

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: around 4,400

4 March 19, 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 [12] 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 [13] which does a systematic mapping of research in that area. For Continuous Integration with Shahin, Ali Babar and Zhu [14] 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 [5] 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 [12]. 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".

- What percentage of open-source projects use CI?
- What is the breakdown of different CI services?
- 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 [7]. 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:

- What are the common errors when loading yaml configuration?
- How are comments used in the configuration?
- 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 daily committing changes. That then a server builds and tests the changes informing you of status of those changes. As well as providing a build for that change of the code that can be saved. In doing so it can pick up on 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 [6]. A key part of the CI is that it enables makes integrate code between a team easier but it does come at a cost of..... what are you wanting to write here????

just because it is done on a server doesn't mean it should integrate. Although it is closer to where we want to be than just talk-

2.2 Usage of Continuous Integration

The actual usage of CI as configuration was looked at by [12]. In this they use three source of information Github repositories, Travis builds and a survey. In order to be do a more systematic study of CI usage than [17]. 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 research questions out of the fourteen. In order to see what difference four years has made to the growth of usage of CI.

2.3 Config as code

Configuration as code or Infrastructure as Code has been an increasing area of research over the last few years. There seems to be slightly more research in infrastructure as code Rahman, Mahdavi-Hezaveh and Williams [13]. The has been a focus on Puppet and Chef, for example in Sharma, Frangkoulis and Spinellis [15] looks at code quality by the measure of "code smell" of Puppet code. This tackles the problem by defining by best practices and analyzing the code against that. In the case of Cito et al. [4] it uses the docker linter in order to be able to analyse the files. For the CI systems we pick we will look into the tooling around that to aid the analysis.

3 Methodology

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

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

Branch: master ▾ New pull request	
JosephLing Create test.yml	
📁 .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 err
📄 main.py	added song beamer co
📄 modernWorship.txt	init TODO: unicode err
📄 notWellKnown.txt	init TODO: unicode err
📄 powerpoint.py	added song beamer co
📄 requirements.txt	init TODO: unicode err
📄 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 [11]. (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.

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

108 In terms which configuration files to pick we based our list from Github
 109 Welcomes all CI Tools blog post in 2017 [8]. In addition we added Github
 110 Actions and Azure Pipelines to list as they are new potentially popular sys-
 111 tems.

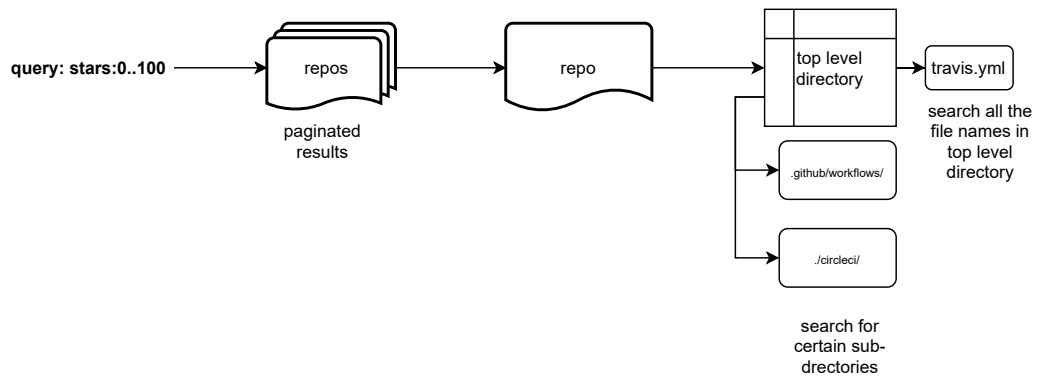


Figure 3: Diagram of the process used to search for projects with CI files in them

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

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

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

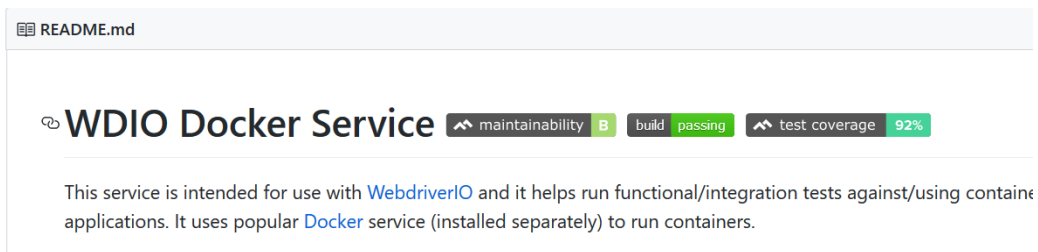


Figure 4: Example of CI tag for Github ReadMe [16]

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

We excluded Wrecker from the search because they represented a very small number of projects in comparison to the other projects. As it seems since the Github survey in 2017 they got bought by Oracle and from doing a search on Github for what we think based on the docs [18] and [9] for their config file naming convention. We were only able to find 20 results so did not include in the scraping script to speed up the process of searching for the other configuration file formats.

As can be seen later in on 5 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

147 a script to combine all together at the end. Making sure to remove any
148 duplicates by filtering on the Github project id.

149 4 Usage of CI

150 4.1 What percentage of open-source projects use 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

Our sample of repositories is 31,494 in comparison to Michael Hilton, Marinov and Dig [12] which had a sample of 34,544. The percentage of CI projects they had was 40.27%. As if you combined the "config file(s)" and "found in ReadMe". However in order to work out if a project might be using CI but the config file wasn't picked a search string is used. Therefore it is not as accurate as finding a config file as their could be false postives.

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

Figure 5 helps give a understanding to the give a depth of the data for where the graph is just blue. This is because on Github you get more repositories with smaller star counts than large ones. In the case of two white bars at the lower end of the stars axis is where we do not have any data. This is due to time constraints on the paper and difficulties discussed in the Methodology (Section 3).

Figure 6 provides insight into the density of the data for between 0 to 25000.

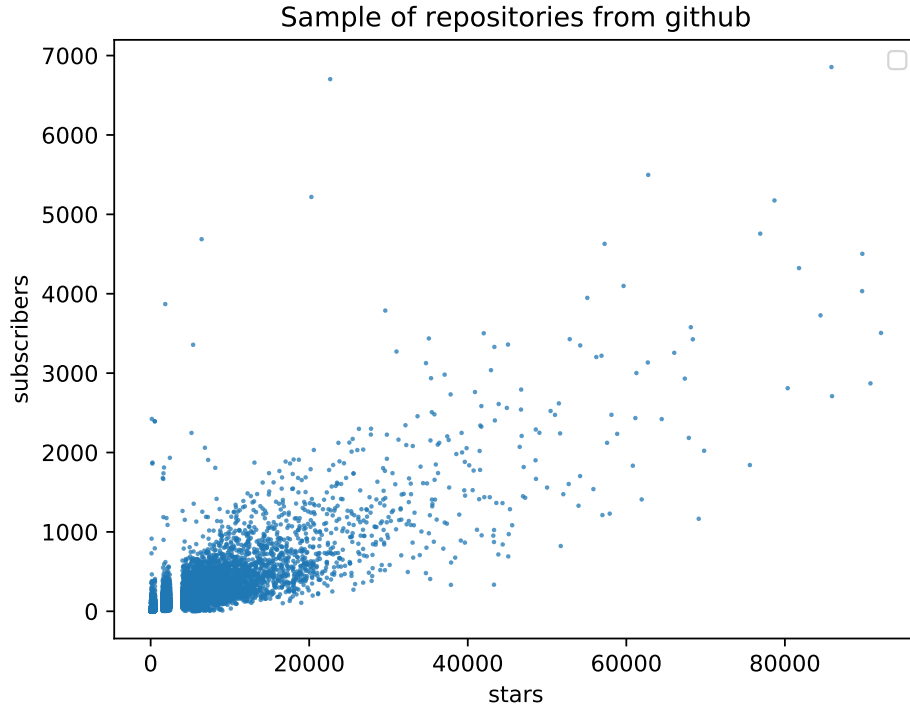


Figure 5: Scatter graph of Github stars against subscribers

171 4.2 What CI systems are projects using?

172 In Table 2 we find like all other research Travis is the most popular CI system
 173 in use. However over the last 4 years since the [8] Circleci has lost out on it's
 174 rough quarter that it owned. In particular the rise of Github actions seems
 175 to have taken second place even though it is still very young in comparison
 176 (DATES). However this might not be down to the Circleci loosing out on their
 177 existing share. But potentially as the rise in CI usage goes up on Github.
 178 Projects are more likely to pick in the built in solutions to Github.

179 Our sample of repositories is 31,494 this means that as it is a representa-
 180 tion of projects on Github so won't account for the whole of it. This means
 181 that although Wrecker had the smallest count of CI when researching of 20
 182 projects. In Table 2 we have configuration types that have lower counts.
 183 This is because that search for the 20 searched the whole of Github but the

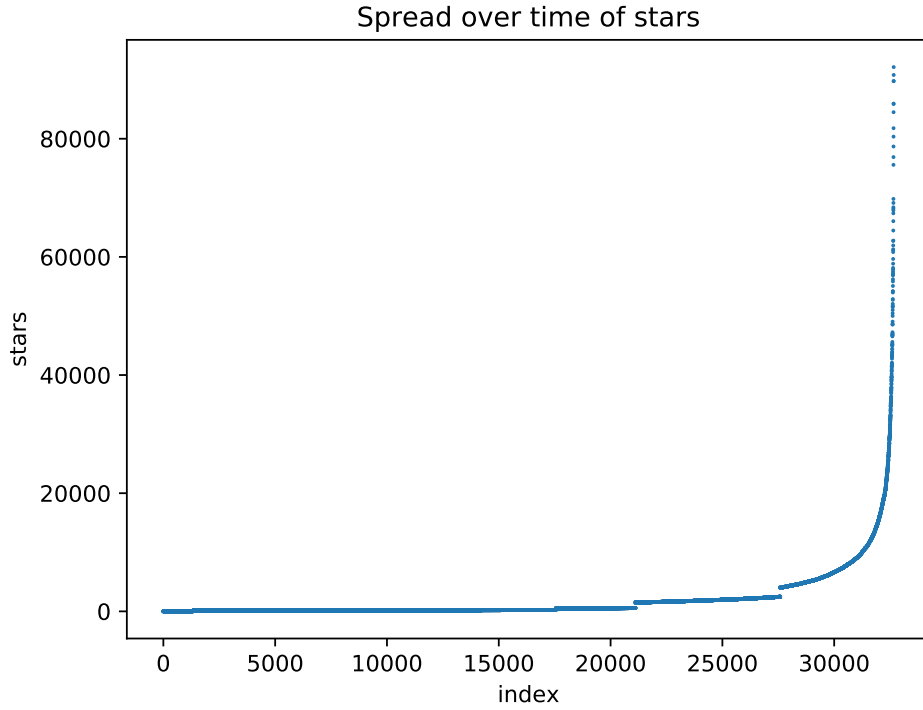


Figure 6: Stars graph

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%

184 scraping was only able to do a small sample. Additionally their potentially
185 could be faults in the scraping causing it show such low numbers for the last
186 3.

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.

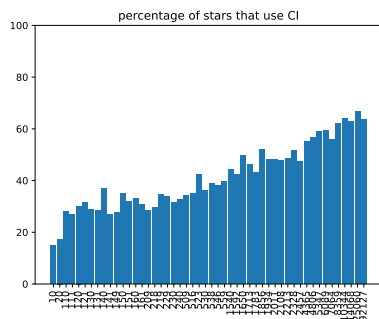


Figure 7: 2020 dataset

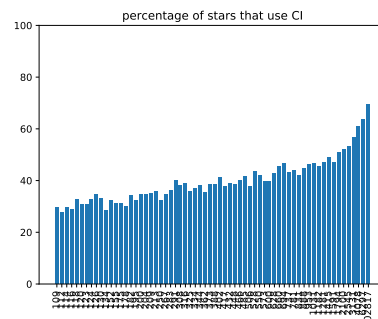


Figure 8: 2016 dataset

Figure 9: In Figure 7 is the results from this research and in Figure 8 is the results from [12].

Here in Figure 7 and 8 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 7 starts at zero stars and 8 starts at 100 stars their is signifacant dip at the start of the first graph.

Figure 10 uses the same method as Figure 7 except is does it based the number of subscribers. Subscribers are used on Github to keep update on the changes on the project. This could range from core team members working on the project to people that want to be notified about a new release. In looking at this metric the hypothesis was that it would have a sharper rise in percentage of projects using CI per subscriber. However that was not the case overall the gradient is not as strong. There is no comparison to [12] because their final corpus does not contain subscriber count for each project.

Table 3: Total count of all programming languages used by projects. It has programming languages that only found once removed.

	total language count	using CI	percentage CI
b'ASP'	7	NaN	NaN
b'ActionScript'	21	2.0	9.523810
b'AppleScript'	6	NaN	NaN
b'Arduino'	21	NaN	NaN
b'Assembly'	47	18.0	38.297872
b'AutoHotkey'	6	NaN	NaN
b'Batchfile'	18	2.0	11.111111
b'C#'	943	180.0	19.088017
b'C'	1278	515.0	40.297340
b'C++'	1571	821.0	52.259707
b'CMake'	17	6.0	35.294118
b'CSS'	650	143.0	22.000000
b'Clojure'	148	74.0	50.000000
b'CoffeeScript'	127	60.0	47.244094
b'Common Lisp'	27	3.0	11.111111
b'Coq'	7	7.0	100.000000
b'Crystal'	13	16.0	123.076923
b'Cuda'	15	2.0	13.333333
b'D'	8	5.0	62.500000
b'Dart'	168	39.0	23.214286
b'Dockerfile'	54	20.0	37.037037
b'Eagle'	7	1.0	14.285714
b'Elixir'	67	55.0	82.089552
b'Elm'	19	12.0	63.157895
b'Emacs Lisp'	104	44.0	42.307692
b'Erlang'	68	41.0	60.294118
b'F#'	24	17.0	70.833333
b'Fortran'	7	5.0	71.428571
b'GLSL'	9	1.0	11.111111
b'Go'	13 1512	1184.0	78.306878
b'Groovy'	37	26.0	70.270270
b'HCL'	20	4.0	20.000000
b'HTML'	856	205.0	23.948598
b'Haskell'	121	84.0	69.421488
b'Haxe'	12	5.0	41.666667

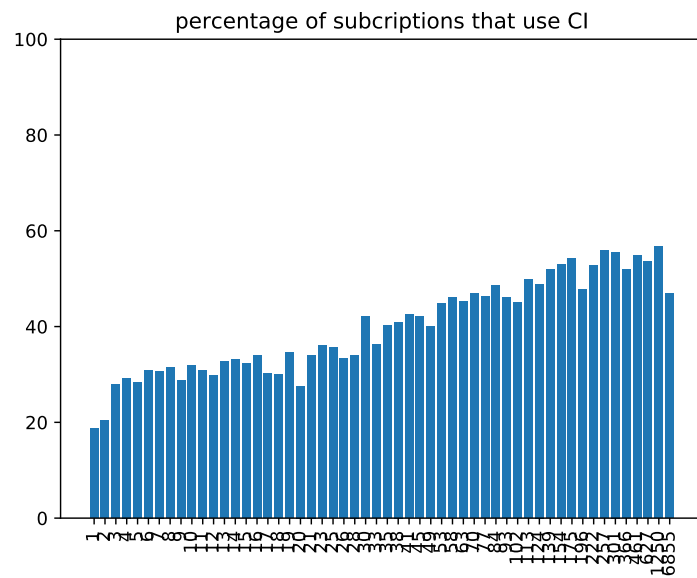
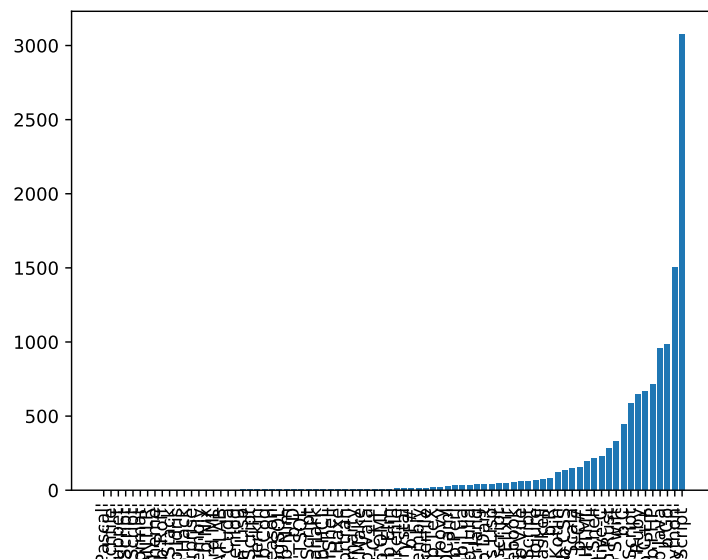


Figure 10: Subs graph



204 5 Structure of configuration files

205 5.1 What are the common errors when loading yaml 206 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: ]
```

207 As can be seen in the Table 4 their our configuration files with yaml
208 errors meaning that the CI for that project will not load. Yet it seems
209 that a very small percentage of projects that have them. For example the
210 two highest configuration types with errors are Drone (36.90%) followed by
211 Travis (0.348%).

Table 4: 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

212 In the case for Drone all the errors are for the same type of error. Po-
 213 tentailly this could be because of how anchors are a lot more common in
 214 Drone.

215 For Travis as it is the largest config type out of the sample by a signifacant
 216 amount it is more likely to contain more errors. Yet with such a small amount
 217 it seems like yaml errors aren't a major problem in CI. Although as they are
 218 required to be fixed in order for the CI to run the chances are the ones
 219 with errors ones that are being changed when the scraping was being done.
 220 Meaning that as the CI has been set up correctly for the other 99.632% as
 221 they are not needing to change because their our no yaml errors in it and
 222 presumbely it is doing what they intend for it to do.

5.2 How are comments used in configuration?

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

226 An example configuration file below for Github actions using the default
227 template slightly altered. Shows two examples of comment usage, the first
228 being including useful information about why a particular version of the
229 programming language was chosen. The second is that the tests have been
230 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
```

231 Initially before we look at the comments it is important to understand
232 how the rest of the file is made up. In the graph below (Figure 12) it shows
233 how each configuration type is made up by mean of each part of the file. For
234 all the yaml based configurations lines of code and number of lines in total
235 are very close. Then for the number of comments being very very small on

236 average.

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

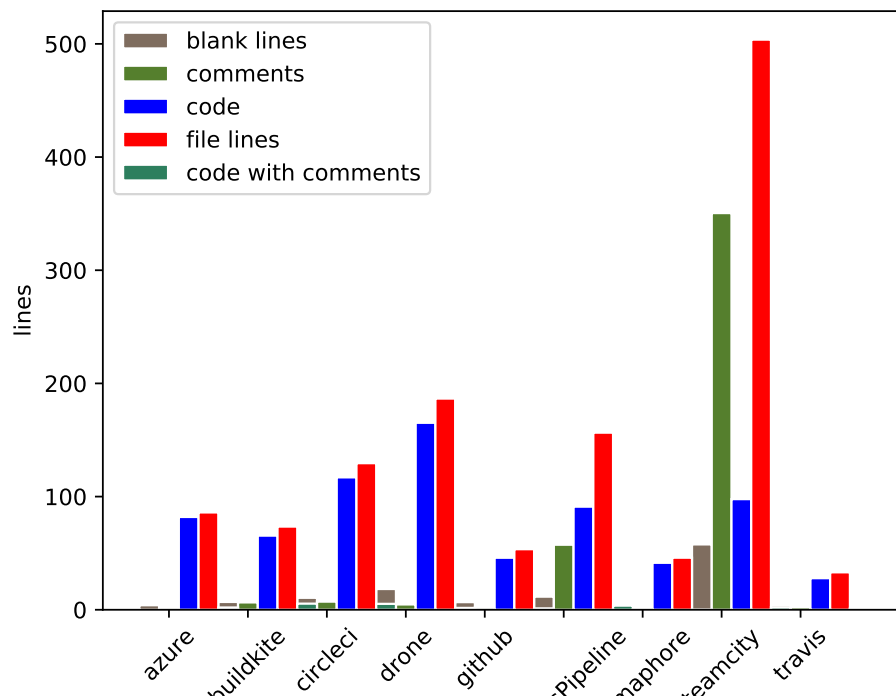


Figure 12: Mean of line counts

239 Ratios:

240 • code: comments

241 • code: line total

242 • code: blank lines

243 • single line comment: multiline comment

244 • single line comment: code with comment

In Figure 13 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. `//===`).

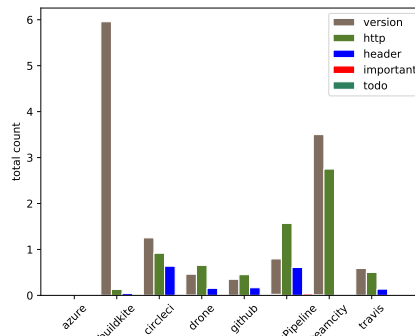


Figure 13: Comment types

245 From labelling the comments in Figure 13 we can see that having com-
 246 ments with versions in and urls is most common. This could indicate com-
 247 ments from templates or how they are commented. Although yet again the
 248 amount of labels found on average is still very low.

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

5.3 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.

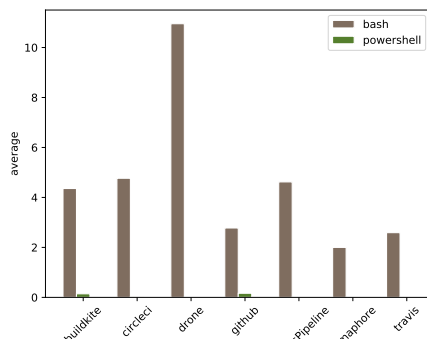


Figure 14: Comment types

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

Figure 15: sum of scripts used

In Figure 14 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 the CI file. Potentially there could be less lines of code in the configuration for files that use scripts. However in Figure 16 we can see that the data is very spiky with outliers. Then in Figure 17 we can see the same affect when

271 trying to see if the more popular a project is affects the chances of it using
272 CI.

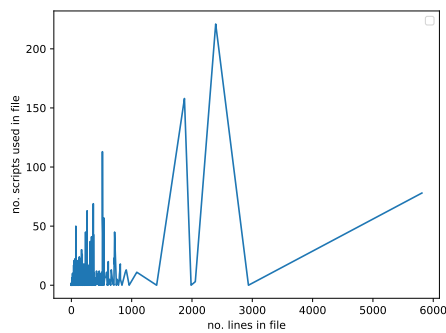


Figure 16: no. scripts to no. lines

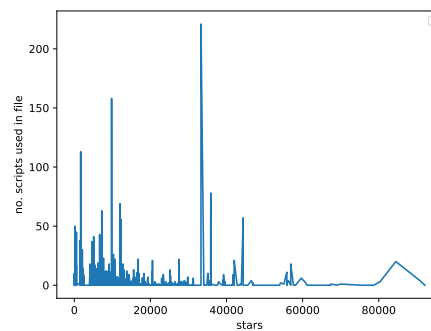


Figure 17: no. scripts to stars

273

percentage of
usage needed
like we had for
comments

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

276 6 Threats to validity

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

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

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

300 Thirdly identifying which projects are programming projects or would
301 have a need for CI. Based on the research [10] it is important to filter out
302 repositories that aren't part of the question being asked. Therefore we could
303 have looked to try and filter out Github static sites and other none software
304 based projects. However if assume a certain type of project won't be using
305 CI then we would be introducing bias when trying to answer how CI is used.
306 For further research better labelling of what kind of projects are which would

307 potentially beneficial though.

308 7 Summary

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

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

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

324 7.1 Discussion and further research

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

328 Further research into usage that we would like to do is look into how
329 the size of the project affects the chance that it uses CI. Then looking at
330 the usage of scripts within CI configuration, for example using a script tag
331 to run a shell script. As while doing the research we found some projects
332 use scripts a lot while others just used the CI config. This would lead to
333 questions around which CI system has a higher amount of scripts used. But
334 also looking at how much they enable them to be used and what is the size
335 of those scripts. The data for the programming language and version(s) is in
336 the config. Therefore it would be possible to work out how much usage each
337 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. Bra [1], Bra [2], Bra [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 [10].

8 Acknowledgement

We wish to thank Michael Hilton in particular for providing the corpus for their research [12].

[section ommitted]

References

[1] (???).

[2] (???).

[3] (???).

[4] 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.

[5] Copeland, P. (2010). Google's Innovation Factory: Testing, Culture, and Infrastructure. In *Proceedings of the 2010 Third International Confer-*

- 363 *ence on Software Testing, Verification and Validation*, Washington, DC,
364 USA: IEEE Computer Society, ICST '10, pp. 11–14.
- 365 [6] Fowler, M. (2010). Continuous integration. In *Continuous Integration*.
- 366 [7] Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous In-
367 tegration Features: An Empirical Study of Projects that (mis)use Travis
368 CI. *IEEE Transactions on Software Engineering*, pp. 1–1.
- 369 [8] Github (2017). Github welcomes all ci tools. In github.com, ed., *Github*
370 *welcomes all ci tools*.
- 371 [9] GitHub (2020). github filename search for wrecker.yml files. In *github*
372 *filename search for wrecker.yml files*.
- 373 [10] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M.
374 and Damian, D. (2014). The promises and perils of mining GitHub.
375 Hyderabad, India: Association for Computing Machinery, MSR 2014,
376 pp. 92–101.
- 377 [11] Ling, J. (2019). Cu worhsip song list creator - a repository taken over
378 for testing. In *CU Worhsip Song List Creator - a repository taken over*
379 *for testing*.
- 380 [12] Michael Hilton, K. H., Timothy Tunnell, Marinov, D. and Dig, D.
381 (2016). Usage, costs, and benefits of continuous integration in open-
382 source projects | Proceedings of the 31st IEEE/ACM International Con-
383 ference on Automated Software Engineering.
- 384 [13] Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A system-
385 atic mapping study of infrastructure as code research. *Information and*
386 *Software Technology*, 108, pp. 65–77.
- 387 [14] Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration,
388 Delivery and Deployment: A Systematic Review on Approaches, Tools,
389 Challenges and Practices. *IEEE Access*, 5, pp. 3909–3943.

- 390 [15] Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Config-
391 uration Code Smell? In *2016 IEEE/ACM 13th Working Conference on*
392 *Mining Software Repositories (MSR)*, pp. 189–200, iSSN: null.
- 393 [16] Tsvilik, S. (2020). wdio-docker-service. In *wdio-docker-service*.
- 394 [17] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015).
395 Quality and productivity outcomes relating to continuous integration
396 in GitHub. Bergamo, Italy: Association for Computing Machinery,
397 ESEC/FSE 2015, pp. 805–816.
- 398 [18] Wrecker and Oracle (2018). Wrecker ci development blog. In *Wrecker*
399 *CI development blog*.