An Empirical Study of Bugs in COVID-19 Software Projects

Akond Rahman [ Tennessee Technological University | arahman@tntech.edu ]
Effat Farhana [ North Carolina State University | efarhan@ncsu.edu ]

Abstract

The dire consequences of the COVID-19 pandemic have influenced development of COVID-19 software i.e., software used for analysis and mitigation of COVID-19. Bugs in COVID-19 software can be consequential, as COVID-19 software projects can impact public health policy and user data privacy. The goal of this paper is to help practitioners and researchers improve the quality of COVID-19 software through an empirical study of open source software projects related to COVID-19. We use 129 open source COVID-19 software projects hosted on GitHub to conduct our empirical study. Next, we apply qualitative analysis on 550 bug reports from the collected projects to identify bug categories. We identify 8 bug categories, which include data bugs i.e., bugs that occur during mining and storage of COVID-19 data. The identified bug categories appear for 7 categories of software projects including (i) projects that use statistical modeling to perform predictions related to COVID-19, and (ii) medical equipment software that are used to design and implement medical equipment, such as ventilators. Based on our findings, we advocate for robust statistical model construction through better synergies between data science practitioners and public health experts. Existence of security bugs in user tracking software necessitates development of tools that will detect data privacy violations and security weaknesses.

Keywords: bugs, covid-19, empirical study, pandemic, software quality

1 Introduction

The novel Coronavirus disease (COVID-19) is a worldwide pandemic that spreads through droplets generated from coughs or sneezes and by touching contaminated surfaces (John Hopkins University, 2020). As of May 31 2020, COVID-19 has caused 370,247 deaths across the world (John Hopkins University, 2020). Apart from causing thousands of deaths and creating long term health repercussions for vulnerable populations, COVID-19 has severely impacted the economic sector. According to a recent study (Erin Duffin, 2020), due to COVID-19 gross domestic product (GDP) will decrease from 3.0% to 2.4% worldwide. As of May 28 2020, nearly 41 million citizens reported unemployment in USA alone (Mitchell Hartman, 2020). More than 3.9 billion people around the world were under some form of stay at home order due to COVID-19 (Alasdair Sandford, 2020).

Health care professionals are at the frontline of combating COVID-19. Practitioners from other domains, such as software engineering have also joined forces to analyze and mitigate the negative consequences of COVID-19. For example, statistical modeling was used to build a software that identifies pneumonia caused by COVID-19 from lung scan images (Tom Simonite, 2020). The software was used in 34 Chinese hospitals (Tom Simonite, 2020). In response to the food insecurity caused by COVID-19, practitioners have created an interactive visualization software that displays free meal sites across USA (Why Hunger, 2020). The creators of the software envision in building a social movement to eradicate hunger and address economic inequalities. As another example, Apple and Google have jointly announced of creating a software framework that will help practitioners build tools to trace COVID-19 infection status of mobile app users (Apple, 2020). The above-mentioned examples show COVID-19 software i.e., software used for analysis and mitigation of COVID-19, to have near-term and long-term effects on public health and society.

Despite the above-mentioned advancements, COVID-19 software projects are susceptible to bugs. Let us consider Figure 1 in this regard. Figure 1 provides a snapshot of a bug report related to statistical modeling (Begley, 2020a). We observe when implementing a statistical model the practitioners did not consider the correlation between intensive care unit (ICU) bed availability and death rate prediction. Furthermore, the number of ICU beds is incorrectly assumed to be 40,000 instead of 1,000.

We hypothesize systematic analysis can reveal bug categories including statistical modeling bugs similar to Figure 1. In prior work researchers (Garcia et al., 2020; Rahman et al., 2020; Linares-Vásquez et al., 2017; Catolino et al., 2019; Thung et al., 2012; Wan et al., 2017) have documented the importance of bug categorization. For example, for autonomous vehicle software Garcia et al. 2020 stated that categorization of bugs can help to construct bug detection and testing tools. Linares-Vásquez et al. 2017 stated categorizing vulnerabilities can help Android practitioners “in focusing their verification and validation activities”. According to Catolino et al. 2019, “understanding the bug type represents the first and most time-consuming step to perform in the process of bug triage”.

In prior work, researchers have categorized bugs for infrastructure as code (IaC) (Rahman et al., 2020), autonomous vehicle (Garcia et al., 2020), and machine learning (Thung et al., 2012; Islam et al., 2019) software. However, COVID-19 software is different from previously studied software in the following aspects: (i) development context: unlike previously studied software projects, COVID-19 software is developed in response to a pandemic that infected 6.1 million individuals in five months (John Hopkins University, 2020), and (ii) public health: unlike previously studied software projects, COVID-19 software has direct implications on pub-
Wang ware projects.

We refer to COVID-19 software projects as software projects related to COVID-19. An empirical study of open source software projects related to COVID-19 can yield (i) manifestation of bugs unique to the COVID-19 reality, and (ii) bug frequency associated with these projects. Our hypothesis is that the utility of COVID-19 software projects and the urgency associated with these projects can yield (i) manifestation of bugs unique to the COVID-19 reality, and (ii) bug resolution time. Furthermore, from our empirical analysis what categories of bugs appear for what types of COVID-19 software projects? How frequently do the identified bug categories appear? What is the resolution time for the identified bug categories?

The goal of this paper is to help practitioners and researchers improve the quality of COVID-19 software through an empirical study of open source software projects related to COVID-19. We answer the following research questions:

- **RQ1**: What categories of open source COVID-19 software projects exist? We identify seven categories of software projects related to COVID-19: aggregation, education, medical equipment, mining, user tracking, statistical modeling, and volunteer management.

- **RQ2**: What categories of COVID-19 software projects? How frequently do the identified bug categories appear? What is the resolution time for the identified bug categories? We identify eight bug categories: algorithm, data, dependency, documentation, performance, security, syntax, and user interface. Except for mining and medical equipment projects, for types of COVID-19 software projects the most frequently occurring bug category is UI.

- **RQ3**: How similar are the identified bug categories to that with previously studied software projects? Identified bug categories for COVID-19 software projects also appear for other software types, but their manifestation of the bugs is different for COVID-19 software projects.

**Contributions**: We list our contributions as follows:

- A categorization of bugs that appear in COVID-19 software projects;
- A categorization of OSS projects related to COVID-19;
- An empirical study that identifies what category of bugs appear for what category of COVID-19 software projects; and
- A comparison of bug categories for COVID-19 software projects to that with previously studied software projects.

We organize rest of the paper as follows: We discuss related work in Section 2. We provide the methodology to answer the three research questions in Section 3 and provide the results in Section 4. We discuss our results with a summary of our findings in Section 5. We provide the limitations of our paper in Section 6. Finally, we conclude the paper in Section 7. Our constructed dataset is available as a public, citable repository (Rahman and Farhana, 2020).

**Overview of the Empirical Study** An overview of our paper is available in Figure 2. First, we mine software projects related to COVID-19 from GitHub by applying a filtering criteria based on number of issues, number of developers etc. Next, we apply qualitative analysis technique called open coding (Saldana, 2015) on the README files of the collected open source software (OSS) projects to identify what categories of OSS projects exist related to COVID-19. After characterizing the collected software projects, we again apply open coding on 550 bug reports from the collected OSS projects to identify bug categories. We also quantify the frequency and resolution time of each bug category across the identified project categories. Finally, we conduct a scoping
review (Munn et al., 2018) to find the similarities in bug
categories between COVID-19-related software projects and
other categories of software projects.

2 Related Work

Our paper is related with prior research that has focused on
categorization of bugs in OSS projects. Mockus et al. 2002
studied the contribution nature in OSS Apache and Mozilla
projects. They (Mockus et al., 2002) observed contributors
who submit bug reports are approximately 8.2 times higher
in number than contributors who address bugs in bug reports.
Ma et al. 2017 investigated Python GitHub projects that are
used in the scientific domain, and observed developers to use
stack traces, as well as communicate with upstream developers,
to identify root causes of bugs. Zhang et al. 2019 ex-
amined bug reports for mobile and desktop software hosted
on GitHub, and identified differences on how the reports are
constructed. Ray et al. 2014 studied the correlations between
bugs and the language the software is being developed, and
reported a modest correlation using an empirical study of
729 GitHub projects. Categorization of domain-specific OSS
bugs has also been investigated: Thung et al. 2012, Garcia et
et al. 2020 in separate research papers used OSS projects
to classify bug categories respectively, for machine learning,
autonomous vehicle, blockchain, deep learning, and IaC.

Our paper is also related with publications that have in-
vestedigated the impact of COVID-19 on software develop-
ment. Ralph et al. 2020 surveyed 2,225 practitioners and re-
ported fear related to COVID-19 to affect productivity of
software practitioners. Butler and Jaffe 2020 conducted a di-
ary study with 435 practitioners and reported practitioners to
face challenges, such as having too many meetings and feel-
ing overworked while working from home due to COVID-19.
Oliveira et al. 2020 surveyed 413 practitioners from Brazil
and reported practitioners’ perceived productivity to increase
due to fewer interruptions.

From the above-mentioned discussion we observe bugs
in software projects related to COVID-19 to be an under-
explored area. While there exists several bug categorization
studies (Thung et al., 2012; Garcia et al., 2020; Wan et al.,
2017; Islam et al., 2019; Rahman et al., 2020) no studies ex-
ist for COVID-19-related projects. The bug categorization-
related studies for IaC, block chain, and deep learning moti-
vated us to derive bug categories and quantify the identified
bug categories. Wan et al. 2017’s paper on blockchain bugs
motivated us to study bug resolution time for each identified
bug category. In our paper, we study COVID-19 software
bugs in the following manner:

• categories of bugs;
• frequency of identified bug categories;
• resolution time of identified bug categories; and
• categories of software projects.

3 Methodology

In this section we provide the methodology to answers re-
search questions: RQ1, RQ2, and RQ3.

3.1 Methodology for RQ1: What categories of
open source COVID-19 software projects exist?

We define COVID-19 software projects as software projects
used for analysis and mitigation of COVID-19. We hypothe-
size multiple categories of COVID-19 software projects to
exist in the OSS domain. We validate our hypothesis by sys-
tematically categorizing COVID-19 software projects. Our
categorization will provide insights on how the software de-
velopment community has responded to the COVID-19 pan-
demic. We answer RQ1 by completing the following steps:

3.1.1 Dataset Collection

We conduct our empirical analysis by collecting COVID-
19 software projects hosted on GitHub. To collect these
projects we use GitHub’s search utility (GitHub, 2020c),
where we first identified projects tagged as ‘covid-19’. We
use the search string ‘covid-19’, as it is a topic designated
for COVID-19 by GitHub (GitHub, 2020a). Our assumption
is that by using a GitHub-designated tag we can collect OSS
projects hosted on GitHub that are related to COVID-19.

OSS projects hosted on GitHub are susceptible to quality
issues, as GitHub users often host repositories for personal
purposes that are not reflective of real-world software de-
velopment (Munaiah et al., 2017). Upon collection of the
projects we apply a set of filtering criteria so that we can
identify projects that contain sufficient data for analysis. We
describe the filtering criteria below:

• Criterion-1: The project must have at least 2 developers.
  Our assumption is that this criterion will filter out projects
  used for personal purposes.
• Criterion-2: The project has at least 5 open issues. We use
  this filtering criterion to identify projects that are actively
  maintained. Our assumption is that by using this criterion
  we will be able to identify COVID-19 software projects that
  are not used for personal purposes as well as projects that
  are active. Prior research (Agrawal et al., 2018) has also
  used the count of issues to filter OSS projects hosted on
  GitHub to conduct empirical studies.
• Criterion-3: The project must have at least two commits
  per month. Munaiah et al. 2017 used the threshold of at
  least two commits per month to determine which projects
  have enough development activity for software organiza-
tions. We use this threshold to filter projects with short de-
development activity.
• Criterion-4: The README of the project is written in En-
  glish. README projects related to COVID-19 can be non-
  English. We do not include non-English projects as raters
  who will perform categorization are not familiar with non-
  English languages, such as Spanish and Cantonese.
3.1.2 Qualitative Analysis of README files

We apply a qualitative analysis called open coding (Saldana, 2015) on the content of README files for each of the downloaded projects from Section 3.1.1. README files describe the content of the project and give GitHub users an overview of the software project (Prana et al., 2019). We hypothesize that by systematically analyzing the content of the README files we can derive what types of software projects are developed that are related to COVID-19.

In open coding a rater identifies and synthesizes patterns within unstructured text (Saldana, 2015). We select open coding because we can obtain detailed information on the software project categories. We use a hypothetical example to demonstrate our process of open coding in Figure 3. First, we collect text from the README files for each of the collected projects from Section 3.1.1. Next, we extract text snippets that describe the purpose of the software project. For example, from the raw text ‘The COVID-19 Vulnerability Index (CV19 Index) is a predictive model that identifies people who are likely to have a heightened vulnerability to severe complications from COVID-19’ we extract the text snippet ‘a predictive model’, as the extracted text snippet describes the purpose of the software project. Next, from the text snippets ‘a predictive model’ and ‘modelling estimated deaths’ we generate an initial category called ‘Models to predict’. Two initial categories ‘Models to predict’ and ‘Models to understand’ are combined into one category ‘Statistical modeling’, as they both indicate the descriptions of the software projects to be related with statistical modeling.

We apply closed coding (Crabtree and Miller, 1999) to identify which project maps to the identified categories from Section 3.1.2. Closed coding is the qualitative analysis technique where a rater maps an artifact to a pre-defined category by inspecting the artifact (Crabtree and Miller, 1999). The first and second author separately conduct closed coding on the collected README files. Both authors use Excel spreadsheets to conduct closed coding. After completing the closed coding process the first and second authors identify agreements and disagreements. Agreement rate is calculated using Cohen’s Kappa (Cohen, 1960). During the discussion phase both authors agreed present their justification, and recheck the category derivation based on the discussion and revisiting content. The mapping determined upon discussion is considered final. One project can map to multiple categories.

3.1.3 Closed Coding

We apply closed coding (Crabtree and Miller, 1999) to identify which project maps to the identified categories from Section 3.1.2. Closed coding is the qualitative analysis technique where a rater maps an artifact to a pre-defined category by inspecting the artifact (Crabtree and Miller, 1999). The first and second author separately conduct closed coding on the collected README files. Both authors use Excel spreadsheets to conduct closed coding. After completing the closed coding process the first and second authors identify agreements and disagreements. Agreement rate is calculated using Cohen’s Kappa (Cohen, 1960). Disagreements are resolved using discussion. During the discussion phase both authors present their justification for disagreements. Next, based on the discussion the authors recheck the labeling based on the justification and content analysis. The categorization determined upon discussion is considered final.

3.1.4 Rater Verification

The derived categories are susceptible to the bias of the first and second author. We mitigate the limitation by allocating an additional rater who applied closed coding for a subset of the README files. The additional rater who is not an author of the paper, is a fourth year PhD candidate in the Department of Computer Science at Tennessee Technological Uni-

---

**Figure 2.** An overview of our empirical study.

- **Criterion-5:** The project is related with COVID-19. We use the ‘topic’ feature of GitHub to search and identify COVID-19 software projects. However, practitioners can mislabel projects using the ‘topic’ feature of GitHub potentially including projects in our dataset that are not related with COVID-19. For example, from manual inspection we observe the ‘RehanSaeed/Schema.NET’ project to be tagged as ‘covid-19’, even though it is not related with COVID-19. In fact, the project is used to convert blob objects into C# classes.

---

1[https://github.com/topics](https://github.com/topics)
versity. The rater has a professional experience of 2 years in software engineering and has conducted qualitative analysis on software artifacts, such as bug reports. We randomly allocate a set of 100 README files mined from 100 projects to the rater. The rater applies closed coding on the content of the README files, to identify the mapping between each project and identified categories. Upon completion of closed coding we calculate Cohen’s Kappa (Cohen, 1960) between the rater and the first author, as well as with the second author, separately.

### 3.2 Methodology for RQ2: What categories of bugs exist in open source COVID-19 software projects? How frequently do the identified bug categories appear? What is the resolution time for the identified bug categories?

In this section, we answer “**RQ2: What categories of bugs appear in COVID-19 software projects? How frequently do the identified bug categories appear? What is the resolution time for each bug category?**” A categorization of bugs for COVID-19 software projects can inform practitioners and researchers about how software related to COVID-19 is developed and in which areas they can help. Furthermore, educators can learn about the software bugs that occur in a software related to a pandemic and disseminate these findings in the classroom. Frequency of the identified bug categories can help us understand what categories of software tend to contain what types of software bugs and provide quality improvement suggestions accordingly. Quantifying the resolution time for bugs in software projects can help software engineering researchers provide actionable guidelines to practitioners. For example, Wan et al. 2017 observed that for blockchain software projects security bugs can take longer to fix compared to other bug categories. Based on their findings Wan et al. 2017 recommended that blockchain project maintainers can adopt security analysis and repair tools to fix security bugs quickly. We provide the methodology to identify bug categories, quantify bug category frequency, and bug resolution time below:

**Methodology to Identify Bug Categories:** We identify bug categories using the following steps:

- **Step#1-Filtering:** We collect the 4,405 issue reports from the 129 projects and manually inspect each issue report. We do not rely on automated approaches, such as keyword search or using bug labels, as automated approaches tend to generate false positives, which may bias research results (Herzig et al., 2013). While inspecting each issue report we use the following IEEE definition for bugs: “an imperfection that needs to be replaced or repaired” (IEEE, 2010), similar to prior work (Rahman et al., 2020). By completing this step we will obtain a set of closed issues reports that correspond to bugs. We use closed reports because as open bug reports are often incomplete and may not help in identifying bugs (Wan et al., 2017). The first and second author manually inspect individually to identify what issue reports correspond to bugs. We record agreement rate and Cohen’s Kappa (Cohen, 1960) between the first and second author. Disagreements between the first and second author are resolved through discussions. The process is subjective and susceptible to the bias of the first and second author. We mitigate the bias by using an additional rater, who inspected randomly inspected 100 issue reports and classified them as bug reports and non-bug reports. The additional rater is the fourth year PhD candidate at Tennessee Technological University who is also involved in rater verification for RQ1.

\[
\text{BugPropAll}(x) = \frac{\text{# of bug reports labeled as category } x}{\text{total # of bug reports}} \times 100\%
\]
We apply the following steps to quantify the frequency of bug categories, similar to Wan et al. (2019) on the content of the collected bug reports from Step#1. Our open coding process is illustrated in Figure 4 using an example. First, we extract raw text from bug report titles and description, from which we generate initial categories. Next, we merge initial categories based on the commonalities and generate categories. Similar to deriving project categories, the first and second author separately apply the process of open coding to generate bug categories. Upon completion of the process we quantify agreement rate and measure Cohen’s Kappa (Cohen, 1960). For disagreements we conduct discussion. Generated categories upon discussion is considered final.

### Methodology to Quantify Bug Category Frequency:

We apply the following steps to quantify the frequency of identified bug categories:

- **Step#1-Closed coding:** We apply closed coding (Crabtree and Miller, 1999) to map each identified category to the bug reports that we study. The first and second author separately apply closed coding for the collected bugs from Step#1. Upon completion, we calculate the agreement rate and Cohen’s Kappa (Cohen, 1960). Disagreements are resolved using discussion.

- **Step#2-Metric calculation:** We quantify the frequency of the identified bug categories using two metrics: 'BugPropAll' and 'BugPropCateg'. We use Equations 1 and 2 to respectively calculate 'BugPropAll' and 'BugPropCateg'. The 'BugPropAll' metric refers to the proportion of bugs across all projects, and provides a holistic overview of the frequency of identified bug categories. The 'BugPropCateg' metric refers to the proportion of bugs for a certain project category, and provides a granular overview of bug category frequency for each software project types identified from Section 4.1.2.

- **Step#3-Rater verification:** The use of first and second author as raters to conduct closed coding is susceptible to rater bias. We mitigate this limitation by allocating an additional rater. We assign randomly selected 250 bug reports to the additional rater who apply closed coding. We provide the additional rater with a document that provides definitions of each identified category with examples. Similar to our process of rater verification for project categorization, the additional rater is the fourth year PhD candidate in the Department of Computer Science in Tennessee Technological University. The fourth year PhD candidate is involved in the rater verification process for identifying project categories and labeling issue reports as bug reports.

### Methodology to Quantify Bug Resolution Time

We use the open and closing timestamp for each closed bug report in our dataset to quantify the resolution time for each bug category, similar to Wan et al. (2017). We calculate bug resolution time by computing the number of hours that have elapsed between when the bug report is opened and closed, and not re-opened again, as per our dataset, which was downloaded on April 04, 2020. We report bug resolution time for all bug categories, as well as for bug reports that belong to certain categories of software projects.

### 3.3 Methodology to Answer RQ3: How similar are the identified bug categories to that with previously studied software projects?

We conduct a scoping review of publications related to software bug categorization. Using a scoping review, researchers can synthesize results using a limited search (Anderson et al., 2008). According to Munn et al. 2018 "Researchers may conduct scoping reviews instead of systematic reviews where the purpose of the review is to identify knowledge gaps, scope a body of literature, clarify concepts or to investigate research conduct.”. Unlike a systematic literature review, a scoping review is less comprehensive, and can be used as a precursor to conduct a systematic literature review. Scoping review can be useful to collect emerging evidence, which eventually can be used to inform further research decisions (Anderson et al., 2008). For example, if a researcher is inexperienced in the domain of software fuzzing, and wants to get an understanding of existing topics such as practices and techniques to implement fuzzing, then a scoping review could be useful to that researcher of interest.

We conduct a scoping review by identifying well-known venues where software engineering research is published. We select five conferences: International Conference on Software Engineering (ICSE), Symposium on Foundations of Software Engineering (FSE), International Conference on Automated Software Engineering (ASE), International Conference on Mining Software Repositories (MSR), and International Symposium on Software Testing and Analysis (ISSTA). We select these conferences because these conferences are considered reputed venues to publish literature related to software engineering (Emery Berger, 2021), and sponsored by special interest groups of the Association of Computing Machinery (ACM). We select conferences as they tend to have a shorter review cycle and are more likely to include recent advances in the field of interest (Vardi, 2009). We conduct the review by applying the following steps:

- **Step-1:** We download all papers from 2010 to 2020 for each of the four conferences. We select papers from 2010 to 2020 to identify and synthesize state of the art bug taxonomies and categories used for a wide range of software projects. Papers that studied bug categories prior to 2010 may not give us an understanding of the state of art. Our hypothesis is that by identifying papers from the last 10 years we will get a better overview of what types of bugs appear for a wide range of software projects.

\[
\text{BugPropCateg}(x, y) = \frac{\# \text{ of bug reports labeled as } x, \text{ of project type } y}{\# \text{ of bug reports for project type } y} \times 100 \%
\]
4 Results

In this section, we provide answers to the three research questions, RQ1, RQ2, and RQ3.

4.1 Answer to RQ1: What categories of open source COVID-19 software projects exist?

We answer RQ1 by first providing summary statistics of our dataset in Section 4.1.1. Next, we report categories of the projects in Section 4.1.2.

4.1.1 Summary of Dataset

Altogether we download 129 projects for analysis. Using the search feature we identify 3,276 public projects upon which we apply our filtering criterion. A complete breakdown of our filtering criterion is available in Table 1. Attributes of the projects are available in Table 2. ‘Languages’ in Table 2 correspond to the count of main programming languages of the collected projects as determined by GitHub’s linguist tool (GitHub, 2020b). Example languages include JavaScript, Python and R.

A temporal evolution of the 129 COVID-19 software projects based on creation date is available in Figure 5. We observe sharp increase in project creation after Feb 29, 2020.

4.1.2 Categorization of COVID-19 Software Projects

We identify 7 categories of COVID-19 software projects. We describe each of the categories below in alphabetic order:

1: Aggregation: This category includes software projects that curate data related to COVID-19 and present collected COVID-19 data in an aggregated format using visualizations. The purpose of these projects is to help users understand the spread of the COVID-19 disease over time and location. Software projects that belong to this category can be country specific as done in ‘juanmnl/covid19-monitor’ (juanmnl, 2020) and ‘dsfsi/covid19za’ (Marivate and Combrink, 2020) respectively, for Ecuador and South Africa. Aggregation of COVID-19 data can also be at a global level, for example, ‘boogheta/coronavirus-countries’ (boogheta, 2020) is a software that aggregates COVID-19 data across the world and allows software users to compare the reported cases on a country-by-country basis.

2: Education: This category includes projects that provide utilities on educating people about COVID-19. Lack of knowledge related to infections and symptoms can contribute to rapid spreading of COVID-19. The purpose of

---

Table 1. Filtering of COVID-19 projects used in paper.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Initial</th>
<th>GitHub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Criterion-1 (Devs &gt;= 2)</td>
<td>3,276</td>
<td></td>
</tr>
<tr>
<td>Criterion-2 (Open issues &gt;= 5)</td>
<td>1,287</td>
<td></td>
</tr>
<tr>
<td>Criterion-3 (Commits/month &gt;= 2)</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>Criterion-4 (README is English)</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Criterion-5 (Actually COVID-19)</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>Final</td>
<td>129</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Attributes of studied COVID-19 projects.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commits</td>
<td>38,152</td>
</tr>
<tr>
<td>Developers</td>
<td>2,243</td>
</tr>
<tr>
<td>Duration</td>
<td>12/2019-03/2020</td>
</tr>
<tr>
<td>Files</td>
<td>24,839</td>
</tr>
<tr>
<td>Issues</td>
<td>4,405</td>
</tr>
<tr>
<td>Languages</td>
<td>18</td>
</tr>
<tr>
<td>Releases</td>
<td>286</td>
</tr>
<tr>
<td>Projects</td>
<td>129</td>
</tr>
</tbody>
</table>
these projects is to build software, where users can ask questions and obtain answers. We observe two categories of software: first, question and answer websites similar to Stack Overflow 3, such as ‘nthopinion/covid19’ (nthopinion, 2020), where users can ask questions about COVID-19, and other users answer such questions. Second, we observe bot-specific software, such as ‘deepset-ai/COVID-QA’ (deepset ai, 2020) that provides answers for questions related to COVID-19 automatically.

- **III: Medical equipment**: This category includes projects to curate and maintain source code for the design and implementation of medical equipment used to treat COVID-19. The purpose of these projects is to create designs of COVID-19 related medical equipment, such as ventilators at scale, so that the growing need of medical equipment in hospitals is satisfied. One example of such repository is ‘makers-for-life/makair’ (makers-for-life, 2020), which states the following in its README page: “Aims at helping hospitals cope with a possible shortage of professional ventilators during the outbreak. Worldwide. ... We target a per-unit cost well under 500 EUR, which could easily be shrunk down to 200 EUR or even 100 EUR per ventilator given proper economies of scale, as well as choices of cheaper on-the-shelf components”. The project includes design of the proposed ventilators as CAD files, as well as relevant firmware available as C++ code files.

Another example is the ‘popsolutions/openventilator’ (popsolutions, 2020), which aims to provide cheap but reliable ventilators to treat COVID-19 in economically under-developed regions of the world. The software project initiated from a Facebook group called ‘Open Source COVID-19 Medical Supplies’ 4, where members discussed the scarcity of ventilators and the importance of creating cheap ventilators through efficient design. In the project we notice developers to create, build, and share designs using OpenSCAD scripts. OpenSCAD is an open source tool to build computer-aided design (CAD) objects 5.

- **IV: Mining**: This category includes projects that provide APIs to mine COVID-19 data from data sources, such as the US Center for Disease Control and Prevention (CDC) 2020, the World Health Organization (WHO) 2020, and data reported from local institutions. The purpose of this category of software is to provide utilities for software developers so that they can get real-time access to COVID-19 data to build aggregation software, discussed above. Because of the nature of the pandemic, access to real-time data is pivotal for accurate aggregation and analysis. The mining tools help developers to get such support. Mining software can be location specific, for example ‘dsfsi/covid19africa’ (Marivatte et al., 2020) is dedicated to curate and collate COVID-19 related data for African countries.

- **V: User tracking**: This category includes software projects that collects information from users regarding their COVID-19 infection status. Tracking of user information can happen voluntarily, where the user voluntarily self-reports COVID-19 infection status. The ‘enigmampe/SafeTrace’ (enigmampe, 2020) software is an example where users self report their infection status as well as location history. Tracking of user information can also be done using inference, as done in ‘OpenMined/covid-alert’ (OpenMined, 2020), where the software collects user’s location information to predict if the user is in a location with high infection density. One utility of these projects is to identify high-risk locations so that users can have an understanding of which nearby location can be avoided. Self reporting software have yielded benefits for China and South Korea (Huang et al., 2020).

- **VI: Statistical modeling**: This category includes software that use statistical models to predict attributes related to COVID-19. The purpose of the projects is to make predictions for the future based on existing data. Example usage of statistical models include (i) predicting death rate as done in ‘ImperialCollegeLondon/covid19model’ (Imperial-CollegeLondon, 2020), (ii) automating the process of lung segmentation with computerized tomography (CT) scan, as done in ‘JoHof/luongmask’ (JoHof, 2020), (iii) predicting the impact of the COVID-19 pandemic on hospital demands as done in ‘neherlab/covid19_scenarios’ (neherlab, 2020), and (iv) predicting presence of COVID-19 with X-ray images using deep learning as done in ‘elcronos/COVID-19’ (elcronos, 2020).

- **VII: Volunteer management**: This category includes software used to efficiently manage volunteering effort. The
purpose of this software is to build software platforms so that users can volunteer and participate in activities to help distressed families and communities. One example is the ‘covid-volunteers’ (helpwithcovid, 2020) software, which provides a web portal where users can sign up for 650 projects that include donation of masks, personal protective equipment (PPEs), and testing of COVID-19. Platforms can be global, such as ‘covid-volunteers’, and also regional, for example ‘Applifting/pomuzeme.si’ (Applifting, 2020) creates a web portal so that people inside Czech Republic can volunteer.

4.1.3 Frequency of the Identified Categories

Based on project count aggregation is the most frequent category. Along with project count, we provide summary statistics of projects that belong to each category in Table 3. We also observe on average user tracking projects to be more frequently released compared to other project types.

We identify four software projects that belong to multiple categories. As an example, the ‘soroushchehresa/awesome-coronavirus’ (soroushchehresa, 2020) project belongs to the categories: aggregation, mining, and statistical modeling.

4.1.4 Rater Agreement

We report agreement rate for three steps: open coding, closed coding, and rater verification.

**Open coding:** After completing open coding, the first and second author respectively, identified 7 and 10 categories. The agreement rate is 70.0%, and the Cohen’s Kappa is 0.7, indicating ‘substantial’ agreement (Landis and Koch, 1977). The authors disagreed on ‘Volunteering software related to local communities’, ‘Education bots’, and ‘Aggregated visualizations’, additional categories identified the second author.

Disagreements were resolved through discussion. Both authors provided justifications for their categorization. The first author pointed out that the category ‘Education bots’ can be merged with ‘Education’ as the category ‘Education’ encompasses all categories of knowledge software, such as bots and web applications. The first author also pointed out that ‘Volunteering software related to local communities’ can be merged with ‘Volunteer management’, as the category is an extension of the category ‘Volunteer management’. Furthermore, the first author also pointed out that ‘Aggregated visualizations’ can be merged with ‘Aggregation’, as ‘Aggregation’ includes software that aggregates COVID-19 data and displays aggregated data with visualizations. The second author was convinced by the first authors’ justification and updated her derived list of categories.

**Closed coding:** During closed coding the first and second authors mapped each of the 129 projects to an existing category. The agreement rate is 93.8%. The Cohen’s Kappa is 0.92. The authors disagreed on the labeling of 8 projects, which are resolved through discussion. During the discussion phase both authors agreed to present their justification, and recheck the labeling based on the justification and content analysis. The categorization determined upon discussion is considered final.

**Rater verification:** We also measured the agreement rate between an additional rater and the authors for categorizing README files of projects. Cohen’s Kappa between the additional rater and the first author for a randomly selected set of 50 README files is 0.73, indicating ‘substantial’ agreement (Landis and Koch, 1977). Cohen’s Kappa between the additional rater and the second author for a randomly selected set of 50 README files is 0.73, indicating ‘substantial’ agreement (Landis and Koch, 1977). The agreement rate between the additional rater and the first and second author is respectively, 78.0% and 76.0%.

4.2 Answer to RQ2: What categories of bugs exist in open source COVID-19 software projects? How frequently do the identified bug categories appear? What is the resolution time for the identified bug categories?

We answer RQ2 by first providing a breakdown of how we obtained our bug reports in Table 4 and 5. As shown in Table 5, the categories with the most and least bug reports are respectively, aggregation and medical equipment. One project can belong to multiple categories, and that is why the total count of bug reports does not total 550.

Next, we describe the identified bug categories in Section 4.2.1 by applying open coding on the collected 550 bug reports. The frequency of the identified bug categories is provided in Section 4.2.2. We provide details of rater verification in Section 4.2.3. Finally, we provide the bug resolution time in Section 4.2.4.

4.2.1 Bug Categories of COVID-19 Projects

We identify 8 bug categories, which we describe below alphabetically:

- **Algorithm:** This category corresponds to bugs when implementation of an algorithm does not follow expected behavior. An algorithm is a sequence of computational steps that transform input into output (Cormen et al., 2009). We observe algorithm bugs to include two sub-categories: (i) bugs related to statistical modeling algorithms, where statistical modeling results are incorrect due to incorrect assumptions and/or implementations, and (ii) bugs related to incorrect logic implemented in the software.

**Example:** We provide examples for the two sub-categories:

- Statistical modeling: In a bug report titled “Death rates should increase when ICU’s are overwhelmed” (Begley, 2020a), a practitioner describes how incorrect assumption can result in incorrect modeling behavior. The practitioner discusses that bed space is correlated with estimation of fatality rate. When bed space of hospitals are exhausted hospitals will not be able to treat new COVID-19 new patients, which could potentially increase the fatality rate. The bug report provides evidence that if the context of COVID-19 is not correctly incorporated in statistical models, those models will provide incorrect results. Incorrect statistical models can be consequential,
Table 3. Summary statistics of projects that belong to each category. Based on project count ‘Aggregation’ is the most frequent category as highlighted in green.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>50</td>
<td>14,985</td>
<td>663</td>
<td>8,641</td>
<td>938</td>
<td>72</td>
</tr>
<tr>
<td>Mining</td>
<td>35</td>
<td>9,671</td>
<td>894</td>
<td>6,714</td>
<td>515</td>
<td>21</td>
</tr>
<tr>
<td>Stat. model.</td>
<td>22</td>
<td>7,214</td>
<td>429</td>
<td>3,464</td>
<td>491</td>
<td>38</td>
</tr>
<tr>
<td>Education</td>
<td>9</td>
<td>4,550</td>
<td>196</td>
<td>1,696</td>
<td>406</td>
<td>14</td>
</tr>
<tr>
<td>User track</td>
<td>9</td>
<td>2,020</td>
<td>152</td>
<td>2,291</td>
<td>119</td>
<td>286</td>
</tr>
<tr>
<td>Volunteer.</td>
<td>7</td>
<td>2,186</td>
<td>143</td>
<td>2,041</td>
<td>320</td>
<td>0</td>
</tr>
<tr>
<td>Med. equip.</td>
<td>3</td>
<td>859</td>
<td>38</td>
<td>790</td>
<td>14</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 4. Filtering of bug reports from COVID-19 software projects.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>4,095</td>
</tr>
<tr>
<td>Criterion-1 (Closed issues)</td>
<td>2,965</td>
</tr>
<tr>
<td>Criterion-2 (Valid bug reports)</td>
<td>550</td>
</tr>
<tr>
<td>Final</td>
<td>550</td>
</tr>
</tbody>
</table>

Table 5. Count of bug reports for each category of COVID-19 software projects. Aggregation-related projects have the highest amount of bug reports.

<table>
<thead>
<tr>
<th>Project Category</th>
<th>Count (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>220 (40%)</td>
</tr>
<tr>
<td>Mining</td>
<td>150 (27.3%)</td>
</tr>
<tr>
<td>Stat. Model.</td>
<td>98 (17.8%)</td>
</tr>
<tr>
<td>Education</td>
<td>58 (10.5%)</td>
</tr>
<tr>
<td>Volunteer.</td>
<td>40 (7.3%)</td>
</tr>
<tr>
<td>User Track</td>
<td>31 (5.6%)</td>
</tr>
<tr>
<td>Med. Equip.</td>
<td>4 (0.7%)</td>
</tr>
</tbody>
</table>

as countries are adopting public health policies specific to COVID-19. For example researchers have critiqued the statistical models derived by the Institute for Health Metrics and Evaluation at the University of Washington (IHME), and advised USA policymakers to use the modeling results with caution (Begley, 2020b).

- Incorrect logic: In a bug report titled “Fix Prefecture Sorting” (reustle, 2020), a practitioner describes a sorting bug which occurs when trying to visualize COVID-19 cases based on prefectures in Japan. A prefecture is an administrative jurisdiction in a country similar to a state or province (Hu and Qian, 2017). The bug occurred due to an incorrect logic that did not perform sorting by prefectures.

**III: Data:** This category corresponds to bugs that occur during mining and storage of COVID-19 data. As discussed in Section 4.1.2 we observed our dataset to include projects that mine and aggregate COVID-19 data. We observe four sub-categories of data bugs: (i) storage: bugs that occur while storing data in a database, (ii) mining: bugs that occur while retrieving data from data APIs, (iii) location: bugs where location information in stored data is incorrect, and (iv) time series: bugs that correspond to missing data for a certain time period.

**Example:** We provide examples for each of these sub-categories below:

- Storage: In a bug report titled “Temperature data not saved in the backend” (pavel ilin, 2020), a practitioner describes a bug where patient temperature data is inserted in the front-end but not stored in the database.


  - Location: In a bug report titled “Rajasthan District names are wrong”, a practitioner describes that inserted location data for an Indian state called ‘Rajasthan’ is wrong (SinghRajenM, 2020).

  - Time series: Missing data was reported for a project and reported in a bug report titled “Data has a gap between 2020-3-11 and 2020-3-24” (branieczeck, 2020).

**IV: Documentation:** This category corresponds to bugs that occur when incorrect and/or incomplete information is specified in release notes, maintenance notes, and documentation files, such as README files.

**Example:** In a bug report titled “Missing PostGIS” (v-aclavpavlicek, 2020), a practitioner describes that installation and execution of the software is prohibited due to a software package called ‘PostGIS’, which is used to store spatial and geographic measurements, such as area, distance, polygon, and perimeter in PostgreSQL databases.

**V: Performance:** This category corresponds to bugs that cause performance discrepancies for the software. Performance bugs are manifested in slow response of the web or mobile app.

**Example:** In a bug report titled “Cluster animation slowing down the browser: It also takes much time”, a practitioner describes how a performance bug related to an animation feature is slowing down a Firefox browser on Windows 10 (Subratappt, 2020). The performance bug was reported for a website called ‘covid19india.org’ 7, which aggregates COVID-19 data for India and displays them.

**VI: Security:** This category corresponds to bugs that violate confidentiality, integrity, or availability for the software.

**Example:** In a bug report titled “Fix password reset procedure” (landovsky, 2020), a practitioner describes a password reset bug, where the password reset procedure ends arbitrarily after 500 login attempts.

**VII: Syntax:** This category corresponds to bugs related

---

7https://www.covid19india.org/
with the syntax of the programming languages used to develop the software.

**Example:** We notice bugs related to data types in `neherlab/covid19_scenarios`. In the bug report titled “Fix types and linting errors” ([ivank](https://github.com/ivanaksamentov)), a practitioner describes how linting and type checking was disabled for the project, which led to bugs related to linting and type checking.

**Example:** In a bug report titled “accessibility fixes” ([abquirarte](https://github.com/abquirarte), 2020) describes a UI bug related to accessibility. According to the bug report, a screen reader incorrectly renders check marks and crosses in front of the “Do’s and Don’t as M’s and N’s”.

### 4.2.2 Frequency of Identified Bug Categories

Based on the ‘Proportion of Bugs Across All Projects (BugPropAll)’ metric we observe UI bugs to be the most frequent category, whereas documentation is the least frequent category. We provide a complete breakdown of the metric in Table 6. Data bugs have four sub-categories: storage, mining, location, and time series. The frequency for storage, mining, location, and time series is respectively, 4.7%, 5.8%, 87.2%, and 2.3%. Algorithm bugs have two sub-categories: statistical modeling and wrong logic. The frequency for statistical modeling and wrong logic is respectively, 42.3% and 57.7%.

We observe bug category frequency to vary for different categories of projects. We provide the ‘Proportion of Bugs For a Certain Project Category (BugPropCat)’ values for each project category in Table 7. ‘AGG’, ‘MINE’, ‘STA’, ‘EDU’, ‘TRAK’, ‘VOL’ and ‘EQU’ respectively, correspond to the seven project categories: aggregation, mining, statistical modeling, education, user tracking, volunteer management system, and medical equipment.

According to Table 7, except for mining and medical equipment software, the dominant bug category is UI. One possible explanation can be the analyzed software projects have UIs, which may have contributed to the frequency of UI bugs. For mining software the dominant bug category is data bugs i.e., bugs that occur due to storing and processing of COVID-19 data. For medical equipment software the dominant bug category is dependency. We also notice algorithm bugs to be the second most frequent bug category for statistical modeling software. Similar to prior work on machine learning ([Thung et al.](https://example.com), 2012), we expected algorithm bugs to be the most dominant category for statistical modeling. Statistical modeling software also have UIs for user interaction, and the count of UI bugs may have foreshadowed the count of algorithm bugs.

### 4.2.3 Rater Agreement and Verification

We report agreement rate for four steps: issue labeling, open coding, closed coding, and rater verification.

**Labeling issues as bugs:** While labeling collected issue reports as bug reports and non-bug reports the agreement rate is 96.5% and the Cohen’s Kappa is 0.9.

**Open coding to identify bug categories:** The first and second author respectively, identified 9 and 10 categories. The agreement rate is 72.7%, and the Cohen’s Kappa is 0.70, indicating ‘substantial’ agreement ([Landis and Koch](https://example.com), 1977). The first author identified ‘database’ as a category not identified by the second author. Upon discussion both authors agreed that ‘database’ is related to data storage and belongs to the data category. The second author identified two additional categories ‘Public health data’ and ‘Type errors’. After discussing the definitions of all categories both authors agreed that ‘Public health data’ and ‘Type errors’ can respectively, be merged with data and syntax.

**Closed coding to quantify bug category frequency:** During closed coding the first and second author mapped each project to an existing category. The agreement rate is 95.1% and the Cohen’s Kappa is 0.93. The authors disagreed on the labeling of 27 bug reports, which are resolved through discussion.

**Rater verification:** For the randomly selected 250 issue reports we allocate an additional rater who manually identified which of the issue reports are bug reports and non-bug reports. The Cohen’s Kappa between the additional rater and the first author is 0.80, indicating ‘substantial’ agreement ([Landis and Koch](https://example.com), 1977). The Cohen’s Kappa between the additional rater and the second author is 0.84, indicating ‘perfect’ agreement ([Landis and Koch](https://example.com), 1977). The agreement rate between the additional rater and the first and second author is respectively, 89.0% and 93.0%.

We have also measured the agreement rate between an additional rater and the authors for categorizing bug reports. Cohen’s Kappa between the additional rater and the first author for a randomly selected set of 250 bug reports is 0.65, indicating ‘substantial’ agreement ([Landis and Koch](https://example.com), 1977). Cohen’s Kappa between the additional rater and the second author for a randomly selected set of 250 bug reports is 0.68, indicating ‘substantial’ agreement ([Landis and Koch](https://example.com), 1977). The agreement rate between the additional rater and the first and second author is respectively, 78.0% and 81.6%.

### 4.2.4 Resolution Time of Identified Bug Categories

We provide bug resolution time as measured in hours for all bug categories in Table 8. From Table 8 we observe that based on min and median bug resolution times security bugs take the longest to resolve, followed algorithm bugs. We also observe data bugs to take as long as 548 hours to resolve.

A breakdown of bug resolution time across the seven project categories is provided in Table 9. The ‘All’ row in...
We report our findings in Table 11. The ‘Bug category’ column presents the bug categories identified for COVID-19 software projects, whereas, the ‘Other software projects’ column presents the software projects for which the bug category was observed according to papers identified from our scoping review. We observe bug categories for COVID-19 software projects to also be observable for other categories of software projects, such as deep learning and automated vehicle.

5 Discussion

In this section, we first provide a summary of our findings in Section 5.1. Next, we provide a discussion on the implications of our findings in Section 5.2.

5.1 Summary

Table 7. Bug category frequency for each identified project type. All values are presented in (%).

<table>
<thead>
<tr>
<th>Bug category</th>
<th>AGG</th>
<th>MINE</th>
<th>STA</th>
<th>EDU</th>
<th>TRAK</th>
<th>VOL</th>
<th>EQU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>6.8%</td>
<td>6.7%</td>
<td>22.4%</td>
<td>3.4%</td>
<td>0.0%</td>
<td>2.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Data</td>
<td>28.6%</td>
<td>60.6%</td>
<td>13.2%</td>
<td>15.5%</td>
<td>0.0%</td>
<td>12.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Dependency</td>
<td>16.3%</td>
<td>18.0%</td>
<td>18.3%</td>
<td>24.1%</td>
<td>9.7%</td>
<td>27.5%</td>
<td>75.0%</td>
</tr>
<tr>
<td>Document</td>
<td>0.9%</td>
<td>1.3%</td>
<td>1.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>10.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Performance</td>
<td>2.7%</td>
<td>2.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>3.2%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Security</td>
<td>1.8%</td>
<td>0.0%</td>
<td>3.0%</td>
<td>3.4%</td>
<td>6.4%</td>
<td>12.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Syntax</td>
<td>5.9%</td>
<td>3.3%</td>
<td>14.3%</td>
<td>17.2%</td>
<td>3.2%</td>
<td>10.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>UI</td>
<td>50.0%</td>
<td>12.0%</td>
<td>34.7%</td>
<td>44.8%</td>
<td>77.4%</td>
<td>32.5%</td>
<td>25.0%</td>
</tr>
</tbody>
</table>

Table 8. Resolution time of identified bug categories. Resolution times is measured in hours. Median resolution time is highest for security bugs.

<table>
<thead>
<tr>
<th>Bug category</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>1.240</td>
<td>13.9</td>
<td>144.6</td>
</tr>
<tr>
<td>Algorithm</td>
<td>0.041</td>
<td>13.5</td>
<td>172.7</td>
</tr>
<tr>
<td>Syntax</td>
<td>0.004</td>
<td>12.1</td>
<td>174.2</td>
</tr>
<tr>
<td>UI</td>
<td>0.003</td>
<td>11.8</td>
<td>254.2</td>
</tr>
<tr>
<td>Data</td>
<td>0.003</td>
<td>8.4</td>
<td>548.0</td>
</tr>
<tr>
<td>Performance</td>
<td>0.961</td>
<td>7.1</td>
<td>104.4</td>
</tr>
<tr>
<td>Dependency</td>
<td>0.014</td>
<td>2.4</td>
<td>379.4</td>
</tr>
<tr>
<td>Documentation</td>
<td>0.013</td>
<td>1.4</td>
<td>76.8</td>
</tr>
</tbody>
</table>

Table 9. Resolution time of bug categories grouped by project categories. We measure resolution time in hours. Median bug resolution time is highest for projects related to medical equipment software.

<table>
<thead>
<tr>
<th>Project category</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical Equipment</td>
<td>5.0</td>
<td>29.3</td>
<td>46.4</td>
</tr>
<tr>
<td>Volunteer Management System</td>
<td>0.013</td>
<td>21.1</td>
<td>174.2</td>
</tr>
<tr>
<td>User Tracking</td>
<td>0.124</td>
<td>16.5</td>
<td>294.5</td>
</tr>
<tr>
<td>Education</td>
<td>0.121</td>
<td>11.2</td>
<td>294.5</td>
</tr>
<tr>
<td>Aggregation</td>
<td>0.003</td>
<td>8.7</td>
<td>379.4</td>
</tr>
<tr>
<td>Statistical Modeling</td>
<td>0.004</td>
<td>7.2</td>
<td>168.3</td>
</tr>
<tr>
<td>Mining</td>
<td>0.005</td>
<td>2.5</td>
<td>548.1</td>
</tr>
<tr>
<td>All</td>
<td>0.003</td>
<td>7.4</td>
<td>548.0</td>
</tr>
</tbody>
</table>

Table 9 shows the minimum, median, and maximum bug resolution time for all bug categories measured in hours. In Table 9 we observe four instances where the minimum bug resolution time is less than 6 minutes (< 0.1 hours). One possible explanation can be practitioners’ habit of opening a bug report after they have developed the fix for a bug (Wan et al., 2017; Thung et al., 2012). In such cases, practitioners notice the bug early, construct the fix for the bug, and then submit the bug report by opening and closing the bug report promptly.

Median bug resolution duration for each project type and bug category is provided in Table 10. ‘AGG’, ‘MINE’, ‘STA’, ‘EDU’, ‘TRAK’, ‘VOL’ and ‘EQU’ respectively, corresponds to the seven project categories: aggregation, mining, statistical modeling, education, user tracking, volunteer management system, and medical equipment. We observe median bug resolution time to vary across bug categories as well as for project categories.

4.3 Answer to RQ3: How similar are the identified bug categories to that with previously studied software projects?

We report our findings in Table 11. The ‘Bug category’ column presents the bug categories identified for COVID-19 software projects, whereas, the ‘Other software projects’ column presents the software projects for which the bug category was observed according to papers identified from our scoping review. We observe bug categories for COVID-19 software projects to also be observable for other categories of software projects, such as deep learning and automated vehicle.

5 Discussion

In this section, we first provide a summary of our findings in Section 5.1. Next, we provide a discussion on the implications of our findings in Section 5.2.

5.1 Summary

Table 7. Bug category frequency for each identified project type. All values are presented in (%).
We discuss the implications of our findings below:

Security and privacy implications of user tracking software: From Table 3 we observe 9 projects to be related with user tracking. While the benefits of user tracking software have been documented for countries, such as Russia and South Korea (Crowell Morning, 2020), this category of software can have negative impacts on privacy of end-users. Data generated from user tracking software can be leveraged for marketing purposes. We make the following recommendations to preserve privacy of user data in user tracking software:

• Policymakers should construct policies specific to COVID-19 software that collects user data.
• Practitioners who develop user tracking software should leverage existing privacy policy frameworks, such as the ‘National Institute of Standards and Technology (NIST) Privacy Framework’ 2020.
• Privacy researchers can build tools that will automatically detect and report privacy policy violations.

Evidence from Table 7 shows that security bugs to exist for user tracking software. We advocate security researchers to systematically investigate if user tracking software includes security bugs. Recent news articles suggest that user tracking software, such as contact tracing apps may become more and more prevalent as Apple and Google are already providing frameworks to build software that tracks user data (Apple, 2020). Our hypothesis is that availability of these frameworks will facilitate rapid development and deployment of mobile apps that collect user data. Security weaknesses in these apps can provide malicious users opportunity to conduct large-scale data breaches. We notice anecdotal evidence in this regard: a researcher has identified vulnerabilities in a user tracking app that could leak user location data (Greenberg, 2020). Panelists at EuroCrypt 2020, a cryptography research conference, discussed limitations of user tracking mobile apps for COVID-19 with respect to API design, indoor location tracking, and informing users about privacy risks (EuroCrypt, 2020a) (EuroCrypt, 2020b).

Towards constructing correct statistical models: From Section 4.2.1 we have observed statistical modeling bugs to exist. Bugs related to statistical modeling can be consequential because based on the predictions generated by statistical models, policymakers enforce public health policies. One possible explanation for buggy statistical models can be attributed to the quality of datasets using which statistical models are build (Koerth et al., 2020). For example, fatality prediction models that are built using the ‘Diamond Princess Cruise Ship Dataset’ may not be applicable for a specific geographic region with low population density. Another possible explanation can be a lack of context and knowledge related to public health specific that hinders model builders to identify appropriate independent variables to construct the models. Incorrect estimation of hospital beds from our discussion in Section 4.2.1 is one example. Other examples of independent variables related to public health includes staff availability, count of known cases, hospitalization rate etc. (Attia, 2020). According to a health expert (Attia, 2020), statistical models that predicted 2.4 million US residents to die, assumed a hospitalization rate of 15-20%, which in reality was 5%.

Based on our findings and above-mentioned explanations we make two recommendations:

• Automated testing for COVID-19 modeling: We hope to see novel research in the domain of COVID-19 that will test the correctness of constructed statistical models used in forecasting in an automated manner. In recent years, we have seen research efforts that test deep learning models (Tian et al., 2018; Pei et al., 2017; Ma et al., 2018). We expect similar research pursuits for COVID-19 statistical modeling.
• Better synergies between data science and public health...
We provide the differences that we have noticed between COVID-19 software projects and other software projects, which we discuss in the following subsections:

### 5.3.1 Differences in Bug Manifestation

A non-COVID-19 software project does not have the context of public health consequences that are associated with a COVID-19 software project. We define a COVID-19 software project to be a software project that is related with analyzing and mitigating the consequences of COVID-19. By definition, we include software projects that directly captures the consequences related to public health, which is absent from a traditional software project. We observe empirical evidence that shows the unique context of COVID-19 to yield differences in bugs and bug resolution time when compared with other software projects.

Let us consider the case of algorithm bugs. Algorithm bugs manifest in COVID-19 projects as well as in machine learning and autonomous vehicle projects. A machine learning project that uses statistical modeling can have algorithm bugs that generate erroneous predictions. For a COVID-19 software project that predicts death rates, a bug related to the modeling algorithm can have serious consequences, as public health policies are derived based on these models, as it occurred during incorrect estimation of hospitalization rate (Atta, 2020). As discussed in Section 4.3 algorithm-related bugs also appear for autonomous vehicles but presence of such bugs manifest in components unique to autonomous vehicle projects, such as lane positioning and navigation, and traffic light processing.

We have observed that data bugs appear for both deep learning projects and COVID-19 software projects. The difference is for COVID-19 we have the concepts of location, as practitioners tend to miss important location-related data for COVID-19, e.g., not able to identify states in India that are observing an outbreak of COVID-19. In the case of deep learning projects, data bugs are related with structure and type of training data.

As another example, dependency-related bugs appear for both IaC scripts and COVID-19 software projects. In the case of IaC, dependency-related bugs are related to an IaC-related artifact, such as Puppet manifest, class, or a module, upon which execution of an IaC script is dependent upon (Rahman et al., 2020). For COVID-19 software project dependencies are related with API and build artifacts, such as Maven dependencies. This difference with respect to dependent artifacts also highlight the differences between COVID-19 software projects and IaC-based software projects.

In short, our findings suggest that while commonalities for bug categories between COVID-19 software projects and other software projects, the manifestation and artifacts related to the bug categories are different from other categories of software projects.

### 5.3.2 Difference in Bug Resolution Time

Our findings indicate that median bug resolution time is lower for COVID-19 software projects than that of blockchain and machine learning projects. Based on our findings, we conjecture that the sense of urgency might have motivated practitioners to fix bugs in COVID-19 software projects.
5.3.3 Differences with Existing Healthcare-related Software Projects

Our findings also demonstrate differences between COVID-19 software projects and other projects related to healthcare domain. To illustrate these differences we use Janamanchi et al. 2009’s work. Janamanchi et al. 2009 studied 174 open-source software projects related to the health domain and identified 11 categories of software projects that do not include the three categories of projects that we have identified for COVID-19 software projects: volunteer management, user tracking, and education. The inception and spread of COVID-19 have motivated software practitioners to create a wide range of software projects, such as projects related to user tracking and volunteer management so that people are aware about the consequences and hygiene practices related to COVID-19. In the context of COVID-19 software projects, projects related to user tracking focus on tracking user location data emitted from smartphones to assess the proximity of individuals who might be exposed to COVID-19. Software projects related to volunteer management are related with managing volunteers to address COVID-19-related societal issues, such as food banking. A pandemic of this nature was not experienced by health professionals prior to 2020. Existing research related to software projects that belong to health domain were not able to perform characterization of COVID-19 software projects and identify project categories unique to COVID-19. Janamanchi et al. 2009 did not systematically study the types of bugs that appear in healthcare software projects. Our paper complements Janamanchi et al. 2009’s work by studying healthcare-related projects that are related with COVID-19 by characterizing the bugs and the types of software projects related to COVID-19 in which the bugs appear in.

6 Threats to Validity

We describe the limitations of our paper as following:

**Conclusion validity:** We have used raters who derived the software and bug categories. Both raters are authors of the paper. Our derived categories are susceptible to the authors’ bias. We mitigate this limitation by allocating another rater who is not the author of the paper we verified our ratings.

Our categories might not be comprehensive because our categorization for projects and bugs is limited to the dataset that we collected. The bug resolution time could be limiting as our dataset includes projects that have a duration of four months.

We use the topic ‘covid-19’ to identify and filter COVID-19 software projects from GitHub. Any software project that is not labeled as ‘covid-19 will not be included in our dataset.

Our datasets have limited lifetime as the COVID-19 was discovered in December 2019, and the lack of maturity in our datasets may influence our analysis. We mitigate this limitation by identifying projects using a filtering criteria so that we can identify projects with sufficient development activity.

**Internal validity:** For RQ1 and RQ2 we use ourselves, the authors of the paper, as raters who conduct open and closed coding on README files and bug reports. Our research is susceptible to mono-method bias, as our categorization and labeling may be influenced by the authors’ implicit expectations and hypotheses about the study.

**External validity:** Our findings are not comprehensive. We have not analyzed projects hosted outside GitHub and private projects hosted on GitHub. We mitigate this limitation by analyzing 129 software projects that belong to 7 categories. Also, as we have used open coding to determine categories, our findings may not be identified by other raters. We mitigate this limitation by conducting rater verification, where we use a rater who is not the author of the paper.

7 Conclusion

The COVID-19 pandemic has impacted people all over the world causing thousands of deaths. Software practitioners have joined the fight in combating the spread and mitigating the dire consequences of COVID-19. An understanding of COVID-19 software categories and software bugs can give us clues on how the software engineering community can help even further in combating COVID-19.

We conduct an empirical study with 129 COVID-19 software projects hosted on GitHub. We identify 7 categories of software projects: aggregation, mining, statistical models, education, volunteer management, user tracking, and medical equipment. By applying open coding on 550 bug reports, we identify 8 categories of bugs: algorithm, data, dependency, documentation, performance, security, syntax, and UI. We observe bug category frequency to vary with project categories, e.g., for mining projects data-related bugs is the most frequently occurring category.

Our findings have implications for educators, practitioners, and researchers. Educators can use our categorization of COVID software projects and related bugs to educate students about the security and privacy implications of COVID-19 software. Privacy researchers can build tools that will check if user tracking software related to COVID-19 are not leaking user data. Practitioners in the data science domain can learn from our categorization of statistical modeling bugs to understand limitations of constructed statistical models and verify underlying assumptions that accompany constructed statistical models. Based on our findings we also advocate for better synergies between data scientists and public health experts so that statistical modeling bugs can be mitigated. We hope our paper will advance further research in the domain of COVID-19 software.

Acknowledgements

We thank the PASER group at Tennessee Technological University for their useful feedback. We also thank Farzana Ahamed Bhuiyan of Tennessee Technological University for her help as an additional rater. The research was partially supported by the National Science Foundation (NSF) award # 2026869.
References


Mitchell Hartman (2020). Covid-19 jobless claims are now over 40 million. many


An Empirical Study of Bugs in COVID-19 Software Projects

Rahman and Farhana 2021

com/Applifting/pomuzeme.si/issues/164. [Online; accessed 10-Mar-2021].


