The Impact of Reusing Open-Source Software Model in Software Maintenance

Nedhal A. Al-Saiyd

Abstract—**Open Source Software (OSS) is becoming very popular and using OSS in maintenance activities has become one of the OSS important issues. Researches in the area of reuse-based maintenance using OOS are rare. The attributes and metrics of OSS maintenance processes are identified depending on literature study. We have proposed a maintenance framework model for reusing of OSS applications in software maintenance, because there are no well-specified guidelines to follow for reuse-based maintenance using OOS. We have tried to shorten the gap between the change request and the reuse-based maintenance using OOS. The paper also shows how to analyze OOS modularity using functional points.**

Index Terms—**Open-source software, maintenance process models, reuse-base maintenance, OSS customization.**

I. Introduction

Open Source Software (OSS) is becoming very popular since it was introduced in 1998 and used widely in companies, communities, homes and governments. OSS is software source code produced by a collaborative process, where many developers and users can participate in developing and posting it to trusted repositories on the internet, and the users have the right to run, copy, distribute, change local copies and improve the software under the OSS license policies [1]. OSS is adopted in most aspects of world's computing technologies. OSS has become the underlying basis of mobile applications and mobile devices, next generation databases, cloud computing, software-as-a-service, and the internet [2], [3]. Open-source software is copyrighted software and has approved license from Open Source Initiative (OSI), which gives right to other developers to copy, modify and distribute it to others, or integrate it with other software components [4], [5]. The programming code and the compiled program are available and used in different software domains; such as well-known applications: MS Internet Explorer, Firefox, MySQL, Linux, and Audacity. Open-source software has standards in their employed field, even some of them lack of documentation. The predominant license "General Public License" (GPL), which is used by programmers and end users, serves 70% of open-source software [6].

From the OSS development and maintenance view point, OSS is unlike a systematic software engineering approach; where the development process is done by a particular developers and maintainers with predefined roles. In OOSs, any volunteer developer or user can develop, do post-delivery source code modification, analyze defects or disseminate them. It is done in collaborative manner [7]. OSS development life cycle model consists of the following main phases [1], [8], [9]:

1) Cathedral Phase, the project initiation is done by core developer, and project manager in the closed environment without outside participants.
2) Transition Phase, it is considered as a link between cathedral phase and bazaar phase. When the design becomes stable and modular, the transition phase is initiated and a prototype is ready to use.
3) Bazaar Phase, OSS is now ready for distribution, where bug detecting and reporting, modification and analyzing tasks occur.
4) Maturity Phase, when OSS is working accurately popular, and has less bug rate, less changes.

OSS Development paradigm not only has decreased the development effort and cost, but it increased software quality too [10]. There is no standardized life cycle approaches support OSS development and maintenance processes. OSS development and maintenance are performed depending on different developers' skills, experiences and end users needs. The absence of OSS development/maintenance organizational structure can affect on the quality of development or maintenance processes. Also, OSS processes are poorly described and the boundaries between the activities are not well-specified and clear [11]. Using OSS in maintenance activities have become one of the OSS important issues.

The purpose of this paper is to analyze technical attributes and characteristics of OSS maintenance processes and propose a general descriptive maintenance processes framework model. The evaluation of framework according to reliability performance, usability and efficiency is then studied.

The rest of the paper describes the flow of information in following sections. In Section II will present the review of the current research work of about the reusing open source software in software maintenance. Section III the reuse-based software maintenance is presented and discussed. It will give precise view of the various reuse-based software maintenance methodologies. In Section IV, the proposed framework model and process work flow are described. Finally, the conclusion is presented in Section V.

II. Open Source Software Literature Review

Usually software maintenance is defined as any modification made on a software system at post-delivery [12]. OSS development model can support effectively various issues associated with management and maintenance processes and can increase the success rate of software. To use OSS as a reusable component in software maintenance

Manuscript received May 30, 2015; revised November 15, 2015.

Nedhal A. Al-Saiyd is with the Faculty of Computer Science, Applied Science University, Amman Jordan (e-mail: nedhal_alsaiyd@asu.edu.jo, nedhal_alsaiyd@yahoo.com).

process, the maintainer will need to know what OSS model is used in developing. In bazaar model, the development is unstructured management and documentation assets, which will decrease project success rate. While top-down cathedral development model, top-down is more structure and moving toward comprehensive policies that can improve OSS success and quality [13], [14].

Due to the openness nature, OSSs are increasingly growing and frequently changing as the software evolves and more people participate in developing and changing source code. Consequently, the success of OSS development depends heavily on the ability to decompose the software into different well-defined components and to document the specifications clearly [15].

Often the goal of most of the organizations is to have reliable source code; since the source code is freely available with few restrictions, and not to do modification on source code. The unplanned modification can lead to problems when having unmanageable OSS versions, or using different versions of OSS simultaneously in the same company. Therefore, OSS projects need to define appropriate configuration management, for version and managing changes for software evolution [16].

Koponen and Hotti have studied two well-known OSS projects, Mozilla Web browser and Apache HTTP server, to identify OSS maintenance activities. The OSS maintenance activities were divided into two pre- and thirteen post-delivery activates. They assume they are similar to ISO/IEC maintenance process [17].

Software maintainer concentrates in his work on using two documentation types during software maintenance; the software source code and the embedded readable comments, and the documentation of logical data model and requirement specification The executable program is accompanied with source codes so that it can be easily modified when it is required to enhance the software, fix bugs or reuse the source code.

A study shows they are more important than software architecture. Among software maintainers, these documentations are considered as important artifacts that help to understand a system [9], [18].

Other researchers studied the ISO maintenance process and OSS maintenance process using 1) Defect Management System (DMS); which is used to store the defect reports, and 2) Version Management System (VMS); where developers are able to restore the older version if the later changes are not correctly implemented. They showed that OSS maintenance process there did not have retirement and migration stages, and modification acceptance occurred after implementation [19], [20].

III. Overview of Reuse-Based Software Maintenance

A. *Reuse-Based Maintenance Models*

Many of the present-day new software versions are obtained by modifying one or several components of the old software components using previously developed software components and possibly adding new components. The reusable components are obtained from component repositories of commercial off-the-shelf (COTS) components, and OSS components.

The old software system can be reused by [21]:

1) *Quick-fix model:* Due to time constraint, first identify the problem, modify the code to fix the defects as quickly as possible then modify the documentation.
2) *Iterative-enhancement model:* the model is required when the requirements are not fully-understood. The documentation is modified and then them modification is done on the code level. It supports reuse.
3) *Full-reuse model:* It requires fully understanding of the system parts. A new system is built from components of the old system and from components available in the component repository.

Fig. 1 shows the main stages of component-based software engineering that uses the reusable components that already developed by other developments.

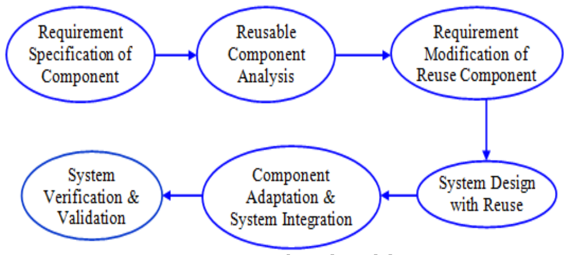

Requirement Specification of Component
Reusable Component Analysis
Requirement Modification of Reuse Component
System Design with Reuse
Component Adaptation & System Integration
System Verification & Validation

Fig. 1. Reuse-based model.

B. *Benefits of Reuse-Based Maintenance* [21]

1) Reusable components Increase reliability.
2) Reduced process risk
3) Increase productivity
4) Reusing code can improve its compliance with standards
5) Accelerated development time
6) Reusable components possess modularity, low coupling, high cohesion, and consistent programming style.

IV. General Framework

First, we need to determine the factors that influence OSS reuse-based maintenance activities and help in decision making:

A. *OSS Reuse Influence Factors*

1) Organizational Metrics: Development/maintenance organizational structure that reflects the actor's roles, actors, responsibilities for technical decisions and efforts of maintenance effective activities.
2) OSS processes are poorly described.
3) OSS Functional Size: as a size becomes very large and the algorithm complexity will grows non-linearly and be very complex. Software size is measured by three factors: the data, the function and the control behaviors (dynamic behaviors) [22].
4) Data manipulation complexity to produce the expected output data. This issue is influence by the internal data structures and external logical files, the number of inputs, and the number of outputs.
5) OSS distributed functionality and number of semantic processing steps and transformations.
6) The application domain and the OSS type; as in real-time software.
7) OSS architecture.

8) Component Engineering
9) Skills and experiences of software maintainer.
10) The expected software lifetime.
11) Diverse programming methodologies and programming languages.
12) Up-to-Date Documentation. Documentation introduces visibility and communication within software phases. Documentation may vary between those process models.
13) Ease of installation and interoperability with the system being developed or maintained.
14) Distributed development process of OSS ad OSS community and authority structure (who does what and why).
15) Quality assessment methods to support the selection among several OSS application
16) OSS application confidentiality

B. Analyzing OOS Modularity Using Functional Points

To examine the OOS modularity, OSS application is converted into graph, where each function is represented by a node on a graph, and the conceptual communications among the functions are represented by links in the graph. Let us examine the group of functions in OSS application that share common features. The functional points are defined into a two-dimensional matrix '*F*'. We assume we have '*n*' intercommunicating functional points. The communication among functional points of OSS component can be represented by graph, where each node represents the semantic functional point and each link represents the interface connection among nodes. This can be represented by a symmetric matrix of size = $n \times n$ and its element F_{ij} represents all the links between the functions (nodes) inside the OSS application; that is the connections between (Function$_i$) in node *i* to (Function$_j$) in node *j*. F_{ii} represent all the links inside the function.

$$F = \begin{pmatrix} F_{11} & \cdots & F_{1n} \\ \vdots & \ddots & \vdots \\ F_{n1} & \cdots & F_{nn} \end{pmatrix} \tag{1}$$

The summation of all the links in the OSS is represented by matrix' row (or Matrix' column can be used), which shows the links that connect among different functions (i.e. connecting functional points). It is defined by:

$$a_i = \sum_{j=1}^{n} F_{ij} \tag{2}$$

To measure the modularity strength '*M*' of the OSS structure, '*M*', which is defined as an objective measure:

$$M = \sum_i \left(F_{ij} - a_i^2 \right) \tag{3}$$

where:

a_i^2: represents the summation of the functions of OSS, which is represented by matrix elements (expected value of modularity).

F_{ii}: represents the links that connect nodes exist inside a function, which in turn exists inside OSS application (i.e. OSS network graph).

Minimizing the number of intercommunication among functional points will minimize the modular structure complexity and therefore will decrease the execution time.

If $M = 0$, it means it is randomly structure.

If $M = 1$, it means functions are low coupling and has modularity.

Therefore, the measure can reduces complexity of source code and removing the unused code, define patterns that are easier to understand code and maintain, and produced well-structure design.

The algorithm can be performed on the produced OSS graph depending on breadth-first strategy.

First: The initial node is represented by 'I_n' and the distance is assigned as 0; Dis(I_n)=0 and its weight; Weight(I_n) = 1.

Second: The distance of every Node *i* from I_s that is adjacent to I_n is calculated as:

Dis(Node$_{i)}$ = Dis(I_n) + 1 = 0 + 1 =1, and its weight is calculated by:

Weight (Node$_i$) = Weight (I_n) = 1.

Third:

Repeat

For all rest Nodes; there is a Node *j* that is adjacent to any Node *i*, one of the following is applied:

If *j* has not yet been assigned a distance, it is assigned distance Dis(Node$_j$) = Dis(Node$_i$) + 1 and weight Weight(Node$_j$) = Weight(Node$_i$).

If *j* has already been assigned a distance and Dis(Node$_j$) = Dis(Node$_i$) + 1, then the Node's weight is increased by Weight(Node$_i$): Weight(Node$_j$) = Weight(Node$_j$) + Weight(Node$_i$).

If *j* has already been assigned a distance and Dis(Node$_{j)}$ < Dis(Node$_i$) + 1, then do nothing.

Until no Nodes remain

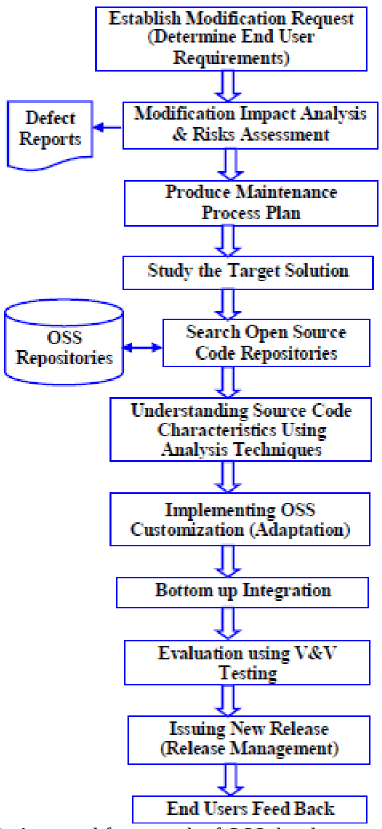
The weight in the OSS graph represents the distance of nodes from the initial node.

C. A General Framework of OSS Development Model

The proposed framework shows the main guidelines to use OSS component in software maintenance within a user organization using OSS reuse-based model. Fig. 2 illustrates these stages.

1) **Establish Modification Request**: Determine Specific change Requirements from end users. It is to understand the Problem to identify the required function and services
2) **Modification Impact Analysis & Risks Assessment:** Identify parts of the software that can be affected by the proposed changes and track the effect of changes.
3) **Study the Target Solution**. Define the OSS desired function and structure, which depends on identifying the functionality, logic of solving-methods, OSS structure, data structure, dependencies, interactions, platforms, pre-conditions and post-conditions.
4) **Produce Maintenance Planning:** Formal and informal plans can be produced at this stage. Planning uses time-, cost- and effort-based analyses.
5) **Search Open Source Code Repositories:** Searching for OSS is part of the development or maintenance process. A variety of reuse-based OOS solutions are available so the best suited one is adopted, which meet t meet several criteria of modularity, low-coupling, highly cohesion,

desired functionalities and interfaces. The availability may lead to decrease the cost.



Establish Modification Request (Determine End User Requirements)
Defect Reports
Modification Impact Analysis & Risks Assessment
Produce Maintenance Process Plan
Study the Target Solution
OSS Repositories
Search Open Source Code Repositories
Understanding Source Code Characteristics Using Analysis Techniques
Implementing OSS Customization (Adaptation)
Bottom up Integration
Evaluation using V&V Testing
Issuing New Release (Release Management)
End Users Feed Back

Fig. 2. A general framework of OSS development model.

6) **Understanding Source Code Characteristics Using Analysis Techniques:** The maintainer needs to build a complete and accurate mental model. The model depends on programming style, comments, coding standard, and code control flow. Maintainers spend approximately from 50% to 90% of software maintenance estimated time on program understanding [23]. Understanding a large OOS application is not an easy task. Maintainers depend in their work on source code analysis techniques, such as, control and data flow and call graphs. It is necessary to identify the problem domain, estimate resources, help to choose algorithms, and to find cause-effect relations.
7) **Implementing OSS Customization:** customization is needed to adapt OSS to work in a new environment. Refactoring techniques are used in this stage. The deployment of Application Programming Interfaces (APIs), which are provided by the vendor, restricts the customizations. But there are no good guidelines to follow by companies when customizing of OSS.
8) **Bottom up Integration:** Integration OSS with the other components to produce the new version of software System.
9) **Evaluation to update the current OSS.** Testing is done to keep the code working efficiently and effectively. Unit testing are necessary. The important set of OSS quality review attributes is: usability, functionality, reliability, performance, accessibility, security, and maintainability. They are analyzed and reviewed individually. Some of the review attributes must be done in the first release while others are done in the later versions. Testing reduces the impacts to users
10) **Issuing New Release (Release Management):** The new release ensures that all changes are tested, deployed and the users use the services efficiently and effectively. The major or minor release consists of the changes on the old software components or/and the new functionalities according to the amount of changes or/and additions on the software requirements.
11) **End Users Feedback:** Feedback is the start of new maintenance and evolution cycle to improve the software usability and quality.

V. Conclusion

Open Source Software (OSS) is becoming very popular and using OSS in maintenance activities has become one of the OSS important issues.

1) The open source software components can be reused in maintaining the old or legacy software system. The main three reuse-based maintenance models are discussed and the benefits are defined.
2) The technical, personnel, environmental, and project managerial factors that influence reusing OSS-based maintenance processes; are categorized, analyzed and determined in this research. These key factors influence the success of the OSS-based software maintenance and evolution.
3) Since the objective of maintenance is not to increase the complexity of the structural design of the new software product, the Modularity of the reused-OOS is analyzed using their functional points and the links among them. The modularity strength is measured. The algorithm is performed on the produced OSS graph depending on breadth-first strategy. The OSS modularity measure can reduces complexity of source code and removing the unused code, define patterns that are easier to understand code and maintain, and produced well-structure design.
4) A general descriptive maintenance processes framework model is proposed and the sequence of work flow of OSS reuse-based guidelines is described clearly.
5) The OSS customization and the reusing OSS in software maintenance is evaluated according to reliability, usability, accessibility and efficiency is studied.

Acknowledgment

The authors are grateful to the Applied Science University in Amman, Jordan for the financial support granted to cover the publication fee of this research article.

References

[1] S. Mandal, S. Kandar, and P. Ray, “Open incremental model — A open source software development life cycle model (OSDLC),” *International Journal of Computer Applications*, vol. 21, no. 1, pp. 33-39, May 2011.

[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 International Symposium of Industrial Electronics (ISIE)*, 2010, pp. 3817-3819.

[3] Saini and K. Kaur, “A review of open source software development life cycle models,” *International Journal of Software Engineering and Its Applications*, vol. 8, no. 3, pp. 417-434, 2014.

[4] Open Source Initiative: The open source definition. (2003). [Online]. Available: http://www.opensource.org/docs/definition.php

[5] J. E. Corbly, "The free software alternative: Freeware, open source software, and libraries," *Information Technology and Libraries*, vol. 33, no. 3, pp. 65-75, Sep. 2014.

[6] M. Ueda, "Licenses of open source software and their economic values," in *Proc. Applications and the Internet Workshops*, 2005, pp. 381-383.

[7] V. Potdar and E. Chang, "Open source and closed source software development methodologies," in *Proc. International Conference of Software Engineering (ICSE)*, 2004, pp. 105-109.

[8] 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.

[9] A. Capiluppi and M. Michlmayr, "From cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects," *The European Journal for the Informatics Professional*, vol. 8, no. 6, pp. 8-17, 2007.

[10] E. Capra, C. Francalanci, and F. Merlo, "An empirical study on the relationship between software design quality, development effort and governance in open source projects," *IEEE Transactions on Software Engineering*, vol. 34, no. 06, pp. 765-782, November/December, 2008.

[11] K. Crowston and J. Howison. (Feb. 2005). The social structure of open source software development. *First Monday*. [Online]. Available: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1207/1127

[12] P. Grubb and A. A. Takang, *Software Maintenance: Concepts and Practice*, 2nd ed., U.K.: World Scientific Publishing Co. 2003, ch. 1.

[13] P. Kamthan, "On the prospects and concerns of integrating open source software environment in software engineering education," *Journal of Information Technology Education*, vol. 6, pp. 45-64, 2007.

[14] D. Wong, K. L. Shephard, and P. Phillips, "The cathedral and the bazaar of e-repository development: encouraging community engagement with moving pictures and sound," *Research in Learning Technology*, vol. 16, no. 1, pp. 31-40, Mar. 2008.

[15] J. Feller and B. Fitzgerald, "A framework analysis of the open source software development paradigm," in *Proc. Twenty First International Conference on Information Systems (ICIS)*, USA, 2000, pp. 58-69.

[16] S. P. Alpar, "Customization of open source software in companies," in *Proc. IFIP International Federation of Information Processing, Advances in Information and Communication Technology*, 2009, vol. 299, pp. 129-142.

[17] T. Koponen and V. Hotti, "Open source software maintenance process framework," in *Proc. the Fifth Workshop on Open Source Software Engineering (WOSSE)*, 2005, pp. 1-5.

[18] S. C. De Souza, N. Anquetil, and K. M. de Oliveira, "Which documentation for software maintenance?" *Journal of the Brazilian Computer Society*, vol. 13, no. 2, pp. 31-44, 2007.

[19] H. Lintula, T. Koponen, and V. Hotti, "Exploring the maintenance process through the defect management in the open source projects-four case studies," in *Proc. International Conference on Software Engineering Advances*, Oct. 2006, p. 53.

[20] S. Batra, "Study of maintainability process in open source software," *International Journal of Scientific & Engineering Research*, vol. 3, no. 5, pp. 1-4, May 2012.

[21] P. Tripathy and K. Naik, *Software Evolution and Maintenance: A Practitioner's Approach*, Canada: John Wiley & Sons Inc., 2015, ch. 3, p. 85.

[22] G. Levesque, A. Abran *et al*., "Measuring software functional size: Towards an effective measurement of complexity," in *Proc. International Conference on Software Maintenance*, 2002, pp. 370-376.

[23] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the NLP techniques for source code data preprocessing," in *Proc. the 3rd International Workshop on Evidential Assessment of Software Technologies*, 2014, pp. 32-39.

Nedhal A. Al-Saiyd got her B.Sc. degree in computer science from University of Mosul, Iraq in 1981, M.Sc. and PhD degrees from University of Technology, Baghdad-Iraq in 1989 and 2000 respectively. She is an associate Prof. at the Computer Science Dept., Faculty of Information Technology, in the Applied Science University, Amman, Jordan. She has got more than 24 years of teaching experience. Her research interests include software engineering, ontology engineering, intelligent systems, user authentication, security and speech processing.

She is a member in the Society of Digital Information and Wireless Communications (SDIWC), senior member in Universal Researchers in Engineering (URENG), senior member in Universal Association of Computer and Electronics Engineers (UACEE), member of Editorial and Reviewer Board of International Journal of Modern Education and Computer Science (IJMECS), and senior member in Emirates Association of Computer, Electrical & Electronics Engineers (EACEEE).