Monorepos: A Multivocal Literature Review

Gleison Brito[1], Ricardo Terra[2], Marco Tulio Valente[1]

[1]Federal University of Minas Gerais, Belo Horizonte, Brazil

[2]Federal University of Lavras, Lavras, Brazil

{gleison.brito,mtov}@dcc.ufmg.br, terra@dcc.ufla.br

Abstract. *Monorepos (Monolithic Repositories) are used by large companies, such as Google and Facebook, and by popular open-source projects, such as Babel and Ember. This study provides an overview on the definition and characteristics of monorepos as well as on their benefits and challenges. Thereupon, we conducted a multivocal literature review on mostly grey literature. Our findings are fourfold. First, monorepos are single repositories that contains multiple projects, related or unrelated, sharing the same dependencies. Second, centralization and standardization are some key characteristics. Third, the main benefits include simplified dependencies, coordination of cross-project changes, and easy refactoring. Fourth, code health, codebase complexity, and tooling investments for both development and execution are considered the main challenges.*

1. Introduction

Monorepos (Monolithic Repositories or Multi-Package Repositories) are commonly described as a single repository containing more than one project, in contrast to the single-repository-per-project model [9]. This model is adopted for several large software companies, including *Facebook*, *Google*, and *Microsoft* [2, 5, 8]. Some popular *open-source* projects also adopt this model to manage their repositories (e.g., Babel and Ember).

The discussion about the adoption of monorepos is an emerging theme in the developer community. The theme is discussed in several forums and blogs, where the adoption of monorepos is either defended or rebutted. There are also much debate about the migration of multiple repositories to a single one, mainly motivated by the adoption of the monorepo model by large companies such as Google and Facebook. However, there is no consensus on the benefits and challenges of this repository model.

In view of such context, this paper provides an overview on monorepos. Thereupon, we conduced a Multivocal Literature Review based on 21 grey literature and two academic papers in order to: (i) undestand the definitions of monorepo, (ii) identify characteristics of monorepos, (iii) investigate benefits of monorepos, and (iv) investigate challenges of monorepos.

Regarding (i), monorepos are usually defined as a single repository that contains multiple projects related or unrelated, but there are two kinds of monorepos: "Monstrous" monorepos, which are commonly used by large companies, such as Google and Facebook, and Projects monorepos, which are used by medium-size open-source projects, such as React and Babel. Regarding (ii), the projects into a Monorepo share the same managing tools and the same version of dependencies, besides all projects are visible to contributors. Regarding (iii), benefits include simplified dependencies, better managing of cross-project changes, easy refactoring, simplified organization, improve overall work culture,

better coodination between developers, and better support of build tools to manage the repository. Regarding (iv), challenges include difficulties with code heath, codebase complexity, tooling investments, loss of version information, build, deploy and test tasks, and migration of many repositories to only one.

The remainder of this paper is organized as follows. Section 2 describes our research metodology. Section 3 reports the results of our multivocal literature review. Section 4 discusses the threats to validity of our study and Section 5 concludes.

2. Research Metodology

In this section, we describe our research methodology. We also provide an overview of the systematic approach used to gather relevant literature.

2.1. Literature Review

After an initial search in the literature to learn more on the topic of monorepos, we could not find a substantial body of academic research on the topic. We therefore decided to conduct a Multivocal Literature Review (MLR), which is based on all accessible literature on a topic [7]. This includes—but is not limited to—blogs, white papers, articles, and academic literature. By using this wide spectrum of literature, the results will give a more broad view at the topic since they include the voices and opinions of academics, practitioners, independent researchers, development firms, and others who have experience on the topic [7].

Garousi et al. [3] emphasize the importance of MLRs in the Software Engineering (SE) field by stating that SE practitioners produces grey literature on a great scale, but most are not published in academic vehicles. Therefore, they argue that not including that literature in systematic reviews implies in researchers missing out important current state-of-the-art practice in SE.

2.2. Research Questions

We conducted this MLR to obtain an understanding of what a monorepo model is, what are the key characteristics of this model, and what are the benefits and challenges of adopting it. In order to achieve the goal, we formulate four research questions:

RQ #1. How does the literature define monorepos?

RQ #2. What are the characteristics of monorepos?

RQ #3. What are the main expected benefits of adopting monorepos?

RQ #4. What are the main expected challenges of adopting monorepos?

2.3. Study Protocol

This section describes the systematic protocol we followed to retrieve the literature used in our study. We describe the databases, the search strategy used to find related literature, the inclusion and exclusion criteria used to find the most relevant literature, and the process in which we catalogued the literature.

Databases. We relied on Google's search engine to find relevant literature:

- **Google Search**[1] to locate grey literature (white papers, blogs, articles, etc.)
- **Google Scholar**[2] to specifically locate academic literature.

We chose Google's search engines instead of more traditional search engines (like Springer Link, ACM Digital Library, IEEE Explore, etc.) because Monorepo is a very new topic and limited academic research is available. We therefore knew before-hand that this literature review would rely mostly on grey literature.

Search Terms. The search string were built following the steps proposed by Brereton et al. [1]:

1. Derive major terms from the research questions by identifying the main concepts.

2. Identify alternative spellings and synonyms for major terms.

3. Check the keywords in any relevant papers we already have.

4. Use the boolean OR to add alternatives spellings and synonyms.

5. Use the boolean AND to link the major terms.

Since the search strings aim to find relevant literature related to the RQs, we defined them as follows:

```
("monorepo" OR "monolithic repository" OR "multi-package repository")
                               AND
("definition" OR "definitions" OR "characteristic" OR "characteristics"
OR "benefit" OR "benefits" OR "challenge" OR "challenges")
```

Study Selection. After retrieving the results of the initial search, we excluded irrelevant articles using the following inclusion and exclusion criteria:

- Inclusion criteria:

    - Literature that explicitly discuss monorepos;

    - Literature that explicitly discuss the challenges and benefits of monorepos;

    - Literature that discuss the definition of monorepos;

    - Literature published after 2014; and

    - Literature that appears in the first five pages on Google's search.

- Exclusion criteria:

    - Inaccessible literature;

    - Results that Google Search deems to similar to other results; and

   - Vendors tool advertisements.

Search Procedure. As illustrated in Figure 1, we first perform an advanced search in Google Search and Google Scholar. For a better focus on each RQs, we split our search

---

[1]http://www.google.com/
[2]http://scholar.google.com/

term in four parts. We analyze 20 pages in total, five pages for each part. We therefore applied the inclusion and exclusion criteria, selecting only relevant literature for the primary study. In order to identify and incorporate the most relevant grey literature in our MLR, we again use the guidelines defined by Garousi et al. [4].
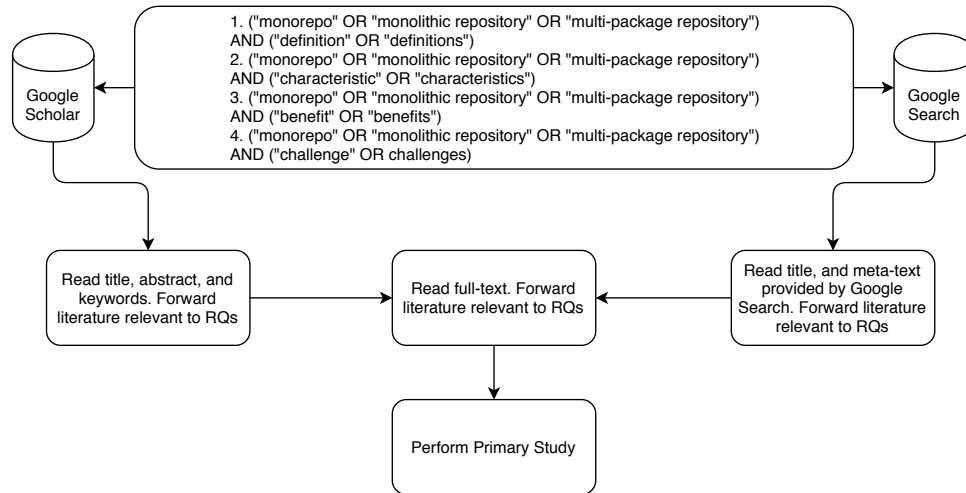
Figure 1. Search process to find relevant literature.

We performed our search procedure in June, 2018. From a total of 255 publications, we applied the inclusion and exclusion criteria to select 23 publications, as properly reported in Table 1.

3. Multivocal Literature Review

Based on the research methodology described in the previous section, this section reports the results of our multivocal literature review.

3.1. Definition of Monorepo (RQ #1)

The most common definition of Monorepo is a single repository that contains multiple projects (publications #4, #11, #13, and #19). These projects can be related or unrelated, but the fact is that they should share the same dependencies. Particularly, publications #11 and #13 define monorepos as a single repository that contains more than one logical project. The projects managed in a Monorepo can depend on each other (such as React and the react-dom package) or they can be completely unrelated (such as the Google search algorithm and Angular).

Monorepos can also be classified as *Monstrous monorepos* or *Project monorepos*, according to publication #13. Monstrous monorepos regards the sheer size to which monorepos at organizations can grow and Project monorepos describes single repositories that are used to manage the core functionality of a project and all of its components. Google and Facebook repositories are examples of Monstruous monorepos, as discussed in publications #12 and #22. Project monorepos are commonly adopted by open-source projects with many modules, such as Babel and Ember. According to publication #14, the monorepo model allows the maintenance of multiple related packages within a single repository.

---

[3]We have double checked all URLs on July 2nd, 2018.

Table 1. List of selected publications[3]

Literature	#	Publications
Grey	1	Anderson, B. (2017). **Code Repositories and Yak Shaving**. http://iamtherealbill.com/2017/01/repo-yak-shaving/
	2	Belagatti, P. (2016). **Microservices: Mono repo vs. multiple repositories**. https://jaxenter.com/microservices-mono-repo-vs-multiple-repositories-130148.html
	3	Karanth, D. (2016). **Microservices: Pros and Cons of Mono Repos**. https://dzone.com/articles/microservices-pros-and-cons-of-mono-repos
	4	Eberlei, B. (2015). **Monorepos**. https://qafoo.com/talks/15_10_symfony_live_berlin_monorepos.pdf
	5	Long, C. (2017). **Multirepo vs Monorepo**. https://chengl.com/multirepo-vs-monorepo/
	6	Libbey, B. (2017). **Monorepo, Manyrepo, Metarepo**. http://notes.burke.libbey.me/metarepo/
	7	Johnson, N. (2017). **Monorepo**. https://www.yonson.io/post/monorepo/
	8	Farina, M. (2016). **Dangers of Monorepo Projects**. https://dzone.com/articles/dangers-of-monorepo-projects
	9	Das, S. (2017). **Code repository for micro-services: mono repository or multiple repositories**. https://medium.com/@somakdas/code-repository-for-micro-services-mono-repository-or-multiple-repositories-d9ad6a8f6e0e
	10	Pendleton, B. (2017). **Big news in the world of source control**. http://bryanpendleton.blogspot.com/2017/02/big-news-in-world-of-source-control.html
	11	Saase, S. (2015). **Monorepos in Git**. https://developer.atlassian.com/blog/2015/10/monorepos-in-git/
	12	Goode, D. (2014). **Scaling Mercurial at Facebook**. https://code.facebook.com/posts/218678814984400/scaling-mercurial-at-facebook/
	13	Oberlehner, M. (2017). **Monorepos in the Wild**. https://medium.com/@maoberlehner/monorepos-in-the-wild-33c6eb246cb9
	14	Vepsäläinen, J. (2017). **Managing Packages Using a Monorepo**. https://survivejs.com/maintenance/appendices/monorepos/
	15	Seibel, P. (2017). **Repo style wars: mono vs multi**. http://www.gigamonkeys.com/mono-vs-multi/
	16	Luu, D. (2015). **Advantages of monorepos**. https://danluu.com/monorepo/
	17	MacIver, D. (2016). **Why you should use a single repository for all your company's projects**. https://www.drmaciver.com/2016/10/why-you-should-use-a-single-repository-for-all-your-companys-projects/
	18	Fabulich, D. (2017). **We'll Never Know Whether monorepos Are Better**. https://redfin.engineering/well-never-know-whether-monorepos-are-better-2c08ab9324c0
	19	Szorc, G. (2014). **On Monolithic Repositories**. https://gregoryszorc.com/blog/2014/09/09/on-monolithic-repositories/
	20	Beigui, P. (2016). **Mono-Repos @ Google. Are they worth it?**. https://medium.com/@pejvan/monorepos-85e608d43b57
	21	Lucido, A. (2017). **The Journey To Android Monorepo: The History Of Uber Engineering's Android Codebase Organization**. https://eng.uber.com/android-monorepo/
Academic	22	Potvin, R., & Levenberg, J. (2016). **Why Google stores billions of lines of code in a single repository**. Communications of the ACM, 59(7), 78-87.
	23	Jaspan, C. et al. (2018). **Advantages and Disadvantages of a Monolithic Repository - A case study at Google**. 40th International Conference on Software Engineering (ICSE), Software Engineering in Practice (SEIP) Track, 225-234.

3.2. Characteristics of Monorepos (RQ #2)

According to publications #22 and #23, the five main characteristics of monorepos are:

- *Centralization*: Codebase is in a single repository encompassing multiple projects.

- *Standardization*: A shared set of tools govern how engineers interact with the code, including building, testing, browsing, and reviewing code.
- *Visibility*: Code is viewable and searchable by all engineers in the organization.
- *Synchronization*: The development process is trunk-based; engineers should always commit to the head of the repository.
- *Completeness*: Any project in the repository can be built only from dependencies also checked into the repository. Dependencies are unversioned; projects must use whatever version of their dependency at the repository head.

3.3. Benefits of monorepos (RQ #3)

This section provides an overview on the benefits of monorepo model.

- *Simplified dependencies*: As discussed in publications #6 and #16, monorepo model proposes to have one universal version number for all projects. Since atomic cross-project commits are possible, the repository is always in a consistent state. Library versioning is de-emphasized. Instead, a library is expected to maintain a stable API and migrate its callers when this API changes. This depends on being able to make atomic commits.
- *Cross-project changes*: Changing APIs that are used in multiple internal projects is more simply in monorepos than in multiple repositories. According to publications #16 and #17, developers can change an API and all its callers in one commit.
- *Easy refactoring*: According to publications #2, #16 and #17, a well-organized unique repository is likely to have modular code and hence refactoring is likely to be easier in monorepos than in multiple repositories. Restructuring is also easier as everything is neatly in one place and easier to understand.
- *Simplified organization*: In monorepos, projects can be organized and grouped to be more logically consistent, as described in publications #2 and #16.
- *Improved overall work culture*: Monorepos encourage the team unification and hence each member can contribute more specifically towards the goals and objectives of the organization, as discussed in publication #2.
- *Better coordination between developers*: Developers run the entire project on their machine, which helps them understand all services and how they work together. As a result, developers tend to find more bugs locally, before sending pull requests, according to publication #2.
- *Tooling*: In monorepos, all code has a fixed path in a single shared hierarchy, which facilitate building tools that operate on multiple projects, as discussed in publications #2, #6, and #16.

3.4. Challenges of monorepos (RQ #4)

This section discuss some challenges and trade-offs of monorepos.

- *Code health*: In monorepos, according to publication #22, it is easier to add dependencies. However, (i) this reduces the incentive for software developers to produce stable and well-defined APIs; (ii) code cleanup is even more error-prone because it is common for teams to do not think about their dependency graph; and (iii) purposeless dependencies increase project exposure to downstream build breakages, leading to binary size bloating, and creating additional work in building and testing.

- *Codebase complexity*: According publication #14, the main challenge of monorepos is to manage all projects in a single repository. Although the understanding organization of the code in monolithic repositories is easy, it is a complex task to determine where new code should be placed. Besides, publication #7 criticizes monorepos since their high codebase complexity does not necessarily increase productivity.

- *Tooling investments*: A huge repository requires managing tools to scale. Publications #1 discusses the high cost of running these tools.

- *Loss of version information*: Publications #8 argues that it is dangerous to lose version information. Basically, since details of imported libraries can be lost, it may be hard to deal with updates.

- *Build, Test Bloat, and Deploy*: Due to its size, publications #3 and #8 question the cost of building and testing on monorepos, whereas publication #7 questions the cost of deploy.

- *Migration*: Publications #10 introduces "the monorepo problem". It refers to the fact that migrating of many repositories to only one has a high cost, because it is necessary to modularize all code. This is too critical that publication #21 reports a migration study case.

4. Threats to Validity

To answer the RQs, we investigated different aspects from monorepos to support generalization of our discussions. This research is mostly based on grey literature, which means that most of the material have not been subject to rigorous peer-review, as academic research usually is. However, (i) the inclusion of the grey literature in our review overcame the scarse works available in the digital databases of scientific literature and (ii) we analyzed the grey literature in a systematic way by following the guidelines proposed by Garousi et al. [4].

5. Conclusion

This study presented a Multivocal Literature Review on monorepos based mostly on grey literature. We investigate (i) how monorepos are defined; (ii) what are the characteristics of monorepos; (iii) what are the benefits to adopt monorepos; and (iv) what are the challenges to adopt monorepos.

Regarding (i), monorepos are usually defined as a single repository that contains multiple related or unrelated projects. In this sense there are two kinds of monorepos: Monstrous monorepos, which contains several unrelated projects and millions of lines of code, and Projects monorepos, which contains related components of a specific project.

Regarding (ii), we can enumerate *centralization* since a single repository encompasses multiple projects, *visibility* since everything is visible by all contributors, *synchronization* since the development process is trunk-based, *completeness* since any project can be built using resources available in the repository, and *standardization* since engineers usually share the same set of tools in monorepos.

Regarding (iii), the main benefits include *simplified dependencies* since library versioning is easy, *simplified organization* since projects are organized in a more consistent way, *easy refactoring* since modular repositories foster modular code, *improved*

overall work culture since monorepos encourage the team unification, *tooling* since a single shared hierarchy facilitate building tools that operate on multiple projects, *better coordination between developers* since developers can easily understand all projects and how they work together, and *better cross-project changes* since an API and its callers can be refactored in a single commit.

Regarding (iv), the main challenges include *codebase complexity* since managing all projects in a single repository is more difficult, *tooling investments* since the cost of running managing tools is large, *code health* since unnecessary dependencies can create additional building and testing work, *loss of version information* since it is dangerous to lose some version information, *build, test, and deploy* since these tasks can take a long time, and *migration* since the cost of migration of many repositories to only one is high.

Several publications compare monorepos with multirepos (#2, #5, #6, #9, #15, and #18). These studies argue that choosing monorepos or multirepo is hard because each model has its own set of principles and practices and its own challenges. Monorepos and multirepos not only have different tooling requirements, but also vary in their engineering culture and philosophy. On the one hand, some companies, such as *Netflix*, value freedom and responsability [6], thus they prefer multirepos. On the other hand, *Google* values consistency and code quality, thus it prefers monorepos.

Ideas for future work include: (i) to conduct surveys with practitioners to expand our understanding on monorepos; (ii) to conduct a study comparing monorepos and multiple repositories models; and (iii) to investigate the adoption of monorepos in open-source projects.

References

[1] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[2] Rain Brian Harrys. The largest git repo on the planet. https://blogs.msdn.microsoft.com/bharry/2017/05/24/the-largest-git-repo-on-the-planet/, 2017. [accessed 15-June-2018].

[3] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In *20th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, page 26, 2016.

[4] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. Guidelines for including the grey literature and conducting multivocal literature reviews in software engineering. *arXiv preprint arXiv:1707.02553*, 2017.

[5] Durham Goode. Scaling mercurial at Facebook. https://code.facebook.com/posts/218678814984400/scaling-mercurial-at-facebook/, 2014. [accessed 15-June-2018].

[6] Netflix. Netflix culture. https://jobs.netflix.com/culture, 2009. [accessed 15-June-2018].

[7] Rodney T. Ogawa and Betty Malen. Towards rigor in reviews of multivocal literatures: Applying the exploratory case study method. *Review of Educational Research*, 61(3):265–286, 1991.

[8] Rachel Potvin and Josh Levenberg. Why Google stores billions of lines of code in a single repository. *Communications of the ACM*, 59(7):78–87, 2016.

[9] StackOverflow. Monorepo. https://stackoverflow.com/tags/monorepo/info, 2017. [accessed 15-June-2018].