Journal of Software Engineering Research and Development, 2021, 9:3, doi: 10.5753/jserd.2021.827

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. Linraes-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-

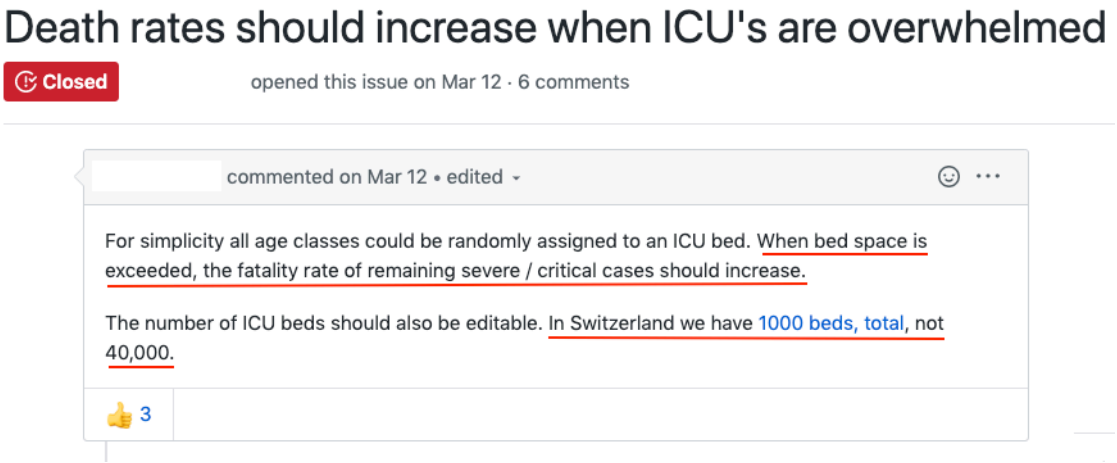

Figure 1. An example of a bug report related to statistical modeling in a software project called 'neherlab/covid19_scenarios'.

lic health and relevant policy making for inhabitants in 188 countries.

In response to the pandemic, researchers have conducted studies related to modeling (Dehning et al., 2020; Yang and Wang, 2020; Tamm, 2020), biological science (Jin et al., 2020; Wang et al., 2020; De Clercq, 2006; Helms et al., 2020), social science (Van Bavel et al., 2020; Pulido et al., 2020; Evans et al., 2020; Will, 2020; Jarynowski et al., 2020), and policy making (Corey et al., 2020; Mello and Wang, 2020; Rourke et al., 2020; Kraemer et al., 2020). However, characterization of bugs in COVID-19 software remains an unexplored area.

The scope of our paper is to get a systematic understanding of bugs in COVID-19 software projects. In our paper, we refer to COVID-19 software projects as software projects that were created to analyze and mitigate the consequences of COVID-19. These projects were created in response to a global pandemic that created a worldwide impact on public health, economy, and societal activities. 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.

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 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 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 examined 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 al. 2020, Wan et al. 2017, Islam et al. 2019, and Rahman 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 investigated the impact of COVID-19 on software development. Ralph et al. 2020 surveyed 2,225 practitioners and reported fear related to COVID-19 to affect productivity of software practitioners. Butler and Jaffe 2020 conducted a diary study with 435 practitioners and reported practitioners to face challenges, such as having too many meetings and feeling 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 exist for COVID-19-related projects. The bug categorization-related studies for IaC, block chain, and deep learning motivated 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 research 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 hypothesize multiple categories of COVID-19 software projects to exist in the OSS domain. We validate our hypothesis by systematically categorizing COVID-19 software projects. Our categorization will provide insights on how the software development community has responded to the COVID-19 pandemic. 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 development (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 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 organizations. We use this threshold to filter projects with short development activity.
- **Criterion-4**: The README of the project is written in English. 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.

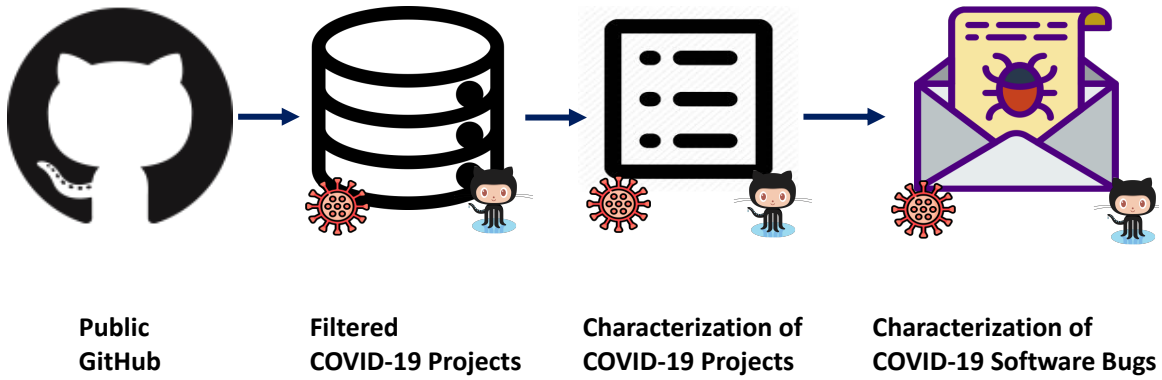

Figure 2. An overview of our empirical study.

- **Criterion-5**: The project is related with COVID-19. We use the 'topic' [1] 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' [2] 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.

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.

[1] https://github.com/topics
[2] https://github.com/RehanSaeed/Schema.NET

The first and second authors conduct the open coding process separately. Both authors used Excel spreadsheets to conduct the open coding process manually. The first and second authors respectively an experience of 10 and 6 years in software engineering and has experience in conducting open coding upon software project artifacts, such as commit messages (Rahman et al., 2020) and Stack Overflow posts (Farhana et al., 2019). Upon completion of the open coding process, the first and second authors identify agreements and disagreements. Disagreements are resolved upon discussion, 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 recorded 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-

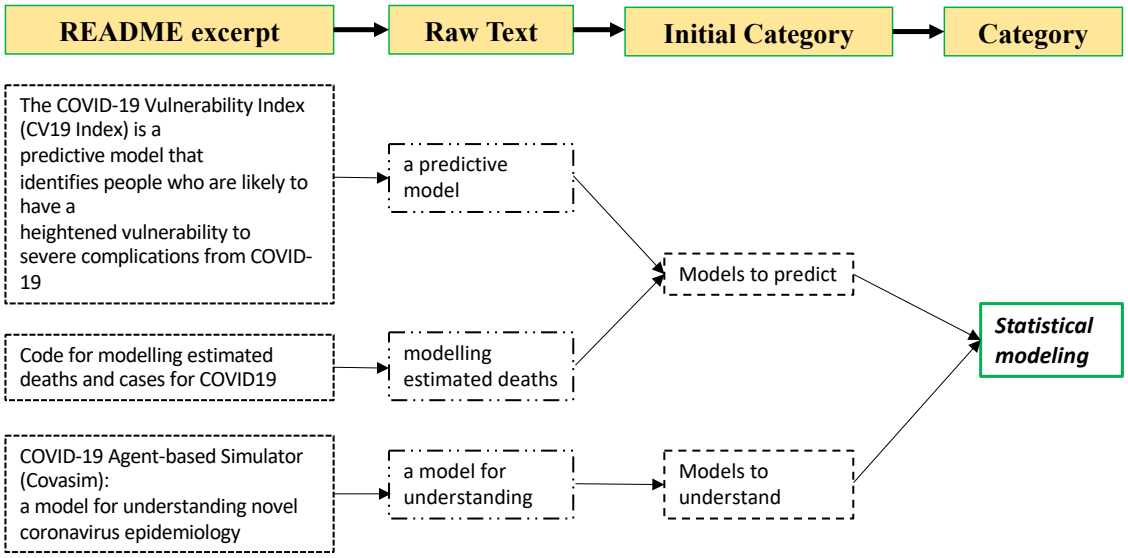

Figure 3. A hypothetical example to demonstrate our process of open coding to categorize COVID-19 software projects.

versity. The rater has a professional experience of 2 years in software engineering and has conduced 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.

$$\text{BugPropAll}(x) = \frac{\text{\# of bug reports labeled as category } x}{\text{total \# of bug reports}} * 100\% \quad (1)$$

$$\text{BugPropCateg}(x, y) = \frac{\text{\# of bug reports labeled as } x, \text{ of project type } y}{\text{\# of bug reports for project type } y} * 100\% \quad (2)$$

- *Step#2-Open coding*: We apply open coding (Saldana, 2015) 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.

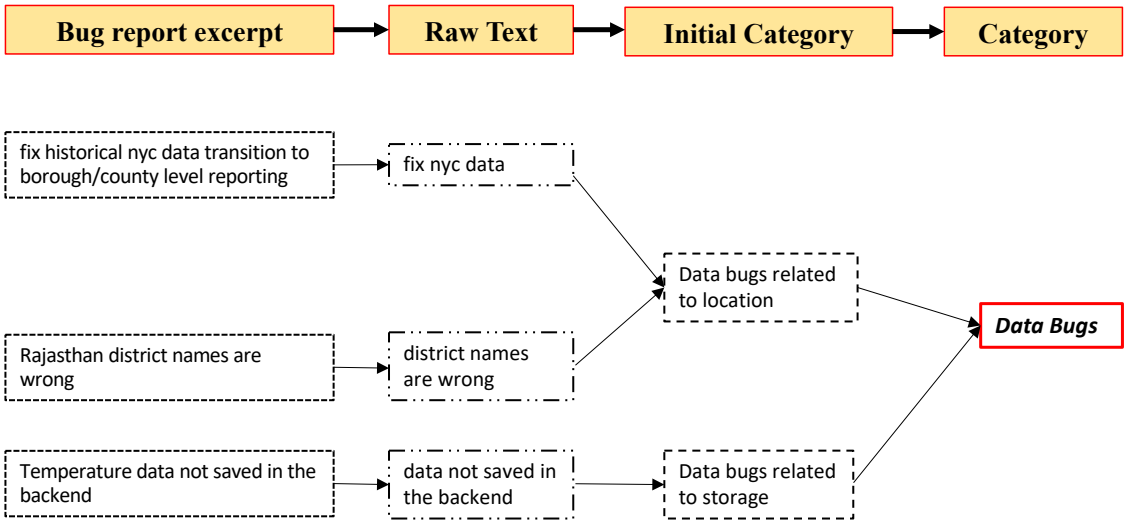

Figure 4. A hypothetical example to demonstrate our process of open coding to identify bug categories for software projects.

- *Step-2*: We read the title, abstract, and keywords to determine if the downloaded papers are related to software bug categorization.
- *Step-3*: Upon completion of Step-2, one rater reads each collected paper, and identifies topics discussed in the paper of interest using qualitative analysis. For each paper the rater determines if the paper focuses on bug categorization. If so, the rater documents the bug categories for the reported software project.

Upon completion of the above-mentioned steps, we derive reported bug categories for multiple software projects.

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.

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

Criteria	GitHub
Initial	3,276
Criterion-1 (Devs >= 2)	1,287
Criterion-2 (Open issues >= 5)	169
Criterion-3 (Commits/month >= 2)	154
Criterion-4 (README is English)	131
Criterion-5 (Actually COVID-19)	129
Final	129

Table 2. Attributes of studied COVID-19 projects.

Attributes	Total
Commits	38,152
Developers	2,243
Duration	12/2019-03/2020
Files	24,839
Issues	4,405
Languages	18
Releases	286
Projects	129

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:

I: ***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.

II: ***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

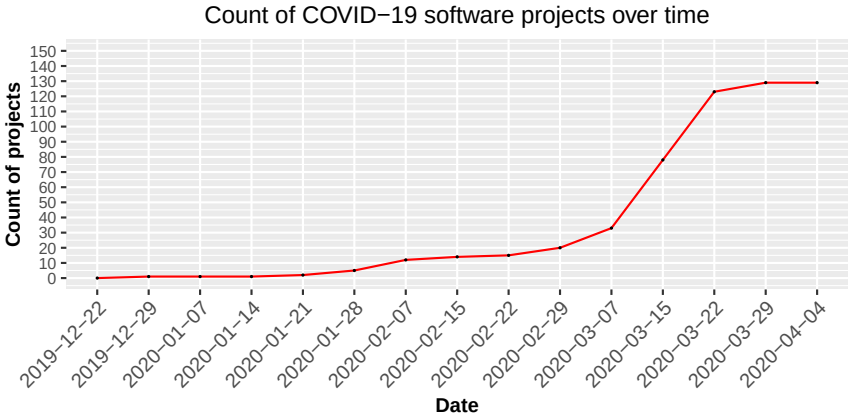

Figure 5. Temporal evolution of COVID-19 software projects based on their creation date. We observe sharp increase in project creation after Feb 29, 2020.

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 it's 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' (Marivate 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 'enigmampc/SafeTrace' (enigmampc, 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' (ImperialCollegeLondon, 2020), (ii) automating the process of lung segmentation with computerized tomography (CT) scan, as done in 'JoHof/lungmask' (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

[3] https://stackoverflow.com/
[4] https://www.facebook.com/groups/opensourcecovid19medicalsupplies/
[5] https://www.openscad.org/

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 [6]. 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:

I: ***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,

[6] https://helpwithcovid.com/medical

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.

Proj. Categ.	Projects	Com.	Devs	Files	Iss.	Rele.
Aggregation	50	14,985	663	8,641	908	72
Mining	35	9,671	894	6,714	515	21
Stat. model.	22	7,214	429	3,464	491	38
Education	9	4,550	196	1,696	406	14
User track	9	2,020	152	2,291	119	286
Volunteer.	7	2,186	143	2,041	320	0
Med. equip.	3	859	38	790	14	63

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

Initial	4,095
Criterion-1 (Closed issues)	2,965
Criterion-2 (Valid bug reports)	550
Final	550

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

Project category	Count (%)
Aggregation	220 (40%)
Mining	150 (27.3%)
Stat. Model.	98 (17.8%)
Education	58 (10.5%)
Volunteer.	40 (7.3%)
User Track	31 (5.6%)
Med. Equip.	4 (0.7%)

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.

🐞 **II**: ***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.
- Mining: Bugs occur when COVID-19-related data is being mined. A practitioner describes a mining bug in a bug report titled "*CDC Children scraper is outdated*" (Timoeller, 2020). The mining tool mines data related to children affected by COVID-19.
- 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*" (zbraniecki, 2020).

🐞 **III**: ***Dependency***:: This category corresponds to bugs that occur when execution of the software is dependent on a software artifact that is either missing or incorrectly specified. For COVID-19 projects, an artifact can be an API or a build artifact.

Example: In a bug report titled "*Missing PostGIS*" (vaclavpavlicek, 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.

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

Example: In a bug report titled "*Missing code of conduct*", a practitioner describes a 'CODE_OF_CONDUCT.md' file to be missing in a Markdown file that describes how practitioners can contribute to the project (mdeous, 2020).

🐞 **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

[7] https://www.covid19india.org/

Table 6. Frequency of identified bug categories. UI-related bugs are the most frequent.

Bug category	BugPropAll (%)
UI	38.2
Data	30.9
Dependency	18.9
Algorithm	7.8
Syntax	6.7
Security	2.5
Performance	1.6
Documentation	1.4

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*" (ivan aksamentov, 2020), a practitioner describes how linting and type checking was disabled for the project, which led to bugs related to linting and type checking.

🐞 **VIII**: ***UI***:: This category corresponds to bugs that involve the user interface (UI) of the software. UI bugs include navigation-related bugs on web pages, bugs related to accessibility, displaying incorrect images, links, and color, and responsiveness.

Example: In a bug report titled "*accessibility fixes*" (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, corresponds 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., 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, 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, 1977). The Cohen's Kappa between the additional rater and the second author is 0.84, indicating 'perfect' agreement (Landis and Koch, 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, 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, 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

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

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

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

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

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.

Project category	Min	Median	Max
Medical Equipment	5.0	29.4	46.4
Volunteer Management System	0.013	21.1	174.2
User Tracking	0.124	16.5	294.5
Education	0.121	11.2	294.5
Aggregation	0.003	8.7	379.4
Statistical Modeling	0.004	7.2	168.3
Mining	0.005	2.5	548.1
All	0.003	7.4	548.0

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

Project category: Aggregation
Definition: Aggregate COVID-19 data and present using visualizations
Count : 50 out of 129 (38.7%)
Most frequent bug category: UI bugs
Median bug resolution time: 8.7 hours

Project category: Mining
Definition: Mine COVID-19 data
Count : 35 out of 129 (27.1%)
Most frequent bug category: Data bugs
Median bug resolution time: 2.5 hours

Project category: Statistical modeling
Definition: Use of statistical models to make COVID-19 predictions
Count : 22 out of 129 (17.0%)
Most frequent bug category: UI bugs
Median bug resolution time: 7.2 hours

Project category: Education
Definition: Educate people about COVID-19
Count : 9 out of 129 (6.9%)
Most frequent bug category: UI bugs
Median bug resolution time: 11.2 hours

Project category: User tracking
Definition: Track user data related to COVID-19
Count : 9 out of 129 (6.9%)
Most frequent bug category: UI bugs
Median bug resolution time: 16.5 hours

Project category: Volunteer management
Definition: Efficiently manage volunteering effort related to COVID-19
Count : 7 out of 129 (5.4%)
Most frequent bug category: UI bugs
Median bug resolution time: 21.1 hours

Project category: Medical equipment
Definition: Source code for design and implementation of medical devices
Count : 3 out of 129 (2.3%)
Most frequent bug category: Dependency bugs

Table 10. Median bug resolution time for each bug category and each project type measured in hours. '—' indicates categories for which no bug reports exist.

	AGG	MINE	STA	EDU	TRAK	VOL	EQU
Bug cat.							
Algorithm	9.8	10.8	13.9	10.1	—	13.5	—
Data	12.2	4.4	15.2	17.0	—	42.0	—
Dependency	5.6	0.1	0.3	4.5	5.3	2.9	22.4
Document	1.3	39.0	1.5	—	—	6.9	—
Performance	7.1	36.6	—	—	1.5	—	—
Security	8.1	—	3.1	84.1	13.9	20.4	—
Syntax	12.1	4.7	11.4	8.6	16.9	79.3	—
UI	8.3	2.7	13.1	16.8	18.7	21.9	46.4

Table 11. Comparison of bug categories of COVID-19 software projects with that of other software project categories.

Bug category	Other software projects
Security	IaC (Rahman et al., 2020), OSS GitHub projects (Ray et al., 2014)
Algorithm	Autonomous vehicle (Garcia et al., 2020), OSS GitHub projects (Ray et al., 2014)
Syntax	IaC (Rahman et al., 2020), deep learning (Islam et al., 2019), OSS GitHub projects (Ray et al., 2014)
UI	Blockchain (Wan et al., 2017)
Data	Deep learning (Islam et al., 2019)
Performance	OSS GitHub projects (Ray et al., 2014)
Dependency	IaC (Rahman et al., 2020)
Documentation	Autonomous vehicle (Garcia et al., 2020), IaC (Rahman et al., 2020)

Median bug resolution time: 29.4 hours

5.2 Implications

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 contract 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*

practitioners: Construction and verification of COVID-19 statistical modeling should involve practitioners from public health and data science. Public health practitioners within a specific locality can provide necessary context that data scientists can incorporate in their statistical models.

Implications for Educators: Our findings have implications for educators involved in teaching the following topics:

- *Data science*: Educators who teach data science can use the examples of statistical modeling bugs to highlight the value of considering the full context and related limitations that accompany statistical modeling.
- *Information security and privacy*: User tracking software can be discussed in information security and privacy courses to demonstrate the value of protecting user data. Such discussion can also include privacy policy frameworks that are already in place, such as the NIST Privacy framework (National Institute of Standard and Technology, 2020).
- *Software engineering*: Our categorization of bugs related to COVID-19 software development can be discussed to demonstrate that understanding and repair of bugs requires contextualization.

Benchmark for practitioners and researchers: Tables 6— 10 can be used as a measuring stick by practitioners and researchers who are involved with COVID-19 software projects. Practitioners can estimate their bug resolution efforts by comparing median resolution times for bugs in their COVID-19 software projects to that of Tables 8, 9, and 10.

Compared to prior work related to blockchain and machine learning (Thung et al., 2012; Wan et al., 2017), median bug resolution time is lower for COVID-19 software projects. We provide two possible explanations: one possible explanation can be related to the sense of urgency. Practitioners may have realized that bugs in COVID-19 software projects could hamper the analysis or mitigation of COVID-19, and therefore, needs immediate attention. Another possible explanation can be the limitations of our dataset. The age of our software projects does not exceed four months and that may have biased median bug resolution time. We advocate for future research that will confirm or refute our explanations.

Recurrence-related implications: Researchers (Kissler et al., 2020; Chen et al., 2020) have provided evidence that support the recurring nature of COVID-19. About the recurrence of COVID-19 Kissler et al. 2020 stated "*a resurgence in contagion could be possible as late as 2024.*". We hypothesize that COVID-19's recurrence will lead to more COVID-19 software building. Whether or not our findings hold for these newly constructed COVID-19 software can be validated through a replication of our paper. We expect to observe more categories of COVID-19 software projects as well as more bug categories.

5.3 Differences between COVID-19 Software Projects and Other Software Projects

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 generates 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 (Attia, 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 OVID-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. Janmanchi et al. 2009 did not systematically study the types of bugs that appear in health care 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 who 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

abquirarte (2020). accessibility fixes. `github.com/cagov/covid19/issues/137`. [Online; accessed 10-May-2020].

Agrawal, A., Rahman, A., Krishna, R., Sobran, A., and Menzies, T. (2018). We don't need another hero?: The impact of "heroes" on software development. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, pages 245–253, New York, NY, USA. ACM.

Alasdair Sandford (2020). Coronavirus: Half of humanity now on lockdown as 90 countries call for confinement. `https://www.euronews.com/2020/04/02/`. [Online; accessed 17-Apr-2020].

Anderson, S., Allen, P., Peckham, S., and Goodwin, N. (2008). Asking the right questions: scoping studies in the commissioning of research on the organisation and delivery of health services. *Health research policy and systems*, 6(1):7.

Apple (2020). Privacy-preserving contact tracing. `https://www.apple.com/covid19/contacttracing`. [Online; accessed 25-May-2020].

Applifting (2020). pomuzeme.si. `github.com/Applifting/pomuzeme.si`. [Online; accessed 09-May-2020].

Attia, P. (2020). Comparing covid-19 to past pandemics, preparing for the future, and reasons for optimism. `https://peterattiamd.com/ameshadalja/`. [Online; accessed 21-May-2020].

Begley, S. (2020a). Death rates should increase when icu's are overwhelmed. `https://github.com/neherlab/covid19_scenarios/issues/7`. [Online; accessed 10-May-2020].

Begley, S. (2020b). Influential covid-19 model uses flawed methods and shouldn't guide u.s. policies, critics say. `https://www.statnews.com/2020/04/17/`. [Online; accessed 10-May-2020].

boogheta (2020). boogheta/coronavirus-countries. `https://github.com/boogheta/coronavirus-countries`. [Online; accessed 09-May-2020].

Butler, J. L. and Jaffe, S. (2020). Challenges and gratitude: A diary study of software engineers working from home during covid-19 pandemic.

Catolino, G., Palomba, F., Zaidman, A., and Ferrucci, F. (2019). Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software*, 152:165 – 181.

CDC (2020). Cases, data, and surveillance. `https://www.cdc.gov/coronavirus/2019-ncov/cases-updates/index.html`. [Online; accessed 09-May-2020].

Chen, D., Xu, W., Lei, Z., Huang, Z., Liu, J., Gao, Z., and Peng, L. (2020). Recurrence of positive sars-cov-2 rna in covid-19: A case report. *International Journal of Infectious Diseases*, 93:297 – 299.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Corey, L., Mascola, J. R., Fauci, A. S., and Collins, F. S. (2020). A strategic approach to covid-19 vaccine r&d. *Science*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Crabtree, B. F. and Miller, W. L. (1999). *Doing qualitative research*. sage publications.

Crowell Morning (2020). Mobile applications for covid tracking & tracing – balancing the need for personal information and privacy rights in the time of coronavirus. `https://www.crowell.com/NewsEvents/AlertsNewsletters/all/`. [Online; accessed 20-May-2020].

De Clercq, E. (2006). Potential antivirals and antiviral strategies against sars coronavirus infections. *Expert review of anti-infective therapy*, 4(2):291–302.

deepset ai (2020). deepset-ai/covid-qa. `https://github.com/deepset-ai/COVID-QA`. [Online; accessed 09-May-2020].

Dehning, J., Zierenberg, J., Spitzner, F. P., Wibral, M., Neto, J. P., Wilczek, M., and Priesemann, V. (2020). Inferring change points in the spread of covid-19 reveals the effectiveness of interventions. *Science*.

elcronos (2020). elcronos/covid-19. `https://github.com/elcronos/COVID-19`. [Online; accessed 09-May-2020].

Emery Berger (2021). Csrankings: Computer science rankings. `http://csrankings.org/#/index?all&us`. [Online; accessed 31-February-2021].

enigmampc (2020). Safetrace. `github.com/enigmampc/SafeTrace`. [Online; accessed 09-May-2020].

Erin Duffin (2020). Impact of the coronavirus pandemic on the global economy - statistics & facts. `https://www.statista.com/topics/6139/covid-19-impact-on-the-global-economy/`. [Online; accessed 08-May-2020].

EuroCrypt (2020a). Eurocrypt 2020 program. `https://eurocrypt.iacr.org/2020/program.php`. [Online; accessed 16-May-2020].

EuroCrypt (2020b). s-212 panel discussion on contact tracing. `https://youtu.be/Xt4P8E_Y-xc`. [Online; accessed 16-May-2020].

Evans, A. B., Blackwell, J., Dolan, P., Fahlén, J., Hoekman, R., Lenneis, V., McNarry, G., Smith, M., and Wilcock, L. (2020). Sport in the face of the covid-19 pandemic: towards an agenda for research in the sociology of sport.

Farhana, E., Imtiaz, N., and Rahman, A. (2019). Synthesizing program execution time discrepancies in julia used for scientific software. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 496–500.

Garcia, J., Feng, Y., Shen, J., Almanee, Sumaya Xia, Y., and Chen, Q. A. (2020). A comprehensive study of autonomous vehicle bugs. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE '20. to appear.

GitHub (2020a). Covid-19 : Github topics. `https://github.com/topics/covid-19`. [Online; accessed 07-May-2020].

GitHub (2020b). Language savant. `https://github.com/github/linguist`. [Online; accessed 07-May-2020].

GitHub (2020c). Search : Covid-19. `https://github.com/search?q=covid-19`. [Online; accessed 07-May-2020].

Greenberg, A. (2020). India's covid-19 contact tracing app could leak patient locations. `https://www.wired.com/story/india-covid-19-contract-tracing-app/`. [Online; accessed 23-May-2020].

Helms, J., Kremer, S., Merdji, H., Clere-Jehl, R., Schenck, M., Kummerlen, C., Collange, O., Boulay, C., Fafi-Kremer, S., Ohana, M., et al. (2020). Neurologic features in severe sars-cov-2 infection. *New England Journal of Medicine*.

helpwithcovid (2020). helpwithcovid/covid-volunteers. `https://github.com/helpwithcovid/covid-volunteers`. [Online; accessed 09-May-2020].

Herzig, K., Just, S., and Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, page 392–401. IEEE Press.

Hu, F. Z. and Qian, J. (2017). Land-based finance, fiscal autonomy and land supply for affordable housing in urban china: A prefecture-level analysis. *Land Use Policy*, 69:454 – 460.

Huang, Y., Sun, M., and Sui, Y. (2020). How digital contact tracing slowed covid-19 in east asia. `https://hbr.org/2020/04/how-digital-contact-tracing-slowed-covid-19`. [Online; accessed 09-May-2020].

IEEE (2010). Ieee standard classification for software anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pages 1–23.

ImperialCollegeLondon (2020). Imperialcollegelondon/covid19model. `https://github.com/ImperialCollegeLondon/covid19model`. [Online; accessed 09-May-2020].

Islam, M. J., Nguyen, G., Pan, R., and Rajan, H. (2019). A comprehensive study on deep learning bug characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 510–520, New York, NY, USA. Association for Computing Machinery.

ivan aksamentov (2020). Fix types and linting errors. `https://github.com/neherlab/covid19_scenarios/issues/101`. [Online; accessed 10-May-2020].

Janamanchi, B., Katsamakas, E., Raghupathi, W., and Gao, W. (2009). The state and profile of open source software projects in health and medical informatics. *International Journal of Medical Informatics*, 78(7):457–472.

Jarynowski, A., Wójta-Kempa, M., Płatek, D., and Czopek, K. (2020). Attempt to understand public health relevant social dimensions of covid-19 outbreak in poland. *Available at SSRN 3570609*.

Jin, Z., Zhao, Y., Sun, Y., Zhang, B., Wang, H., Wu, Y., Zhu, Y., Zhu, C., Hu, T., Du, X., et al. (2020). Structural basis for the inhibition of sars-cov-2 main protease by antineoplastic drug carmofur. *Nature Structural & Molecular Biology*, pages 1–4.

John Hopkins University (2020). Corona Virus Resource Center. `https://coronavirus.jhu.edu/`. [Online; accessed 31-May-2020].

JoHof (2020). Johof/lungmask. `https://github.com/JoHof/lungmask`. [Online; accessed 09-May-2020].

juanmnl (2020). covid19-monitor. `github.com/juanmnl/covid19-monitor`. [Online; accessed 09-May-2020].

Kissler, S. M., Tedijanto, C., Goldstein, E., Grad, Y. H., and Lipsitch, M. (2020). Projecting the transmission dynamics of sars-cov-2 through the postpandemic period. *Science*.

Koerth, M., Bronner, L., and Mithani, J. (2020). Why it's so freaking hard to make a good covid-19 model. `https://fivethirtyeight.com/features/why-its-so-freaking-hard-to-make/`. [Online; accessed 22-May-2020].

Kraemer, M. U., Yang, C.-H., Gutierrez, B., Wu, C.-H., Klein, B., Pigott, D. M., du Plessis, L., Faria, N. R., Li, R., Hanage, W. P., et al. (2020). The effect of human mobility and control measures on the covid-19 epidemic in china. *Science*, 368(6490):493–497.

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174.

landovsky (2020). Fix password reset procedure. `https://github.com/Applifting/pomuzeme.si/issues/99`. [Online; accessed 10-May-2020].

Linares-Vásquez, M., Bavota, G., and Escobar-Velasquez, C. (2017). An empirical study on android-related vulnerabilities. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 2–13, Piscataway, NJ, USA. IEEE Press.

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., and Wang, Y. (2018). Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111.

Ma, W., Chen, L., Zhang, X., Zhou, Y., and Xu, B. (2017). How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, page 381–392. IEEE Press.

makers-for life (2020). makers-for-life/makair. `https://github.com/makers-for-life/makair`. [Online; accessed 09-May-2020].

Marivate, V. and Combrink, H. M. (2020). Use of available data to inform the covid-19 outbreak in south africa: A case study. *Data Science Journal*, 19(1):1–7.

Marivate, V., Nsoesie, E., Bekele, E., and open COVID-19 data working group, A. (2020). Coronavirus COVID-19 (2019-nCoV) Data Repository for Africa.

mdeous (2020). Missing code of conduct. `https://github.com/reach4help/reach4help/issues/135`. [Online; accessed 10-May-2020].

Mello, M. M. and Wang, C. J. (2020). Ethics and governance for digital disease surveillance. *Science*.

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

are still waiting for unemployment benefits. https://www.marketplace.org/2020/05/28/covid-19-jobless-claims-unemployment-benefits-waiting/ [Online; accessed 31-May-2020].

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346.

Munaiah, N., Kroh, S., Cabrey, C., and Nagappan, M. (2017). Curating github for engineered software projects. *Empirical Software Engineering*, pages 1–35.

Munn, Z., Peters, M. D., Stern, C., Tufanaru, C., McArthur, A., and Aromataris, E. (2018). Systematic review or scoping review? guidance for authors when choosing between a systematic or scoping review approach. *BMC medical research methodology*, 18(1):143.

National Institute of Standard and Technology (2020). Nist privacy framework. https://www.nist.gov/privacy-framework. [Online; accessed 24-May-2020].

neherlab (2020). covid19_scenarios. github.com/neherlab/covid19_scenarios. [Online; accessed 09-May-2020].

nthopinion (2020). nthopinion/covid19. https://github.com/nthopinion/covid19. [Online; accessed 09-May-2020].

Oliveira, E., Leal, G., Valente, M. T., Morandini, M., Prikladnicki, R., Pompermaier, L., Chanin, R., Caldeira, C., Machado, L., and de Souza, C. (2020). Surveying the impacts of covid-19 on the perceived productivity of brazilian software developers. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, SBES '20, page 586–595, New York, NY, USA. Association for Computing Machinery.

OpenMined (2020). covid-alert. github.com/OpenMined/covid-alert. [Online; accessed 09-May-2020].

Paul, R., Baltes, S., Gianisa, A., Torkar, R., Kovalenko, V., Marcos, K., Nicole, N., Yoo, S., Xavier, D., Tan, X., et al. (2020). Pandemic programming. *Empirical Software Engineering*, 25(6):4927–4961.

pavel ilin (2020). Temperature data not saved in the backend. https://github.com/COVID-19-electronic-health-system/Corona-tracker/issues/351. [Online; accessed 10-May-2020].

Pei, K., Cao, Y., Yang, J., and Jana, S. (2017). Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 1–18, New York, NY, USA. Association for Computing Machinery.

popsolutions (2020). popsolutions/openventilator. https://github.com/popsolutions/openventilator. [Online; accessed 09-May-2020].

Prana, G. A., Treude, C., Thung, F., Atapattu, T., and Lo, D. (2019). Categorizing the content of github readme files. *Empirical Softw. Engg.*, 24(3):1296–1327.

Pulido, C. M., Villarejo-Carballido, B., Redondo-Sama, G., and Gómez, A. (2020). Covid-19 infodemic: More retweets for science-based information on coronavirus than for false information. *International Sociology*, page 0268580920914755.

Rahman, A. and Farhana, E. (2020). Dataset for Paper: COVID-19-EMSE. https://figshare.com/s/7044678e1d7e7feb1efb. [Online; accessed 22-January-2021].

Rahman, A., Farhana, E., Parnin, C., and Williams, L. (2020). Gang of eight: A defect taxonomy for infrastructure as code scripts. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE '20. to appear.

Ray, B., Posnett, D., Filkov, V., and Devanbu, P. (2014). A large scale study of programming languages and code quality in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, New York, NY, USA. ACM.

reustle (2020). Fix prefecture sorting. https://github.com/reustle/covid19japan/issues/15. [Online; accessed 05-Mar-2021].

Rourke, M., Eccleston-Turner, M., Phelan, A., and Gostin, L. (2020). Policy opportunities to enhance sharing for pandemic research. *Science*, 368(6492):716–718.

Saldana, J. (2015). *The coding manual for qualitative researchers*. Sage.

SinghRajenM (2020). Rajasthan district names are wrong. https://github.com/covid19india/covid19india-react/issues/321. [Online; accessed 10-May-2020].

soroushchehresa (2020). soroushchehresa/awesome-coronavirus. github.com/soroushchehresa/awesome-coronavirus. [Online; accessed 16-May-2020].

Subratappt (2020). Cluster animation slowing down the browser. it also takes much time. https://github.com/covid19india/covid19india-react/issues/497. [Online; accessed 10-May-2020].

Tamm, M. V. (2020). Covid-19 in moscow: prognoses and scenarios. *FARMAKOEKONOMIKA. Modern Pharmacoeconomic and Pharmacoepidemiology*, 13(1):43–51.

Thung, F., Wang, S., Lo, D., and Jiang, L. (2012). An empirical study of bugs in machine learning systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, pages 271–280.

Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 303–314, New York, NY, USA. Association for Computing Machinery.

Timoeller (2020). Cdc children scraper is outdated. https://github.com/deepset-ai/COVID-QA/issues/43. [Online; accessed 10-May-2020].

Tom Simonite (2020). Software that reads ct lung scans had been used primarily to detect cancer. now it's retooled to look for signs of pneumonia caused by coronavirus. https://www.wired.com/story/chinese-hospitals-deploy-ai-help-diagnose/. [Online; accessed 08-May-2020].

vaclavpavlicek (2020). Missing postgis. https://github.

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

Van Bavel, J. J., Baicker, K., Boggio, P. S., Capraro, V., Cichocka, A., Cikara, M., Crockett, M. J., Crum, A. J., Douglas, K. M., Druckman, J. N., et al. (2020). Using social and behavioural science to support covid-19 pandemic response. *Nature Human Behaviour*, pages 1–12.

Vardi, M. Y. (2009). Conferences vs. journals in computing research. *Communications of the ACM*, 52(5):5–5.

Wan, Z., Lo, D., Xia, X., and Cai, L. (2017). Bug characteristics in blockchain systems: A large-scale empirical study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 413–424.

Wang, C., Li, W., Drabek, D., Okba, N. M., van Haperen, R., Osterhaus, A. D., van Kuppeveld, F. J., Haagmans, B. L., Grosveld, F., and Bosch, B.-J. (2020). A human monoclonal antibody blocking sars-cov-2 infection. *Nature Communications*, 11(1):1–6.

WHO (2020). Global research on coronavirus disease (covid-19). https://www.who.int/emergencies/diseases/novel-coronavirus-2019/global-research-on-novel-coronavirus-2019-ncov. [Online; accessed 09-May-2020].

Why Hunger (2020). Why hunger. https://whyhunger.org/map.php. [Online; accessed 08-May-2020].

Will, C. M. (2020). 'and breathe...'? the sociology of health and illness in covid-19 time. *Sociology of Health & Illness*.

Yang, C. Y. and Wang, J. (2020). A mathematical model for the novel coronavirus epidemic in wuhan, china. *Mathematical Biosciences and Engineering*, 17(3):2708–2724.

zbraniecki (2020). Data has a gap between 2020-3-11 and 2020-3-24. https://github.com/covidatlas/coronadatascraper/issues/375. [Online; accessed 10-May-2020].

Zhang, T., Chen, J., Luo, X., and Li, T. (2019). Bug reports for desktop software and mobile apps in github: What's the difference? *IEEE Software*, 36(1):63–71.