A Development Process for Building OSS-Based Applications

Meng Huang[1,2], Liguang Yang[1,2], and Ye Yang[3]

[1] Lab for Internet Software Technologies, Institute of Software,
The Chinese Academy of Sciences, P.O.Box 8718, Beijing, 100080, China
[2] Graduate School of the Chinese Academy of Sciences,
Beijing 100039, China
{hm, yanglg}@itechs.iscas.ac.cn
Http://www.cnsqa.com
[3] Center for Software Engineering, University of Southern California,
Los Angeles CA 90089-0791,USA
yangy@sunset.usc.edu

Abstract. It has become great prominence that business organizations are considering open source software (OSS) when looking for software system solutions. However, building applications based on open source software remains an essential issue for many software developers since the new development process differs from traditional in-house development. In this paper, we present a development process based on our experience on using open source software in application development. The new process emphasizes the early assessment to improve the architecture stability and project manageability by assessing available OSS. A set of measurable assessment criteria is established in assessing OSS candidates and making optimal decisions in the development process. A case study is discussed to show the application of this process.

1 Introduction

It is becoming more popular that business organizations apply open source software (OSS) for their IT needs [1,2,3,4,5]. Some important motivation include cost reduction, technology reuse, organizational and environmental considerations. In many cases, users' needs cannot be fulfilled exactly by the existent OSS. Building applications basing on existent OSS is essential tasks for the developers in such scenarios. Though the approach of building OSS-Based Applications (OBAs) has been widely adopted, there is a lack of well-established development process that accommodates the many distinguished OSS features which can lead a seemingly simple development to disaster, if not handled properly. OBAs' developers often feel confused about what to do next when blindly following certain traditional process models, and results in schedule delay and cost overrun due to tremendous rework over time. Building software applications from OSS requires modifying, improving and integrating existent OSS instead of reinventing wheels. Traditional in-house development cannot work well in such scenarios.

M. Li, B. Boehm, and L.J. Osterweil (Eds.): SPW 2005, LNCS 3840, pp. 122 – 135, 2005.

What is open source software is still defined unclearly [1]. In this paper, we adopt a loose concept of OSS that includes publicly available source code and community-source software that can be freely distributed and modified. We define OBAs as "applications including one or more OSS". Goals of building OBAs are :(1) building or improving an OSS' functionality, performance or qualify to fulfill customers' needs; (2) Integrating multiple OSS to deliver a more complex integrated system solution; (3) including (1) and (2).

In most case, building OBAs is similar to developing COTS-Based Applications (CBA). The essential tasks consist of selecting suitable OSS or COTS product, adjusting them as needed, and integrating them with custom components, OSS, or COTS. However, this is a significant difference between OBAs and CBA development. In developing CBA, developers cannot modify COTS components for their special needs because they rarely have access to the source code of COTS products, therefore are hardly able to modify COTS [6,7]. In building OBAs, developers not only need to integrate existent OSS, but also need to modify existent OSS because their quality are irregular or their functionality cannot fully satisfy desired system functionality. Therefore, developers cannot use the process of developing CBA to build OBAs directly.

Our paper offers a development process of building OBAs that is analogous to the process of developing CBA [7]. The new process emphasizes the early assessment to improve the architecture stability and project manageability by assessing available OSS. A set of measurable assessment criteria is established in assessing OSS candidates and making optimal decisions in the development process.

2 Related Work

COTS-based development is based on the acquisition and integration of commercial off-the-shelf products over in-house development [8]. Acquiring COTS products includes identifying alternative candidate COTS products; assessing candidates; and designing architecture based on selected COTS. Integrating COTS products includes tailoring COTS components and developing glue code.

Assessing candidate COTS components is key factors of system development lifecycle (SDL) [8,9,10]. Some recent research focuses on assessment method [10,11,12]. These methods establish assessment criteria according to requirements specification. Some methods in them assume the requirements specifications pre-existed before doing COTS assessment [11,12], while others propose that establishing assessment criteria and eliciting requirement should be performed concurrently [10]. Tradeoff between desired requirements and available COTS packages is usually the key to success within COTS-Based development [8,13]. Multiple criteria are needed to establish in assessment process [14,11,15]. Which are used to collect data from various COTS candidates for doing such tradeoff analysis [8,15]. Some recent researchers focused on designing CBA development process that includes evaluation and integration [7,16,17,18,19]. They believe an iterative process for developing CBA should meet volatility of COTS products and concurrent evolution of requirements, architecture, and COTS choices. Those process frameworks put acquisition and integration of COTS in the background of SDL and support flexible and concurrent development process.

The following actions are carried out in developing CBA by developers:

- Assessing candidate COTS according to criteria
- Negotiating with customer about requirements and available COTS and making decisions
- Designing architecture on available COTS
- Integrating COTS products
- Merging the process of developing CBA into SDL

Those actions are performed similarly when building OBAs. OBAs approach may accelerate development and cut costs, but the consequences of selecting the wrong component can erase these benefits [3]. There are some researches about how to assess OSS [1,3,5, 20] although they only focus on deploying OBAs in business organizations. Some reported the importance of tradeoff among requirements, available OSS and other project factors such as cost and schedule [3,5]. Further, although developers can design architecture for OBAs, OBAs' architecture relies upon available OSS largely. Integrating OSS into OBAs also like integrating COTS components into CBA except OBAs' developers should improve selected OSS in functionality or quality. Merging the process of building OBAs into SDL is a challenge for OBAs' developers also [21].

Those analogous actions imply that OBAs' developers could use a similar process for developing CBA. Nevertheless, there are some differences between OSS and COTS that require appropriate adjustments to accommodate the special OBA development needs. These adjustments include the following three aspects:

1. Designing high-level architecture should be implemented before assessment.

Developers cannot afford to freeze high-level architecture design in early stage of CBA development because interfaces among selected COTS cannot be fixed and system architecture should be designed concurrently while assessing candidate COTS products. However, in building OBAs, it is possible and reasonable to have high-level architecture selected early in the development since interfaces between OSS can be adjusted, customized, and even modified to meet particular system needs.

2. Particular attributes of OSS should be considered carefully during assessment.

In developing a CBA, developers evaluate candidate COTS packages by a set of established evaluation criteria. Evaluation criteria often include attributes of system functionality and performance, organization constraints, and priorities [22]. These are equally important criteria for assessing different OSS if they are appropriately adjusted. Additionally, in building an OBA, developers have to read and change source code for their needs. Reading and changing source code require considering some other attributes. In section 3, we explain which particular attributes should be considered.

3. The process of integrating OSS is similar to that of integrating COTS, but with greater complicity.

In developing CBA, the integration actions include tailing COTS components and developing glue code to make them work together. In this case, COTS is integrated as black-box reuse. Since COTS is often not free, it always comes with sufficient documentation and vendor support. However, since integrating OSS frequently involves changing OSS source code without any "vendor" support, developer must read through and understand the source code, determine which part should be modified and how to modify, consider the resultant issues possibly cause by code modification. In section 3, we define related actions during integrating OSS.

3 A Development Process for Building OBAs

Firstly, a development process for building OBAs is presented in 3.1, which is an analogous process of developing CBA. Because the special OBAs development needs should be considered, we discuss the important issues about high-level architecture design, special attributes of OSS and OSS integration of the process for meeting those special needs in 3.2.

3.1 Descriptions of the Development Process for Building OBAs

The process for building OBAs is shown in Figure 1.

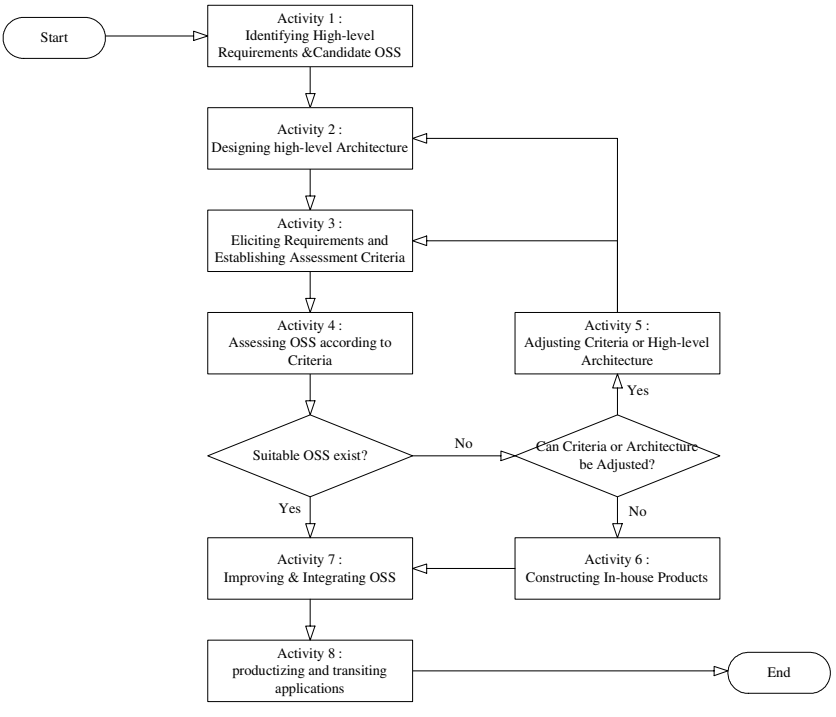

Fig. 1. A Development Process for building OBAs

Activity 1: Identifying High-level requirements & Candidate OSS

Input	· Conceptions of desired system and project restraints*
Step	1.1 Identify high-level requirements 1.2 Identify Candidate OSS according to OSS project profiles and high-level requirements**
Output	· High-level requirements and project restraints which describe boundary of the project · An inventory of candidate OSS that may be used possibly in the applications

**Project restraints include cost, schedule and software & hardware environments.*

***The project profiles [1,20,25] include project' age, application domain, programming language, size, developers' state, number of users, modularity level, documentation level, popularity, status, and vitality. By OSS project profiles and high-level requirements, a group of OSS, which may be used in the application, is sorted out initially.*

Activity 2: Designing High-Level Architecture

Input	· High-level requirements, project restraints and inventory of candidate OSS
Step	2.1 Dividing the desired system into a set of modules. Each module is an OSS and reflects one or more high-level requirements. 2.2 Sorting candidate OSS for every module in high-level architecture
Output	· High-level architecture

Activity 3: Eliciting Requirements & Establishing Assessment Criteria

Input	· High-level architecture
Step	3.1 Eliciting low-level requirements and distributing them into modules' attributes of high-level architecture 3.2 Define total weights of the desired system as distributable benchmark 3.3 Assign weights for each requirement and project restraints*** 3.4 Accumulating requirements weights in each module's attributes as assessment criteria
Output	· a set of OSS assessment criteria for modules in high-level architecture

****Developers must negotiate with users about how much weights are assigned to a requirement.*

Activity 4: Assessing OSS according to Criteria

Input	· Candidate OSS · Assessment criteria
Step	4.1 Assessing candidate OSS according to criteria 4.2 Choosing the suitable OSS for architecture
Output	· Selected OSS

Activity 5: Adjusting Criteria or High-level Architecture

Input	· If assessment criteria or high-level architecture can be adjusted
Step	5.1 Developers return to Activity 3 to negotiate with customers and adjust criteria 5.2 If criteria cannot be adjusted, developers go to Activity 2 to reconsider architecture
Output	· Adjusted assessment criteria and high-level architecture

Activity 6: Constructing In-house Products.

Input	· Requirements cannot be met by selected OSS
Step	6.1 Building in-house products for those requirements
Output	· In-house products****

*****Those in-house products may be OSS or not according to the OSS' license of other part in the applications and organizations' policy.*

Activity 7: Improving & Integrating OSS.

Input	· Selected OSS · In-house Product

Step	7.1 Defining Necessary Interfaces of OSS. 7.2 Analyzing OSS' architecture, source code, and location of modification. 7.3 Improving Code and Unit Testing. 7.4 Integrating OSS and Integration Testing.
Output	· OBAs system

Activity 8: productizing and transiting applications.

Input	· OBAs system
Step	8.1 Doing βtest, documenting user manual 8.2 Collecting users' feedback and fixing reported bugs. 8.3 Transiting products to operation.
Output	· OBAs product

3.2 Important Issues in the Process

Because the special OBAs development needs should be considered, we discuss the important issues about high-level architecture design, special attributes of OSS and OSS integration of the process for meeting those special needs.

3.2.1 Designing High-Level Architecture Before Assessment

In activity 2 of the process, the OBAs were divided into modules by high-level architecture. Modularity improves maintainability of OBAs and development efficiency by adding alternative OSS.

Although OBAs' low-level architecture relies on selected OSS largely, Developers still design high-level architecture early since interfaces between OSS can be adjusted, customized, and even modified to meet particular system needs by changing source code.

Basing on requirements and currently candidate OSS, developers start to design high-level architecture. Developers will divide the desired system into a set of modules. Each module is according to a set of requirements. The degree of success of this activity is decided by developers' knowledge and experience on candidate OSS.

3.2.2 Attributes of OSS

In activity 3 of the process, attributes of OSS are used to build assessment criteria. Attributes of COTS components are suitable for OSS also if they are adjusted slightly. Furthermore, reading and changing source code in building OBAs require considering more other attributes. The stability of OSS' architecture should be considered because modularity and repairing architecture of OSS are important issues while OSS evolving. OSS' architecture instability will prevent developers to reuse OSS [2,23]. Another attribute that should be considered is the development tools of OSS. OSS' low cost has contributed to the widespread adoption of sophisticated tools [21]. Developers should accept the development tools of selected OSS if they want to change source code. Otherwise, transferring the selected OSS to their familiar development circumstance would consume many development resources. The third attribute is understandability and revisability of source code; the quality of source code is an important problem in reusing OSS [1,5,24]. Chaos in source code cuts the productivity of reading and changing source code. Beside all these attributes coming from reading and changing source code, attribute of OSS' legality should be considered too. A risk in reusing OSS is third parties' patents or other intellectual-property rights [25].

Based on COTS' attributes and OSS' particular attributes, our process uses the following set of attributes to assess OSS (Table 1).

It is very important to assign weight appropriately for every attribute based on stakeholders' agreement to facilitate the quantitative evaluation analysis later on. Detailed scales, either qualitative or quantitative, should also be defined for each criteria in order to measure candidate OSS' score with respect to that criteria.

Table 1. Attributes of OSS

Attributes of OSS	Description
Correctness of Functionality	If the functionality of OSS consists with its statement?
Flexibility of Interface	The easy degree that the OSS' interfaces can be used in different environment.
Availability/Robustness	The degree that the OSS operate correctly when using.
Installation/Upgrade Ease	The easy degree that install/upgrate the OSS within a hardware or software environment.
Security	The degree that the OSS prevents unauthorized access and harm
Portability	The easy degree that the OSS can be transferred from one environment to another.
Product Performance	OSS' performance in execution, data capacity, response time, and so on.
Functionality	The degree that the OSS fulfills function needs.
Understandability of Interface	Document quality, simplicity, and testability of development interfaces.
Ease of Use in Interface	The easy degree that developers use the OSS' interface.
Maturity	The length of time that the OSS is available and team of OSS exist.
Version Compatibility of Interface	Compatibility of interface between earlier and later versions.
Inter-Component Compatibility	The easy degree that OSS exchange data with other OSS
Training	The degree that OSS' vendor/developers provide training in using the OSS
Cost of Procurement and Maintenance	The cost of procurement and periodic maintenance
Developers' Support	Response time for problem, capacity in dealing with problem
Architecture Stability	Evolution speed of the OSS' architecture
Usability of Development Tools	The easy degree that in-house developers use the OSS' development tools
Understandability and Revisability of Source Code	Quality of source code and their documents
Legality of Source Code	The legality of OSS' source code

3.2.3 Improving and Integrating OSS

In activity 7 of the process, selected OSS and in-house products are improved and integrated to OBAs. If selected and in-house products can be integrated to fulfill customers' needs without improvement in interfaces or quality, developers integrate them and test the integrated system against defined test cases. Else, developers improve the selected OSS in interfaces or quality with custom coding.

This activity consists of the following steps, which accommodates the differences of OSS integration from COTS integration, as illustrated in Figure 2.

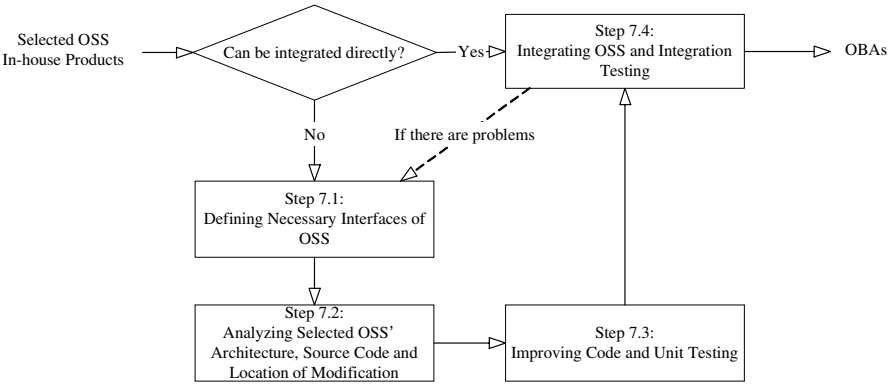

Fig. 2. Actions of Improving & Integrating OSS

Step 7.1: Defining Necessary Interfaces of OSS.
If it is needed to integrate multiple OSS with custom coding, necessary interfaces between multiple OSS should be defined according to high-level architecture.

Step 7.2: Analyzing OSS' Architecture, source code and location of modification.
Developers analyze architecture and source code of selected OSS that needs to be improved, then locate where code will be improved for implementing new interface or enhancing quality of selected OSS.

Step 7.3: Improving Code and Unit Testing.
After locating where code will be improved, developers perform code modification as needed. Then, the units testing are carried out to verify the correct modification.

Step 7.4: Integrating OSS and Integration Testing.
All OSS are integrated and OBAs are tested. If there are problems during integration, developers should enter Step 7.1 to improve the system.

4 Case Study

The case was a project of building an email system using open source software. The process introduced in Section 3 was used to build the system, which kept the project under control and end up with an extremely satisfying OBA. The most important part of the email system, Mail Transfer Agent (MTA), is used as an example to show how to assess the candidate OSS.

4.1 Identifying High-Level Requirements and Candidate OSS

After eliciting requirement from customer, the high-level requirements were identified. High-level requirements include:

1. Mail Transfer Agent (MTA) which supporting SMTP (Simple Mail Transfer Protocol) to receive and relay email.
2. To allow users to receive and send email from their remote computers, the email system supports the standard email protocols -- Interactive Message Access Protocol (IMAP4) and Post Office Protocol (POP3).
3. To allow users to receive and send email by web browser, the email system supports webmail.
4. To improving the security of the email system, the email system provides SMTP authentication and defend spam and virus by content filter.

The project restraints include platform (LINUX), cost (less than 20PM), schedule (less than 3 month), etc.

We searched current email systems and efficient authentication technologies; secure technologies; configuration technologies, etc. Then we identified a set of candidate OSS (Table 2).

4.2 Designing High-Level Architecture

High-level architecture design of the email system is illustrated in Figure 3. The high-level architecture design based on common experience of email system.

The results of sorting candidate OSS for architecture are shown in Table 2.

Table 2. Candidate OSS

Modules in High-level Architecture	Candidate OSS
MTA	Sendmail Postfix Qmail
User Authentication	Syrus SASL Courier
POP3/IMAP	Syrus IMAPD Free pop3 Gavamail Server Mercur POP3 and IMAP Server
Webmail	TWIG webmail SQwebmail IMP webmail Open webmail
Email Database	Unified Mail Queue*****

*****This is not an OSS, but a standard for mail database. We chose the standard for keeping compatibility among modules.*

At the time, there was not an OSS offering content filter. We decided to build a content filter by in-house development.

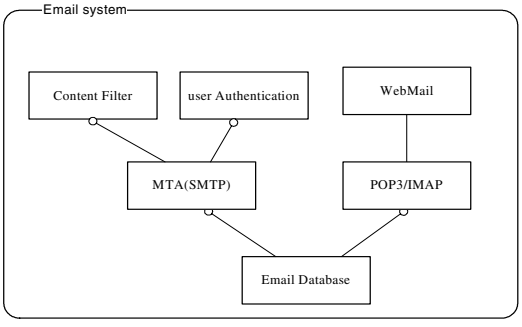

Fig. 3. High-Level Architecture of the Email System

4.3 Eliciting Requirements and Establishing Assessment Criteria

After designing high-level architecture, we elicited requirements and established assessment criteria of the email system. We first elicited low-level requirements and distributed them into modules' attributes. For an example, some functionality requirements distributed into MTA functionality attributes are shown in table 3. Then, we defined 4,000 as total weights for distributable benchmark and assigned weights to requirements basing on discussions between team members and customers. Lastly, each attributes' weights of module in high-level architecture were calculated as assessment criteria. The table 4 was an example for the assessment criteria of MTA; the last column presents corresponding weights assigned.

Table 3. An Example for Weights of Some Functionality Requirements That Were Distributed into MTA Functionality Attribute

No.	Requirements	Weights
1	Email Queue Manager	15
2	Smtpd Parameters Control	15
3	Address Verification	10
4	Maillist Support	10
5	Clients Connect Control	10
6	Relay Rules	10
7	Proxy Support	10
8	Queue Configuration	15
9	Mailbox Limit	10
10	Icp Control	10
11	Email Head Control	10
12	UCE (unsolicited commercial email) Control	15
13	Address Classes	10
14	Access Policy Delegation	10
15	Address Rewriting	10
16	Multi Domain Support	10
17	Virtual Domain Support	10
18	Log Support	10

Table 4. Assessment Criteria of MTA

No.	Attributes	Weights
1	Correctness of Functionality	150
2	Flexibility of Interface	80
3	Availability/Robustness	100
4	Installation/Upgrade Ease	60
5	Security	120
6	Portability	80
7	Product Performance	60
8	Functionality	200
9	Understandability of Interface	80
10	Ease of Use in Interface	60
11	Maturity	80
12	Version Compatibility of Interface	100
13	Inter-component Compatibility	80
14	Training	100
15	Cost of Procurement and Maintenance	80
16	Developers' Support	60
17	Architecture Stability	150
18	Usability of Development Tools	80
19	Legality of Source Code	60
20	Understandability and Revisability of Source Code	100
Total weights		**1880**

Table 5. Result of Assessing Candidate MTA

No.	OSS attributes	Scores of Candidates		
		Sendmail	**Postfix**	**Qmail**
1	Correctness of Functionality	150	150	120
2	Flexibility of Interface	60	80	80
3	Availability/Robustness	90	90	80
4	Installation/Upgrade Ease	30	50	60
5	Security	100	110	80
6	Portability	60	80	70
7	Product Performance	50	60	40
8	Functionality	180	180	160
9	Understandability of Interface	60	80	70
10	Ease of Use in Interface	40	60	50
11	Maturity	80	70	60
12	Version Compatibility of Interface	90	80	80
13	Inter-component Compatibility	70	80	80
14	Training	90	90	60
15	Cost of Procurement and Maintenance	80	80	80
16	Developers' Support	60	60	50
17	Architecture Stability	150	150	120
18	Usability of Development Tools	70	70	60
19	Legality of Source Code	60	60	60
20	Understandability and Revisability of Source Code	90	90	80
	Total	**1650**	**1720**	**1540**

4.4 Assessing Candidate OSS

We assessed the candidate OSS according to the set of assessment criteria. For an example, assessment results of MTA are shown in table 5. According to assessment result, we chose Postfix as MTA of the email system.

As the same process, we chose Cyrus IMAPD as the POP3/IMAP, Cyrus SASL as SMTP user authentication, and Sqwebmail as webmail.

4.5 Improving, Integrating OSS and Productizing

The selected OSS cannot fully meet customers' needs in content filter; therefore, after designing and implementing content filter by in-house development, we defined necessary interfaces between content filter and Postfix. At the same time, Postfix is needed to improve in Chinese language support and users configuration modules. We analyzed the architecture and source of Postfix. After analyzing, we realized those new interfaces between content filter and Postfix by coding functions such as GetHeadOfMail, GetContentOfMail, FilterMailByKeywords, FilterMailByDB, etc while improved the code in Chinese language support and users configuration modules. Then, we combined those new functions and improved code into Postfix. After unit testing and integration testing, we delivered the email system to users.

4.6 Lessons Learned

1. Building OBAs from existent OSS can cut cost, accelerate schedule, and improve quality of products. If we developed the email system from the first line code, it would use more than 200 PM based on experiential estimation. However, by using existent OSS, we cost only 15 PM to finish our product. We got an email system with high performance and security. Now it ran steadily in 2 companies.
2. OBAs' developers must select OSS carefully during building OBAs on existent OSS. Many researches in deploying OSS indicate qualities of existent OSS are irregular and functionality of them contrast clearly with each other. Our experiences in the case validate those findings.
3. During building OBAs, there are many tradeoffs among available OSS, requirements, designing, and other projects factors such as cost and schedule. Those tradeoffs aggravate the complexity of selecting OSS.
4. During building the OBAs, we adapt a strategy that we only considered stable versions of OSS in identifying candidate OSS. The benefits of the strategy are we need not consider those new versions that are validated deficiently during identifying candidate OSS. The disadvantages are we cannot fully utilize the OSS' features in new versions. We believe the strategy will induce a bit of extra works in future maintenance and upgrade of our email system.

5 Conclusions

With many excellent OSS coming forth, it is more popular that deploy OSS in business organizations. Many developers build OBAs on existent OSS instead of build them from beginning. The process of in-house development cannot work well in such scenarios.

In most case, the process that builds OBAs on existent OSS is largely similar with the process of developing CBA. Selecting correct OSS, adjusting them as needed, and integrating them are essential actions during system development lifecycle. OBAs' developers can use an analogous process with developing CBA to build OBAs. However, there is a significant difference between OSS and COTS in right and motivation of modifying source code. Therefore, developers cannot use the process of developing CBA to build OBAs directly.

Our paper offers a development process of building OBAs. The process is analogous to the process of developing CBA in emphasizing the early assessment of OSS. For difference exists between OSS and COTS, we adjusted the process of developing CBA to accommodate OBAs development in early high-level architecture design, attributes of OSS, and OSS integration. A case study is discussed to show the application of this process.

OSS community is a rapidly evolving world that complicates the OBAs development reality. As for future work, we plan to enhance the process by introducing accommodating mechanisms and strategies to address the OSS refreshment during both development phase and maintenance phase. Future experimental studies will also be performed on more projects to validate the applicability of our process and further refine it accordingly.

Acknowledgement. Work reported in this paper is supported by the National Natural Science Foundation of China under grant Nos. 60273026, 60473060 and the Hi-Tech Research and Development Program (863 Program) of China under grant Nos. 2004AA112080, 2005AA113140.

References

1. Wang, H., Wang, C.: Open Source Software Adoption: A Status Report. IEEE Software, Vol. 18, No. 2 (2004) 90-95
2. Fitzgerald, B.: A Critical Look at Open Source. IEEE Computer IEEE Software, Vol. 37, No. 7 (2004) 92-94
3. Norris, J., Kamp, P.: Mission-Critical Development with Open Source Software: Lessons Learned. IEEE Software, Vol. 21, No. 1 (2004) 42-49
4. Dedrick, J., West, J.: An Exploratory Study into Open Source Platform Adoption. Proceedings of the 37th Annual Hawaii International Conference on System Sciences. 2004
5. Madanmohan, TR., De', R.:Open Source Reuse in Commercial Firms. IEEE Software, Vol. 21, No. 6 (2004) 62 -69
6. Barros, M., Werner, C., Travassos, G.: Scenario Oriented Project Management Knowledge Reuse within a Risk Analysis Process. International Conference on Software Engineering and Knowledge Engineering (SEKE'01). (2001)
7. Yang, Y., Port, D., Boehm, B., Buhta, J., Abts, C.: Composable Process Elements for COTS-Based Applications, 5th International Workshop on Economics-Driven Software Engineering Research (EDSER-5). (2003).
8. Alves, C., Finkelstein, A.: Challenges in COTS decision-making: a goal-driven requirements engineering perspective. Proceedings of the 14th international conference on Software engineering and knowledge engineering. (2002) 789-794

9. Dean, J. Vidger, M.: COTS Software Evaluation Techniques. Proceedings of The NATO Information Systems Technology. Symposium on Commercial Off-the-shelf Products in Defence Applications. (2000).
10. Ncube, C. Maiden, N.: PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm. International Workshop on Component-Based Software Engineering. (1999).
11. Kontio, J.: A COTS Selection Method and Experiences of Its Use. Proceedings of the 20th Annual Software Engineering Workshop. (1995).
12. Kunda, D., Brooks, L.: Applying Social-Technical Approach for COTS Selection. Proceedings of the 4th UKAIS Conference. (1999).
13. Carney, D.: COTS Evaluation in the Real World. SEI Interactive, Carnegie Mellon University,December (1998).
14. Alves, C., Castro, J.: CRE: A Systematic Method for COTS Components Selection. XV Brazilian Symposium on Software Engineering (SBES). (2001).
15. Sivzattian, S., Nuseibeh, B.: Linking the Selection of requirements to Market Value: A Portfolio-Based Approach. 7th International Workshop on Requirements Engineering: Foundation for Software Quality. (2001).
16. Basili, V., Boehm, B.: COTS Based System Top 10 List. IEEE Computer, Vol. 34, No. 5 (2001) 91-93.
17. Benguria, G., Garcia, A., Sellier, D., Tay, S.: European COTS Working Group: Analysis of the Common Problems and Current Practices of the European COTS Users. COTS-Based Software Systems (Proceedings, ICCBSS 2002), Springer Verlag, Dean, J., Gravel, A. (eds.). (2002)44-53.
18. Albert, C., Brownsword, L.: Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview. CMU-SEI-2002-TR-009, (2002).
19. Morisio, M., Seaman, C., Parra, A., Basili, V., Kraft, S., Condon, S.: Investigating and Improving a COTS-Based Software Development Process. Proceedings of the 22nd International Conference on Software Engineering (ICSE 22). (2000) 32-41.
20. Capiluppi, A., Lago, P., Morisio, M.: Characteristics of Open Source Projects. Proceedings of the 7th European Conference on Software Maintenance and Reengineering(CSMR 2003). (2003)317-328
21. Spinellis, D., Szyperski, C.: How Is Open Source Affecting Software Development?. IEEE Software (Vol. 21, No. 1): (2004)28-33.
22. Boehm. B., Abts, C., Brown, A.W., Chulani, S., Clark,B., Horowitz, E., Madachy, R., Reifer, D., Steece,B.: Software Cost Estimation with COCOMO II. Prentice Hall. (2000).
23. Tran, J., Godfrey, M., Lee, E., Holt, R.: Architectural Repair of Open Source Software. Proceedings of the 2000 International Workshop on Program Comprehension (IWPC'00). (2000)
24. Godfrey, M., Tu, Q.: Evolution in open source software: A case study. Proceedings of the International Conference on Software Maintenance (ICSM'00). (2000) 131-142
25. Ruffin, M., Ebert, C.: Using Open Source Software in Product Development: A Primer. IEEE Software, Vol. 21, No. 1(2004)82-86