INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Component Reusability in Extreme Programming Using Microservice Architecture

Nagalambika[1]*, L Manjunath Rao[2]

1 Research Scholar, Department of MCA, Dr. Ambedkar Institute of Technology, Bangalore - 560056, Karnataka, India
2 Professor, Department of MCA, Dr. Ambedkar Institute of Technology, Bangalore, Karnataka - 560056, India

OPEN ACCESS

Received: 29-01-2022

Accepted: 01-09-2022

Published: 22-09-2022

Citation: Nagalambika , Rao LM (2022) Component Reusability in Extreme Programming Using Microservice Architecture . Indian Journal of Science and Technology 15(36): 1808-1814. https://doi.org/10.17485/IJST/v15i36.248

* **Corresponding author**.

nagalambika.swamy@gmail.com

Funding: None

Competing Interests: None

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Background/Objectives: In a distributed development environment, the different teams share the code leading to dependencies and shadowing the purpose of microservices. This study is to propose the technique of component reusability in Online Order Management to measure the reusability in terms of lib reuse, product customization and vendor services. **Methods:** The case study on online order management software was analysed, developed and tested. The mean and standard deviation are calculated to what extent OOM projects can be reused in other application domains like audio, business, communication, games, software development, library management, home utilities and education. The source of our descriptive analysis is the Kruskal-Wallis test. Reusability is calculated by three markers: lib-reuse, Reuse-frequency and throughput. The proposed technique is efficient in terms of throughput. **Findings**: The presented case study is a practical application that illustrates the percentage of reusability of the proposed model. This will not only minimize engineering efforts but also reduces resource costs. Reusability is directly proportional to resource cost. More the reuse percentage lesser the resource cost. The results have shown the reusability percentage for each domain. Projects are selected based on popularity from GitHub and BitBucket. **Novelty:** The novelty of the proposed approach lies in the fact that the specific application domains can be considered as reusing assets from open-source software projects. The proposed approach increases the autonomy between the teams and also helps teams to operate with minimal dependencies. The software professionals can benefit from the proposed methods.

Keywords: Reusability; Microservices; Extreme Programming

1 Introduction

The use of microservices to build an efficient software system, that addresses many modern-day application challenges. Reusability is widely adopted today[1]. Before microservices, monolith architecture was part of the play. The shortcoming of this type was evident in the case of a large and complex application. Understanding and modification are not easy when the size of the application is on the growing side. Also, it is difficult to understand the change as the modularity is affected due to the large codebase. As mentioned in [2], the transition is incremental. But there is the possibility that to update one component, the entire background task is disturbed as the entire application is required to be redeployed. Other problems include inter-module dependencies and scalability issues. It is a self-contained process with unique and different business capabilities. This type of system is inevitable for large systems. Using microservices architecture it is easy to select the features that are best suited for the required functionality (service). It is all about offering customised products and services as needed, which is essentially the same as microservices. It emphasises forming a collaborative development are a team in which all team members communicate with one another and with their clients regularly. Every computer has a physical limit of load to be managed and similarly, there is a limit on the number of functions to be specified, maintained and tested. Functions that exceed such restrictions will be separated into distinct processes. The database tier will be segregated from the application tier, and front-end proxy servers and load balancers will be utilised to coordinate horizontal scalability to numerous application servers, among other things. Service architecture is required for big, complicated, and scalable systems [2].

Several survey-based studies have been conducted so far. Authors[3], have conducted a study using poor documentation as a goal. Along with this inventory of various tools has been analysed. Although quantity and quality of documentation are discussed this study has not touched on the executable documentation area. A real implementation on the taxation department. Not only code but also the performance of the system is improved using this approach. The authors have touched on another different area, i.e. automated acceptance test. In the actual implementation, this approach has helped in the development of an automated acceptance test with a reuse methodology. The development process is agile and requires, testing and development to be tested iteratively. Authors have also proposed two performance measures called precision and recall, but these are not tested yet. Although the line of implementation is different from ours, they have presented a structured review on the security of microservices[4,5].

The research on reusability is the need for present and future as supported by[6]. Descriptive statistics, open coding of qualitative information, and odds ratio have been used to analyze the data, but We found multiple deviations in criteria and methods used for decision making. Hence, this points to a gap between what is done in industry and what is proposed by academia, at least for the cases studied. Thus, more collaborative studies with industry participation are needed. Microservices support agile architecture but if the development isn't properly managed it could affect the speed of the process, reducing productivity and increasing technical debt. This paper investigates a way to manage the employment of shared libraries in microservices to enhance agility throughout development. As an alternative to the use of shared libraries, simple functionalities should be implemented by each microservice, whereas complex functionalities should be implemented by external microservices with well-defined interfaces, good documentation and adequate versioning policies. The external dependencies have not been taken into account which on being included can affect the results [7].

Authors have investigated the potential methods that can bring improvements in software quality for a small-scale software firm. It recommends suggestions to use software craftsmanship to redesign its existing products. An extreme programming approach is implemented for software development to support craftmanship. This approach has not included forward engineering [8].

Authors in [9] have given a detailed comparison of extreme programming and scrum but this survey has not included the distribution for the comparison. To enhance the software development process, the Extreme programming framework is selected over other frameworks. The code developers write testing code before production in test-driven development benefitting by avoiding unnecessary details and complexity [10]. Extreme programming recommends automating testing and integration to make the development process well-organized. TDD gives instant feedback to developers. TDD can be applied to integration tests with mock objects as well as unit tests. This study also lags in implementing the microservices in distributed agile systems [11].

Automatically analysing the use of reusable libraries that are present on the internet and understanding the scope and nature of reuse in practice is challenging. It took a lot of time and effort to manually identify useful and relevant libraries. Worse still, developers may not even be aware of non-popular but reusable libraries. It is mentioned that developers prefer to implement features from scratch rather than reusing third-party features. The use of reusability not only will help increment in the performance but also considering it on the distributed level will help reduce conflicts between the teams [12].

The multiple case study on 596 Java OSS projects is implemented to evaluate their reusability. The research findings of the study infer that the most reuse-friendly components are from the domains of Science and Engineering Applications and Software Development Tools. Such results can act as a guide for selecting appropriate projects for studying or exploiting reuse opportunities [13].

This work will help practitioners to avoid the bad practices altogether or otherwise deal with them more efficiently. We have not only explored the association and interlink between the Extreme Programming framework but also proposed a functional model of reuse. Extreme programming is renowned agile practice. In the development model, iterations are used for development and system requirements are recorded by the customer using user stories. In the planning stage, the developers along with the customer determine the required functionality be developed. This paper attempts to clarify the definitions and differences between Service-oriented architecture and Micro Services by pointing out their pros and cons. The common between both is system integration, but the industry is more aligned toward microservices due to their elastic scalability and independent deployability (14).

A survey on recent advances in microservices focuses on the evolutionary aspects for better documentation understanding of microservices, how these originated and what is their possible future (15). The move towards microservices is an important matter as major companies are either shifting their back-end systems or developing their business model following the microservice paradigm. We will witness a big change in the view in which software is envisioned, and in the technique in which abilities are planned into components. The other advantageous highlights of microservices are scalability, fault tolerance and availability. The gaps shown by the background study are formulated as our research questions.

[**RQ1**] What is the extent to which the OOM project can be reused?

[**RQ2**] What are the differences in the availability of reusable sets of classes extracted from OOM projects?

Answering the aforementioned three research questions, the paper's organization is as follows: Section 1.1 describes the Framework of the Reusable Component Model. Section 1.2 presents the development approach. Sections 2 and 3 include case study and model reuse across different platforms respectively followed by the conclusion.

1.1 The framework of the Reusable Component Model

Because of the trend toward the adoption of microservices, software architecture is witnessing huge change. One of the preferred approaches for capitalising on the popularity of microservices architecture in cloud computing is to transform the architecture from a monolithic to microservices architecture (16). Applications in a monolithic architecture are written in the same source code whereas microservices architecture deploys applications on different source code and other machines. Monolithic architectures are less scalable and difficult to maintain. The conversion of monolithic software architecture to a microservice architecture necessitates a specific strategy and technique (17,18). According to research on legacy software modernization, there are three strategies for transitioning from monolithic to microservice architecture. There are three types: top-down, bottom-up, and hybrid. Bottom-up and hybrid strategies both necessitate legacy software and software reengineering (8). Our research employs a hybrid strategy because it is not only based on research papers but also on the legacy software related to online order management.

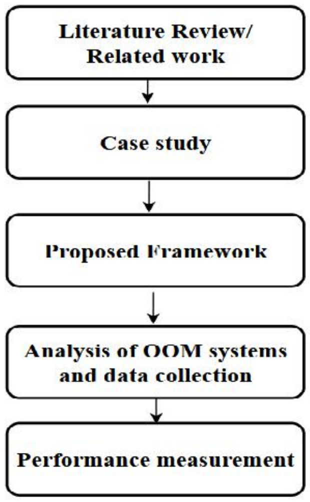
Literature Review/ Related work
Case study
Proposed Framework
Analysis of OOM systems and data collection
Performance measurement

Fig 1. The steps of the Research Methodology

Figure 1, shows the steps in our designed research methodology. The framework proposed is based on both the research paper and OOM software. Micro-services are built for combining small parts of a system to produce a combined system. Different developers work on different microservices. The application and back work independently using a message bus or HTTP API. In our approach, we have implemented HTTP APIs. The separation to make microservices independent has helped in independent authority on the present and future of service.

For large companies that are reliant on technology stack and architecture., there exist similarities in projects like user management and messaging. Instead of doing this in a non-profit way, one of the viable solutions is to fix the problem using microservices. This not only reduces the labour of developers but also saves time by simply cloning a code with the addition of project-specific logic, and a culture of work-sharing by giving focus on innovation. The best-fit approach for such scenarios is to build the system in a modular manner. Common infrastructures such as Naming service, API gateway, Security and Logging offer microservices. Each module is responsible for its business logic. If a new microservice is required to be added. The functionality of the common component can be reused. Each microservice has an associated library service. Library X package repository contains completed reusable libraries. The team member can view, implement and download.

1.2 Development Approach

Now at this phase, we require to keep a copy of reusable code. The biggest issue here is each microservice requires a tool to keep track and synchronize the changes. This issue can be resolved if the components are organized as collection and synchronisation are automatic between them. This approach is implemented by Github. The one advantage of this approach is that teams can focus only on the components required by them as shown in Figure 2.

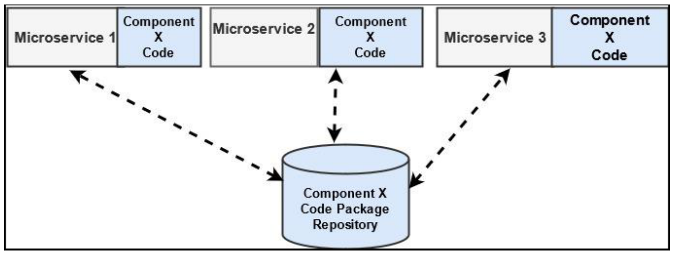

Fig 2. Development approach using Microservices

The reported literature has shown the adaption of extreme programming in large-scale projects. Model reuse seems to be a topic of noteworthy interest in application areas, especially in an online order management system. In this section, we take an online order management system for testing and verifying our approach, along with the design of a case study for exploring the reusability of online order management (OOM) projects. In particular, we examined 8, OOM projects. The key purpose that we performed a case study rather than another type of empirical evaluation is that we wanted to investigate the reuse opportunities offered by real OOM projects. In the multichannel order service in OOM, each service has its database. Order services are called in a dashboard as well as notification services showing the reusability. The required product customizations and vendor support terms are reusability services.

In this paper, we have chosen two software reusability measures called library reuse and reuse frequency. The formula to derive the percentage of reuse is derived by calculating the ratio of the sum of the number of same lines present in the program and the total number of lines present in the program. Most of the software can be developed by considering existing products. Technically it is the exact definition of reusability. Not only the lines of code but we also need to calculate the reusability of the class. A mathematical expression can be written as:

$LOC = (S^{LOC}, A^{LOC})$ (1)

$G^{LOC} = (\{range^{LOC}\}, \{\beta, range^{LOC}\}$ (2)

$R^{LOC} = \beta,$

Mapping $\delta loc = Class \rightarrow range^{LOC}$

$\delta loc\ (Method) \rightarrow range^{LOC}$ (3)

2 Result

This case study's context is the development of OOM. For the proposed framework, we have used a meta-repository to collect data from various projects hosted on BitBucket and GitHub. The unit identification and analysis are performed automatically after dumping the repository's entire database. Data collection is also automatic for each unit of analysis, in some cases by automatic parsing the database is performed, while in other cases manual parsing is required. The criteria that we have used for selecting projects and populating the repository are discussed below:

• Projects should be written in Java and Junit and ranked by popularity in BitBucket and GitHub. The top 8 based on popularity are selected and the first two that met our criteria are filtered.

We have used the following technologies to achieve reusability:

• Application server: Tomcat 8.5.
• Framework: Spring boot.
• Programming language: Open-source Java and Junit.
• IDE: IntelliJ IDE.
• Tools: Maven building tools.
• Code Repository: Bit bucket and Git.
• User services: Running on port 9001 with Url: http://localhost:9001/user?id=10

3 Discussion

To measure the efficiency of the proposed methodology, we have used two factors of reuse called library reuse and reuse frequency. Library reuse is defined as, under the given domain how many external libraries can be used, while the reuse frequency indicates the number of reuse of system files. Table 1, presents the results of our first research question i.e. to which extent OOM projects can be reused in other systems. Reusability is quantified by three indicators: lib-reuse, reuse amount, and reuse frequency. The application domains are audio, business, communication, games, software development, library management, home utilities and education. The source of our descriptive analysis is Kruskal- Wallis test.

Table 1. Software Reuse in different domains

Domain	Lib-reuse		Reuse frequency		Average throughput	New	Reused	Reuse percentage
	Mean	Std.Dev	Mean	Std.Dev	Per second			
Audio	0.029	0.03	0.008	0.03	50.67	7	25	28
Business	0.032	0.02	0.004	0.02	50.88	13	40	3.25
Communication	**0.041**	**0.08**	**0.019**	**0.04**	**50.33**	**33**	**45**	**73**
Games	0.023	**0.03**	0.023	0.04	50.22	15	70	21
Software Development	0.034	0.02	0.011	0.02	45.99	23	34	67
Library Management	0.039	0.01	0.023	0.02	45.07	12	32	37
Home Utilities	0.023	0.02	0.022	0.01	50.34	34	13	2.6
Education	0.011	0.01	0.023	0.02	50.55	34	45	75

The table shows the lib-reuse, reuse frequency and average throughput in different domains. Our approach can be compared to[19] in terms of throughput. The proposed approach has excelled in terms of throughput. The highlighted values show the difference in factors of reusability in terms of Mean and Standard Deviation. Reusability is calculated based on the use reuse of existing classes and new classes used. In the case of the education domain, 34 classes are new while 45 are reused, making the percentage 75. While gathering data on this basis few factors like security, login modules, notifications, cancellations of orders in bulk and updates. These all factors make around more than 100 lines of classes, segregated into separate spaces to make a library this separate library is used across other applications as well. This vaguely makes the reusable percentage around 30 per cent overall. Table 2, shows an evaluation of the documentation in OOM projects accessed using indicators – amount and completeness.

The table shows that the availability of reuse in the OOM model is high in Business and Library management. While documentation is high in the case of communication. Of all the domains considered, the education domain has low availability, mean and standard deviation. This could be another factor for research elaboration. Our results can provide useful insights and feasibility of reuse opportunities to both researchers and practitioners. The results also provide indications of software reusability in different application domains.

On the other hand, the results of this study provide guidance on case selection for researchers, selecting the appropriate application. From all the factors mentioned, only documentation is subjectively evaluated to check the replicability and common understanding of the OOM projects. In this way, the proposed method has not only overcome the weakness of Extreme programming i.e. weak documentation but also clearly defined unambiguous criteria for evaluation of OOM projects.

Table 2. Software Availability and reusability

Domain	**Availability**		**Documentation**				
	Mean	**Standard Deviation**	**Very low**	**Low**	**Moderate**	**High**	**Very High**
Audio	0.012	0.03	15	40	28	45	10
Business	0.024	0.03	23	25	30	23	30
Communication	0.023	0.02	15	23	12	50	40
Games	0.019	0.03	29	40	0	22	29
Software Development	0.015	0.02	30	33	18	34	39
Library Management	0.039	0.04	35	38	0	12	20
Home Utilities	0.023	0.02	40	39	11	24	11
Education	0.011	0.01	30	40	0	40	20

4 Conclusion

To reduce the development time, reusability in microservices is preferred. Formal documentation is difficult to understand and does not contribute much to the end product. In this study, we have worked on reusability percentage. Though Cost is dependent on the percentage of reusability it is trivial because it depends on several factors. Currently, our approach is based on reusability percentage, availability and documentation only. With an online order management model, we put our reusable component model development strategy to the test. The novelty of the approach lies in its enhanced throughput. In several initiatives, our method has proven to be effective for the development chain. The presented model is reusable, as demonstrated by the case study and this practical application. Researchers, for example, can simply identify application domains that need applying methods and tools that improve reusability. Additionally, in future, we will work on making a framework that could be streamlined.

References

1) Lo SK, Liew CS, Tey KS, Mekhilef S. An Interoperable Component-Based Architecture for Data-Driven IoT System. *Sensors*. 2019;19(20):4354–4354. Available from: https://doi.org/10.3390/s19204354.
2) S A. The Economics of Microservices. *IEEE Cloud Computing*. 2016;3(5):16–20. Available from: https://doi.org/10.1109/MCC.2016.109.
3) Theo T, Uwe VH, Paris A. A mapping study on documentation in Continuous Software Development. *Information and Software Technology*. 2022;142:106733–106733. Available from: https://doi.org/10.1016/j.infsof.2021.106733.
4) Berardi D, Giallorenzo S, Mauro J, Melis A, Montesi F, Prandini M. Microservice security: a systematic literature review. *PeerJ Computer Science*;7:e779–e779. Available from: https://doi.org/10.7717/peerj-cs.779.
5) Papoutsoglou EA, Faria D, Arend D, Arnaud E, Athanasiadis IN, Chaves I, et al. Enabling reusability of plant phenomic datasets with MIAPPE 1.1. *New Phytologist*. 2020;227(1):260–273. Available from: https://doi.org/10.1111/nph.16544.
6) Kai P, Deepika B, Syed M, Krzysztof W, Tony G, Efi P. Choosing Component Origins for Software Intensive Systems: In-House, COTS, OSS or Outsourcing? - A Case Survey. *IEEE Transactions on Software Engineering*. 2018;44(3):237–261. Available from: https://doi.org/10.1109/TSE.2017.2677909.
7) Heshmatisafa S, Seppänen M. API Utilization and Monetization in Finnish Industries BT - Agile Processes in Software Engineering and Extreme Programming - Workshops. 2020;p. 23–31. Available from: https://doi.org/10.1007/978-3-030-58858-8_3.
8) Ahmadi A, Budiardjo EK, Mahatma K. Software Craftsmanship Skill using Extreme Programming for Quality Improvement: A Case of Very Small Software Organization. *2021 10th International Conference on Software and Computer Applications*. 2021;p. 94–99.
9) Faiza A, Shabib A, Syed SM, Usman W. Comparative analysis of two popular agile process models: extreme programming and scrum. *International Journal of Computer Science and Telecommunications*. 2017;8(2):1–7. Available from: www.ijcst.org/Volume8/Issue2/p1_8_2.pdf.
10) Kholid H. The extreme programming approach for financial management system on local government. *International Conference on Science and Technology*. 2015;p. 29–34. Available from: https://doi.org/10.1109/TICST.2015.7369335.
11) Kent B. Test-Driven Development By Example. 2002.
12) Mohamed AS, Ali O, Houari AS, Raula GK, Katsuro I, Inoue, et al. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software*. 2018;145:164–179. Available from: https://doi.org/10.1016/j.jss.2018.08.032.
13) Maria E, Stamatia AA, Alexander B, Ioannis C, S. Reusability of open source software across domains: A case study. *Journal of Systems and Software*. 2017;134:211–227. Available from: https://doi.org/10.1016/j.jss.2017.09.009.
14) Tomas C, Michael JD, Michal T. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*. 2018;17(4):29–45. Available from: https://doi.org/10.1145/3183628.3183631.
15) Nicola D, Ivan L, Stephan TL, Manuel M, Ruslan M, Microservices LS. How to make your application scale. *Ershov Informatics Conference*. 2017. Available from: https://doi.org/10.1007/978-3-319-74313-4_8.
16) Lorenzo DL. From monolithic architecture to microservices architecture. *IEEE International Symposium on Software Reliability Engineering Workshops*. 2019;p. 93–96. Available from: https://doi.org/10.1109/ISSREW.2019.00050.
17) Nagalambika S, R KSM, Praveen. Component Based Software Architecture Refinement and Refactoring Method into Extreme Programming. *International Journal of Advanced Research in Computer and Communication Engineering*. 2016;5(12):398–401. Available from: https://doi.org/10.17148/ijarcce.2016.51291.

18) Nagalambika L, Manjunath R. A Study on Development of Software Applications Using Extreme Programming and Devops. *Inspira-Journal of Commerce*. 2021;07(4):27–30. Available from: https://www.inspirajournals.com/issue/downloadfile/2/Volumne-Pages/lPZxNrYwLBBmg1pWVaLG.
19) Andre DC, Ronaldo D, Frank IS, S. An Architecture to Automate Performance Tests on Microservices. *18th International Conference on Information Integration and Web-based Applications and Services*. 2016;p. 422–429. Available from: https://doiorg/10.1145/3011141.3011179.