2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR) | 979-8-3503-1184-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/MSR59073.2023.00058

Keep the Ball Rolling: Analyzing Release Cadence in GitHub Projects

Oz Kilic
Carleton University
Ottawa, ON, Canada
ozkilic@cmail.carleton.ca

Nathaniel Bowness
University of Ottawa
Ottawa, ON, Canada
nbown088@uottawa.ca

Olga Baysal
Carleton University
Ottawa, ON, Canada
olga.baysal@carleton.ca

***Abstract*—Release cadence is the measure of time between software releases, both internal and external. Few studies analyze popular open-source projects' release cadence and use. In this work, we gathered over 8,000 GitHub projects from four popular programming languages; Go, Java, Python, and Ruby. Project were categorized into *slow*, *modern*, *rapid*, and *rapid*+ release cadence groups. We determined that only 13% of projects had a rapid release cadence of under 30 days. Applying NLP and topic modeling, we extracted the top 5 frequent topics for programming languages and obtained insights into their common uses. For example, Go projects are commonly used for Kubernetes tooling, while Ruby projects often leverage Rails for web development. We observed no significant relationship between frequent topics and the release cadence categories. This finding suggests release cadences are independent of the type of software delivered for a programming language.**

***Index Terms*—Release cadence, README, GitHub, programming languages, topic modeling**

I. Introduction

Release engineering encompasses the entire software release cycle, from planning and development to delivering high-quality software to the end user [1]. A big part of modern release engineering is how quickly software can be released to the customer. Release cadence is the measure of time between software releases, both internal and external. Each project typically embraces a particular release cycle and release cadence.

Release engineering [2; 3; 4] and release cadence [1; 5; 6] have been previously studied. Other techniques, such as continuous development [7; 8], and release pipelines [9; 10], have also been shown to have many benefits. While not many peer-reviewed studies have been published on the release cadence of private companies some news articles and blogs discuss the release pipelines and strategies they use [11; 12]. Similar to this paper, some studies analyze the release cadence of OSS projects [13]. However, the existing studies do not offer a large-scale view of release cadences across multiple projects. Without a larger dataset of projects, it is unclear to what extent software development projects adopt modern release cycles. Also, release cadence may depend on not only the language but also the application/purpose. The goal of this work is to address this gap by performing a release cadence analysis at scale, i.e., GitHub projects developed in different programming languages. To the best of our knowledge, this is the first study to analyze the release cadence at scale. The results of this work can provide developers and researchers insights into how quickly software (e.g., applications) is being released on GitHub.

In this paper, we answer the following research questions:

RQ1: What percentage of projects fall into different release cadence categories? Does the percentage vary between languages?

RQ2: What are the most frequent topics covered in a project's README for each language?

RQ3: Is there a relationship between projects' release cadence category and topics?

II. Methodology

A. Data Collection

To answer our research questions, we mined the World of Code (WoC) dataset [14] which includes projects from GitHub, GitLab, and Bitbucket. We considered the WoC's summary-level information, i.e., project properties that can be queried and filtered using MongoDB with specific criteria [14]. We collected the data for Go, Java, Python, and Ruby projects omitting toy-like, inactive, or very recent projects. Thus, our dataset contains projects with at least 150 commits, 20 stars, 10 contributors, and 10 files of the primary programming languages while having the first commit before 2019 and the last commit in 2021.

We used the GitHub API [15] to retrieve all releases for projects returned from the WoC since WoC did not include the release information that we needed to calculate the average release cadence. Since the GitHub data in the WoC dataset covers up to 2021, we limited the README files collected using the GitHub API to the end of 2021.

Unsurprisingly, some of the projects queried from the WoC dataset were no longer available on GitHub as of December 2022. Table I reports the project count for each filter applied during the data collection process. We discuss these filters and restrictions in the following sections.

B. Release Cadence

The *release cadence* defines how often software is delivered to customers/users. In OSS software development, a release is created whenever a software version is ready for deployment. We applied a number of filters on the collected dataset of project releases to ensure the release cadence can be studied.

TABLE I: Data collection and our datasets (# of projects).

Lang.	WoC	With README	With Releases	**With Filtered READMEs**	**With Filtered Releases**	Total Releases
Go	2,358	2,307	1,317	900	1,152	11,218
Java	1,141	1,139	457	2,149	394	23,463
Python	4,663	4,613	2,330	4,317	2,012	33,310
Ruby	927	925	593	1,063	543	6,667

First, projects with less than two releases were removed from the dataset. At least two releases are required to calculate the average release cadence of the repository. The *average release cadence* was calculated as the average number of days between all consecutive releases. Pre-releases were also removed from the dataset.

Due to the numerous software versioning standards [16; 17], all releases published on GitHub, whether patches, minor updates, or major releases, were treated as a individual releases, and only the date and time of the release are considered.

Once the average release cadence for each project was calculated, each project was classified into one of the four categories. These four categories were selected and based on state-of-the-art research on release engineering. Adams and McIntosh [1] have found that modern release engineering is associated with the release frequency of every 2-6 weeks. They also talked about daily software releases by companies like Netflix and Facebook. A study on OSS release cycles on GitHub has defined a rapid release cycle when software is released every 5 to 35 days [13].

Based on the state-of-the-art research on this topic, we have defined four *release cadence categories* such as **rapid+**, **rapid**, **modern**, and **slow**. These categories include projects that are released faster than seven days, between seven and 30 days, between 30 and 90 days, and slower than 90 days, respectively.

C. Topic Modeling

To classify projects in an unsupervised manner and explore textual patterns, we pre-processed the previously gathered README files as markdown. All files were cleaned of non-ASCII characters and HTML tags, and the files were converted to HTML. We discarded files without at least one H1 or H2 element. By manually analyzing samples from the dataset, we observed that the most informative part of the files is the first paragraph, describing the purpose/function of a project. Therefore, we retrieved the first text paragraph that had at least four characters and three token outputs.

After running the files through an NLP pipeline with typical steps (e.g., lemmatization, negation handling) and obtaining tokenized representations of the project README files, we identified a few projects with suspiciously lengthy first paragraphs (more than 250 tokens). We observed that one project had mostly foreign characters in this paragraph, while the rest had paragraphs that were not properly formatted. Such projects were discarded. The remaining projects, grouped by language, had on average, an approximate token count of 12–15 in their relevant paragraph.

Topic modeling is an unsupervised process of extracting topics in text mining. Currently, the most common topic modeling approaches are based on Latent Dirichlet Allocation (LDA) [18], and have been leveraged in the software engineering research [19; 20; 21]. Topic modeling was applied to the remaining 900, 2,149, 4,317, and 1,063 projects written in Go, Java, Python, and Ruby, respectively. We leveraged Biterm Topic Modeling (BTM), an improved version of LDA, due to its superiority with short texts compared to LDA [22]. We used the full paragraphs as our context window to extract biterms. Discarding the bottom 2% occurrences in the corpus, we limited our corpus to 94, 79, 86, and 71 terms for Go, Java, Python, and Ruby, respectively.

We used lower prior alpha probabilities to obtain more cohesive and distinct topics. For each language, we used hyperparameter optimization to decide the number of topics, between 5 and 50, and the alpha division factor, between 2 and 50, where the alpha is 1/(number of topics * division factor). We optimized these parameters to minimize the entropy calculated from the top five topics based on the number of documents assigned to them using the highest topic likelihood. We manually explored the resulting major topic themes for each language by analyzing the top terms and documents for each topic. From the top five topics, we computed the top 10 relevant terms based on the term relevance formula [23]. We used a λ value of 0.6 as suggested. For each model, we used the suggested number of iterations for efficiency [24].

D. Topics and Cadences

We selected projects that had both extracted topics and average release cadences, i.e., 524, 1,067, 1,905, and 384 projects written in Go, Java, Python, and Ruby, respectively. To explore the relationship between topics and release cadences, we conducted two alternative non-parametric tests. First, for each topic, we used the Kruskal-Wallis test to see whether there is a significant rank distribution difference between cadence groups based on their topic likelihoods. We believe this is a more sound approach as it does not strictly pigeonhole repositories into a single topic.

Then, by combining the rapid cadence groups (*rapid* and *rapid+*) and non-rapid cadence groups (*modern* and *slow*), we obtained a rapidness feature. We used this feature to conduct Fisher's exact test for each topic. For this test, we assigned topics on repositories based on the repositories' highest topic likelihoods.

III. Results and Discussion

The dataset for this project is composed of popular GitHub projects from four popular programming languages. After filters, the dataset contained 74,658 releases from 4,101 GitHub projects (45% of the total projects considered) that were analyzed to find the average release cadence. Topic modeling was applied to 8,429 READMEs.

A. What percentage of projects fall into different release cadence categories? Does the percentage vary between languages? (RQ1)

1) Release Cadence Categorization: We analyzed the average release cadence of the projects in two categories. First, we looked at projects with commits before 2019 and after 2021 (all-time). However, it is possible that some projects were not active throughout that entire interval. Some projects were also created before 2015, so they may have been inactive for years but have recently become active again. To address the problem of long-running projects affecting the results, for the same projects, release cadences were also categorized using only the releases from 2022, disregarding earlier releases. For both versions, we later calculated the median release cadence.

The median cadence for all the projects was 100 days when considering all releases, corresponding to the *slow* category. The 25th and 75th percentiles of the average cadences were 50 and 200.3 days. The distribution of the projects in the *slow*, *modern*, *rapid*, and *rapid+* categories were 54.8%, 32.6%, 10.4%, and 2.24%, respectively. The significant majority of projects being *slow* or *modern* may be attributed to the time for which these projects have been active.

The distribution of the release cadence categories using 2022 releases shows a different picture, as all categories except *slow* become significantly more common. The 25th, 50th (median), and 75th percentiles of average release cadences were 22.2, 44.3, and 82 days in 2022. This release cadence improvement is likely to be also attributed to more modern release engineering practices, such as CI/CD, agile methodologies, and a more significant focus on delivering software quickly.

2) Release Cadence per Language: The median release cadence for projects in the four different languages was 75.2, 92.8, 107.8, and 132 days for Go, Java, Python, and Ruby, respectively, shown in Table II. Based on the categorization, only Go has the majority of projects that release with a *modern* or better faster cadence. Java trails Go, with 48% of the projects releasing with *modern* or faster release cadences. Both Go and Java are considered multi-purpose, statically-typed languages. The faster release cadence for projects in these languages may be attributed to their applications and usage. Many OSS projects in these languages, such as Spring [25] and Kubernetes [26], have been available for years and supported by large companies. It should also be noted that Go is the newest of the four programming languages, released as stable in 2012, so it may not be as influenced by long-running GitHub projects [27]. The two languages with the slowest average release cadence are Ruby and Python. Both of these languages can be described as scripting languages. Once again, the applications developed in these languages may influence the average release cadence.

Similarly, the distribution of the release cadence categories from only 2022 shows that all languages have a shorter average release cadence. The most significant change was for Ruby, with a significant percentage of projects moving from *slow* to *modern* or faster cadence categories in 2022.

A possible explanation for Ruby's cadence change can be a result of large companies such as Shopify making significant investments in the language [28]. This work can be extended to study the evolution of the release cadence for a language since programming languages may undergo changes, paradigm shifts, and community support.

TABLE II: Release cadence category (%) per language.

Time	Release Cadence	Go	Java	Python	Ruby	Total
All-time	Slow	43.6	51.6	57.7	64.7	54.8
	Modern	37.2	35	31.4	24.9	32.6
	Rapid	15.5	11.3	8.95	8.12	10.4
	Rapid+	3.68	2.08	1.94	2.28	2.24
2022	Slow	20.6	21	23.4	21.4	22.2
	Modern	48.6	41.4	43.9	35.7	43.4
	Rapid	26.3	29.4	24.6	30.4	26.5
	Rapid+	4.53	8.17	8.14	12.5	7.89

B. What are the most frequent topics covered in a project's README for each language? (RQ2)

Hyperparameter optimization yielded five topics for all languages except Python, for which we found that having eight topics was the best. The top five topics, along with their document counts, top 10 relevant terms, top two-three repositories, and manually generated summaries are reported for each language in Table III.

Our topic terms and manual inspections showed that some READMEs had meta-informative first paragraphs. Rather than explaining what the project does, these paragraphs include but are not limited to explanations about the project not being maintained, version information, dependencies, and directives to users/contributors. Nevertheless, our topic modeling shows it is possible to extract valuable information from README files. Although all languages had one or more topics related to various tools and libraries, some other topics showed relatively niche use cases for the programming language.

Data interchange, API clients, and Kubernetes management were popular domains for Go projects. Considering that both Go and Kubernetes were developed by Google [29; 30], and Kubernetes is written in Go [26], Go seems to be the natural choice of language for Kubernetes-related projects. Furthermore, while it is a relatively small-sized topic, one topic references other programming languages that are written and supported by Go, such as Rum and Tengo.

Although Kotlin was announced to be supported in Android development in 2017 [31], our results suggest that Android-focused projects and libraries dominate Java topics. There were topics referencing software plugins and web applications such as Spring, but they were seen in fewer projects. Another topic, albeit relatively small, was specifically about Android view libraries that provide UI components.

Nowadays, Python is mainly known for its machine learning and scripting uses, which is observable in our topic results in two separate groups. However, the second-largest Python topic indicates a significant portion is oriented toward web applications, specifically Django. Python projects had a greater number of topics after optimization. The largest topics were related to tools/libraries/implementations. Since the topics in Python were much more balanced compared to other languages it may suggest Python has a wider variety of use cases.

Multiple Ruby topics were related to a web framework called Ruby on Rails. This indicates that Rails is a popular framework for web development in the Ruby programming

TABLE III: Summary of the top five topics per language.

Lang.	Topic ID	Documents	Top Terms	Topic Summary	Top Repositories
Go	3	520	standard, json, format, sql, need, implement, database, file, package, build	Data related projects, data interchange protocols, and JSON	capnproto/go-capnproto2, emersion/go-message, goccy/go-json
	0	226	github, com, framework, set, data, cloud, function, enable, open, allow	Tools and frameworks	genuinetools/ghb0t, google/cloud-print-connector, isiaon/esm
	1	84	client, api, version, new, http, please, repository, request, window, change	API clients and projects with meteainformative first paragraphs	bxcodec/faker, cloudfoundry/diego-ssh, cloudfoundry/lager
	4	44	kubernetes, manage, operator, resource, cluster, terraform, platform, provider, like, base	Libraries facilitating managing Kubernetes systems	ccojocar/sso-operator, etungsten/bottlerocket-update-operator, jgramoll/terraform-provider-spinnaker
	2	10	language, feature, framework, time, include, system, command, example, line, native	Languages written in Go and other miscellaneous projects	avelino/gin, d5/tengo, dbl0null/flaggy
Java	3	1502	android, app, user, database, design, format, SDK, google, write, library	Android-focused projects and libraries	alexcohn/ffmpeg-android, atakli/shareviahttp, douglasjunior/android-simple-tooltip
	0	394	build, plugin, gradle, test, maven, system, set, run, help, allow	Plugins for different software written in Java	bitbucket.org/ijabz/jaudiotagger, ChestShop-authors/ChestShop-3, git.eclipse.org/r/emf
	2	132	see, language, apache, development, contain, start, example, please, repository, version	Miscellaneous projects with meta-informative first paragraphs	android-password-store/Android-Password-Store, camunda/camunda-bpm-reactor, ebanx/swipe-button
	4	38	image, high, time, open, feature, way, tool, source, find, need	User interface (UI) and Android view libraries	akshay2211/PixImagePicker, Chufyjj/Ghost, fiji/Stitching
	1	30	spring, service, api, web, client, interface, application, implementation, implement, provide	Web applications and libraries	0SRYPPR0/APKMirror, eclipse-ee4j/cdi, florianberthe/spring-authorization-server
Python	2	1500	data, task, system, format, read, machine, run, one, module, file	Miscellaneous tools, libraries, and implementations	Boulder-Investment-Technologies/lppls, chakki-works/seqeval
	0	821	django, framework, platform, application, project, collection, easy, database, allow, simple	Web application-related projects	awaazde/drf-friendly-errors, aykut/django-bulk-update, bitmazk/django-influxdb-metrics
	1	637	network, learning, model, machine, pytorch, well, develop, data, implement, language	Machine learning applications, various tools and libraries	asteroid-team/torch-audiomentations, by2101/warp-transducer, Ekultek/WhatWaf
	4	494	interface, client, line, command, api, server, library, function, wrapper, provide	Command line interfaces and various tools	aashutoshrathi/git-stalk-cli, dbcli/mycli, dnouri/dstoolbox
	7	352	source, open, program, file, format, analysis, like, tool, language, help	Miscellaneous projects and projects with meta-informative first paragraphs	alisaifee/jira-cli, brunobord/the-black-hack, Egoistically/ALAuto
Ruby	4	621	need, get, want, extension, make repository, add, rail, work, app	Miscellaneous projects, projects with meta-informative first paragraphs	ahoward/map, braintree/pg_ha_migrations, copiousfreetime/amalgalite
	0	197	tool, test, command, line, run, code, manage, help, check, write	Various tools and libraries, command line programs	floraison/fugit, hercules-team/augeas, joenorton/rubyretriever
	3	114	api, official, client, service, request, wrapper, build, implementation, framework, feature	API clients	3scale/3scale_ws_api_for_ruby, basecrm/basecrm-ruby, cburnette/boxr
	1	100	rails, user, class, generate, model, like, object, also, language, easy	Web application libraries and utilities	amoeba-rb/amoeba, asenchi/scrolls, comma-csv/comma
	2	27	officially, following, support, project, library, please, activerecord, use, base, extension	Projects with meta-informative first paragraphs	collectiveidea/delayed_job_mongoid, dry-rb/dry-auto_inject, dry-rb/dry-configurable

language. Apart from Rails, it seems Ruby has utility-related uses in general. The largest topic having meta-informative projects indicates Ruby's topic modeling may have been influenced by general words and usage comments more than other languages. Ruby has typically been associated with scripting and quickly created programs in the past, and this was prevalent in the topics.

Overall, the topic modeling from the README files successfully identified key insights into what projects are used for with minimal overlap between the topics. This includes insights into popular frameworks, tooling, usage patterns, and applications for the different programming languages.

C. Is there a relationship between projects' release cadence category and topics? (RQ3)

We applied non-parametric tests for each topic of a language. Our Kruskal-Wallis test resulted in average statistic values of 3.927, 2.497, 3.264, and 2.727 for Go, Java, Python, and Ruby projects, respectively. Meanwhile, the Fisher's Exact Test reported average odds ratios of 1.089, 0.879, 1.011, and 0.826, in the same order. However, both tests using topic distributions and cadence groups showed there was no significant difference in topics among cadence groups for any of the languages. Therefore, we did not conduct post hoc tests.

IV. CONCLUSION

In this study, we collected open-source GitHub projects written in Go, Java, Python, and Ruby to study their average release cadences. We classified projects into different release cadence categories. The project release categories and topics were then analyzed for any patterns. We found that most of the projects in our dataset fall into the *slow* category. Only 13% of the projects can be categorized as *rapid*. By applying topic modeling, we identified the key applications for the programming languages. However, we did not observe significant release cadence differences among different topics. Our findings indicate the release cadence may be impartial to the software application and rather related to the desired release schedule for the project.

Our study is subject to several limitations. There is no consensus on release cadence intervals; these intervals are based on classifications defined by existing work [1; 13; 32]. Project properties may have changed between 2021 and our collection of READMEs and releases from WoC in December 2022. Projects' current numbers of stars and latest commits were automatically checked before collection to minimize this issue. The replication package including the dataset, analyses, and results is made publicly available [33].

Acknowledgements

Kilic and Baysal acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2021-03809.

References

[1] B. Adams and S. McIntosh, "Modern release engineering in a nutshell – why researchers should care," in *2016 IEEE 23rd Int. Conf. on Softw. Analysis, Evolution, and Reengineering (SANER)*, vol. 5, 2016, pp. 78–90.

[2] G. Schuh and W. Eversheim, "Release-engineering—an approach to control rising system-complexity," *CIRP Annals*, vol. 53, no. 1, pp. 167–170, 2004.

[3] H. K. Wright and D. E. Perry, "Release engineering practices and pitfalls," in *2012 34th Int. Conf. on Softw. Eng. (ICSE)*, 2012, pp. 1281–1284.

[4] A. Dyck, R. Penners, and H. Lichter, "Towards definitions for release eng. and devops," in *2015 IEEE/ACM 3rd Int. Workshop on Release Eng.*, 2015, pp. 3–3.

[5] K. Okumoto and A. L. Goel, "Optimum release time for software systems based on reliability and cost criteria," *Journal of Systems and Softw.*, vol. 1, pp. 315–318, 1979.

[6] S. Yamada and S. Osaki, "Cost-reliability optimal release policies for softw. systems," *IEEE Transactions on Reliability*, vol. R-34, no. 5, pp. 422–424, 1985.

[7] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[8] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Softw.*, vol. 32, no. 2, pp. 50–54, 2015.

[9] L. Bass, I. Weber, and L. Zhu, *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[10] S. Nandgaonkar and V. Khatavkar, "CI-CD pipeline for content releases," in *2022 IEEE 3rd Global Conf. for Advancement in Technol. (GCAT)*, 2022, pp. 1–4.

[11] S. Shankland, "Rapid-release Firefox meets corporate backlash," 2011, https://www.cnet.com/culture/rapid-release-firefox-meets-corporate-backlash/.

[12] ——, "Google ethos speeds up Chrome release cycle," 2010, [Online]. Available: https://www.cnet.com/culture/google-ethos-speeds-up-chrome-release-cycle/.

[13] S. D. Joshi and S. Chimalakonda, "Rapidrelease-a dataset of projects and issues on GitHub with rapid releases," in *2019 IEEE/ACM 16th Int. Conf. on Mining Softw. Repositories (MSR)*. IEEE, 2019, pp. 587–591.

[14] A. Mockus, A. Nolte, and J. Herbsleb, "MSR Mining Challenge: World of Code," 2023.

[15] GitHub, "GitHub REST API," Dec 2022. [Online]. Available: https://docs.github.com/en/rest/

[16] T. Preston-Werner, "Semantic versioning 2.0.0." [Online]. Available: https://semver.org/

[17] C. Commons, "Calendar versioning." [Online]. Available: https://calver.org/

[18] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[19] N. Orii, "Collaborative topic modeling for recommending GitHub repositories," *Inf. Softw. Technol.*, vol. 83, no. 2, pp. 110–121, 2012.

[20] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *Proceedings of the 22nd ACM SIGSOFT Int. Symposium on Foundations of Softw. Eng.*, 2014, pp. 155–165.

[21] M. M. Rahman and C. K. Roy, "An insight into the pull requests of GitHub," in *Proceedings of the 11th Working Conf. on Mining Softw. Repositories*, 2014, pp. 364–367.

[22] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *Proceedings of the 22nd Int. Conf. on World Wide Web*, 2013, pp. 1445–1456.

[23] C. Sievert and K. Shirley, "LDAvis: A method for visualizing and interpreting topics," in *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, 2014, pp. 63–70.

[24] Bitermplus documentation, "Benchmarks." [Online]. Available: https://bitermplus.readthedocs.io/en/latest/benchmarks.html

[25] Spring, "spring-projects/spring-framework," Dec 2022. [Online]. Available: https://github.com/spring-projects/spring-framework

[26] Kubernetes, "Kubernetes/Kubernetes: Production-grade container scheduling and management." [Online]. Available: https://github.com/kubernetes/kubernetes

[27] Go Community, "Release history - the Go programming language," Dec 2022. [Online]. Available: https://go.dev/doc/devel/release

[28] "Shopify invests in research for Ruby at scale," May 2022. [Online]. Available: https://shopify.engineering/shopify-ruby-at-scale-research-investment

[29] "Build simple, secure, scalable systems with Go." [Online]. Available: https://go.dev/

[30] Google, "What is Kubernetes? — Google Cloud." [Online]. Available: https://cloud.google.com/learn/what-is-kubernetes

[31] "Kotlin on Android. Now official: The Kotlin blog." [Online]. Available: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/

[32] D. Chakroborti, S. S. Nath, K. A. Schneider, and C. K. Roy, "Release conventions of open-source software: An exploratory study," *Journal of Software: Evolution and Process*, p. e2499, 2022.

[33] N. Bowness, O. Kilic, and O. Baysal, "Release Cadence: Replication Package," Feb 2023, [Online]. Available: https://osf.io/v7jc4/?view_only=6a88e7d3a17e4ba1857c2389b9465b78.