International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Leveraging Open-Source Reuse: Implications for Software Maintenance

Raghavendra Sridhar

Independent Researcher
Email: *princeraj01[at]gmail.com*

Abstract: ***The increasing popularity of Open Source Software (OSS) has positioned its use in maintenance activities as a significant area of interest. However, dedicated research into reuse-based maintenance strategies specifically for OSS remains limited. To address this, our work began by identifying key attributes and metrics of OSS maintenance processes through a comprehensive literature study. Building on these findings, we have proposed a framework model designed to facilitate the reuse of OSS applications in software maintenance, addressing the current lack of well-specified guidelines for such practices. Our framework aims to streamline the process, effectively bridging the gap between receiving a change request and implementing a reuse-based maintenance solution leveraging OSS.***

Keywords: Open source software (OSS), software maintenance, maintenance process models, reuse-based maintenance, OSS customization, component reuse, software evolution, modularity analysis, change management, software quality

1. Introduction

Since its emergence in 1998, Open Source Software (OSS) has grown in popularity and is now widely used across companies, communities, homes, and governments. OSS is developed through a collaborative model, allowing numerous developers and users to contribute to its creation and distribution via trusted online repositories. Under OSS license agreements, users are granted the freedom to run, copy, modify, and distribute the software, including making improvements to local copies [1]. Today, OSS plays a critical role in nearly every area of computing technology—serving as the foundation for mobile applications and devices, next-generation databases, cloud computing platforms, software-as-a-service (SaaS), and the broader internet infrastructure [2], [3]. As copyrighted software approved by the Open Source Initiative (OSI), OSS licenses permit integration with other software components and redistribution by developers [4], [5]. Both source code and compiled binaries are accessible and commonly applied across various software domains, including popular tools like Firefox, MySQL, Linux, Audacity, and even MS Internet Explorer. Although some open-source projects may lack comprehensive documentation, they typically follow domain-specific standards. The most widely adopted OSS license, the General Public License (GPL), is used in approximately 70% of open-source projects by developers and end users alike [6].

From the standpoint of software development and maintenance, Open Source Software (OSS) diverges significantly from traditional, structured software engineering methodologies. In conventional models, development is typically carried out by designated teams with clearly defined roles and responsibilities. In contrast, OSS development is characterized by a decentralized and collaborative approach, where contributions can be made by any volunteer developer or user. These contributors engage in post-release source code modification, defect identification, analysis, and dissemination of updates, often without centralized oversight [7].

The OSS development life cycle generally comprises four distinct phases [1], [8], [9]:

- **Cathedral Phase**: Initial development is conducted in a closed environment by a core developer or project leader, without external contributions.
- **Transition Phase**: This intermediate stage marks the shift from closed to open development. Once the architecture is sufficiently modular and stable, a prototype is released to facilitate broader collaboration.
- **Bazaar Phase**: The project is made publicly accessible, enabling a wider community to participate in bug reporting, source code modification, and enhancement activities.
- **Maturity Phase**: At this stage, the software reaches a high level of stability and adoption, characterized by a lower defect rate and reduced frequency of changes.

While the OSS paradigm has demonstrated benefits such as reduced development costs and improved software quality [10], it lacks a universally accepted life cycle model for guiding both development and maintenance activities. Instead, the process is largely influenced by the competencies, experiences, and objectives of individual contributors. The absence of a formal organizational structure may introduce variability in the quality and consistency of the maintenance process. Furthermore, OSS practices are often insufficiently documented, with ill-defined boundaries between development and maintenance phases [11]. As OSS continues to gain traction in various domains, effective strategies for its maintenance and evolution have become increasingly critical.

This paper takes a close look at what makes maintenance in Open Source Software (OSS) unique, especially from a technical perspective, and explores the key features that define how OSS maintenance works. Based on this analysis, we introduce a practical framework designed to help teams make the most of OSS when reusing components for maintenance tasks. We also put our framework to the test, evaluating how reliable, user-friendly, and efficient it is in real-world scenarios.

Here's how the paper is organized: In Section 2, we start by reviewing what other researchers have discovered about reusing OSS for software maintenance. Section 3 dives into some of the main methods people use for reuse-based maintenance, giving you a clear overview of the different approaches out there. In Section IV, we lay out our own framework, walking through how it works step by step. Finally, Section 5 wraps things up with a summary of what we found and some thoughts on where this research could go next.

2. A Survey of Open Source Software Research

Software maintenance is generally understood as any modification made to a system after it has been delivered to users. The open and collaborative nature of Open Source Software (OSS) development can offer distinct advantages for managing and maintaining software, often contributing to higher project success rates. However, for maintainers aiming to reuse OSS components effectively, it is important to consider the underlying development model of the OSS in question. For instance, the bazaar model, with its informal management style and limited documentation, may pose challenges that can lower the likelihood of project success, whereas the cathedral model adopts a more structured, top down approach [14], typically supported by comprehensive policies and better documentation, which tends to enhance both the quality and reliability of OSS. The openness of OSS projects means they are constantly evolving, with frequent changes and contributions from a diverse community of developers. As a result, the long term success of OSS initiatives often hinges on the ability to break the software into well defined, modular components and to provide clear, thorough documentation of specifications, factors that are especially critical when considering OSS for reuse in maintenance processes, as they directly impact how easily and effectively the software can be adapted and maintained over time.

For many organizations, ensuring access to reliable source code is a primary objective. Given that open-source software (OSS) is freely available and typically subject to minimal licensing restrictions, organizations often prefer to use OSS without making direct modifications to the source code. However, unplanned or ad hoc modifications can result in significant challenges—such as the proliferation of unmanageable OSS versions or the simultaneous use of incompatible versions within the same organization. To address these issues, OSS projects must implement robust configuration management practices to control versioning and manage software changes effectively, supporting long-term software evolution [16].

Koponen and Hotti conducted an analysis of two prominent OSS projects—Mozilla Web Browser and Apache HTTP Server—to identify core OSS maintenance activities. They categorized these activities into two pre-delivery and thirteen post-delivery tasks, drawing parallels to the ISO/IEC standard software maintenance process [17].

In OSS maintenance, practitioners primarily rely on two forms of documentation: the source code with embedded comments and the documentation related to the logical data model and requirements specification. Access to the source code, along with the executable, enables maintainers to modify the software when needed for bug fixes, enhancements, or component reuse. Studies have shown that such documentation is considered even more critical than software architecture in facilitating system comprehension. Among maintainers, these artifacts are regarded as essential tools for effective software understanding and maintenance [9], [18].

Several researchers have examined and compared the maintenance processes outlined in the ISO standard with those observed in Open Source Software (OSS) projects. In these studies, tools such as Defect Management Systems (DMS), which are used to log and track defect reports, and Version Management Systems (VMS), which enable developers to revert to previous versions when recent changes are problematic, played a central role. The findings indicate that OSS maintenance practices often differ from traditional approaches by lacking formal retirement and migration stages. Additionally, it was noted that, in OSS environments, the acceptance of modifications typically occurs after the changes have already been implemented, rather than beforehand.

3. Exploring Approaches to Software Maintenance Through Component Reuse

3.1. Approaches to Maintenance Through Reuse

Contemporary software updates frequently involve modifying specific components of existing systems while incorporating previously developed software elements and potentially introducing new components. These reusable elements are typically sourced from repositories containing commercial off-the-shelf (COTS) components or Open Source Software (OSS) resources.
The existing software systems can be reused through three primary models:

- **Rapid Resolution Model:** When facing time constraints, this approach prioritizes quickly identifying the issue, implementing code modifications to address defects as expeditiously as possible, and subsequently updating documentation.
- **Progressive Enhancement Model:** This model is appropriate when requirements lack complete clarity. It follows a process where documentation is first modified, followed by corresponding code-level changes. This approach inherently supports component reuse.
- **Comprehensive Reuse Model:** This strategy requires thorough understanding of all system components. It involves constructing a new system by integrating elements from the original system with components available in established repositories.

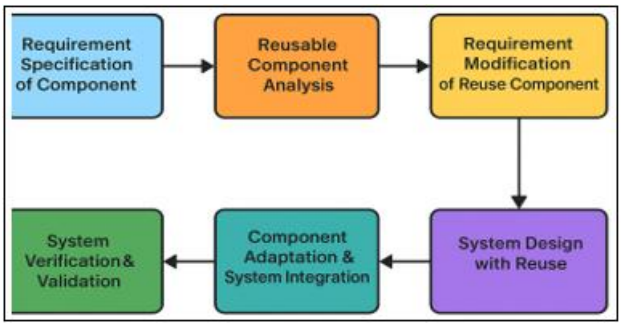

Figure 1: Reuse-based model

Fig 1 illustrates the key phases of component-based software engineering, where development relies on integrating reusable components that have been previously created by other developers.

3.2 Advantages of Maintenance Through Component Reuse

Implementing a reuse-based approach to software maintenance offers several significant benefits:

- **Enhanced System Reliability**: Utilizing previously tested and validated components increases overall system dependability.
- **Minimized Process Risk**: Incorporating established components reduces uncertainties associated with development and maintenance activities.
- **Productivity Enhancement**: Development teams can accomplish more in less time by leveraging existing solutions rather than building from scratch.
- **Improved Standards Compliance**: Reused code often better adheres to industry standards due to its repeated application and refinement.
- **Shortened Development Cycles**: Projects reach completion faster when utilizing pre-existing, proven components.
- **Superior Architectural Qualities**: Reusable components typically exhibit desirable structural characteristics, including well-defined modularity, reduced interdependencies (low coupling), strong internal cohesion, and consistent programming conventions.

4. A Proposed General Framework

4.1 Key Factors Influencing OSS Reuse in Maintenance

To develop an effective framework for Open Source Software (OSS) reuse in maintenance activities, we must first identify the critical factors that influence decision making in this domain. These factors span organizational, technical, and human dimensions that collectively shape the success of OSS integration. At the organizational level, the structure of development and maintenance teams, which includes clearly defined roles, responsibilities, and decision making authority, significantly impacts maintenance effectiveness. This is particularly important given that OSS processes are often poorly documented compared to traditional software development approaches. Technical considerations include the functional size of the OSS, which directly affects complexity as larger systems tend to exhibit non linear growth in algorithmic complexity. Similarly, data manipulation complexity, which is influenced by internal data structures, external logical files, and input output requirements, plays a crucial role in determining reuse suitability.

The distributed nature of OSS functionality, including the number of semantic processing steps and transformations, introduces additional complexity that must be carefully evaluated. Application domain specificity and OSS type (such as real time software) further constrain reuse decisions, as do architectural considerations and component engineering practices. Human factors cannot be overlooked, as the skills and experience of software maintainers directly impact their ability to effectively integrate and maintain OSS components. The expected software lifetime also influences reuse decisions, particularly when considering long term support implications. Additionally, diverse programming methodologies and languages used across OSS projects can present integration challenges that must be addressed.

Documentation quality serves as a critical enabler of visibility and communication throughout the software lifecycle, though its comprehensiveness varies considerably across different OSS projects and process models. Practical considerations such as ease of installation and interoperability with existing systems directly affect implementation feasibility. The distributed development process characteristic of OSS, including community dynamics and authority structures (who does what and why), introduces unique governance considerations. To support objective selection among multiple OSS alternatives, quality assessment methods must be established and consistently applied. Finally, confidentiality requirements of the OSS application must be evaluated, particularly when integrating open source components into systems with sensitive data or functionality.

4.2 A General Framework for Utilizing OSS in Reuse-Based Maintenance

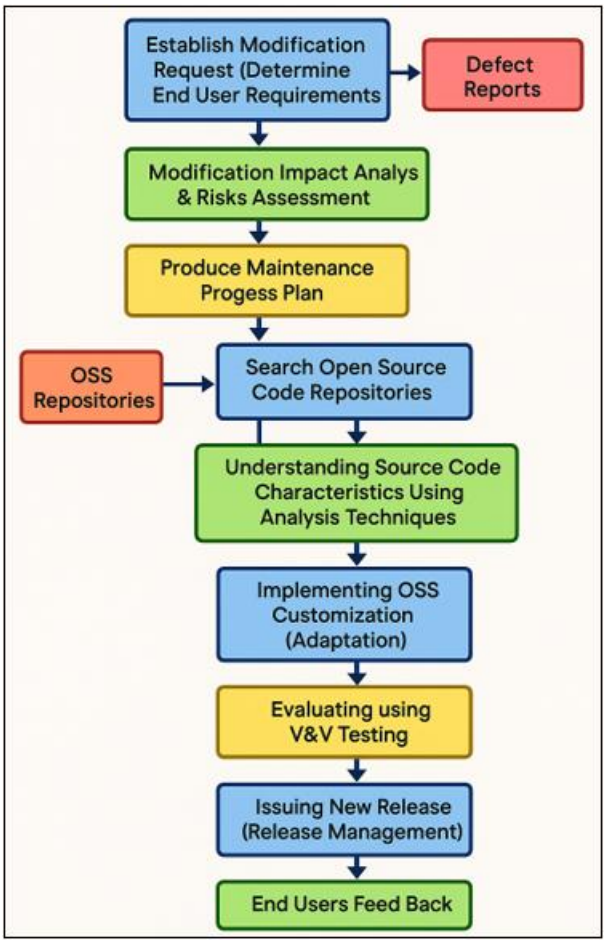

Figure 2: A general framework of OSS development model

The proposed framework outlines key guidelines for integrating Open Source Software (OSS) components into software maintenance activities within an organization, employing a reuse-based model. This process, illustrated in Fig. 2, begins with establishing modification requests, where specific change requirements from end-users are determined to fully understand the problem and identify the necessary functions and services. This is followed by modification impact analysis and risk assessment, which involves identifying software parts potentially affected by the proposed changes and tracking the ripple effects of these modifications. Subsequently, studying the target solution requires defining the desired functionality and structure of the OSS component, a step that depends on clearly identifying its capabilities, problem-solving logic, internal structure, data organization, dependencies, interactions, platform compatibility, and pre- and post-conditions. Finally, producing maintenance planning involves creating formal or informal plans based on analyses of time, cost, and effort.

The next phase focuses on selecting and understanding the OSS component. This involves searching open source code repositories to find suitable OSS solutions that meet criteria such as modularity, low coupling, high cohesion, desired functionalities, and compatible interfaces; the availability of appropriate OSS can significantly reduce costs. Once a candidate is identified, understanding source code characteristics using analysis techniques becomes crucial. Maintainers must build a complete and accurate mental model of the OSS, which is influenced by programming style, comments, coding standards, and code control flow. Given that maintainers can spend 50% to 90% of maintenance time on program comprehension, and understanding large OSS applications is challenging, techniques like control and data flow analysis and call graphs are essential to identify the problem domain, estimate resources, choose algorithms, and find cause-effect relations. Following this, implementing OSS customization is often necessary to adapt the OSS to the new environment, utilizing refactoring techniques. While vendor-provided APIs can restrict customization, there is a general lack of clear guidelines for companies undertaking OSS customization.

The final stages involve integrating and deploying the modified system. Bottom-up integration sees the customized OSS component combined with other system components to produce the new software version. Evaluation to update the current OSS is then performed, involving thorough testing (including unit testing) to ensure the code works efficiently and effectively. Key OSS quality review attributes like usability, functionality, reliability, performance, accessibility, security, and maintainability are analyzed, some in the initial release and others in subsequent versions, to reduce user impact. Issuing a new release (release management) ensures all changes are tested and deployed, allowing users to effectively utilize the services. The nature of the release (major or minor) depends on the extent of changes to old components or additions of new functionalities based on software requirements. Finally, end user feedback is gathered, which initiates a new cycle of maintenance and evolution aimed at further improving software usability and quality.

5. Conclusion

This paper addresses the increasingly important issue of leveraging Open Source Software (OSS) in maintenance activities, driven by its growing popularity. We have demonstrated that OSS components can be effectively reused in maintaining legacy software systems, discussing three main reuse-based maintenance models and their associated benefits. Furthermore, we have categorized, analyzed, and determined the key technical, personnel, environmental, and project managerial factors that influence the success of OSS-based maintenance and evolution. To ensure that maintenance does not inadvertently increase design complexity, we introduced a method for analyzing the modularity of reused OSS using functional points and their interconnections, proposing an algorithm based on a breadth-first strategy to measure modularity strength, which aids in reducing code complexity, removing unused code, defining understandable patterns, and producing well-structured designs. Based on these insights, a general descriptive maintenance processes framework model has been proposed, clearly outlining the workflow guidelines for OSS reuse. Finally, the aspects of OSS customization and its reuse in software maintenance were evaluated considering reliability, usability, accessibility, and efficiency.

References

[1] J. E. Corbly, “The free software alternative: Freeware, open source software, and libraries,” Inf. Technol. Libr., vol. 33, no. 3, pp. 65–75, Sep. 2014.

[2] A. Zoitl, T. Strasser, and A. Valentini, "Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study," in Proc. IEEE Int. Symp. Ind. Electron. (ISIE), 2010, pp. 3817–3819.
[3] Open Source Initiative, "The open source definition," 2003. [Online]. Available: http://www.opensource.org/docs/definition.php
[4] S. Saini and K. Kaur, "A review of open source software development life cycle models," Int. J. Softw. Eng. Its Appl., vol. 8, no. 3, pp. 417–434, 2014.
[5] S. Mandal, S. Kandar, and P. Ray, "Open incremental model — A open source software development life cycle model (OSDLC)," Int. J. Comput. Appl., vol. 21, no. 1, pp. 33–39, May 2011.
[6] C. M. Schweik and A. Semenov, "The institutional design of open source programming: Implications for addressing complex public policy and management problems," vol. 8, no. 1–6, Jan. 2003.
[7] V. Potdar and E. Chang, "Open source and closed source software development methodologies," in Proc. Int. Conf. Softw. Eng. (ICSE), 2004, pp. 105–109.
[8] M. Ueda, "Licenses of open source software and their economic values," in Proc. Appl. Internet Workshops, 2005, pp. 381–383.
[9] A. Capiluppi and M. Michlmayr, "From cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects," Eur. J. Inform. Prof., vol. 8, no. 6, pp. 8–17, 2007.
[10] K. Crowston and J. Howison, "The social structure of open source software development," First Monday, vol. 10, no. 2, Feb. 2005. [Online]. Available: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1207/1127