

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

3 Joseph Ling
jl653@kent.ac.uk



School of Computing
University of Kent
United Kingdom

Word Count: around 5500

4 April 2, 2020

6 Continuous integration (CI) is becoming more popular as software develop-
7 ment moves to an Agile fast paced development life cycle. Most CI is done
8 automatically using a service which run based off configuration. Our major
9 questions is how much is CI acutally being used? As well as how are these files
10 being structured? We got 31,494 open source projects from Github to answer
11 these questions. In doing so compared our results against Michael Hilton,
12 Marinov and Dig [17] work to see if their has been a increase in usage. We
13 found a shift in CI services being used and were able to get similar results
14 to their study. In terms of structure we found that configuration files are
15 written with no comments normally. We suggest at the end further research
16 is needed to get a better understanding of this growing field.

similar is a
bad word to
use to de-
scribe the
comparison

1 Introduction

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, Infrastructure as Code in Rahman, Mahdavi-Hezaveh and Williams [18] which does a systematic mapping of research in that area. For Continuous Integration with Shahin, Ali Babar and Zhu [19] which does another systematic review on how it is used. These two papers demonstrate some of breadth of research that has taken place. In addition you have papers like Google's Innovation Factory: Testing, Culture, and Infrastructure Copeland [9] which demonstrate some of the depth that the papers go into.

Continuous Integration is a process of automatically running compiling, running tests and checking that the product works. This is can be combined with Continous Delivery where the product is deployed or released after it has gone through successfully 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 followed by Xml and Java based scripting formats.

In order to look at our first theme CI usage we looked at In Usage, Costs, and Benefits of Continuous Integration Open-Source Projects [17]. They looked closely at usage of CI as well. As we are looking at CI usage as well we are going answer the first three questions from their theme "Usage of CI".

- **RQ1** What percentage of open-source projects use CI?
- **RQ2** What is the breakdown of different CI services?
- **RQ3** Do certain types of projects use CI more than others?

However the two key differences is that we will be scraping a new data set for the comparison. In doing so gathering slightly more data on the

repositories but not none on pull requests. As well as we didn't conduct a survey. From that additional data we are going to look more closely at the first question of What percentage of open-source projects use CI? As we are asking the same questions, we will use their corpus to compare on what has changed over the last 4 years. For our second theme, structure of CI as configuration we wanted to pick structural components that would be similar between all CI files. It would have been really interesting to do a full in depth analysis of each like Gallaba and McIntosh [11]. However we would like to tie in how the files are structured to how they are used so won't following that style. This led to the following research questions:

- **RQ4** What are the common errors when loading yaml configuration?
- **RQ5** How are comments used in the configuration?
- **RQ6** How are external scripts used within the configuration?

2 Related Works

2.1 Continous Integration

Continous Integration is frequently submitting work normally tied into a feedback loop. For example using version control and committing changes daily. For each changed commmitted a server builds and tests the changes informing you of status of those changes. As well as providing a build which is typically a binary executable of code that can then be saved if necessary. In doing you can reduce the chances of facing the situation off "It works on my machine...". As the building and packaging of the code is done on a server to make sure everything integrates.

An early definition of CI was written up and then updated later by Martin Fowler [10]. A key part of the CI is that allows teams to work on the same code base which without CI could easily lead to integeation bugs and broken builds.

73 To enable to this to happen automation needs to put in place for build,
74 testing and other aspects of the intergeration process in order that a clear
75 peice of feedback (yes or no) can be given about the status of the build. If
76 done with from a version control system if the same commit is built twice
77 (so no changes have happened) it is vital that it produces the same result.
78 Othwerise it is hard for a team to be able to depend on CI if they are getting
79 flakey test results or flakey build results.

80 **2.2 Usage of Continuous Integration**

81 The actual usage of CI as configuration was looked at by [17]. In this they
82 use three source of information Github repositories, Travis builds and a sur-
83 vey. In order to be do a more systematic study of CI usage than [23]. In
84 analysing that data they found that "The trends that we discovered point
85 to an expected growth of CI. In the future, CI will have an even greater
86 influence than it has today." As we are looking at the same question we will
87 use four of the research questions out of the fourteen. In order to see what
88 difference four years has made to the growth of usage of CI.

89 **2.3 Config as code**

90 Configuration as code or Infrastructure as Code has been an increasing area
91 of research over the last few years. There seems to be slightly more research
92 in infrastructure as code Rahman, Mahdavi-Hezaveh and Williams [18]. The
93 has been a focus on Puppet and Chef, for example in Sharma, Fragkoulis and
94 Spinellis [20] looks at code quality by the measure of "code smell" of Puppet
95 code. This tackles the problem by defining by best practices and analyzing
96 the code against that. In the case of Cito et al. [8] it uses the docker linter
97 in order to be able to analyse the files. For the CI systems we pick we will
98 look into the tooling around that to aid the analysis.

99 3 Methodology

100 Initially the project started of as a small piece of research that would aid
 101 looking into how visualise CI systems. Therefore the initial scraping script
 102 was a quick hack to try and get some data initially. This meant that as
 103 we were not initially trying to get lots of data we did not decided to use
 104 Ghtorrent (REFERENCE). However as it quickly started to want to gather
 105 more data and look at different questions it started to form into this paper.

File/Folder	Description
.github/workflows	Create test.yml
output	added song beamer co
.travis.yml	Create .travis.yml
Jenkinsfile	Create Jenkinsfile
README.md	added song beamer co
example.log	powerpoint support ac
hymns.txt	init TODO: unicode ern
main.py	added song beamer co
modernWorship.txt	init TODO: unicode ern
notWellKnown.txt	init TODO: unicode ern
powerpoint.py	added song beamer co
requirements.txt	init TODO: unicode ern
scaper.py	added song beamer co
songbeamer.py	added song beamer co
worshipNight_1.txt	fixed unicode errors an
worshipNight_2.txt	fixed unicode errors an

Figure 1: Example Github repository that has multiple configuration types in it [16]. (This is an old repository that was reused in order to test out the scraper)

```

PATHS = {
    "travis": "travis",
    "gitlab": "gitlab-ci",
    "azure": "azure-pipelines",
    "appVeyor": "appveyor",
    "drone": "drone",

    "jenkinsPipeline": "jenkinsfile",

    "teamcity": ".teamcity/",

    "github": ".github/workflows/",
    "circleci": ".circleci/",
    "semaphore": ".semaphore/",
    "buildkite": ".buildkite/"
}
PATHS_MULTIPLE = ["github", "circleci", "semaphore",
                  "teamcity", "buildkite"]
NONE_YAML = ["jenkinsPipeline", "teamcity"]

```

Figure 2: Python configuration file used to specify what types of configuration to search for. The key specifies the name of the configuration and the value is the location in the repository the config should be found.

106 We chose to use a config file to specify which CI systems config files we
 107 would look for. If it was a directory then it would get all ".yaml" or ".yml"

108 along with any Teamcity ".kts" and ".xml" files. However the script did not
 109 look into any of the sub directories which might be the cause for the low
 110 number of Teamcity configuration files found. In the case that it was a file
 111 that was on the top level directory we matched it the lowercase file name we
 112 found against the query.

113 In terms which configuration files to pick we based our list from Github
 114 Welcomes all CI Tools blog post in 2017 [12]. In addition we added Github
 115 Actions and Azure Pipelines to list as they are new potentially popular sys-
 116 tems.

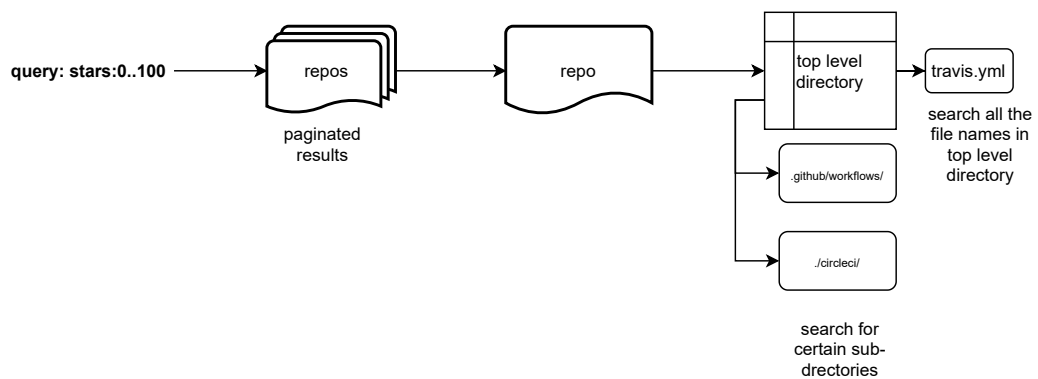


Figure 3: Diagram of the process used to search for projects with CI files in them

117 As can be seen in Figure 3 do a query based on the number of stars a
 118 project has on Github. This is because we need a way of getting a large
 119 sample from Github without introducing too much bias into the sample.
 120 That is not to say that our method is perfect but it provides an easy way
 121 to get a large sample that includes projects with and without CI. Another
 122 potential solution would have been to use the "filename:travis.yaml" search
 123 api. However this did not provide information about which projects did
 124 not use CI. As well as for one unique search there can only be 1000 results
 125 returned by the Github Api. To mitigate that limit we search based stars as
 126 we did do a search for a 1000 results per star count. The limitation of this
 127 though was that there will be over a 1000 repositories that have 0 to 500 or

128 even 500 to 501 stars. That means it is a sample that represents some of the
129 population not a sample of all CI files on Github.

130 As the config could have mistakes in it or we missed out a major CI
131 system. We also saved the ReadMe.md when we scraped each project. A
132 Readme.md is used to describe a project and will be displayed on Github at
133 the bottom of the root directory. As can be seen in Figure 4 some ReadMe's
134 have a label and/or links to the CI system used for that project. Therefore
135 we also save that data when we scrape a project.

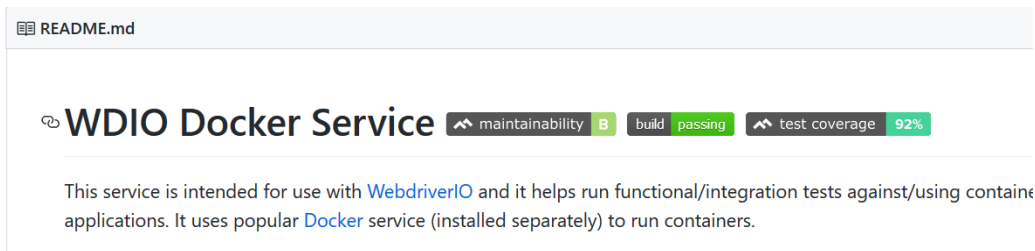


Figure 4: Example of CI tag for Github ReadMe [21]

136 We ended up with a config file with queries for configuration files for the
137 following CI systems: Travis, Gitlab, Azure, App Veyor, Drone, Jenkins,
138 Github, Circleci, Semaphore, Teamcity and buildkite.

139 We excluded Wrecker from the search because they represented a very
140 small number of projects in comparison to the other projects. As it seems
141 since the Github survey in 2017 they got bought by Oracle and from doing a
142 search on Github for what we think based on the docs [24] and [14] for their
143 config file naming convention. We were only able to find 20 results so did
144 not include in the scraping script to speed up the process of searching for the
145 other configuration file formats.

146 Along with information of what CI is being used for a project we also
147 gathered metadata about the project. The available metadata through the
148 api is largely what can be seen on a repository for example in Figure 5.
149 We have the star count which is an indication how popular a project is as
150 users can star projects that they like Borges, Hora and Valente [7]. Then

we have watchers which is users that have subscribed to the project to get notifications about the project.

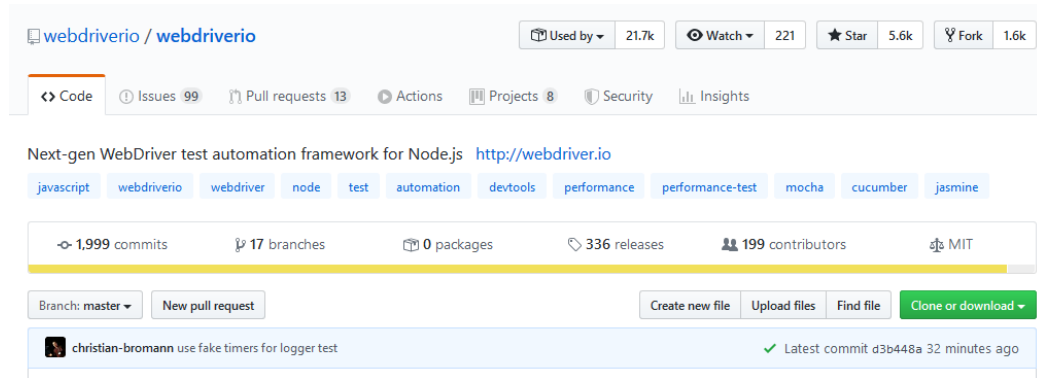


Figure 5: Example Github project description and metadata[22]

3.1 Data corpus

This all produced a sample of 32,660 projects from open source projects on Github. As can be seen in Figure 6 we weren't able to scrape the whole star count range easily. This is because the script would crash when Github gave a 500 error code at us randomly. Along with empty repositories initially causing a problem. In order to mitigate the damage of this the scraper would create a new Comma Separated Value (csv) file search e.g. one for stars:0..1 and another for stars:1..2. As all the csv file contained the same header we ran a script to combine all together at the end. Making sure to remove any duplicates by filtering on the Github project id.

In Figure 7 it shows the density of our the data we got from Github more clearly than Figure 6. As discussed already the number of projects that have 0 stars would easily be over 10,000 however we only could get a maximum of a 1,000 results for each star level.

3.2 comparison corpus info

In Michael Hilton, Marinov and Dig [17] paper they use a similar method of using the Github Api in order to create their corpus. Additionally they

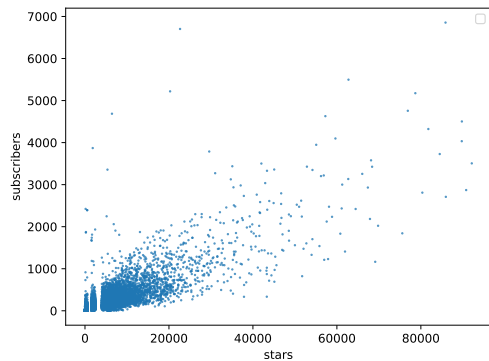


Figure 6: Github stars against subscribers

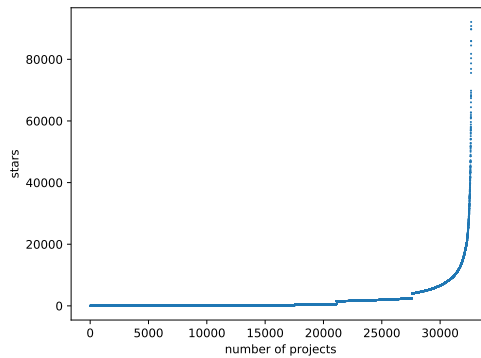


Figure 7: density of stars sampled

170 contacted Cloud Bees (REFERENCE) to get a list of all open source projects
 171 that used their services. This helped them not to miss out on projects that
 172 they would otherwise missed out on. They kindly gave a copy of their final
 173 corpus.

175 Figure 8 provides a visual representation of their corpus in comparison to
 176 our dataset in order to help gain an understanding between the two of them.

so the cloud bees data isn't in our corpus that they gave us... so hwo do we do comparison based on the data. But the paper says that it's 35,544 which is the size of data we have in our

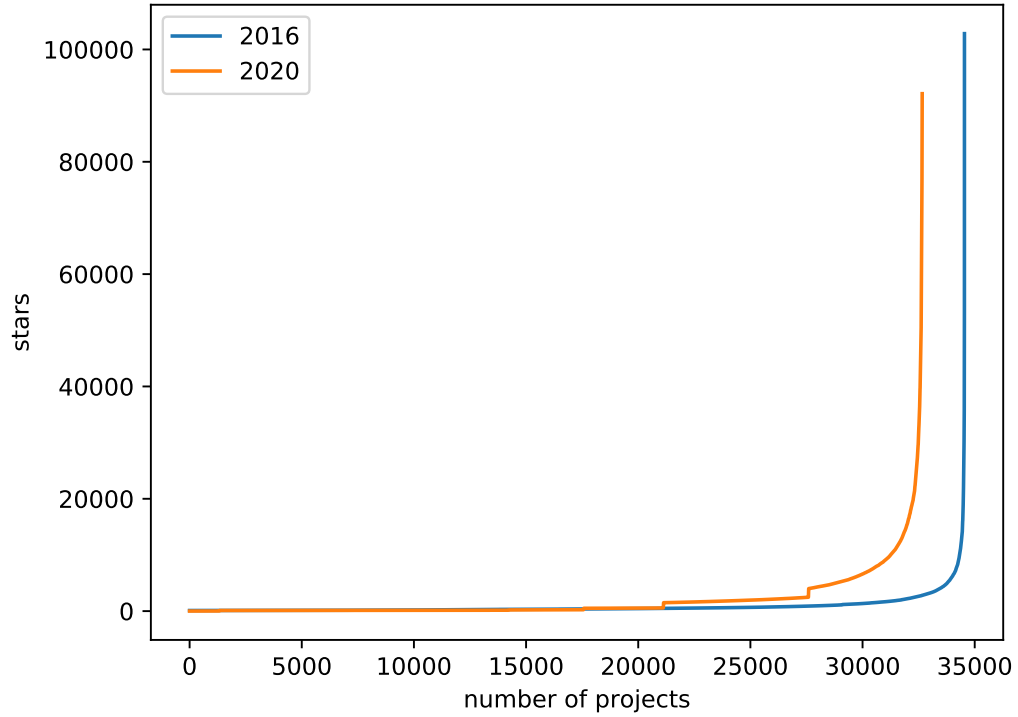


Figure 8: Star comparison line graph between the two datasets

177 4 Usage of CI

178 4.1 RQ1: What percentage of open-source projects 179 use CI?

180 Out of the 32,660 projects 38.51% of them had CI configuration files in them
181 indicating that they used CI.

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 out of a sample of 31,494

182 An interesting factor in Table 1 is the percentage of that 38% that has

multiple CI in them. This is because configuration files can be used to CI or CD and some projects are run a monorepo which means that have multiple projects inside them. Another simple explanation is that although the configuration is stored version control it just hasn't been deleted.

We scraped the "ReadMe.md" files from the projects to check if they had a CI status label in them as shown in Figure 4. To do this we checked for "alt="Build Status"", "alt='Build Status'", "Status" and "status" being in the file. Then if that same line of text contained a url specified by if contained "http://" or "https://" then we counted it as potentially being a project that used CI. However this can lead to false positives as only 6782 (55.92%) of projects that we found CI files in ReadMe had a CI status label that we could find.

In Table 2 is the results from Michael Hilton, Marinov and Dig [17] which had a sample of 34,544.

CI/CD	count	Percentage	no. multiple configs	multiple percent
uses CI(s)	13910	40.27%	1796	12.91%
does not use CI	20634	59.73%		

Table 2: Percentage of CI used for projects out of a sample of 34,544

The percentage of CI projects they had was 40.27%. If you look at Table 1 it shows that we got 38.51% CI projects. This is interesting as we searched for more kinds of CI configuration so there was a potentially a higher chance of having CI.

One possible reason could be because of in RQ3 4.3 it shows that the more popular a project the higher chance it has of using CI. Therefore as their sample contains a few more projects that are popular their they could all be using CI. However that is a weak tangent to make in order to full explain it.

Another possible reason could be if you combined the "config file(s)" and "found in ReadMe" results together you would get 41.28% which is shows

208 that our sample is within the margins of the same results that of Table 2.

209 Another possible reason is that because of Github's growth over the last
210 4 years (Git [4] to 2019 Github [13]) so that Github is now at 40 million
211 active users. It means that there are more projects that are using CD setups
212 for building their static sites and in general Github is being used for more
213 things that wouldn't require CI.

214 Therefore we think that the last two factors are the most likely contribu-
215 tors to why there is less CI usage now. Another important interesting part is
216 despite Github growing so much the CI usage rate has stayed relatively the
217 same.

218 4.2 RQ2: What CI systems are projects using?

219 In Table 3 we find like all other research Travis is the most popular CI system
220 in use. However over the last 4 years since the [12] CircleCi has lost out on it's
221 rough quarter that it owned. In particular the rise of Github Actions seems
222 to have taken second place even though it is still very young in comparison
223 as it was officially released November 13th 2019 but had a closed beta since
224 the summer of 2019. However this might not be down to the CircleCi loosing
225 out on their existing 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.

228 Our sample of repositories is 31,494 this means that as it is a representa-
229 tion of projects on Github so won't account for the whole of it. This means
230 that although Wrecker had the smallest count of CI when researching of 20
231 projects. In Table 3 we have configuration types that have lower counts.
232 This is because that search for the 20 searched the whole of Github but the
233 scraping was only able to do a small sample. Additionally their potentially
234 could be faults in the scraping causing it show such low numbers for the last
235 3.

Table 3: Configuration types spread

	config	percentage
Travis	10607	74%
Github	2301	16%
CircleCi	1109	8%
Jenkins pipeline	161	1%
Drone	84	1%
Buildkite	32	0%
Teamcity	4	0%
Semaphore	2	0%
Azure pipeline	1	0%

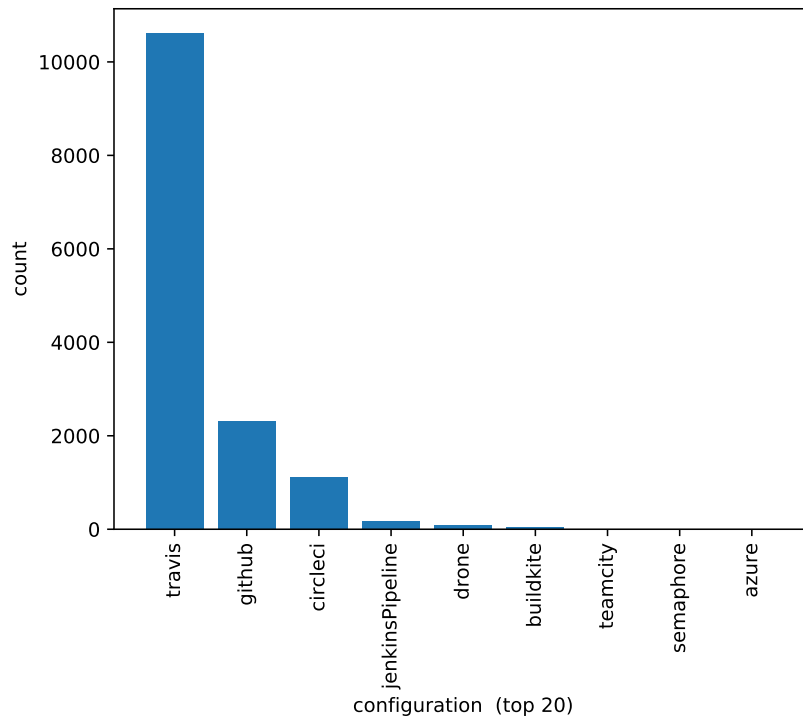


Figure 9: Subs graph

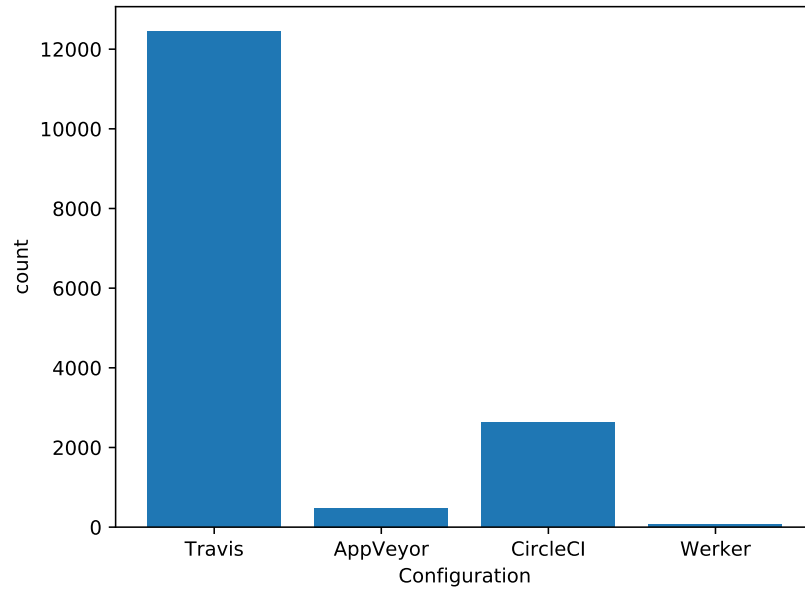


Figure 10: Subs graph

236 4.3 RQ3: Do certain types of projects use CI more 237 than others?

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

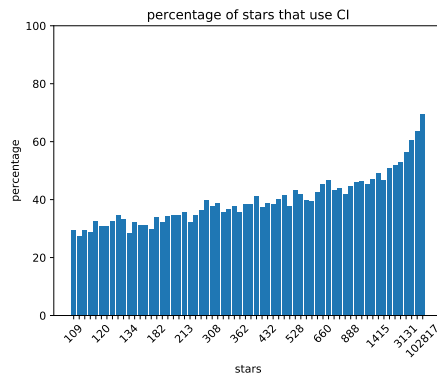


Figure 11: 2016 dataset

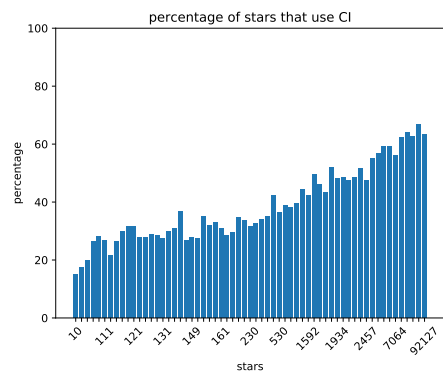


Figure 12: 2020 dataset

Figure 13: In Figure 12 is the results from this research and in Figure 11 is the results from [17].

240 Here in Figure 12 and 11 we are comparing whether or not in the last
 241 4 years the number of stars increases the CI being used. Their seems to a
 242 steeper gradient in the more recent datasets. However as 12 starts at zero
 243 stars and 11 starts at 100 stars their is significant dip at the start of the first
 graph. Figure 15 uses the same method as Figure 12 except is does it based

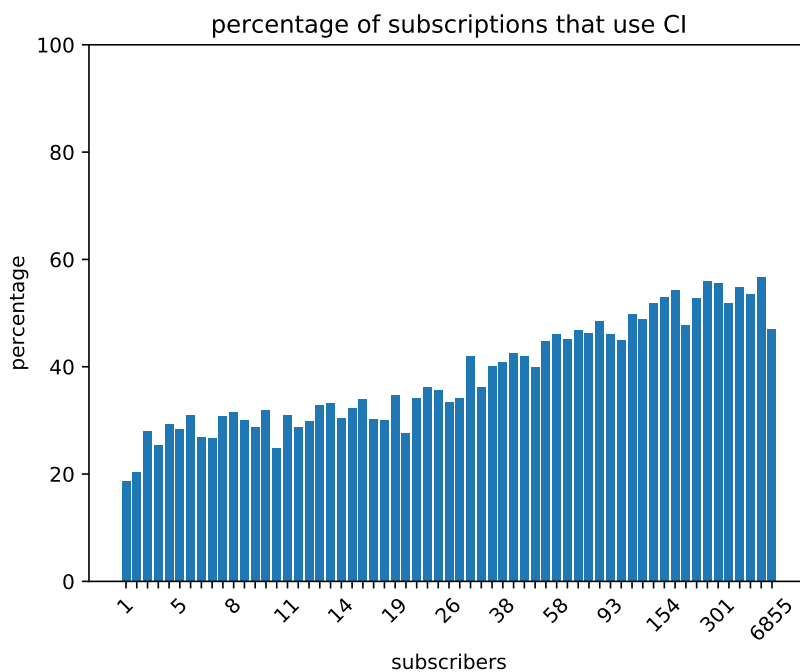


Figure 14: Subs graph

244 the number of subscribers. Subscribers are used on Github to keep update
 245 on the changes on the project. This could range from core team members
 246 working on the project to people that want to be notified about a new release.
 247 In looking at this metric the hypothesis was that it would have a sharper rise
 248 in percentage of projects using CI per subscriber. However that was not the
 249 case overall the gradient is not as strong. There is no comparison to [17]
 250 because their final corpus does not contain subscriber count for each project.
 251 That gives us a good look at how projects can be viewed through Github's
 252 metadata. In terms of what kind of programming languages are being used
 253 for CI? As well as what programming languages where found when creating
 254

when the writ-
 ing is good
 and nearly
 polished
 make this the
 proper size

255 the sample. We can see the top 20 results in Table 4 in that we can see that
 256 Javascript is the most common kind of project. This was too be expected as
 257 in Github’s annual report [13] on the platform they reported that Javascript
 258 has been the most popular for the last 5 years. The interesting part is that
 259 our sample matches the rise in Python over Java. Despite the fact that they
 260 are using ”unique contributors to public and private repositories tagged with
 261 the appropriate primary language” and we are using the count of projects by
 262 primary programming language tag.

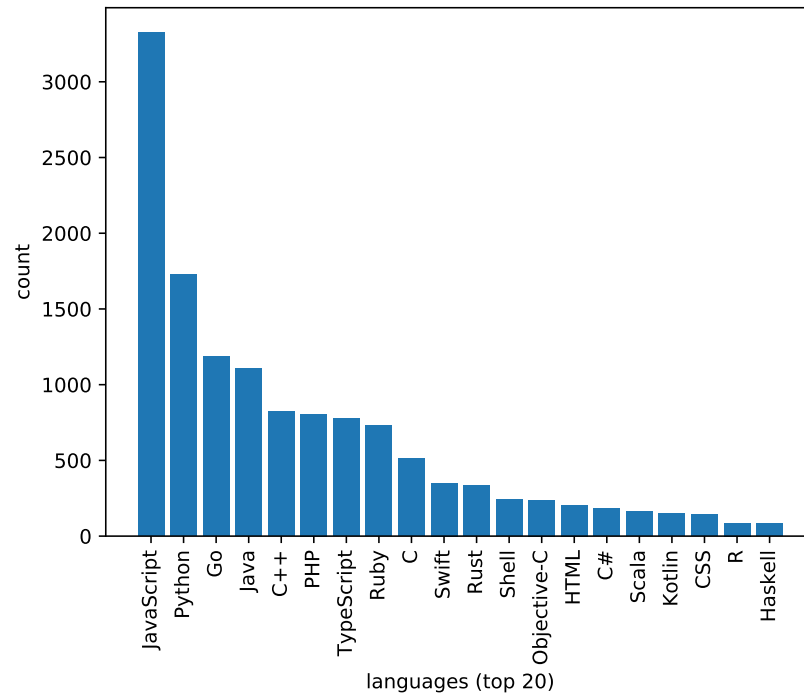


Figure 15: Subs graph

263 In Figure 16 we have grouped together the programming languages by
 264 their type. In order to categories the different programming languages we
 265 used Wikipeda’s page on List of programming languages by type 202 [6].
 266 After that filled in some of the missing gaps for languages they did not have.
 267 However did do it for all the programming types as their were some obscure
 268 projects using uncommon programming languages.

Table 4: Total count of all programming languages used by projects. It has programming languages that only found once removed.

	total count	using CI	percentage CI
JavaScript	6663	3323.0	49.87%
Python	4127	1726.0	41.82%
Java	2963	1108.0	37.39%
C++	1571	821.0	52.26%
Go	1512	1184.0	78.31%
PHP	1337	806.0	60.28%
C	1278	515.0	40.3%
Objective-C	1087	239.0	21.99%
Ruby	1053	732.0	69.52%
C#	943	180.0	19.09%
TypeScript	900	779.0	86.56%
HTML	856	205.0	23.95%
Shell	816	244.0	29.9%
Swift	733	350.0	47.75%
CSS	650	143.0	22.0%
Jupyter Notebook	459	63.0	13.73%
Rust	352	333.0	94.6%
Kotlin	284	150.0	52.82%
Scala	223	162.0	72.65%
Vue	186	58.0	31.18%
Dart	168	39.0	23.21%
Vim script	161	60.0	37.27%
Clojure	148	74.0	50.0%
R	129	86.0	66.67%
CoffeeScript	127	60.0	47.24%
Lua	122	37.0	30.33%
Haskell	121	84.0	69.42%
Emacs Lisp	104	44.0	42.31%
TeX	89	18.0	20.22%
PowerShell	178	6.0	6.82%

269 The interesting factor is the proportion of each category that uses CI. Is
270 that interpreted programming languages are more likely to CI than compiled
271 languages. This can be seen in the 4th and 5th bar which have a 4.51% dif-
272 ference between them. Based on the assumption that compiled languages on
273 the whole will be statically typed our findings go along with Michael Hilton,
274 Marinov and Dig [17] observation. That dynamically typed languages use
275 CI more than statically typed languages. However the interesting part is
276 based on Figure 16 there isn't a big difference between different factors. For
277 example the 4.51% we thought would be larger considering that the top two
278 programming languages were Javascript and Python.

279 However the use of types leading to the lack of CI usage might not be the
280 major factor. As Rust (94.6%), Typescript (86.56%) and Go (78.31%) have
281 the highest percentage of CI usage. In terms of Rust and Go it could be down
282 to their tooling that comes builtin to the language. As that would lead to
283 implementing CI to be a lot easier. Yet Typescript is more a special case as it
284 is a subset of Javascript so uses 'npm' to deal with dependency management
285 which was some of inspiration for Rust's tooling Rust [5]. Older programming
286 languages like Java and C# both have tooling for dependency management
287 but the chances that they use CI is much lower. Therefore an area for further
288 research would be whether or not the use "modern" dependency management
289 systems increases the chance of CI.

290 We found that there is a higher chance that popular projects use CI along
291 with those that have more subscribers to their project. In terms programming
292 language the usage of type system contributed to a slight increase in the
293 chance for the project. Yet it seemed there is area for further research in
294 how the dependency and tooling effects the chances of CI being used.

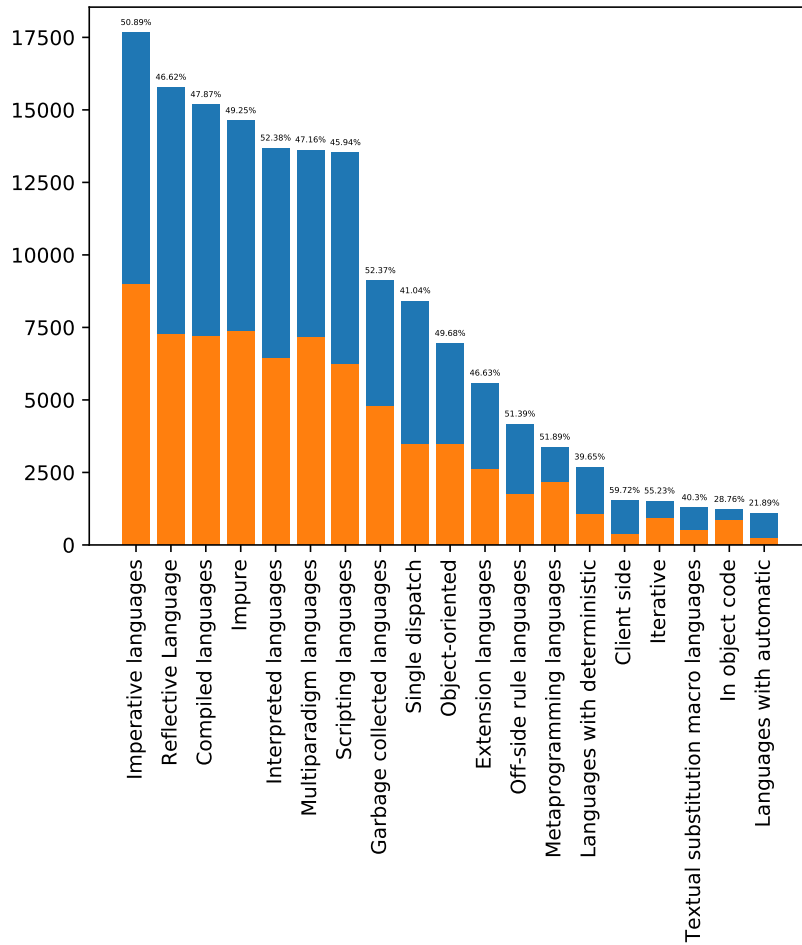


Figure 16: Using categories found on Wikipeda 202 [6] we categorised what kind of programming languages used CI. The key factor is the difference how their our more Interpreted language usage vs Compiled languages. We only included the top 20 categories of programming language.

295 5 Structure of configuration files

296 The following three research questions will just be on the XXXXX CI projects.
 297 In order to be able to ask the questions about the data we filter the sample
 298 to only include CI projects. Then we created a csv table with a row per CI
 299 type in that project as some projects had multiple versions of CI as shown
 300 in REFERENCE-RQ1. Then we processed each CI file to get the necessary

301 data to be able to ask questions about it's structure. As we wanted to be
302 able to process files with or without errors in along with all types of CI. We
303 created a parser to go through each line of the configuration file working out
304 what that line is. For example is it a comment or blank line or does it have
305 code.

306 5.1 RQ4: What are the common errors when loading 307 yaml configuration?

Composer error In the example it has two steps that are using an yaml anchor. This allows for the yaml to be referenced somewhere else. However if you define the anchor twice with the same name it causes an composer error. As you have two references using the same name 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
```

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 5: 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

As can be seen in the Table 5 their our configuration files with yaml errors meaning that the CI for that project will not load. Yet it seems that a very small percentage of projects that have them. For example the two 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 as it is the largest config type out of the sample by a significant amount 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 are the ones with errors ones that are being changed when the scraping was being done. Meaning that as the CI has been set up correctly for the other 99.632% as they are not needing to change because their our no yaml errors in it and presumably it is doing what they intend for it to do.

5.2 RQ5: How are comments used in configuration?

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

327 An example configuration file below for Github actions using the default
328 template slightly altered. Shows two examples of comment usage, the first
329 being including useful information about why a particular version of the
330 programming language was chosen. The second is that the tests have been
331 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 ratio 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
```

332 Initially before we look at the comments it is important to understand
333 how the rest of the file is made up. In the graph below (Figure 17) it shows
334 how each configuration type is made up by mean of each part of the file. For
335 all the yaml based configurations lines of code and number of lines in total
336 are very close. Then for the number of comments being very very small on

337 average.

338 In the case for Jenkins pipelines and teamcity there is a much higher
339 usage of having code with comments.

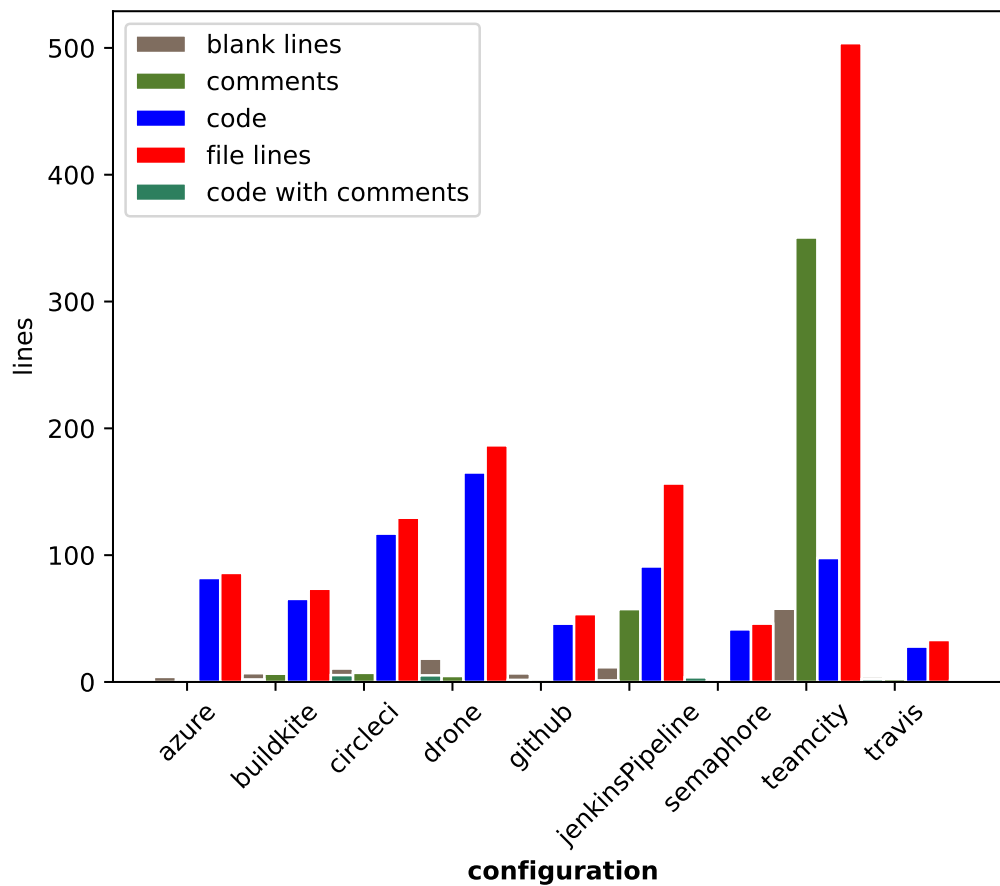


Figure 17: Mean of line counts

340 Ratios:

341 • code: comments

342 • code: line total

343 • code: blank lines

344 • single line comment: multiline comment

345 • single line comment: code with comment

In Figure 18 a regular expression was used to label the comments. There were key different types of comment that we wanted to find. The first being the commented out code which we did by searching for version numbers in comments. The second being useful information about the structure of the CI file such todo, note, important comments (e.g. `//todo`). In order to increase the search for this we included searching for urls and separation comments (e.g. `//===`).

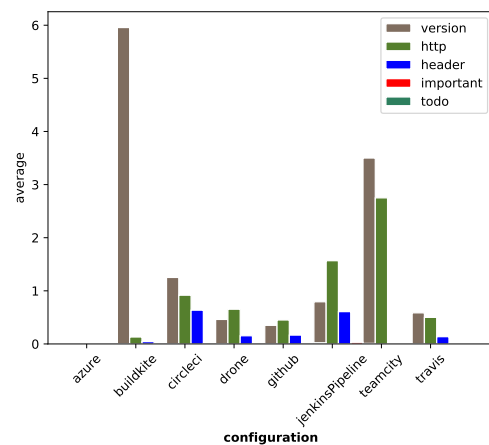


Figure 18: Comment types

346 From labelling the comments in Figure 18 we can see that having com-
347 ments with versions in and urls is most common. This could indicate com-
348 ments from templates or how they are commented. Although yet again the
349 amount of labels found on average is still very low.

350 Overall we have found that comments are not used a lot. In the cases
351 that they are used it's more likely to be from a configuration template or
352 commenting out configuration.

5.3 RQ6: Are external scripts used within the configuration?

An external script is a bash or powershell script typically depending on the operating system. It can be used to build, deploy or do any step that CI takes. The key difference between it and the CI configuration is that it be executed on a users machine. Therefore you do get some setups where you have scripts defined for building and deploying the code that the users and CI both use. Most CI systems allow for "script" tags to be used which could be described as an internal script. Therefore external scripts are defined outside the CI configuration in the directory.

The methodology we used to handle this was too look at how many bash or powershell scripts where used in CI. Using the code the parsed the yaml files for comments we were able to check do a using a regular expression for either of those files.

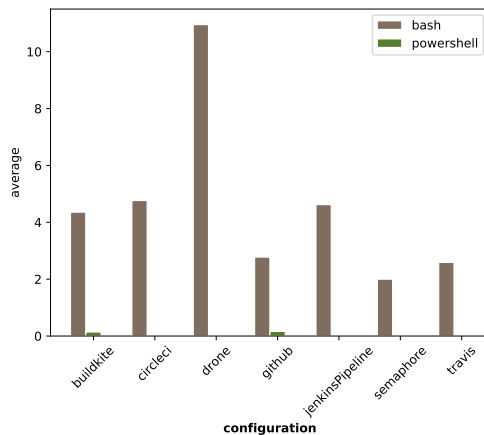


Figure 19: Comment types

Figure 20: sum of scripts used

	bash	powershell
buildkite	61	2
circleci	1497	8
drone	230	0
github	1097	65
jenkinsPipeline	171	0
semaphore	2	0
travis	5937	3

In Figure 19 we have the average number of times a script is used for a configuration file that already has a script being used.

As some of the necessary actions are being done in the scripts and not in

370 the CI file. Potentially there could be less lines of code in the configuration
 371 for files that use scripts. However in Figure 21 we can see that the data is
 372 very spiky with outliers. Then in Figure 22 we can see the same affect when
 373 trying to see if the more popular a project is affects the chances of it using
 374 CI.

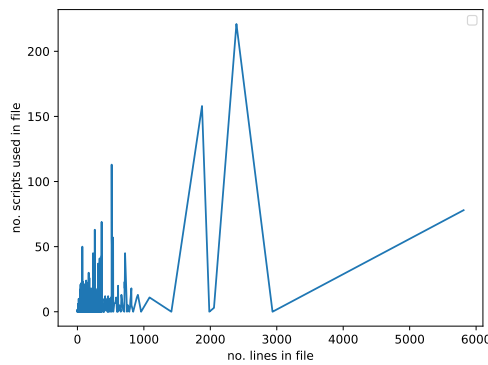


Figure 21: no. scripts to no. lines

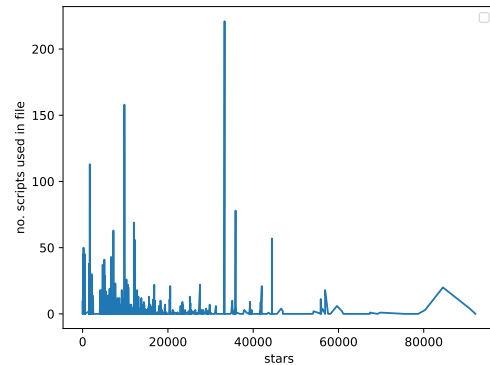


Figure 22: no. scripts to stars

percentage of
 usage needed
 like we had for
 comments

375
 376 Overall we can see that scripts are not used that much. And their no
 377 correlation between lines of code and usage of external scripts.

378 6 Threats to validity

379 The major and most obvious threat is the sample gathered from scraping
380 the data from Github. This has already been touched on in the 3 section but
381 now we are going to look at it in more detail.

382 Firstly if we assume that the scraping works perfectly then it's only at
383 maximum a 1000 open source projects per star. That is excluding closed
384 source projects which would range from personal projects to companies. As
385 well as it is only data from Github not from Gitlab, bitbucket or other version
386 control hosting services. This leads to bias in the data for example if Gitlab
387 was also scraped then we would get a lot more Gitlab ci files. However in
388 order to get best spread of data Github has the best api and most services
389 do not tie you down to use only their service. As well although we could get
390 a 1000 projects per star we were still able to get around 30,000 projects and
391 a wide spread across Github. The key aspect being that because it was a
392 sample we focused on getting a good spread of data.

393 Secondly the scraping script is not perfect in how it finds configuration
394 files. As it only looks in the top level directory for the file name pattern
395 described in their docs or unique folder. Therefore if the systems allowed
396 many different names or different names in past it wouldn't have picked it
397 CI system. Additionally we only decided to scrape for certain CI files. Yet
398 we chose a good scope based on previous research into the top CI files. As
399 well the scraping script has been tested worked on to try and minimise any
400 bugs. In the case that we did not pick up a CI file we ran a regexp against
401 the ReadMe file to get a better understanding of the error bounds.

402 Thirdly identifying which projects are programming projects or would
403 have a need for CI. Based on the research [15] it is important to filter out
404 repositories that aren't part of the question being asked. Therefore we could
405 have looked to try and filter out Github static sites and other non software
406 based projects. However if assume a certain type of project won't be using
407 CI then we would be introducing bias when trying to answer how CI is used.
408 For further research better labelling of what kind of projects are which would

409 potentially beneficial though.

410 7 Summary

411 We got a sample of XXXX open source projects from Github and were able
412 to compare that to a previous study 4 years ago. In doing so we found that
413 usage of CI projects was similar and that more popular a project the higher
414 chance it would be using CI. This lined with the research from 4 years ago.
415 The major change was the increase in popularity of Github Actions taking
416 over second place from Circleci. Additionally we look at whether or not the
417 number of people watching the project had the same effect. It did but to a
418 lesser extent.

419 In terms of structure of CI configuration we looked each line of was used
420 in context of comments. We found that a very few projects use comments in
421 their CI. In terms of how they used scripts, we found the majority of projects
422 do not use external scripts.

423 From this a better understanding of this topic could be gathered by look-
424 ing into the data gathered more. As we found we were faced with a lot more
425 questions while doing this research as we go into below.

426 7.1 Discussion and further research

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

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

Further research into structure could look into the naming of each part of the build process that is used. This would be interesting as it would provided insight into what terms are commonly used. As well an idea into how people plan or don't plan out their configuration files. Additionally CI systems can be designed to run on every commit to version control or only commits to certain branches. Therefore by looking at the branching regexp that are being used an better understanding of how branches are actually used in software development where CI is also used could be found out. In particular looking into which branching method (e.g. [1], [2], [3]) is used more for projects with CI and those that don't. asdf In addition working on pruning our dataset using methods outlined in [15].

8 Acknowledgement

We wish to thank Michael Hilton in particular for providing the corpus for their research Michael Hilton, Marinov and Dig [17].

References

- [1] (???).
- [2] (???).
- [3] (???).
- [4] (???). GitHub State of the Octoverse: 2016.
- [5] (2020). Cargo: Rust's community crate host | Rust Blog.
- [6] (2020). List of programming languages by type. Page Version ID: 946745400.
- [7] Borges, H., Hora, A. and Valente, M. T. (2016). Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016*

464 *IEEE International Conference on Software Maintenance and Evolution*
465 *(ICSME)*, pp. 334–344, iISSN: null.

466 [8] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and
467 Gall, H. C. (2017). An Empirical Analysis of the Docker Container
468 Ecosystem on GitHub. In *2017 IEEE/ACM 14th International Confer-*
469 *ence on Mining Software Repositories (MSR)*, pp. 323–333, iISSN: null.

470 [9] Copeland, P. (2010). Google’s Innovation Factory: Testing, Culture, and
471 Infrastructure. In *Proceedings of the 2010 Third International Confer-*
472 *ence on Software Testing, Verification and Validation*, Washington, DC,
473 USA: IEEE Computer Society, ICST ’10, pp. 11–14.

474 [10] Fowler, M. (2010). Continuous integration.

475 [11] Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous In-
476 tegration Features: An Empirical Study of Projects that (mis)use Travis
477 CI. *IEEE Transactions on Software Engineering*, pp. 1–1.

478 [12] Github (2017). Github welcomes all ci tools. In github.com, ed., *Github*
479 *welcomes all ci tools*.

480 [13] Github (2019). Octoverse - top languages.

481 [14] GitHub (2020). github filename search for wrecker.yml files.

482 [15] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M.
483 and Damian, D. (2014). The promises and perils of mining GitHub.
484 Hyderabad, India: Association for Computing Machinery, MSR 2014,
485 pp. 92–101.

486 [16] Ling, J. (2019). Cu worhsip song list creator - a repository taken over
487 for testing.

488 [17] Michael Hilton, K. H., Timothy Tunnell, Marinov, D. and Dig, D.
489 (2016). Usage, costs, and benefits of continuous integration in open-

490 source projects | Proceedings of the 31st IEEE/ACM International Con-
491 ference on Automated Software Engineering.

492 [18] Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A system-
493 atic mapping study of infrastructure as code research. *Information and*
494 *Software Technology*, 108, pp. 65–77.

495 [19] Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration,
496 Delivery and Deployment: A Systematic Review on Approaches, Tools,
497 Challenges and Practices. *IEEE Access*, 5, pp. 3909–3943.

498 [20] Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Config-
499 uration Code Smell? In *2016 IEEE/ACM 13th Working Conference on*
500 *Mining Software Repositories (MSR)*, pp. 189–200, iSSN: null.

501 [21] Tsvilik, S. (2020). wdio-docker-service.

502 [22] Tsvilik, S. (2020). wdio-docker-service.

503 [23] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015).
504 Quality and productivity outcomes relating to continuous integration
505 in GitHub. Bergamo, Italy: Association for Computing Machinery,
506 ESEC/FSE 2015, pp. 805–816.

507 [24] Wrecker and Oracle (2018). Wrecker ci development blog.