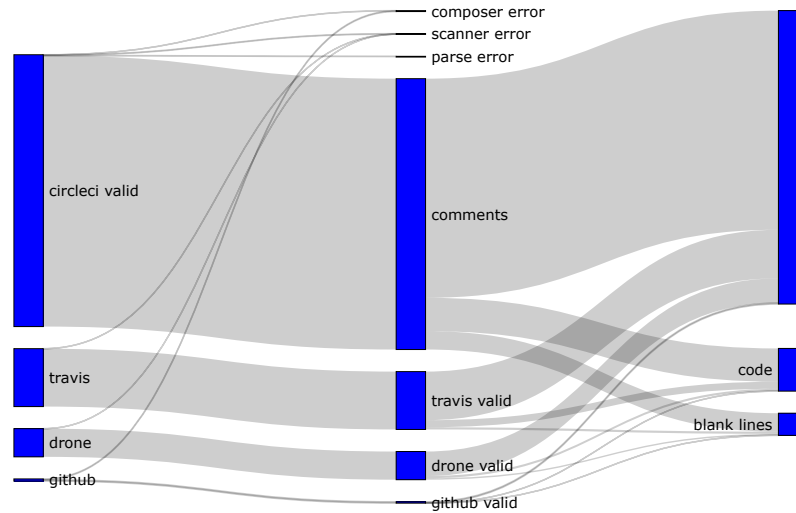


Figure 8: Sankey daigram

214 6 Threats to validatity

215 strength and validalti section - possible issues - baise assume in the data -
 216 e.g. sampling vai star - focusing on github - problematic of scraping tool

217 7 Summary

218 asdfasdf

219 55df26ae2061a09c5830423efc280783897fe8c9

220 7.1 Discussion and further research

221 In the process of writing this paper we kept on considering more research
 222 questions. As there is a lot of meta data that you can get for a single
 223 project, in addition to what was used for this paper.

224 Further research into usage that we would like to do is look into how
225 the size of the project affects the chance that it uses CI. Then looking at
226 the usage of scripts within CI configuration, for example using a script tag
227 to run a shell script. As while doing the research we found some projects
228 use scripts a lot while others just used the CI config. This would lead to
229 questions around which CI system has a higher amount of scripts used. But
230 also looking at how much they enable them to be used and what is the size
231 of those scripts. The data for the programming language and version(s) is in
232 the config. Therefore it would be possible to work out how much usage each
233 version is getting of a particular programming language.

234 Further research into structure could look into the naming of each part
235 of the build process that is used. This would be interesting as it would
236 provided insight into what terms are commonly used. As well an idea into
237 how people plan or don't plan out their configuration files. Additionally CI
238 systems can be designed to run on every commit to version control or only
239 commits to certain branches. Therefore by looking at the branching regexp
240 that are being used an better understanding of how branches are actually
241 used in software development where CI is also used could be found out.

242 In addition working on pruning our dataset using methods outlined in
243 Kalliamvakou et al. (2014).

244 8 Acknowledgement

245 The authors wish to thank Hans-Martin Adorf, Don Rosenthal, Richard
246 Franier, Peter Cheeseman and Monte Zweben for their assistance and ad-
247 vice. We also thank Ron Musick and our anonymous reviewers for their
248 comments. The Space Telescope Science Institute is operated by the Associ-
249 ation of Universities for Research in Astronomy for NASA.

Appendix A. Probability Distributions for N-Queens

[section omitted]

References

- Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem on GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 323–333, iSSN: null.
- Copeland, P. (2010). Google’s Innovation Factory: Testing, Culture, and Infrastructure. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, Washington, DC, USA: IEEE Computer Society, ICST ’10, pp. 11–14.
- Fowler, M. (2010). Continuous integration. In <https://www.martinfowler.com/articles/continuousIntegration.html>.
- Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects that (mis)use Travis CI. *IEEE Transactions on Software Engineering*, pp. 1–1.
- github (2017). <https://github.blog/2017-11-07-github-welcomes-all-ci-tools/>. In github.com, ed., *github welcomes all ci tools*.
- GitHub (2020). github filename search for wrecker.yml files. In *github filename search for wrecker.yml files*.
- Gitlab (2020). <https://docs.gitlab.com/ee/api/lint.html>. In *Gitlab docs*.
- Jenkins (2020). <https://jenkins.io/doc/book/pipeline/development/>. In *Jenkins documentation*.

274 Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M. and
275 Damian, D. (2014). The promises and perils of mining GitHub. Hyderabad,
276 India: Association for Computing Machinery, MSR 2014, pp. 92–101.

277 Michael Hilton, K. H., Timothy Tunnell, Marinov, D. and Dig, D. (2016).
278 Usage, costs, and benefits of continuous integration in open-source projects
279 | Proceedings of the 31st IEEE/ACM International Conference on Auto-
280 mated Software Engineering.

281 Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A systematic
282 mapping study of infrastructure as code research. *Information and Soft-*
283 *ware Technology*, 108, pp. 65–77.

284 Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration, De-
285 livery and Deployment: A Systematic Review on Approaches, Tools, Chal-
286 lenges and Practices. *IEEE Access*, 5, pp. 3909–3943.

287 Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Configuration
288 Code Smell? In *2016 IEEE/ACM 13th Working Conference on Mining*
289 *Software Repositories (MSR)*, pp. 189–200, iSSN: null.

290 travis (2017). travis yaml (old repository). In [https://github.com/travis-](https://github.com/travis-ci/travis-yaml/)
291 [ci/travis-yaml/](https://github.com/travis-ci/travis-yaml/).

292 travis (2020). travis yaml new implementation. In [https://github.com/travis-](https://github.com/travis-ci/travis-yml/)
293 [ci/travis-yml/](https://github.com/travis-ci/travis-yml/).

294 Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015). Quality
295 and productivity outcomes relating to continuous integration in GitHub.
296 Bergamo, Italy: Association for Computing Machinery, ESEC/FSE 2015,
297 pp. 805–816.

298 Wrecker and Oracle (2018). Wrecker ci development blog. In *Wrecker CI*
299 *development blog*.