International Journal of Engineering & Technology, 7 (3.33) (2018) 67-71

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

The Reuse Process of OSS Components

Jong-Bae Kim[1]*, Myung-Jin Bae[2]

[1] *Graduate School of Software, Soongsil University,369, Sangdo-ro, Dongjak-gu, Seoul, 06978, Korea*
[2] *Department of Telecomm. Engr., Soongsil University, Sori Engineering lab, 369, Sangdo-ro, Dongjak-gu, Seoul, 06978, Korea*
**Corresponding author E-mail: kjb123@ssu.ac.kr*

Abstract

In recent years, OSS has been expanding globally, with the advantage of being able to avoid adverse consequences of proprietary software monopolies, and to reduce the budget and procedures for maintenance and upgrades. This can serve as a good opportunity to prevent the monopoly of large corporations and establish a balanced industrial structure through fair competition. despite the many advantages of OSS, we are having difficulties in using OSS due to several reasons. To meet the functionality of the required software, leveraging open source is seen as an effective way to reduce costs and ensure quality. However, in the actual development process, it is difficult to select open source with proper function and quality, to analyze it, to change it, and to integrate it into the target system. In order to solve these problems, this study proposed the procedures and methods for identifying and selecting open source, and the efficient improvement and integration method through minimal modification.

Keywords: *OSS, Open Source Software, Component, Reuse, Process, Evaluation*

1. Introduction

As we can see in Table 1, open source software (OSS) can be divided into commercial and non-commercial software. A non-commercial OSS is an OSS that is not costly to use the software. Commercial OSS is the software that users pay to use the software. The cost that the user pays is the cost of the service that can easily use the package of the relevant software, not the license fee for the usage rights [1-2].

Table 1: Types of Software Licensing and Deployment

Division		Whether the source code is released	
		Yes	No
Software costs paid by the user	Free	Non-commercial OSS	Freeware / Shareware
	Pay	Commercial OSS	Proprietary commercial software

OSS can be defined against these other types of software, but in a more rigorous sense, OSS is defined as 'software that is distributed on a condition that complies with the Open Source Software Definition (OSD)' [3-4]. The OSD can be summarized as Table 2.

Table 2: The Open Source Definition

Division	Definition
Free Redistribution	The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
Source Code	The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.
Derived Works	The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
Integrity of The Author's Source Code	The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
No Discrimination Against Persons or Groups	The license must not discriminate against any person or group of persons.
No Discrimination Against Fields of Endeavour	The license must not restrict anyone from making use of the program in a specific field of endeavour. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
Distribution of License	The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
License Must Not Be Specific to a Product	The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software

	distribution.
The License Must Not Restrict Other Software	The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

In this paper, we propose procedures and methods for identifying, evaluating, selecting, changing and applying appropriate OSS when developing software using open source software.

2. Related Works

2.1 Reuse of Open Source Software

In recent years, OSS has been expanding globally, with the advantage of being able to avoid adverse consequences of proprietary software monopolies, and to reduce the budget and procedures for maintenance and upgrades. Companies have also attempted to apply the OSS-based development approach as a new alternative to address the limitations of existing software development: software quality, development speed and cost [5-6].
In particular, for small and medium-sized businesses with weak capital and organization, open source can be used as a way to improve the efficiency and quality of software development at low cost. This can serve as a good opportunity to prevent the monopoly of large corporations and establish a balanced industrial structure through fair competition.
The development of OBAs (OSS-Based Applications) is similar to the development of commercial-off-the-shelf (COTS) based applications(CBAs). In other words, it is essential to select the appropriate OSS or COTS product, adjust them as needed, and integrate them with custom components, OSS or COTS. However, there is a significant difference between OBA and CBA development in the detailed procedures. In CBA development, developers cannot access the source code of COTS products, so it is almost impossible to modify COTS for specific needs. On the other hand, at the time of OBA construction, developers must integrate and modify existing OSS because the quality or function of existing OSS cannot fully meet the requirements [7]. However, when blindly following the traditional process model in OBA development, it is often confused about what to do next. As a result, a lot of rework occurs over time, resulting in a certain delay and an additional cost burden. In particular, despite the many advantages of OSS, we are having difficulties in using OSS due to several reasons [7] that shown in Table 3.

Table 3: Difficulties in using OSS

Division	Difficulties
Difficult to navigate	It is difficult to know whether the project exists or what the current situation is because advertising is not done in particular.
Uncertainty	For developers who need to identify and select open source that meets the required functionality and quality, uncertainty about quality is a source of hesitation for using open source.
Poor documentation	It is difficult to analyze because it does not have systematic documents compared with commercial software.

2.2 QSOS and OpenBRR

QSOS and OpenBRR assessment methodologies help their users reach a similar goal, that is, select among a list of similar OSS projects for the one best suited to their context. Some researchers review the advantage and disadvantages of both methodologies (Table 4) [8-14].

Table 4: Advantages and Disadvantages of QSOS and OpenBRR

Division	Advantages	Disadvantages
QSOS	Open repository of evaluation scores for various FlOSS projects (this pushes evaluators to collaborate on evaluation and to facilitate cross validation) Extensive list of criteria Interesting innovating nomenclature for the tree hierarchy QSOS methodology is versioned and evaluation mention the QSOS version used	Ambiguous scoring rules for more than half of the criteria Scoring procedure with 3-level scale may make decision making harder Universality of scoring rules is not possible for many criteria
OpenBRR	Allows for tailoring hence better fit one's evaluation context Clearer scoring procedure with fewer ambiguities 5-level scoring scale for about half of the criteria Ask evaluator to perform a quick assessment step to reduce the evaluation effort	No open repository of evaluation (due to possible tailoring) Does not exploit the 5-level scales for more than half of the criteria Terminology is broad and imprecise for the top nodes in the hierarchy OpenBRR does not seem to be versioned. However, this may be left to the evaluator

And they find that both methodologies have a particular important weakness. They do not require evaluators to capture the location of the raw data used to obtain the evaluation scores. This makes it hard to refute or argue the correctness of an evaluation.

3. Results and Discussion

3.1 Open Source Component Selection Process

Currently, OSS is provided by many communities and sites. Therefore, the steps to select an open source for development from a vast amount of open source should be prioritized.
To achieve this, in the requirements acquisition and open source selection phase, activities such as high level requirements analysis, open source identification, low level requirements analysis, evaluation criteria establishment, open source evaluation and selection are carried out. The overall activities for the require ments acquisition and open source selection phase are shown in Fig. 1.
A high level requirements analysis is an activity that collects the requirements of the software to be developed and identifies the approximate requirements of the software to be developed based on the collected requirements. In the early stages of requirements analysis, user requirements are often unclear and abstract. In particular, in the case of a project for the development of its own product, the planning task defines the required function of the system. The definition of such a function is abstract in the initial stage. Therefore, the function that identified through the high-level requirement identification will be used as a keyword for open source search in the next open source identification activity. This process complements the definition of detailed functions. This is possible because open source can provide not only source code, but also various feature ideas.
Open source selection activities are the process of establishing and evaluating criteria for selecting open source that meets the requirements from the candidates and selecting open source.

Table 5 shows an example of OSS survey criteria, which can be used as a primary screening condition for OSS search.

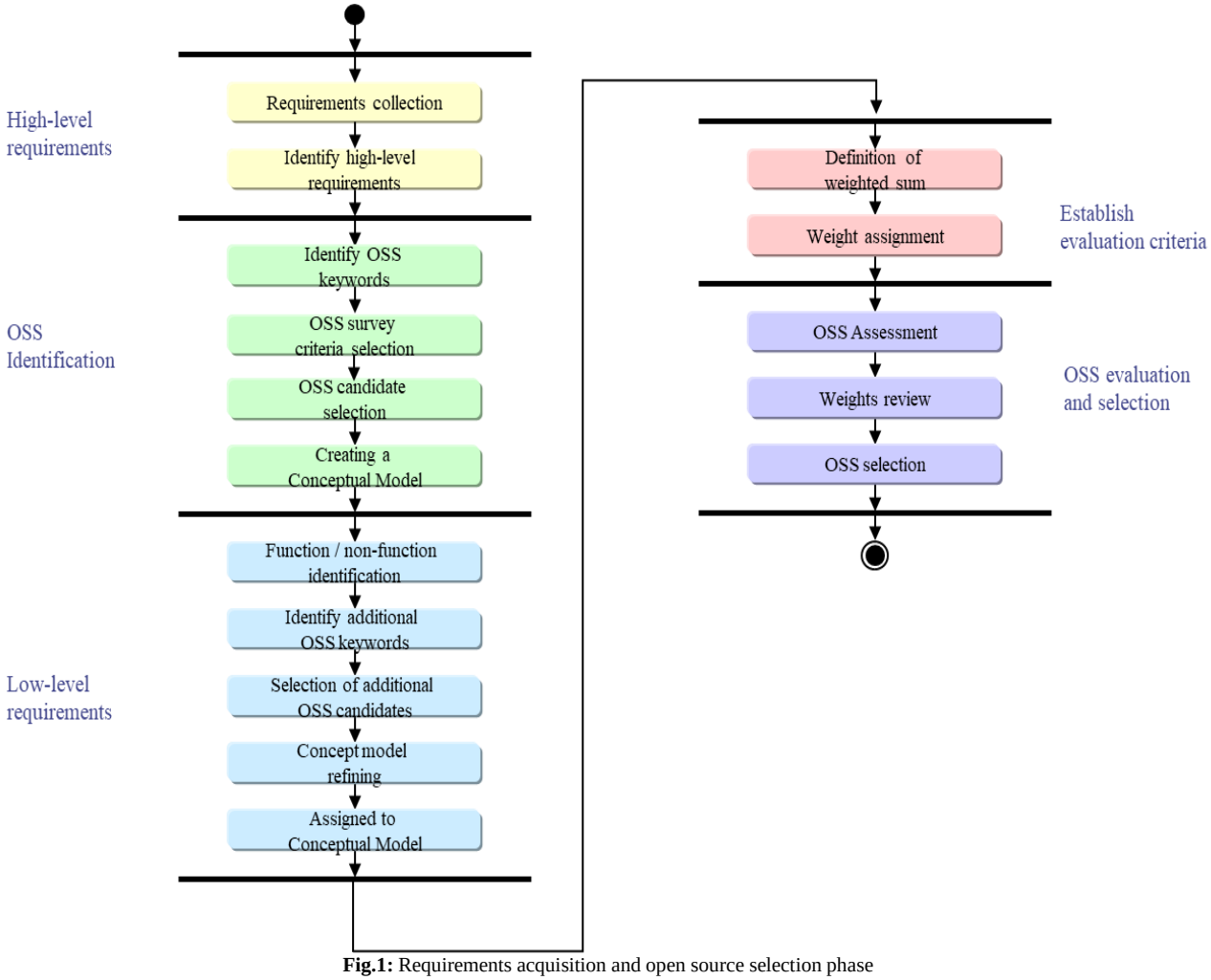

Fig.1: Requirements acquisition and open source selection phase

Table 5: The example of OSS survey criteria

No.	Item	Comments	Remarks
1	Activity	Activity Rate (%)	Exclusion Condition
2	Project Admins	Project Manager (Authority)	Data for Evaluation
3	Developers	More developers, More reliable	Data for Evaluation
4	Development Status	1 - Planning, 2 - Pre-Alpha, 3 - Alpha, 4 - Beta, 5 - Production/Stable, 6 - Mature	Exclusion Condition & Data for Evaluation
5	Intended Audience	-	Data for Evaluation
6	License	-	Exclusion Condition
7	Operating System	-	Exclusion Condition
8	Programming Language	-	Exclusion Condition
9	Topic	Category	Data for Evaluation
10	Translations	Languages for comments and documents	Exclusion Condition & Data for Evaluation
11	User Interface	-	Exclusion Condition
12	Registered	-	Data for Evaluation

The items to be considered in selecting survey criteria and selecting candidates for OSS identification are the distinction between the exclusion condition and evaluation data shown in the survey standard in Table 5.

In general, when introducing commercial software, the most suitable candidates can be selected for a few candidates who have requested the proposal. In this case, only the ranking derivation method through comparative analysis can be used, but in the case of open source, the task of selecting the candidates to be compared should be performed in advance. In particular, even if the keyword search result is minimized, it is necessary to clearly define the "exclusion condition" even for the efficiency of the comparative analysis for future evaluation.

For example, open sources that are less than 50% active, 1-Planning development, or proprietary licenses are unreliable and cost extra to acquire. Therefore, it is better to exclude these open sources from the beginning, and it is necessary to limit the operating system, development language, and user interface according to the system environment to be developed.

In particular, it is the license to be careful when integrating and integrating open source software components. The table 6 compares various features of each license and is a general guide to the terms and conditions of each license. The table lists the permissions and limitations regarding the linking of the licensed code with code licensed under a different license, distribution of the code to third parties, modification of the code by a licensee, protection of licensees from patent claims made by code contributors regarding their contribution, and protection of contributors from patent claims made by licensees, whether modification to the code must be shared with the community or may be used privately, whether modified code may be licensed under a different license (for example a copyright) or must retain the same license under which it was provided, use of trademarks associated with the licensed code or its contributors by a licensee.

Table 6: compares various features of each license

License	Linking	Distribution	Modification	Patent grant	Private use	Sublicensing	TM grant
Academic Free License	Permissive	Permissive	Permissive	Yes	Yes	Permissive	No
Apache License	Permissive	Permissive	Permissive	Yes	Yes	Permissive	No
Artistic License	With restrictions	With restrictions	With restrictions	No	Permissive	With restrictions	No
Beerware	Permissive	Permissive	Permissive	No	Permissive	Permissive	No
BSD License	Permissive	Permissive	Permissive	Manually	Yes	Permissive	Manually
Creative Commons Zero	Public Domain	Public Domain	Public Domain	No	Public Domain	Public Domain	No
CeCILL	Permissive	Permissive	Permissive	No	Permissive	With restrictions	No
Eclipse Public License	Limited	Limited	Limited	Yes	Yes	Limited	Manually
European Union Public Licence	Copylefted, with an explicit compatibility list	Copylefted, with an explicit compatibility list	Copylefted, with an explicit compatibility list	Yes	Yes	Copylefted, with an explicit compatibility list	No
GNU Affero General Public License	GNU GPLv3 only	Copylefted	Copylefted	Yes	Copylefted	Copylefted	Yes
GNU General Public License	GPLv3 compatible only	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes
GNU Lesser General Public License	With restrictions	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes
MIT license / X11 license	Permissive	Permissive	Permissive	Manually	Yes	Permissive	Manually
Mozilla Public License	Permissive	Copylefted	Copylefted	Yes	Yes	Copylefted	No
Sleepycat License	Permissive	With restrictions	Permissive	No	Yes	No	No
Do What The Fuck You Want To Public License (WTFPL)	Permissive/Public domain	Permissive/Public domain	Permissive/Public domain	No	Yes	Yes	No

3.2 Development Process Using Open Source Component

The change design and improvement phase is a key step in using OSS as a process of changing and adjusting selected OSS to meet the requirements of the software to be developed. This step consists of three activities: change request analysis, source code analysis & change design, build and test. The activities and tasks at this stage are shown in Fig. 2.

Because the features and purpose of the software to be developed are different, there is no OSS that exactly matches the target software.

Therefore, the selected open source cannot be used as it is, and the target software should be developed by changing some of the functions, and the non-existing functions must be newly implemented and integrated.

The change request analysis activity analyses change and inconsistency requirements through gap analysis between target software and OSS. Based on this GAP analysis (Fig. 3), we design the abstract-level architecture of target software. These higher level architectures serve as the basis for future development of inconsistency requirements and changes in OSS.

The most effective way to reduce the gap between requirements and components is to use a wrapper or adapter. A wrapper is a method for componentizing and using existing software without modifying the code, and the adapter is used to resolve the inconsistency between the two components.

Also, in order to wrap an existing system, refinement logic is required to solve the inconsistency of data or function between the component and the existing API (Application Program Interface) as in Fig. 4.

Build and test activities change and improve the actual OSS according to the development plan, and perform unit and integration tests of the code that has modified or developed according to the development plan.

Build and test activities apply the open source development approach of staged builds and iterations. The right side of Fig. 2 shows the workflow of build and test activities through incremental builds and iterations. This process changes the source code of the change locations found through analysis, and iterates through the process of integrating, building, and testing it immediately to achieve the goal of rapid modification and integration.

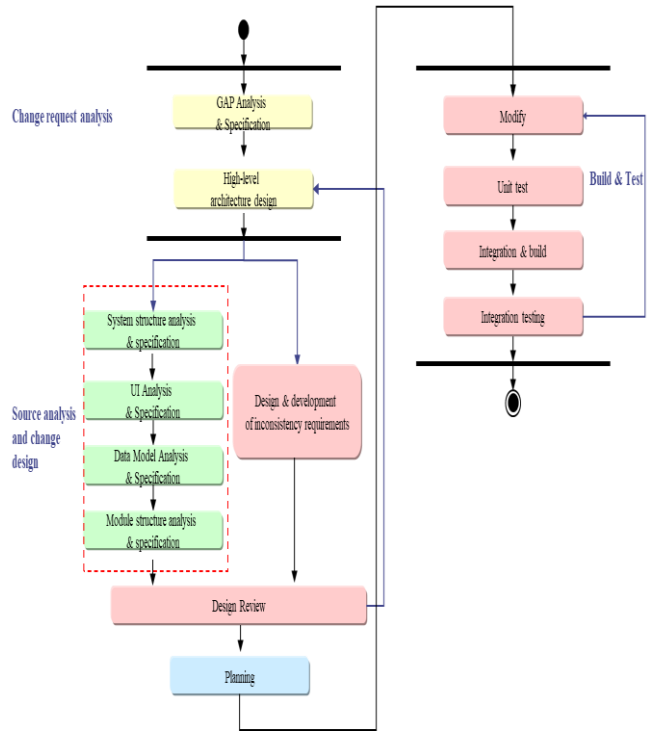

Fig. 2: Change design and improvement phase

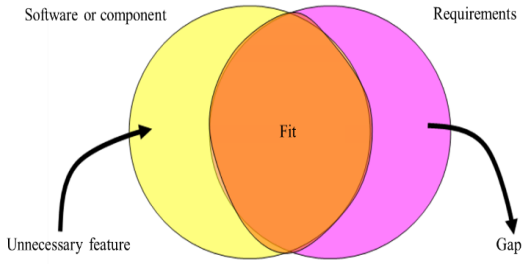

Fig. 3: Concept of gap analysis

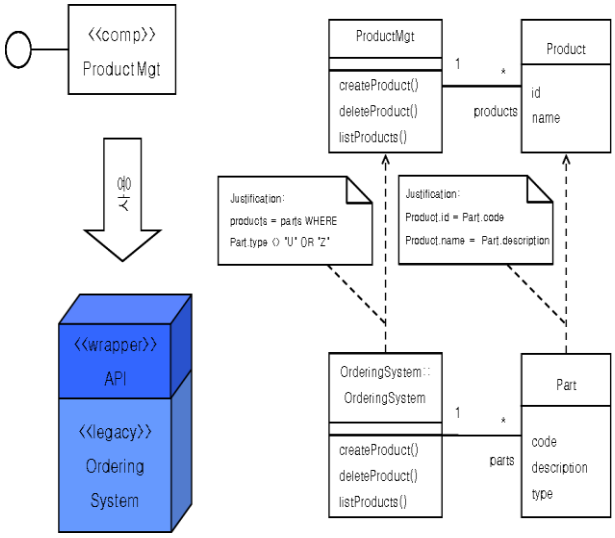

Fig. 4: refinement logic for wrapping

The key to build and test activity is to locate the source code for the change. At this stage, the open source tree structure extracted from the module analysis can be used to analyse the location of the code in the open source to be changed. In addition, because we know the file locations of each operation, class, data structure, and other components through open source analysis, we can directly access the source code that needs to be changed without any further analysis.

4. Conclusion

To meet the functionality of the required software, leveraging open source is seen as an effective way to reduce costs and ensure quality. However, in the actual development process, it is difficult to select open source with proper function and quality, to analyse it, to change it, and to integrate it into the target system.

In order to solve these problems, this study proposed the procedures and methods for identifying and selecting open source, and the efficient improvement and integration method through minimal modification. However, research on metrics for more accurate and objective selection of reusable open source should be reinforced, and detailed procedures and techniques should be defined according to various types of reuse. Further research on the development and application of more specific and practical OSS utilization processes should be undertaken in the future.

References

[1] Kwei-Jay,L., Yi-Hsuan, L., Tung-Mei, K., Examining Open Source Software Licenses through the Creative Commons Licensing Model. Software Applications: Concepts, Methodologies, Tools, and Applications, 6 (2009), 2978-2990.

[2] Josh, L., Jean, T., The Scope of Open Source Licensing. The Journal of Law, Economics, and Organization, Volume 21, Issue 1, (2005), 20–56.

[3] Joseph, F., Brian, F., A framework analysis of the open source software development paradigm. ICIS '00 Proceedings of the twenty first international conference on Information systems, (2000), 58-69.

[4] OSD, "http://www.opensource.org/osd.html"

[5] International Institute of Infonomics University of Maastricht, The Netherlands and Berlecon Research GmbH Berlin. Germany Research, FINAL REPORT, (2002).

[6] Martin, F., The Business and Economics of Linux and Open Source. Prentice Hall PTR, (2002).

[7] MICHAEL, K., Commercializing Open Source Software. ACM Queue, vol.1, no.5, (2003), 51-55.

[8] Jean-Christophe, D., Simon, A., Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS, PROFES 2008, LNCS 5089, (2008), 189–203.

[9] Petrinja E., Sillitti A., Succi G., Comparing OpenBRR, QSOS, and OMM Assessment Models. In: Ågerfalk P., Boldyreff C., González-Barahona J.M., Madey G.R., Noll J. (eds) Open Source Software: New Horizons. OSS 2010. IFIP Advances in Information and Communication Technology, vol 319, (2010).

[10] Stol, KJ., Ali, M., A Comparison Framework for Open Source Software Evaluation Methods. In: Ågerfalk P., Boldyreff C., González-Barahona J.M., Madey G.R., Noll J. (eds) Open Source Software: New Horizons. OSS 2010. IFIP Advances in Information and Communication Technology, vol 319, (2010).

[11] Adewole, A, Sanjay, M., Nicholas, O., A Review of Models for Evaluating Quality in Open Source Software. IERI Procedia, Volume 4, (2013), 88-92.

[12] Cotugno, R., Messina, A., Adapting scrum to the italian army: methods and (open) tools. In: IFIP International Conference on Open Source Systems. Springer, Berlin, Heidelberg, (2014), 61-69.

[13] Syahlie, K., Hypervisors assessment in education industry: Using OpenBRR methodology. In: Information, Communication Technology and System (ICTS), 2014 International Conference on. IEEE, (2014), 303-308.

[14] Zahoor, A., Mehboob, K., & Natha, S., Comparison of open source maturity models. Procedia computer science, 111 (2017), 348-354.