AKADÉMIAI KIADÓ

Comparison between mono and multi repository structures

Ulvi Shakikhanli* and Vilmos Bilicki

Doctoral School of Computer Science, Faculty of Science and Informatics, University of Szeged, Szeged, Hungary

Pollack Periodica • An International Journal for Engineering and Information Sciences

17 (2022) 3, 7–12

DOI:
10.1556/606.2022.00526

ORIGINAL RESEARCH PAPER

Received: December 10, 2021 • Revised manuscript received: April 11, 2022 • Accepted: April 22, 2022
Published online: June 24, 2022

ABSTRACT

The comparison of Mono and Multi Repository structures is a highly debated topic in the software development field. Despite the choice of repository structure is the first main step in development; so far, this comparison has only been made on a small or local scale. Here, Mono and Multi Repository structures have been compared from different aspects using thousands of projects.

First, an algorithm shared for collecting and identifying Mono and Multi Repository projects and save them into the database. Database was used for making different comparisons for example the usage intensity of both structure types over time, the developer's preference over structure type based on their country and so on. Also, all these comparisons have been made according to the team size and development period for each repository structure.

KEYWORDS

mono/multi repository, repository structure, repository identification

1. INTRODUCTION

Development area and information technologies are improving every day. It is all too common for developers to start coding an application without a formal architecture in place [1]. In general, this creates some problems for developers and others in this field for example sometimes certain things as the architecture of application can cause some confusion. It is well known that software design and architectural patterns are general and reusable solutions to the problems, like computer hardware performance limitations, high availability and minimization of business risk [2]. For the sake of clarity, it can be stated that the architectural pattern is a solution that is directed to elements of software architecture. However, there are no patterns for managing the architecture of large projects at the repository level. There are several Version Control Software Systems (VCSSs) and all of them have different characteristics. Some of them are free (for example: Git, Fossil, Mercurial) and they support several features that may be crucial during the version control process [3]. Beside this, several repository structures have appeared since 2009 like Monorepo (also called Mono Repository), Multirepo (also known as Multi Repository or Polyrepo); and the latest architecture, which can be understood as a hybrid of the previous two is Metarepo (also called Meta Repository) [4].

Choosing repository structure is the first step for development period and can be accepted as the foundation of whole project. Since each repository structure has its own features this choice is highly crucial.

Based on the research it has become clear that some of the big companies choose different approaches for their own software projects. In some cases the reason for them choosing a certain repository structure depends on the type of culture they prefer. For instance, Netflix uses the Multi Repository architecture because of its culture called "freedom and responsibility" [5]. There are also other big companies that prefer the Mono Repository

*Corresponding author.
E-mail: ulvi@inf.u-szeged.hu

AKJournals

approach like Microsoft and it also can be stated that its repository is one of the biggest one [6].

The precise definition of Mono and Multi Repository projects is hotly debated but simple definitions will be provided by using current blogs and some academic papers written in this area.

The Mono Repository (Monorepo or Single Repository) is a type of source control pattern where all the components and sections of source code are kept in one repository. And the Multi Repository pattern manages all the packages and source code components in several repositories [7].

There are two key differences between this paper and the others. First of all the comparison was made from different perspectives that are not properly covered or sometimes not even mentioned in the literature. The second, the comparisons did not done on a basis of projects of some specific company or group of developers. The database [8] was created by us and it contains projects gathered from all over the world. These projects vary from small one developer start-up projects to big commercial ones and it allows us to make a comparison on a much bigger scale than has never been attempted before.

Several questions prepared to make comparisons more understandable and they will be stated later on.

2. METHODOLOGY

This chapter presents research methodology, the key questions, algorithm and approaches used in the study.

2.1. Research questions

The main aim of this paper is to compare two basic repository structures and verify them according to the different measures. In order to make the paper clearer and more understandable, the following five questions were defined:

- RQ #1. How does the literature compare the Mono/Multi Repository structure?
- RQ #2. What is the usage intensity of the Mono and Multi Repository approaches?
- RQ #3. Is there any relationship between the developer's country and their choice of repo structure?
- RQ #4. What is the connection between the team size and repository structure?
- RQ #5. What is the connection between the development period and repository structure?

2.2. Literature review

Unfortunately, there is a shortage of academic studies that make a comparison between Mono and Multi Repository structures. This is why using the Multivocal Literature Review (MLR) was preferred to get the same basic understanding [9]. Therefore, it means that mostly different blogs were used and posts for the literature review. First, let us see what the literature says about the definition of Mono and Multi Repository systems.

The Mono Repository (Monorepo or Single Repository) is a type of source control pattern where all the components and sections of source code are kept in one repository. In contrast, the Multi Repository pattern manages all packages and source code components in several repositories [7].

2.2.1. Accessibility (visibility). The well-organized hierarchical structure is the main advantage of Mono Repository structure [4]. In this case, it had to be assumed that all the developers have access to all parts of the project (of course this may not so depending on the company policy). The Survey conducted among Google developers can shed light on this case [10]. It told us that most developers greatly value the visibility and access possibilities in different parts of the code, but this is also possible in a Multi Repository approach. The main reason for this is that developers can check the effect and workflow of different components in the repository and gain an overall understanding of the project this way. However, as it was shown in the survey, all of these benefits may become drawbacks in some cases and create problems in the Multi Repository architecture. One of these might be dependency problems. In big companies like Google, the size of Mono Repository projects may be enormous (e.g., 2 billion lines of code) and changing one dependency in a project can cause huge difficulties like the diamond dependency.

2.2.2. Testing for security and functionality. This may be one of the best advantages of the Multi Repository structure. Independent repositories allow isolated testing for module security and functionality [11]. Nevertheless, with this type of flexibility it also introduces complexity during version control and this usually creates some drawbacks.

2.3. Identifying and collecting Mono Repository projects

Collecting Mono Repository projects is not as easy as the Multi Repository one, so that is why it is going to be discussed first. First, it had to be mentioned that the Github platform was used as source of finding project. Github is a free version control system, which contains millions of projects so it is something like a gold mine for this type of software research. Before dive in the steps of algorithm it had be mentioned that in order to be able to compare the collected results and check accuracy of algorithm. Several Mono and Multi Repository projects have been collected from different resources to be used as examples. Those projects were chosen according to the definition of structure types.

The whole procedure of identifying and collecting Mono Repository projects can be separated into three parts:

1. Get a list of possible users;
2. Check each repository of user;
3. Add newly founded projects to the database.

2.3.1. Get a list of possible users. Github Application Programming Interface (API) can provide us with JavaScript

Object Notation (JSON) documents. Those documents contain several parameters of repositories. As it was mentioned previously, Github has millions of users and it is not possible just try to get any list of users. In this case, it is needed to find those users that may have mono or multi repository projects. There are also some additional steps for user selection. Users are excluded from the list if it has no repositories or repositories don't belong any of the structures.

2.3.2. Check each repository of a user. This section is about how to identify Mono Repository projects in Github user accounts.

In order to be able to understand this part the definition of a mono repository project had to be remembered. Earlier it was said that a Mono Repository project should contain all parts of the project in one folder or repository.

An algorithm had to be constructed to define a repository that is either a mono or not. The file structure of the repository was used for this. The file structure is a list of names of folders and files in a repository. These are folder names and they may be listed like "Client, Server, UI, Front, Back, API, Frontend, and Backend".

As it might be expected, these are names of folders, which contain components of either the frontend or the backend parts of the project.

The steps of algorithm are:

- Step 1: Find the repository of the user;
- Step 2: Collect the name of folders of the repository;
- Step 3: Compare the collected folder names with the previously defined list;
- Step 4: Add the project to the database if there are any matches.

Here it should be mentioned that Step 3 is the most important part and it can be carried out in several ways depending on the programming language and framework.

2.3.3. Add the newly founded project to the database. This is the last step. Unique database has been created for this research. There are several features of the repository that can be added to the database depending on the requirement. In addition, of course all these features can be obtained by the Github API if the name of the repository and the user is known.

2.4. Identifying and collecting Multi Repository projects

It had already been remarked that the collection and identification process of the mono and multi repository projects is quite similar. Steps below show the shows the flow of procedure.

1. Get a list of possible users;
2. Collect properties of each repository;
3. Match the repositories;
4. Add newly founded projects to the database.

The first step is almost the same so there is no need to explain them again. Let us now go directly to the third step.

2.4.1. Collect properties of each repository. As it had already been said, one can get all the properties of repository using the Github API. In this stage, a simple database will be created for users and add all the repositories with their main properties to this database. *Main properties* can be listed as follow: *name*, *created and finished date*, *database used* and so on.

2.4.2. Match the repositories. This stage is probably one of the most difficult and time-consuming part of this study. Therefore, after collecting all the repositories of a particular user, there are two main questions:

- Q1: How frontend and backend repositories can be identified and grouped?
- Q2: How to define which frontend and backend repositories belong to the same project?

The Machine Learning (ML) algorithm based on the file structure of the repository provides the answer for the first question. ML models trained based on thousands of repositories and it can define the type of repository with almost 90% accuracy.

The repository sets which are demonstrated here can be seen as rough sets [12]. So the answer of the second question lies with classification. After the first stage, repositories divided into two groups, the frontend and backend repositories. Repositories in those two groups have been classified using K Nearest Neighbor (KNN) method based on the name of the repository and content of the readme file. After lots of tests and measurements, an accuracy score of nearly 90% acquired. For a better understanding of this stage, tests can be presented as follows:

The database contains several features of the repository ("*Name of repository*", "*Created date*", etc.). The main idea was to apply these features and make a classification based on them. Since variations can be endless, it was extremely hard to find the right features. In order to do this, several approaches have been tried out and most of them abandoned. For reasons of space, results of those tests are presented in Table 1.

Table 2 shows us the list of abbreviations while Table 1 explaining results of tests. As it can be seen in Table 1, all the important features were utilized in the database and the success rate was reasonably low. In the second case, "*Readme*" was removed, the result also fell with it, and of

Table 1. The success rate of different feature combinations

Used features	Success rate
RN + L + F + DT + D + FS + R	36%
RN + L + F +DT + D + FS	32%
RN + L + F + DT + D +R	48%
RN + L + F + D + R	48%
RN + L + F + R	52%
RN + L + R	56%
RN + R	89%
RN + L	28%
RN + L + F	24%

Table 2. List of abbreviations

Name of feature	Abbreviation of it
Repo Name	RN
Language	L
Framework	F
Database type	DT
Developers	D
File structure	FS
Readme	R

course, it increased again when it included it. As it might be expected, the number of combinations may be several hundred. This is why there is no need to include all of them here. The highest score was obtained when *"Repository Name"* and *"Readme"* were used together. In this test, it was nearly 89%, but depending on the test subject, it may plus or minus five percent. The utility of the algorithm lies not only in its accuracy, but also with it being most probably the only algorithm and this may be one of the main steps for matching front and backend repositories.

3. A COMPARISON OF THE MONO AND MULTI REPOSITORY APPROACHES

This section answers to the questions RQ #2 - RQ #5 from different aspects. In order to be able to get a better insight specific group of projects that are called "Big Projects" were used. For the sake of simplicity, these projects have been classified in a *3-month development period* and used *10 Mbytes of memory.*

3.1. An intensity comparison of the repository structure

Now, the usage level of the repository structure was described over time from 2015 to 2021 and address RQ #2. The whole database was utilized (including Big Projects) to get better results. As it can be seen in Fig. 1, the multi repository approach became popular from 2017.

The Multi Repository approach practically did not exist in 2015 and it started to become popular and rival the Mono Repository approach in 2018. Surprisingly, the Multi Repository approach got a huge boost in 2019 and afterwards the Mono Repository approach was never quite able to catch up. Therefore, it means the Multi Repository approach is much more popular than the Mono Repository one, but this does not mean that there is no need to compare both approaches.

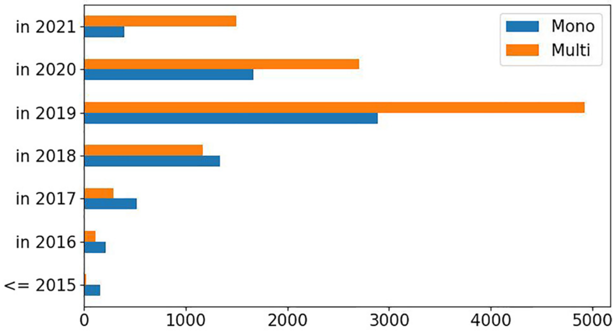

Fig. 1. The usage level of repository structure by years

3.1.1. A comparison of the developer countries and the structure type. Firstly, Fig. 2 provides some rather interesting results. It quickly gives us the answer to RQ #3. As it can be seen, the Multi Repository approach is quite popular in almost all the major countries but there are notable exceptions. For example, while the number of developers from China is not only much bigger than the others, they also have the highest percentage of Mono Repository choices. Here, it can be observed that the choice of repository structure is not just based on company policy but also on the "programming culture" of the given country.

3.1.2. A comparison of the team size in Mono/Multi Repository structures. The aim of this section is to answer RQ #4. Before examining this comparison again there are a few things, which need to be mentioned. In order to be able to get good results the projects with fewer than five developers were eliminated because this type of projects are mostly for personal use and they do not have any real commercial value.

The projects with a team size of between 5 and 10 developers were compared in Fig. 3. Here, the results indicate that the Multi Repository approach has the upper hand. Despite this, the share of Multi Repository projects starts to decrease when the team size increases. Hence, it can be said

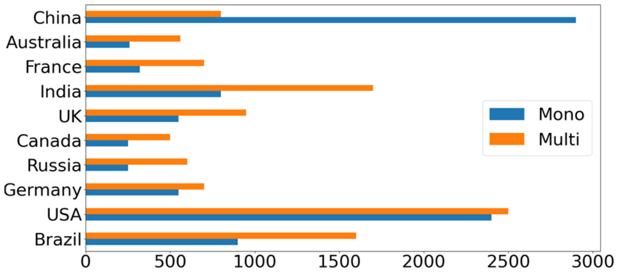

Fig. 2. The countries of developers for each structure type

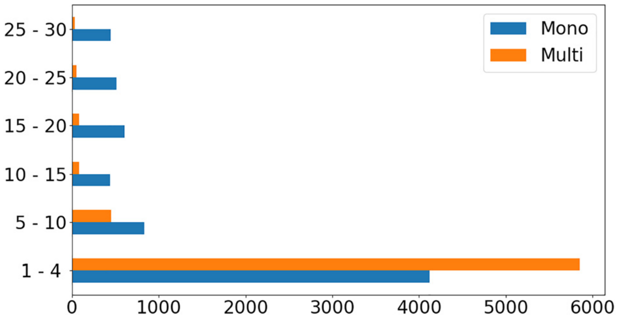

Fig. 3. A comparison of the team size

that developer teams with a bigger size prefer to work on the Mono Repository structure rather than the Multi one.

3.1.3. A comparison of the development period. With the last comparison, RQ #5 was answered. Figure 4 tells us the percentage of development period for each approach based on the whole database. For example, in Fig. 4a it can be observed that almost 64% of all the *Mono Repository* projects are completed in under a month and this figure is nearly 66% for *Multi Repository* cases. Moreover, figures for other cases like the development period between 3 and 6 months and over 12 months were practically the same Fig. 4b. It is again worth to remind that these pie charts were created based on the whole database.

In Fig. 5 the *Big Projects* were examined, which have been mentioned earlier took up 10 Mb of memory and the development period was over 3 months. In the previous case, the whole database was examined and it appeared that the percentage values were closer to each other and now surprisingly similar values can be observed here as well.

Naturally, there are some differences but they are no more than 3% and can be ignored. Based on the results, it can be hypothesized that the repository structure of the project does not significantly affect the development period. Of course, while more comparisons are required to test this statement, it does show that some interesting results could be acquired by examining real projects carried out by different developer teams.

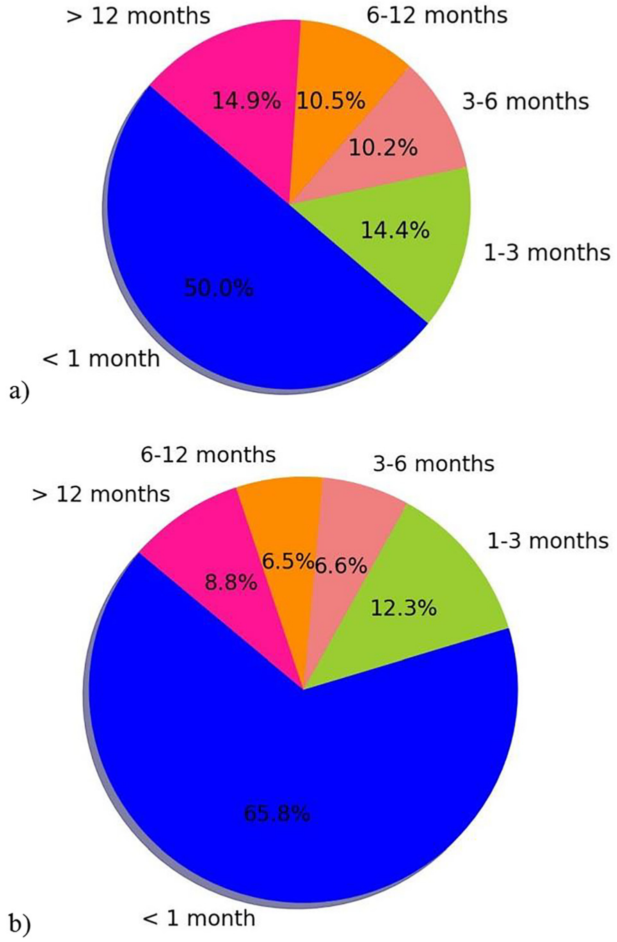

Fig. 4. The development period in terms of two approaches used, a) Mono Repository, b) Multi Repository

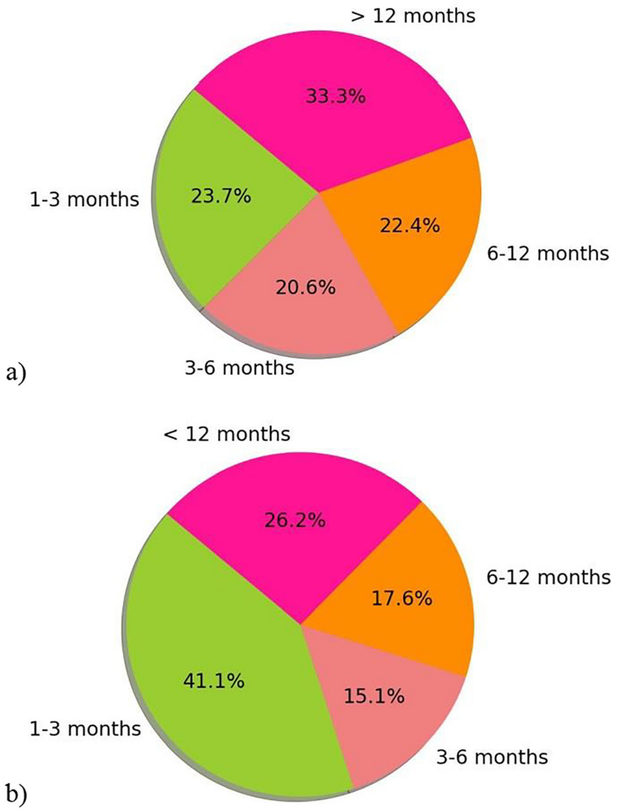

Fig. 5. The development period comparison of two approaches (Big Projects), a) Mono Repository, b) Multi Repository

4. CONCLUSION

In this study, followings were presented

a) A comparison of two repository structure types from quite different perspectives using databases containing different types of projects;
b) A unique algorithm for identifying and collecting mono and multi repository projects from Github;
c) Databases were created with the help of this method and they can be used by other researchers for different purposes.

Regarding a), most of the comparisons between Mono and Multi Repository approaches were made using rather small or local scales. Related papers, which were found in the literature, focused only projects of one company or a specific group of developers. Here, new approach implemented to this comparison by using a new database that contains projects taken from different backgrounds. This way the resulting comparisons were much more general and produced some things that were never taken into account in similar studies. According to the results it can be said that Multi Repository projects takes less time than Mono Repository ones and additionally the team sizes in Mono Repository projects are much bigger than Multi one. Beside these choice of structure compared according to the regions of developers and that can be useful for project planners when they want to take regions of developers in account.

Regarding b), the procedure of identifying a Mono Repository project is based on its file structure. As it was mentioned, in the definition all, the components of the project are kept in one repository and obviously, they were inside specific folders. They can easily be defined by examining the file structure. The identification of multi repository projects was based on clustering algorithms. In this algorithm, the name and the content of the readme file of the repository were used. Later only needed to use Github API commands to collect the specific features of projects and repositories present in the database.

Regarding c), database was created that can be used for different purposes and in this paper; the difference between two repository structures was tried to ascertain. At first, the degree of usage of the repository structure from 2015 to 2021 was checked. Secondly, the effects of structures were compared based on aspects like *development period*, *number of developers in the team* and *regions of developers*.

There are several ways two repository structures can be composed but almost all of them are based on views of developers and some statistics prepared for certain companies (like Google and Facebook). In this study, not only a way was proposed for identifying and collecting mono and multi repository projects, but also created a unique database to go with it. . Other researchers can do additional comparisons according to the database.

Plans for future work include the following. Now, the algorithm does not work on specific types of projects and it needs to be extended it to get results that are more objective. In addition, there is plans for creating metrics for each repository structure and after that, it would be possible to get a clearer view of how the two different repository structures can affect the development period

REFERENCES

[1] Comparison of version-control software, Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_version-control_software. Accessed: Dec. 6, 2021.

[2] G. Kokrehel and V. Bilicki, “The impact of the software architecture on the developer productivity,” *Pollack Period.*, vol. 17, no. 1, pp. 7–11, 2022.

[3] Architectural pattern, Wikipedia, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Architectural_pattern. Accessed: Nov. 25, 2021.

[4] B. Libbey, Monorepo, Manyrepo, Metarepo, 2019. [Online]. Available: https://notes.burke.libbey.me/metarepo/. Accessed: Nov. 1, 2021.

[5] R. Hastings, Netflix culture: Freedom & responsibility, Reed Hastings. Slideshare 2019. [Online] Available: https://www.slideshare.net/reed2001/culture-1798664/2-Netflix_CultureFreedom_Responsibility2. Accessed: Nov. 15, 2021.

[6] H. Brian, Scaling Git (and some back story), Microsoft Devblocks, 2017. [Online]. Available: https://devblogs.microsoft.com/bharry/scaling-git-and-some-back-story/. Accessed: Nov. 15, 2021.

[7] P. L. Scott, Mono-repo or multi-repo? Why choose one, when you can have both? Medium.com, 2017. [Online]. Available: https://patrickleet.medium.com/mono-repo-or-multi-repo-why-choose-one-when-you-can-have-both-e9c77bd0c668. Accessed: Oct. 4, 2021.

[8] Multi/Mono Repository Database, Github 2021. [Online]. Available: https://github.com/Shakikhanli/Mono-Multi-Repository-Database. Accessed: Dec. 10, 2021.

[9] R. T. Ogawa and B. Malen, “Towards rigor in reviews of multivocal literatures: Applying the exploratory case study method,” *Rev. Educ. Res.*, vol. 61, no. 3, pp. 265–286, 1991.

[10] C. Jaspan, M. Jorde, A. Knight, C. Sadowski, E. K. Smith, C. Winter, and E. Murphy-Hill, “Advantages and disadvantages of a Monolithic Repository: A case study at Google,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, Gothenburg, Sweden, May 27–June 3, 2018, pp. 225–234.

[11] T. Holmes, Terraform mono repo vs. multi repo: The great debate, Hashi Corp. [Online]. Available: https://www.hashicorp.com/blog/terraform-mono-repo-vs-multi-repo-the-great-debate. Accessed: Jan. 28, 2021.

[12] D. Nagy, T. Mihálydeák, and L. Aszalós, “Graph approximation on similarity based rough sets,” *Pollack Period.*, vol. 15, no. 2, pp. 25–36, 2020.