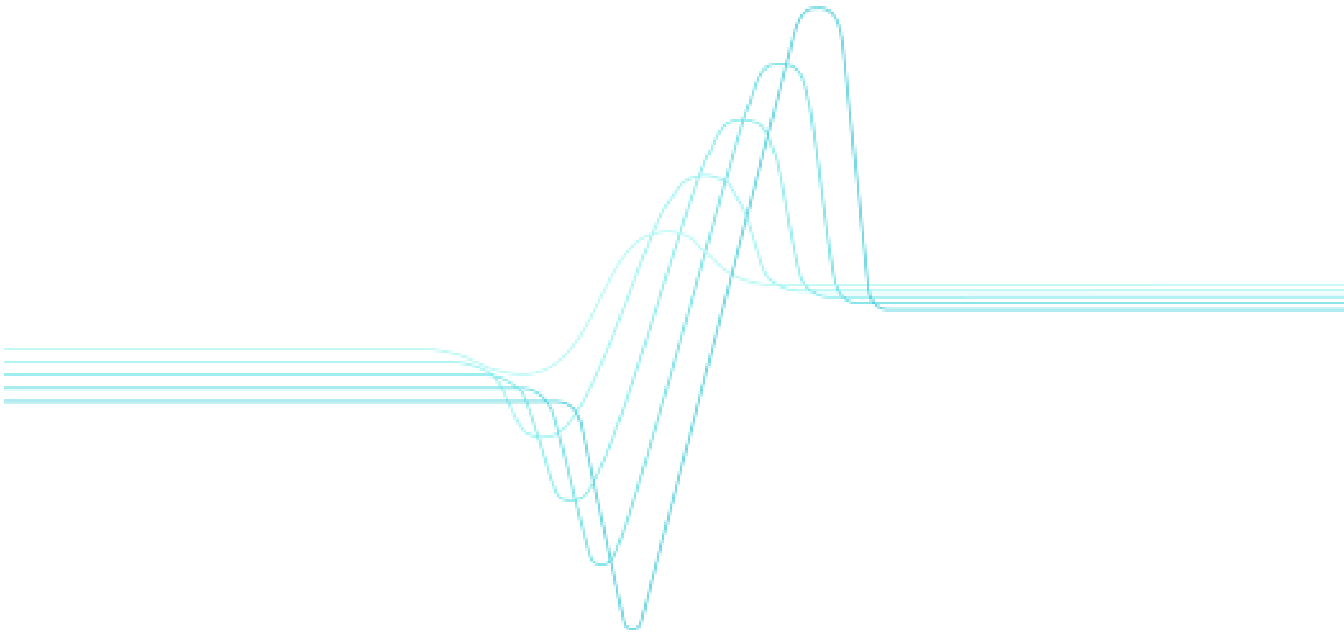
ESPOO 2003 VTT PUBLICATIONS 489

Leena Arhippainen

Use and integration of third-party components in software development



VTT PUBLICATIONS 489

Use and integration of third-party components in software development

Leena Arhippainen

VTT Electronics

ISBN 951–38–6032–9 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6033–7 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Leena Ukskoski

Otamedia Oy, Espoo 2003

Arhippainen, Leena. Use and integration of third-party components in software development. Espoo 2003. VTT Publications 489. 68 p. + app. 16 p.

Keywords component-based software development (CBSD), Commercial-Off-The-Shelf (COTS), glue software, Open Source (OS), third-party components

Abstract

Reuse of software components has been seen as an effective way of decreasing costs and reducing software development cycle-time. Nowadays, reusable components are often acquired from outside organisations. Consequently, the use of third-party software components brings new aspects to software development. One aim of this research was to clarify the activities, roles and possible problems related to the use of third-party software components.

The main purpose of this study was to research the software development process when using third-party components. As a result, the third-party component-based software development process is presented. Furthermore, the use of third-party components is clarified in the particular projects at Nokia Networks. This case study offered an opportunity to research the issues involved in integrating third-party components into a software product. As a result of this study, the glue software development process is described.

Preface

This research was carried out in the Minttu (Software component products of the electronics and telecommunication field) project at VTT Electronics, based on co-operation between Nokia Networks and the University of Oulu. The project started in September 2001 and I had a great opportunity to study third-party components issues during the project's first year. I would gracefully like to thank VTT Electronics and Nokia Networks for giving me the opportunity to study this interesting topic.

This work is made based on my graduate thesis of the use of third-party software components. Thus, I would like to thank Mrs. Päivi Parviainen from VTT Electronics and Professor Veikko Seppänen from the University of Oulu, for their support and guidance during this study. Especially, I wish to express my warmest gratitude to Dr. Pekka Abrahamsson who has encouraged me during this research year.

Oulu, 20.12.2002

Leena Arhippainen

Contents

List of abbreviations

API	Application Programming Interface
CARE	COTS aware requirements engineering
CBSD	Component-Based Software Development
CEP	Component Evaluation Process
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
CSA	Current State Analysis
EJB	Enterprise JavaBeans
EUR	Euro
IEEE	Institute of Electrical and Electronics Engineers
IMN	IP Mobility Networks
I/O	Input/Output
IP	Internet Protocol
J2E	Java 2 Enterprise
NET	Nokia Networks
OCM	Original Software Component Manufacturing

ORB	Object Request Broker
OS	Open Source
OSD	Open Source Definition
OSI	Open Source Initiative
OTSO	Off-The-Shelf-Option
PC	Personal Computer
PORE	Procurement-Oriented Requirements Engineering
Pri2mer	Practical process improvement for embedded real-time software
SEL	Software Engineering Laboratory
USD	United State Dollar
VTT	Technical Research Centre of Finland

1. Introduction

The world of software development has changed a lot in the last few decades. Nowadays, users neither have to develop their own home-brewed modelling language nor do they need to develop programming language to support components. Also, they don't have to develop operating systems or computer platforms to support component management. Currently, software developers can use existing methods, languages and tools for software product development. So, is the software development much easier and faster now? (Heineman & Councill 2001.)

In recent years, new software technologies are developed and standards initiated that enables the use of ready-made components. Component-based software development (CBSD) has become a significant aspect of improving quality and reducing the cost of the software development process. Furthermore, the use of third-party software components in software development has increased rapidly. Deployment of third-party components has been seen as a solution especially when the size and complexity of systems increases. Therefore, the use of the COTS software has become significant in state-of-the-art and state-of-the-practice software and system development. (Heineman & Councill 2001; Morisio et al. 2000; Chung & Cooper 2001; Ochs et al. 2001; Feller & Fitzgerald 2002.)

Third-party component-based software development means exploitation of external components as a part of own product development. A component is one of the parts that make up a system and it may be hardware or software and may be subdivided into other components (IEEE 1990). The purpose of this study is to investigate the use of third-party software components from the integrator's point of view. This means all those aspects which component’s user has to take into account when making system from third-party components. Examination concentrates especially on the use of Commercial Off-The-Shelf (COTS) and Open Source (OS) software components.

1.1 Background

Software reuse has been practised for many years, and it has evolved when industry has matured. The purpose of software reuse is to implement and update software systems using existing software assets. Earlier software reuse was focused on source code and reuse practices were done ad hoc. Nowadays, software reuse practices are well planned and are done systematically. Currently, systematic reuse is recognised as an effective technology for saving money and time and improving software productivity and quality. Reusable assets can be any product of the software life cycle for instance architecture, requirement, design, code, tools, test cases and documentation. (Reifer 1997; Morisio et al. 2002.)

Software reuse is commonly divided to for reuse and with reuse software development. In for reuse development, software components are planned and implemented so that they could be used in other contexts. For reuse development is supported by extending the following activities to the ordinary development process:

- analyse the variability in requirements between different reusers of the components
- analyse the costs and benefits of incorporating these different requirements
- design the components with the appropriate level of generality for all reusers.

Furthermore, for reuse development contains activities for re-engineering existing components, qualifying and classifying the components so that they are ready to be reused. (Karlsson 1995.)

In contrast to for reuse, with reuse software development means building systems using reusable components. With reuse development includes the following activities (Karlsson 1995):

- searching a set of candidate components
- evaluating a set of retrieved candidate components to find the most suitable one
- adapting the selected components to fit the specific requirements.

Use of third-party components is a form of software reuse. The main difference between use of COTS components and in-house reusable assets is the way in which they are managed. Reusable software assets are managed as a shared resource whereas COTS products are managed as a purchased component. Thus, COTS components are more difficult to interchange than reusable software assets. (Reifer 1997.)

Software reuse can be classified in three different approaches (Figure 1). The scope of reuse has different values such as general, domain and product-line reuse. The scope defines the area of potential reuse for component. General reuse means that components can be reused whatever the context of the system. The domain and product-line reuse includes components that can be used in a particular environment. The domain reuse means that components are developed for a specific application domain for both in-house and sale purposes. Currently, the product-line reuse has only been applied in-house.

The reuse target (Figure 1) has divided two values, internal and external approaches. The internal (in-house) approach means that organisation develops for reuse components and provides them internally. The external approach means that organisation develops for reuse components and provides them on the external market. Then organisation has to put more effort for communication with another company. The third approach of reuse classification is granularity which includes fine-grained (small-scale) and coarse-grained (large-scale) categories. The fine-grained components are generic and domain independent such as I/O functions, file and database-access functions and individual object classes. The coarse-grained components are more specific and an example of component can be a user-interface package. (Karlsson 1995.)

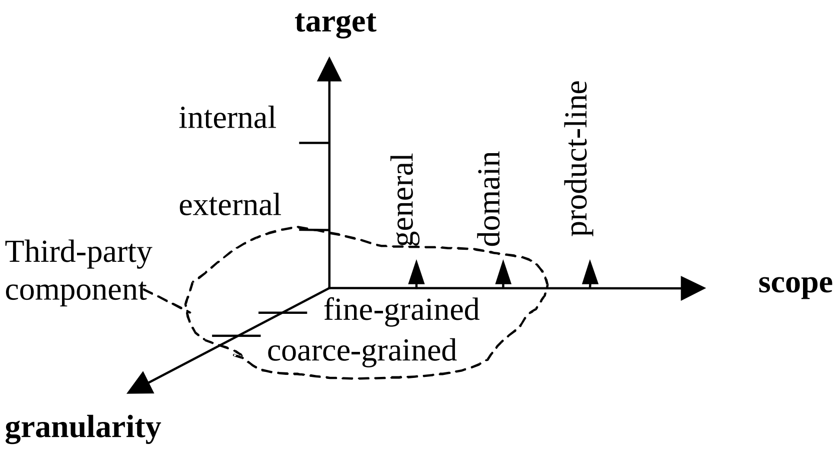


Figure 1. Third-party components in reuse area (adapted from Karlsson 1995).

Third-party components can be general or domain components (Figure 1), which increases components' usability and portability. Several companies have been developed general and domain components for in-house and for sale purposes. Currently, general components are often available free from OS community. Third-party components are provided externally, when communication between component's provider and customer may become complicated. The granularity of COTS and Open Source component could be varying.

Systematic reuse can be organised in the three different ways (Figure 2). The first approach is Project-oriented company where projects are made both for and with reuse. The other way is to organise a Component Production Team, which acts as a kind of subcontractor to each project. This Centralised component production-oriented approach separates the project team from the component production team. The third approach is Domain-oriented company where there is a special component production team for instance such domain as network, database and user-interface. Project teams build applications by acting as integrators of components. (Karlsson 1995.)

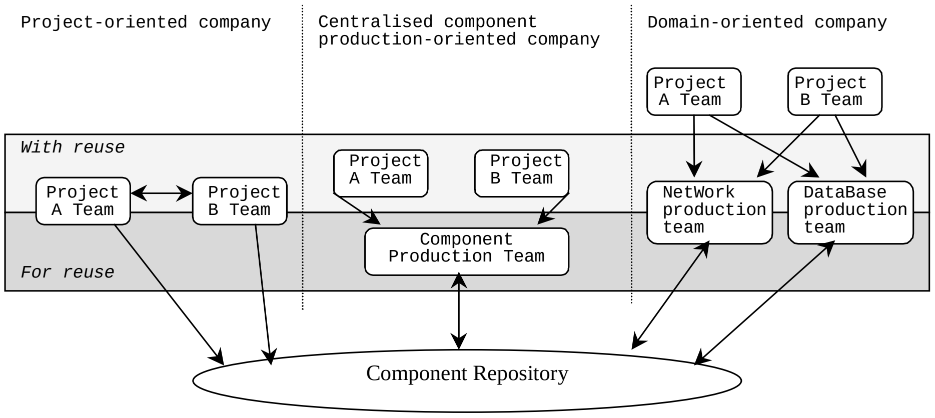


Figure 2. Three different reuse organisation models (adapted from Karlsson 1995).

The use of third-party components can be organised as the Centralised component production-oriented company (Figure 2). An organisation can arrange a third-party component team, which support the use of components, collect experiences and information of them and is responsible for Component Repository. Project's members work in a co-operation with Component Production Team, when they use third-party components.

In the past few years the acquisition of third-party components has been an important research subject. This is of no wonder, because the success of third-party component-based software development mostly depends on well-acquired components. That demands systematic component evaluation and selection processes and comprehensive negotiation processes between the COTS vendor and the OS community. (Meyers & Oberndorf 2001; Morisio et al. 2000.)

The use of COTS and Open Source software components causes changes in the software development process. Thus, not changing the traditional process may be a failure factor. (Morisio et al. 2000.) More emphasis is need to put on requirements specification and integration phases. Therefore, organisations have to improve these processes to ensure that the use of third-party components is beneficial. This study clarifies problems and needs of the use of third-party software components from the integrator's point of view.

Recently, the use of third-party software components has increased. Companies want decrease software development costs and time cycle. Products have to get time to market faster and faster. Nevertheless, a quality of the products has to be ensured. Furthermore, organisations want to focus on their own core competence and the rest of the software can be taken from outside.

Increased use of COTS components have caused such questions to arise as (Morisio et al. 2000):

- How standard development practices should be modified to fit this new development paradigm?
- What methods are now effective?
- How cost saving expected should be quantified?
- What metrics are worth collecting from COTS projects?

1.2 Research problem and methods

The purpose of the research is to investigate how the use of the third-party components affects the software development process and what special characteristics have to be taken account in the COTS process from integrator's point of view. The goal of this research will be achieved by answering to the following research questions:

1. How does the use of the third-party components affects the software development process?

The question can be divided into the following sub-questions:

- What are the differences between the use of COTS and Open Source components?
- What are requirements for third-party components?
- What kind of process is third-party component-based software development according to the literature?

2. How the status of the third-party component-based software process can be analysed and what are the most important process improvement targets in the case organisation?

The question can be divided to the following sub-questions:

- How third-party components are used in the case organisation?
- How components are integrated together?
- What problems can occur in component integration?

The first research question will be solved via theoretical analysis (Figure 3), which based on a current literature of third-party components, software development and software reuse (Järvinen 2001).

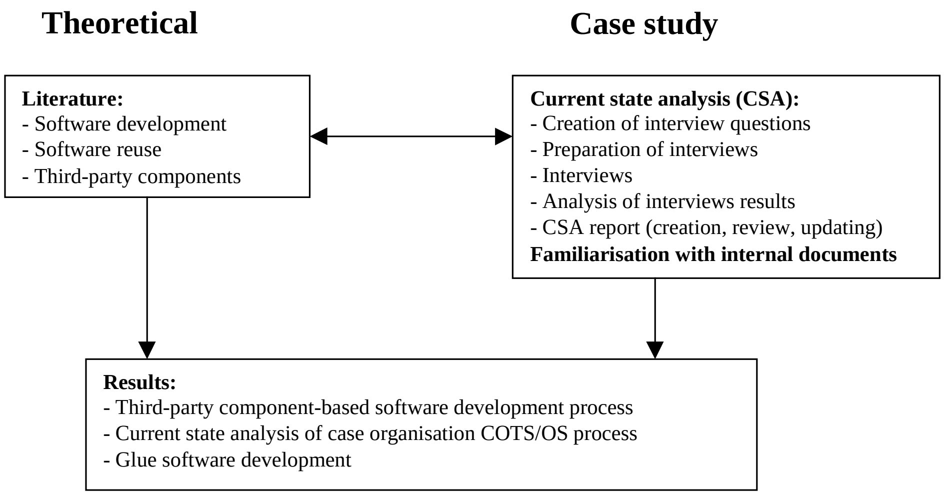

Theoretical
Case study
Literature:
- Software development
- Software reuse
- Third-party components
Current state analysis (CSA):
- Creation of interview questions
- Preparation of interviews
- Interviews
- Analysis of interviews results
- CSA report (creation, review, updating)
Familiarisation with internal documents
Results:
- Third-party component-based software development process
- Current state analysis of case organisation COTS/OS process
- Glue software development

Figure 3. The research methods.

The second research question will be solved via a case study, where third-party component-based development is analysed in several product projects in case organisation. Data is gathered by interviewing employees face to face or using conference calling. Interviewees are selected in a co-operation with contact person from case organisation. A questionnaire (Appendix A) is made based on literature and used as a framework for interviews. In addition, case organisation's internal documents and procedures are studied for current state analysis. The analysis of a qualitative data is rather different than quantitative analysis. Firstly, the results of open interviews have to be recorded and then observation simplified. Thus, material is considered from particular perspective and an interpretation is made based on clues found. A constructive part of this study is developed based on theoretical and empirical data, where new process for glue software development has been created. (Alasuutari 1994; Järvinen 2001.)

1.3 Scope

The research clarifies in general what differences the use of third-party components causes to the traditional software development. The research concentrates on the COTS and Open Source software deployment processes. Original software component manufacturing (OCM) components, which are made in co-operation between the supplier and the customer (Seppänen et al. 2001), are beyond the scope of this study.

Systematic reuse can been described as four concurrent processes (Figure 4). In Create process unit, reusable assets (for reuse) are developed and Reuse process (with reuse) deploys assets in the product development. Support process manages and maintains the reusable asset collection and support communication between Create and Reuse. The purpose of the manage process is to plan, initiate, resource, track and co-ordinate the other processes.

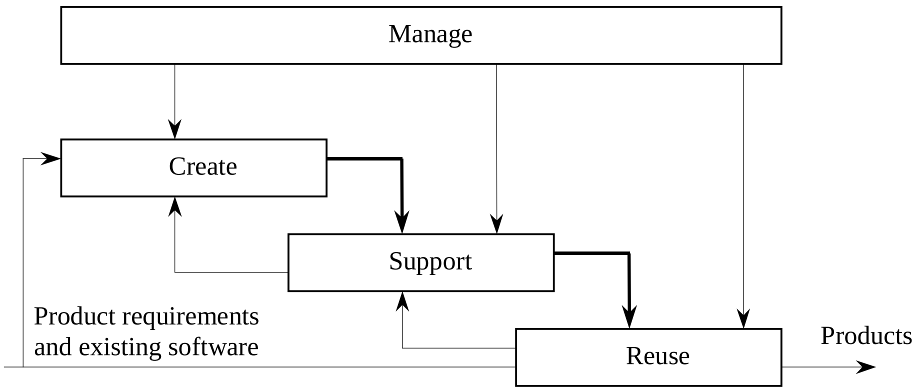


Figure 4. Four concurrent processes of systematic reuse (Jacobson et al. 1997).

Examination considers the use of COTS and Open Source software from Reuse process (with reuse) perspective (Figure 4), where the customer has to integrate third-party components into their own product software. These third-party components come outside of own organisation. Instead, for reuse software development of COTS and OS components is out the scope of this research.

1.4 Structure

The publication is divided into four main parts. The purpose of the structure (Figure 5) is to introduce the frame of reference for the study and general process of third-party component-based software development. In the case study, the current COTS and OS component-based software development process is presented.

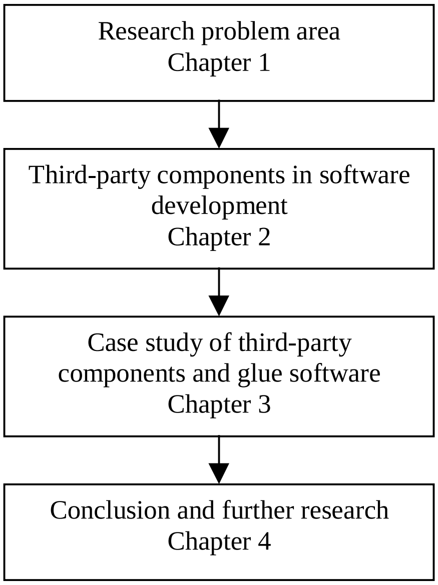

Research problem area
Chapter 1
Third-party components in software development
Chapter 2
Case study of third-party components and glue software
Chapter 3
Conclusion and further research
Chapter 4

Figure 5. The structure.

Chapter 1 introduces this topic to the work and presents the scope of the research area.

Chapter 2 presents the theoretical description of third-party components-based software development and differences to the traditional software development. Examination concentrates on the two different kinds of components; Commercial-Off-The-Shelf (COTS) and public available Open Source (OS) software components.

Chapter 3 gives a short description of the case organisation Nokia Networks IP Mobility Networks (IMN) and presents currents state analysis process and results of the use third-party components in software development. Furthermore, Chapter 3 describes glue software development in third-party component-based software development.

Chapter 4 builds up conclusions of the research answering the research problems. Possible future work is also discussed.

2. Third-party component-based software development

The objective of this chapter is to introduce and analyse the key concepts of third-party component-based software development. Examination focuses on such third-party components as COTS components and Open Source components. Currently, they are the most common non in-house software. Firstly, third-party components are introduced and benefits and risks are discussed. Secondly, the use of third-party software components is compared to traditional software development. As a conclusion, a process model for third-party component-based software development is presented.

In software development the developer creates application of the in-house components. In the component-based software development a developer role changes to a customer, who purchases the COTS product and only integrates them to the own product software. The COTS product is like a black box, when the customer cannot see the functions of the product. The customer sees a view of the product's interfaces and what it produces. There are some differences (Table 1) between developer (vendor) and customer (integrator). (Meyers & Oberndorf 2001.)

Table 1. Developer and Customer approaches (Meyers & Oberndorf 2001).

Developer	**Customer**
1. Identify requirements. 2. Define unique interfaces. 3. Develop custom implementations. 4. Integrate custom implementations. 5. Use and support system of developed implementations.	1. Identify requirements. 2. Adopt standard interfaces based on market research. 3. Procure implementations based on standards. 4. Integrate procured implementations. 5. Use and support system of procured implementations.

In software development an organisation develops a system for a customer. In a COTS-based system, the organisation should build up the system from small pieces that are built by others. Thus, an importance of a component identification, purchase and integration will be emphasised. (Meyers & Oberndorf 2001.)

In addition to commercial components, component-based software development can utilise Open Source software, which is a publicly available source code and it is freely distributable (Meyers & Oberndorf 2001; Schmidt & Porter 2001). Open Source software includes a provision that modifications that are made are shared with the community (Wang & Wang 2001). Open doesn't mean that Open Source is an (open) standard, but when a piece of OS software becomes very popular it may be regarded eventually as a standard implementation (White et al. 2002).

The following conditions are outlined to the Open Source Definition (OSD) (Feller & Fitzgerald 2000; OSI 2002):

- The software has to be redistributable (Open Source software release is given with full rights to reproduce and redistribute software).
- The source code has to be available to the user (product should include source code, or it has to be available by free, public Internet download).
- Open Source software has to be modifiable and derivative works permitted.
- The license must not discriminate against any person, group or fields of endeavor.
- The license has to apply to all parties to whom the software is distributed.
- The license can't restrict aggregations of software.

Nowadays, development of Open Source software is very popular and estimation of the amount of developers is 750 000. In practice, the amount of OS developers is difficult to estimate, and that is in fact, one of the main failings in the OS research domain. Likewise, the number of OS user is more complex to estimate. (Feller & Fitzgerald 2002).

Feller & Fitzgerald (2002) identify four main stakeholders involved in Open Source software, which are developer communities, user communities, commercial and non-commercial organisations. Usually, OS software users are also developers and vice versa of course. Likewise, commercial and non-commercial organisations use and develop products as well. Instead of the amount of developers, the demographic profiles of Open Source software users are small.

2.1 Benefits and risks

In this section, benefits and risks of use of third-party components are described. COTS components and Open Source software seem to bring several advantages, but unfortunately risks as well. Scott Hissam express well this situation: "users can get the latest and greatest and the fastest fixes. But on the other hand, the disadvantage is that users have to get the latest and greatest and the fastest fixes". (Weinstock et al. 2000.)

General opinions of the deployment of Open Source components are often negative for instance senior managers usually think that what is free is unreliable. Support of OS components is also seen as a problematic aspect, because organisations tend to need a commercial company to sue if something goes wrong. Also, the lack of control of OS developers may be a negative aspect in OS deployment. Furthermore, motivation of OS developers has been based less on money than recognition. That makes them unpredictable for instance they may suddenly stop current work and start to develop new more exciting Open Source software. Also, they may not see commercial issues as so important for example backward compatibility and interoperability needs. However, all these negative views have little actual substance, although will affect the success of Open Source utilisation. Thus, deployment of Open Sources requires more publicity so that respected organisations can overcome doubts. (Peeling & Satchell 2001.)

2.1.1 Business

The use of COTS and Open Source components decreases software development cycle-time and costs. In addition, Open Source software reduces quality assurance costs. Open Source software decreases costs of software development because OS user will avoid software licensing costs. Absence of negotiations with license managers will save time for other tasks of OS-based software development. Users also avoid costly management of licenses which includes several activities. Firstly, users don't face the legal costs and risks of checking and signing licenses. Nor do they have to ensure that license conditions are adhered to, and that all relevant licenses have been purchased and updated. They also don't have to buy future upgrades. (Meyers & Oberndorf 2001; Schmidt & Porter 2001; Peeling & Satchell 2001.)

Open Source software is available free, but it still has several different types of license and each of these imposes a different set of restrictions. Therefore, this could impede project capabilities such as internal reuse, proprietary custom extension and resale. Although Open Source is a freely available source code, it doesn't mean that the deployment of Open Source software is totally without costs. From a project management standpoint, indirect costs including development, technical support and maintenance effort have to be considered. Furthermore, component integrators' existing expertise in OS technologies (e.g. Unix, Perl) is critical to be clarified. This is important because a lack of familiarity with OS technologies would require developers' retraining and the adoption of a new software and development philosophy. Consequently this may mean significant costs and resources. (Wang & Wang 2001.)

The use of third-party components elicits new risks to the integrator's software development projects. From the project management perspective, schedule estimation is difficult because of the new third-party component tasks and vendor dependence. The other risk is that a purchased component may turn out incapable. A component may not perform as well as expected. This kind of problem can be avoided by proper evaluation, benchmarking and communicating with other component's users before purchase. (Morisio et al. 2000; Reifer 1997.)

In component-based software development, maintenance and enhancement differ from conventional software development. Thus, the integrator may be relying on sources outside its own organisation. Drivers for maintenance enhancement to the system could be acquired from users or the marketplace. The integrator has to react to outside forces for instance if standards or commercial components change, he has to decide whether to change the whole component or adopt the changes. Likewise, a supplier may change business plans or even go out of business. Thus, these issues should be taken into account in the risk analysis and strategy for the use of third-party components. (Meyers & Oberndorf 2001.)

2.1.2 Software quality

An advantage of Open Source is a robust source code, which encourages a large market of early adopters. These users help to debug the software, and thus make it highly robust at an early phase of its development. In addition, the upgrade cycle for Open Source is faster than the cycle of commercial software, which may take typically 12–18 months. Also, Open Source upgrades can often be much more responsive to user requirements than commercial software. COTS components quality issues could be hard to ensure and components could have side effects to the integrator's final product. (Peeling & Satchell 2001; Morisio et al. 2000.)

When use third-party components portability will be increased, which means that component can be transferred easily from one hardware or software environment to another. Likewise, interoperability between components will be grown, which means that components are more capable to exchange information between them and to use the information that has been exchanged. The use of third-party components increases productivity as well and it enables companies to focus on their own core competencies. (IEEE 1990; Meyers & Oberndorf 2001; Ochs et al. 2001.)

Usually, Open Source software is written portable and thus is available to different kinds of platforms. OS components offer opportunity to a wider choice of platforms and potentially easier upgrade to new technology. (Peeling & Satchell 2001.) Infrastructure software, such as Osage (http://osage.sourceforge.net/), is usually directly usable by a wide range of companies because it addresses a common technical need (Ambler 2000).

Open Source is not dependent on related products. Usually, customers perceive that the application works best with other products from the same manufacturer. Open Source software gives the possibility to acquire other products avoiding being locked into having components from a particular manufacturer. (Peeling & Satchell 2001.)

2.1.3 Maintenance

Maintenance of commercial and Open Source software is somewhat different. Commercial software might become unmaintainable if its developer leaves the organisation. Instead, the Open Source software life cycle is much longer as it is the property of community. One reason is that Open Source software is often better structured and documented than commercial. (Peeling & Satchell 2001.)

When acquiring COTS components the maintenance issues have to be negotiated with a vendor. The COTS vendor could be economically unstable and go out of this business area. As a precaution, the integrator better make sure that it has access to the course code if the vendor goes out of business or stops supporting the product. In addition, a COTS vendor could be inflexible in implementing an integrator's enhancements needs. (Morisio et al. 2000; Och et al. 2001; Reifer 1997.)

One Open Source disadvantage to users is that there is an organisation missing who would be interested in supporting software. The other reason for the lack of Open Source deployment is that ease-of-use features often arrive later than for commercial components. Therefore, the use of OS components may feel more complex and hard for users. (Peeling & Satchell 2001.)

In deployment of Open Source components, there is no single proprietary source of software support and upgrades. Thus, the support of OS components is different than commercial products. Firstly, there is no risk that the one organisation that supports the software shall stop supporting it or go out of business. Secondly, this enables market competition between companies, which offer support services. Support of Open Source components is often based on an OS community where code fixes could be done rapidly. Furthermore, the source code is available to component integrators making system integration much easier and cheaper. (Peeling & Satchell 2001.)

Naturally, access to source code is one of the main advantages of Open Source product (Peeling & Satchell 2001). Thus, users can fix bugs and make needed changes directly. Of course, the benefit of this feature depends on the size of component. To small and middle size components the bug fixes and changes are more manageable than to large systems for instance operating systems.

From the project management perspective, long-term maintainability issues have to be considered when using third-party components. In Open Source component-based development, standard interfaces, availability of support and development status all affect the long-term manageability of an organisation's projects. (Wang & Wang 2001.)

2.1.4 Summary of benefits and risks

The use of third-party components brings several benefits to the software development (Table 2). On the other hand, different risks have to be considered as well (Table 3).

Table 2. Benefits of the use of third-party component.

Benefits	**COTS**	**OS**
Decreases software development time cycle.	X	X
Decreases software development costs.	X	X
Decreases software quality assurance costs.		X
Increases portability.	X	X
Increases interoperability.	X	X
Increases productivity.	X	X
Is easy to use.	X	
Offers a robust source code.		X
Upgrade cycle is fast.		X
Offers an access to source code.		X

Table 3. Risks of the use of third-party component.

Risks	COTS	OS
Requires software licensing costs and negotiations.	X	
Require familiarisation with new technologies.		X
Schedule estimation is difficult.	X	X
A purchased component may turn out incapable.	X	
A vendor may change business plans or go out of business.	X	
The vendor may be inflexible to upgrade component	X	
Increases indirect costs.		X
There isn't an organisation who would support acquired software.		X
Can cause side effects to the integrator's final product.	X	X

2.2 Different kinds of COTS processes

The use of the COTS component brings essential differences in comparison to the traditional software development. New activities are product evaluations and familiarisation and vendor interaction, which means technical, administrative and commercial issues. In COTS-based development, the main differences appear in the requirements definition, COTS selection, high-level design, integration and testing. (Morisio et al. 2000.)

2.2.1 New roles

Morisio et al. (2000) present new roles and responsibilities to COTS-based software development projects. The COTS team role is at the organisational level and depending on the size of the organisation, COTS team could be a group or a person. The team works like a repository of history, knowledge and skills about COTS activities for instance component evaluation and selection. Evaluations tend to be constricted, if they are done by individual project, because it focuses only on packages, which are familiar to project members. In addition, evaluations require techniques and skills that projects haven't got. (Morisio et al. 2000.)

The COTS team is also responsible to keep a history record of COTS evaluations. This means a catalogue of COTS known, provided functions, information of vendor, cost, location and project using it. The catalogue has to develop as maintainable, because market places change rapidly. Furthermore, the COTS team should act as a contact point between projects so that their members can get and change experiences of previous COTS projects. (Morisio et al. 2000.)

COTS projects need administrative, managerial and commercial support for the component acquisition as well. The aim of the COTS team is to define a repeatable process for vendor interaction. The other role is to manage the interface with the vendor, which is responsible at the project level. Projects should design a single point of contact with the component vendor. This role follows a predefined and documented process by the COTS team and records interactions with the vendor. Likewise, the role includes the creation of a partnership with the vendor. The COTS team supports this role for instance in non-technical and acquisition issues. (Morisio et al. 2000.)

2.2.2 COTS process models

The Software Engineering Laboratory (SEL) has researched the COTS-based process for several years. Their process (Figure 6) for COTS-projects includes four main phases: Requirements, Design, Coding and Integration. COTS specific activities are described above the horizontal line and traditional software development activities below that line. Requirements phase includes component identification, evaluation and selection. In Design phase integration requirements and effort are identified. Coding phase includes implementing glue software and interfaces. In Integration phase, all components are integrated and tested. (Morisio et al. 2000.)

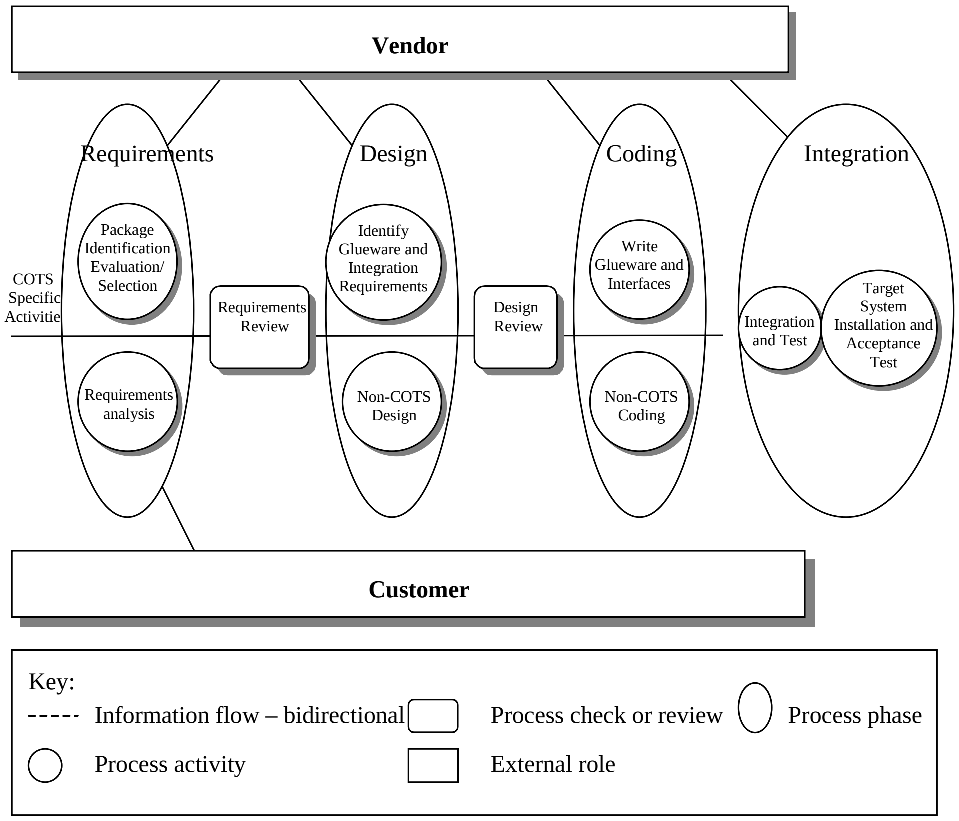


Figure 6. COTS process (Morisio et al. 2000).

According to Meyers and Oberndorf (2001) the COTS process includes such elements as Requirements, Reference models, Components and interfaces, Standards, Implementations, Integration and testing and Deployment and support (Figure 7). In the Standards element, market research is reviewed and standards are evaluated and selected against predefined criteria. It is also important to ensure co-operation between standards groups and user groups, because it helps to get the latest information about the standard relevant to the system. (Meyers & Oberndorf 2001.)

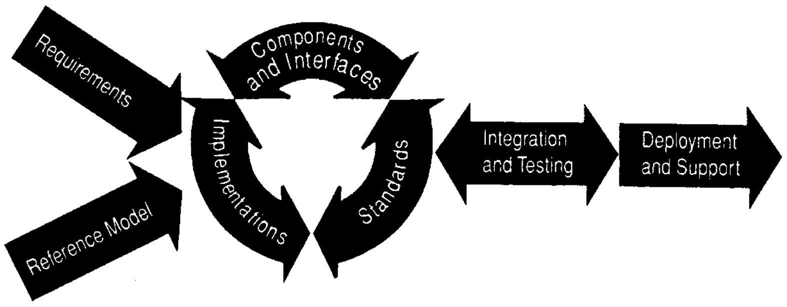


Figure 7. COTS-based software development process (Meyers & Oberndorf 2001).

Ochs et al. (2001) present COTS process parallel to the traditional development. They have divided the COTS-based development process into these four phases: COTS Assessment and Selection, COTS Tailoring, COTS Integration (glue code) and System maintenance (Figure 8).

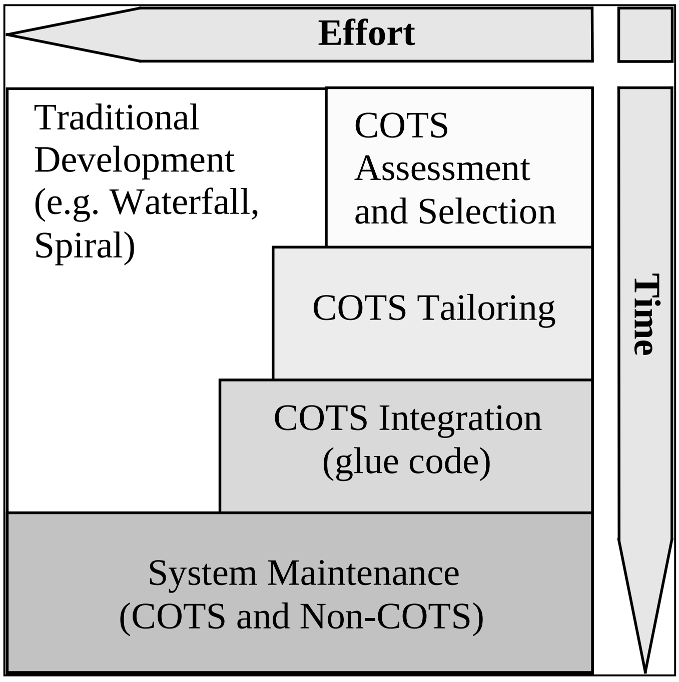

Effort
Time
Traditional Development (e.g. Waterfall, Spiral)
COTS Assessment and Selection
COTS Tailoring
COTS Integration (glue code)
System Maintenance (COTS and Non-COTS)

Figure 8. Phases of COTS-based and traditional software development (Ochs et al. 2001).

Based on the presented COTS-based software development processes, the particular phases can be seen relating to third-party component-based software development. The deployment of Open Source component can follow the same process than COTS components. As a conclusion, third-party component-based software development includes the following phases:

- Requirements definition
- Component searching, evaluation and selection
- Design of integration
- Integration
- Testing and maintenance.

2.3 Requirements definition

According to Morisio et al. (2000) the Requirement phase (Figure 6) includes such activities as Make versus buy decision I, Requirements definition, COTS identification and selection, COTS familiarisation, Feasibility study and Make versus buy decision II. Whether or not to use the COTS in a project is a key decision, which influences all subsequent phases and the success of the project as well. Thus, the decision should consider both the technical and non-technical aspects. The purpose of requirements definition is to guide the identification of the COTS component. COTS components are identified and evaluated using reviews, vendor documentation, suggestions and peer experiences. The aim of this activity is to decrease the amount of candidates to be deeply evaluated, because the amount of deep evaluations has to be kept low for cost and schedule issues. COTS familiarisation means that selected COTS are actually used. This will help to better understand the functionalities of the component. (Morisio et al. 2000.)

In the Feasibility study, a product is described at an appropriate level to make the second Make vs. buy decision. It should include a complete requirements definition, a high level architecture, an effort estimation and a risk assessment model. The high-level architecture description is essential because it allows the team to outline dependencies among the COTS component, incompatibilities and integration effort. Incompatibilities, vendor dependability, and COTS dependability are inputs for the risk model and integration effort is input for the effort estimation model. The feasibility study should be repeated for all product variants for instance product without COTS and products with COTS A and with COTS B. In Make vs. buy decision II, attributes are considered in more detailed level. The result of this activity is that the product variant will be developed including requirements and COTS selected. Each product variant is a different trade-off among requirements satisfied, risks accepted and cost. The purpose of Make vs. buy decision is to analyse in detail these trade-offs. The Requirements phase ends with a requirements review. This review is essential because in COTS-based development several decisions (selection, requirements) are made early, with limited information available. (Morisio et al. 2000.)

According to Meyers and Oberndorf (2001) Requirements (Figure 7) definition includes three key tasks: Current system baselining, specification of system requirements and requirements ranking by priority. The purpose of baselining is to determine the openness of each component and interface in the system. The aim of the interface is to connect two or more components together in order to pass information from one to the other (IEEE 1990). Baselining will help to identify possibilities to insert open system technologies in the system.

COTS-based software development causes changes to the traditional requirements engineering process. In the traditional software development process the requirements have been collected from stakeholders and then described as a system that is going to be useful. These requirements are called native requirements. When developing a system with COTS components, requirements are called foreign requirements. Thus, the requirements for a COTS-based system need to consist of a mix of foreign and native requirements. In order to deliver requirements to the customer using COTS components, foreign components are expected to be much more than native components. (Chung et al. 2001; Chung & Cooper 2001.)

In this approach, the requirements for the system need to be developed by considering the capabilities of available COTS products (Chung et al. 2001). Thus, requirements engineering and elicitation of features of COTS products are conducted in parallel and COTS software is selected that almost exactly fits the requirements (Ochs et al. 2001). However, COTS-based software development introduces new problems for requirements engineers, including when to acquire new customer requirements and when to reduce the number of candidate products. Requirements engineering acquires models and validates functional and non-functional requirements. System design specifies the functional and physical architectures as well as the hardware and software design to meet these requirements. (Kontio et al. 1996.)

2.3.1 Requirements for COTS components

The reliability, functionality and quality of third-party components have to be evaluated before integration into a system. COTS software has to operate under a guaranteed set of reliability requirements – reliability related to individual components and the COTS system as whole. Several highly reliable components linked together may not guarantee a reliable system. Also, after component integration, a lack of immediate system failure does not necessarily imply that the component will provide long-term reliability. (Hardy 2000.)

A COTS component should be easy to use, flexible enough and include appropriate functionality. In general, the most significant goal of a component is to ensure that it will do everything required. Usually, COTS components include too much functionality, which have to be covered by the integrator. However, when evaluating functional requirements, the integrator has to know what components do without knowing how. That requires communication with the COTS vendor, so that the integrator could better know beforehand if a certain component can be used in a particular context. (Hardy 2000.)

When a component is integrated into the system, the resulting quality of the operation has to be assessed – for example, precision, time of processing and average response. The quality of a component can be defined subjectively by a manager or integrator without knowing the underlying behaviour. Although quality is difficult to measure, it has to be considered during the design of a COTS component-based system. (Hardy 2000.)

2.3.2 Requirements for Open Source

Open Source components have to be evaluated according to technical and non-technical evaluation criteria. Technical criteria should include architecture, development and operational issues. When an organisation adopts Open Source software candidates for a commercial purpose, it has to ensure that technical support is available for the commercial product. This could include training, documentation, real-time support, bug fixes and professional consulting. (Wang & Wang 2001.)

A software product that is developed using OS components has to be upgradeable, so that future functionality is easier to add. The purpose is that OS-based software development is continuous and enables future upgrade capabilities. Also, backward compatibility is significant so that future versions of the OS components require minimal recoding and reintegration with the existing system. (Wang & Wang 2001.)

In a larger and more complex context, all components have to be employed using a particular standard. Open Source component development has to adhere to the various open standards and future revisions. Thus integration of OS components will be easier. (Wang & Wang 2001.)

An Open Source component must be flexible enough that it could be customised or integrated into widely different technical environments. The component may also need to be extended to include additional functionality. Furthermore, the Open Source component dependency on operating system, development tools and other components has to be considered. It is also important to be aware of whether or not an OS component can be integrated with commercial components and a software product. The assumption is that all software must be able to integrate together. In addition, high reliability is one requirement for an OS component. Thus adaptable OS software must be widely used and its performance evaluated and reviewed. (Wang & Wang 2001.)

2.3.3 A technique for requirements definition

Chung et al. (2001) presents a technique for COTS-based requirement engineering. The COTS aware requirements engineering (CARE) technique is an initial process, which is based on the scenario used to develop system requirements with COTS. An author has searched for available COTS components. The author takes care that components match the functional and non-functional requirements of the system being developed. If more than one suitable component is available, the author selects the appropriate components. The advantages of the CARE methodology appear near of the software development lifecycle, in the detailed design, implementation and unit testing phases. The COTS aware requirements engineering technique may use more time in the requirements analysis phase. (Chung et al. 2001.)

2.4 Component searching, evaluation and selection

According to Ochs et al. (2001) in the COTS-based software development the most crucial phase is COTS Assessment and Selection (Figure 8). In this phase, long-term decisions on which COTS will be used in the software system have to be made. The importance selecting the optimal COTS product will be emphasised because selection affects the next phases. If the use of a non-optimal COTS product is detected in the later phases of the system development the consequences can be extremely expensive. Thus, repeatable and systematic methods to assess and select COTS software are important issues in COTS-based software development. (Ochs et al. 2001.)

According to Meyers and Oberndorf (2001) in the Components and interfaces phase (Figure 7), components are ranked by priority because the amount of components has to be decided for instance costs reasons. Also, architectural approaches have to be considered from various perspectives such as: organisation's existing architecture, general architecture styles (e.g. client-server) and domain-specific architectures. In this phase, components of the current system and similar systems are identified. This enables the possibility of sharing knowledge of components. The other key tasks of phase are market research, prototyping critical aspects and architecture documentation. (Meyers & Oberndorf 2001.)

Meyers and Oberndorf (2001) called non in-house component an implementation (Figure 7). Implementations have to evaluate and select by looking for such aspects as conformance with selected standards, availability on selected platforms and vendor reputation. Criteria also include analysis of costs, performance, reliability, stability and supportability and licensing information. In addition, implementations have to test (conformance test) that they fulfil the requirements. (Meyers & Oberndorf 2001.)

Open Source software should be evaluated against traditional software selection test in order to ensure it meets the necessary requirements. When using Open Source component as the basis for a commercial product it is important to outline a strategy for its selection, use, upgrade and support. Open Source should be treated as if you wrote it. There is no other party that will be held responsible if it does not perform as expected or if something goes wrong. (White et al. 2002.)

2.4.1 Methods for COTS evaluation and selection

In COTS-based development the evaluation and selection activities are crucial for the process success. For instance, the use of non-optimal COTS software in the development can become extremely costly for the organisation. Therefore, the organisation has to make long-term decisions on which COTS will be used in a software system. (Ochs et al. 2001.)

PORE

Ncube & Maiden (1999) present an iterative method, Procurement-Oriented Requirements Engineering (PORE) that supports requirements engineering and product evaluation and selection processes. PORE integrates different methods, techniques and tools for requirements acquisition and product identification, evaluation and selection. The PORE approach has three main components. The first is the process model that identifies four essential goals that should be achieved and prescribes generic processes to achieve each of these goals. The second part is the method box, which includes methods, techniques and tools that are available to help undertake and achieve each of the processes. The last is the product model that provides semantics and syntax for modelling software products. PORE focuses on requirement engineering and COTS acquisition at the same time. (Ncube & Maiden 1999; Chung et al. 2001.)

<u>OTSO</u>

Kontio et al. (1995) have developed Off-The-Shelf-Option (OTSO) method for evaluating and selecting reusable components for software development. OTSO provides specific techniques for defining the evaluation criteria, comparing the costs and benefits and consolidating the evaluation result for decision making. This multi-phase approach to COTS selection begins during requirements elicitation (Dean 2000). The OTSO method's (Figure 9) phases are search, screening, evaluation and analysis. The purpose is to identify and find all potential component candidates using existing guidelines and criteria. Selected candidates are evaluated according to the evaluation criteria (e.g. cost estimate, qualitative tests) and evaluation results are analysed which leads to the final selection of COTS components. (Kontio et al. 1995.)

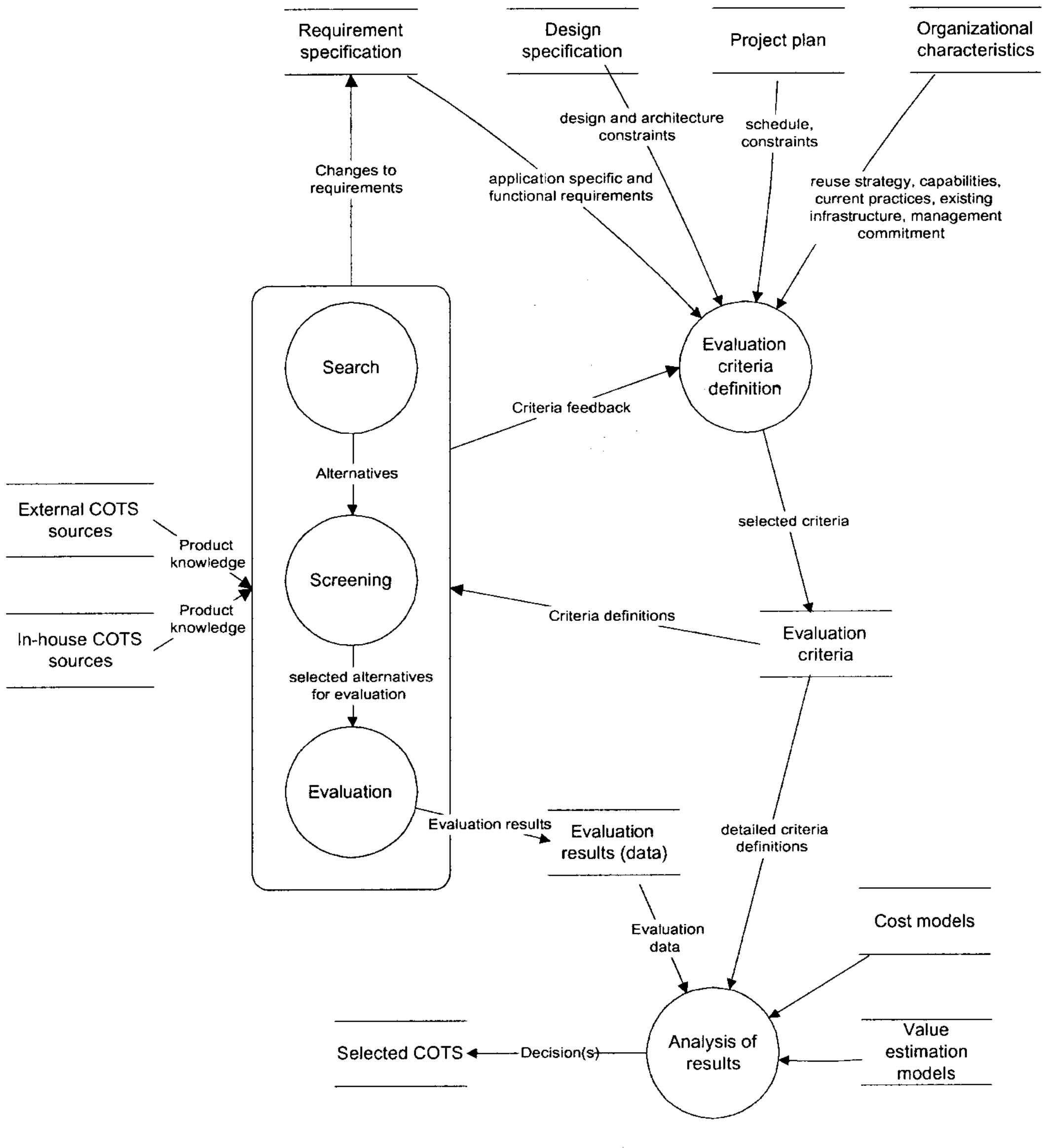


Figure 9. OTSO method (Kontio et al. 1995).

<u>CEP</u>

Other method for COTS selection the Component Evaluation process (CEP), which is an advancement of the OTSO method. The enhancements are (Ochs et al. 2001):

- A planning facility for evaluating the available components alternatives
- An activity for developing, executing, analysing evaluation scenarios
- The use of an additional mathematical method for evaluation data analysis.

Taweel and Brereton method

The Taweel and Brereton method (1999) is an eight-phased product evaluation process that can be used by software consumers or developers to evaluate and select products or off-the-shelf components. In this method the requirements that are not met by any COTS software candidate are systematically dropped. The maximally covered set of requirements fits the COTS software selected (Ochs et al. 2001). Taweel and Brereton (1999) presents some advantages of the method:

- It needs no requirements list.
- Much of the information generated can be reused.
- It provides a mechanism to reassess the evaluation.
- It facilitates the inclusion of additional products or components at any stage during the evaluation process.

2.5 Design of integration

The design phase of the COTS process (Figure 6) includes high-level design activities for instance definition of COTS integration. Design effort will be emphasised when several COTS components are involved and each of them may have different architectural styles and constraints. In COTS design review essential issues are decisions of architecture, COTS integration and glue software. The Make vs. buy decision is reviewed as well because now more information is available about COTS components and integration issues. The Review includes re-estimations of risks, integration effort and costs. The decision could lead to situation where the selected COTS component is impossible to use because of integration or cost reason for example. Thus, one would have to return back to the requirements phase. (Morisio et al. 2000.)

2.6 Integration

Integration is a process of combining software components, hardware components or both into an overall system (IEEE 1990). Software component integration is the assembly of components using a component infrastructure that provides the binding to form a system from the disparate components without modifying the components' source code. The component could be acquired from different sources, such as from a COTS vendor, an OS community or ones’s own organisation repository. Two commercial components could be difficult to integrate together, thus glue software may have to be created. (Meyers & Oberndorf 2001.)

In third-party component-based software development, the integration of products is different and more difficult than in traditional development because the integrator may have less insight into commercial products. Sometimes, a vendor may not even be willing to give products for integration with others because it may want to retain market dominance. (Meyers & Oberndorf 2001.)

Communication, functionality, interoperability and security issues for the overall system have to be concerned when integrating COTS components. In order to be absolutely interchangeable, components have to be aware of their surroundings at run-time, but they not have to require knowledge of the surrounding world until a task is executed. Interface of a COTS component has to be described so that the component can be used. Interface represents the moment at which component no longer needs to be aware about the underlying or surrounding system. A consistent method of interface definition and exportation has to be established by all components of a system in order to maintain independence from another. Exportation of component interface is possible make using standard commercial technologies, for instance CORBA (Common Object Request Broker Architecture), COM (Common Object Model) or Enterprise JavaBeans (EJB). (Hardy 2000.)

The purpose of component communication is to pass logistical messages between each other. A message-passing abstraction is used as the glue software, which connects objects and determines the interaction between components. On general interaction level, components offer and request services and generate and observe events. Procedures and event handling capabilities of components have to be in place to handle any of these combinations of event/service interactions. Thus, components can be integrated together. (Hardy 2000.)

When components are integrated into a particular system, implicit dependencies between components can appear. Consequently, these dependencies may limit COTS components reuse because components become then more difficult to reassemble or portable to another environment. In order to minimise several possible problems during COTS component integration, components may have to be concerned in special ways for instance their communication and functionality is carefully monitored and controlled. Therefore, the integrator has to minimise and report all possible dependencies between components. (Hardy 2000.)

Security issues should be concerned for each COTS component because any type of commercial component may have a security impact on the overall system. The component may for instance access unauthorised resources or access a resource in an unauthorised way. Also, component may have a wrong or bad functionality. As a precaution for these kinds of problems, component rights should be carefully inspected and tightly controlled. In addition, an insufficient component documentation may be a quite risk for development because correct security integration may be difficult or even impossible without good guidelines. Wrapper software could be used for protecting a component from a security breach, but still gaps could be left in the wrapper software leaving the component vulnerable. (Hardy 2000.)

2.6.1 Incompatibilities between components

Integration can become difficult because of incompatibilities between components. Yakimovich et al. (1999) present a classification of software components incompatibilities for COTS integration. This classification of mismatches or incompatibilities could be useful for COTS selection and integration. Thereby, the right component selection means less integration effort. In this case, incompatibilities are failures of components' interactions (Figure 10). Thus, finding and classifying these interactions help to find and classify the incompatibilities. Yakimovich et al. (1999) present three aspects of inter-component interactions and incompatibilities, which are type of interacting component, layer (syntax or semantic-pragmatic), and number of component participating in the interaction.

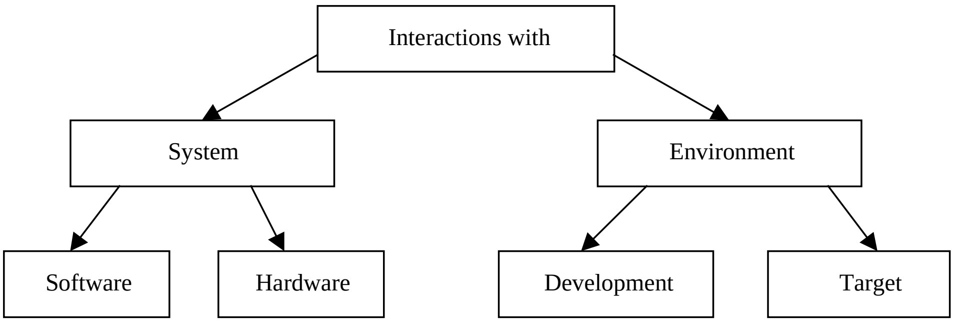


Figure 10. Interactions of software components (Yakimovich et al. 1999).

Firstly, components interact with other system components (software or hardware) and with the system environment. The system environment could be of the development phase, which includes debuggers, compilers and other development tools. It also could be the environment of the target system, which contains for instance operating systems, interpreters, virtual machines, and other applications. The parts of these environments could be considered components as well. (Yakimovich et al. 1999.)

The second aspects of inter-component interactions and incompatibilities are layers. Syntax defines the representation of the syntax rules of the interaction for example the functions' name, types, order of the parameters or data fields in the message. Semantic-pragmatic determines the functional specifications of the interaction for instance component functionality. Lastly, an incompatibility can elicit an interaction involving a particular number of participating components. A syntax incompatibility can appear because of syntactic difference between two components. Instead, a semantic-pragmatic incompatibility can occur either by one component, two mismatching components, or several conflicting components. (Yakimovich et al. 1999.)

Semantic-pragmatic incompatibilities are divided to the following types (Yakimovich et al. 1999):

- 1-order semantic-pragmatic incompatibility or an internal problem. This kind of component disregards the components it is interacting with. A component either doesn't have required functionality (not matching the requirements) or its invocation could be faulty (an internal fault).

- 2-order semantic-pragmatic incompatibility or a mismatch. This happens when two interacting components cause incompatibility. Components may not have 1-order semantic-pragmatic incompatibilities and they may work well in the other environment.

- N-order semantic-pragmatic incompatibility or a conflict. In this case, incompatibilities occur in interaction with several components. Components may not have 1-order and 2-order semantic-pragmatic incompatibilities, but their cumulative interaction can create a failure. For instance, processes together may require more memory than available.

However, several different kinds of incompatibilities (Table 4) can be elicited in the integration, and thus it is an important and difficult phase in third-party component-based software development.

Table 4. Interactions incompatibilities (Yakimovich et al. 1999).

Type of incompatibility	**Type of component**			
	System		Environment	
	Software	Hardware	Development	Target
Syntax	1.1	2.1	3.1	4.1
semantic-pragmatic 1-order	1.2a	2.2a	3.2a	4.2a
semantic-pragmatic 2-order	1.2b	2.2b	3.2b	4.2b
semantic-pragmatic n-order	1.2c	2.2c	3.2c	4.2c

2.6.2 Glue and wrapper software

Glue and wrapper software have been used when integrating two or more components together. Use of the wrapper software is the most popular way to modify commercial component without access to source code. It acts as shells that buffer the system from the commercial component. So commercial system looks like layered and it has to be considered together with component dependencies and interaction. (Hardy 2000.)

There are two types of wrappers. One type averts inputs that could cause errors. The other validates component output before passing it onto the next component. Wrapper software filters unwanted inputs that could make commercial components to behave unappropriately. In addition, they can be used to control the levels of component interaction and limit the undesirable dependencies. On the other hand, wrappers may become as or more complex than the encapsulated element. Degree of complexity involved in interfacing with the component and this may deter integrators from using wrappers in COTS integration. Wrappers cannot prevent against undesired events that are unanticipated by developers. Some unwanted functionality may exceed the wrapper's ability to buffer the system from the component. This problem has to be carefully assessed during the system design phase to ensure appropriate security measures are taken at every level of component interaction. (Hardy 2000.)

Luqi et al. (2000) present an automatic generation of wrapper and glue software which based on designers specifications (Figure 11). It reduces interoperability gaps between individual COTS components. Wrapper software offers a common message-passing interface for components that frees developers from the error prone tasks of implementing interface and data conversion for individual components. The glue software schedules time-constrained actions and arranges the communication between components. (Luqi et al. 2000.)

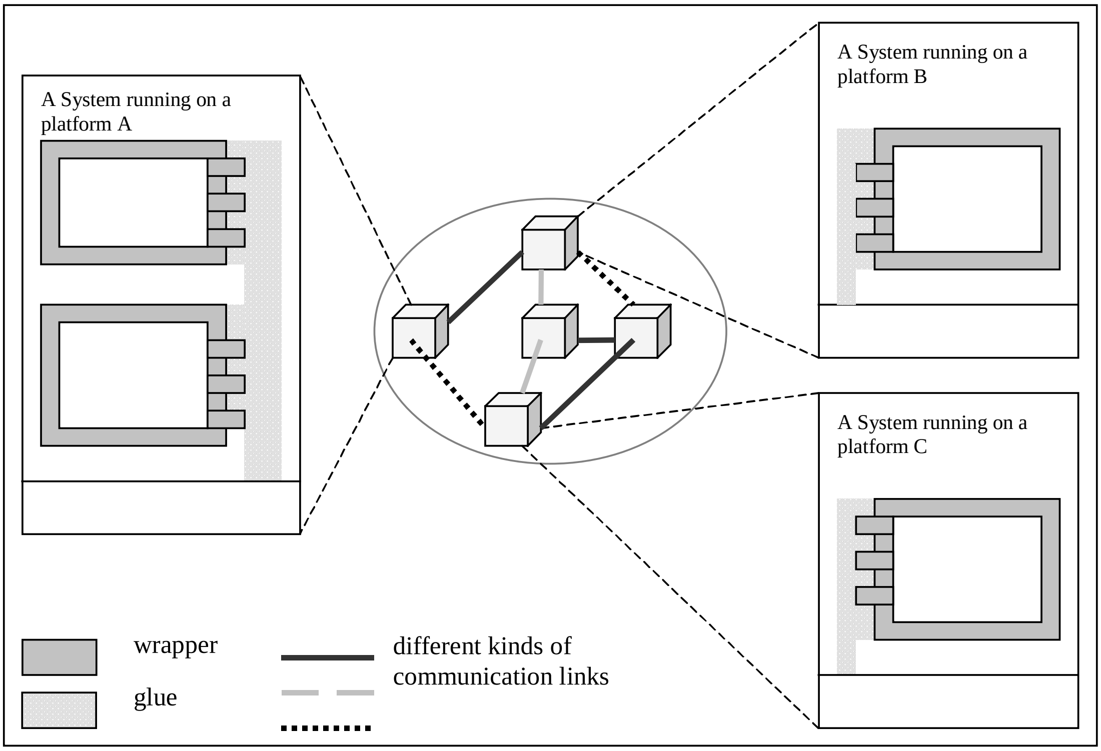

A System running on a platform A
A System running on a platform B
A System running on a platform C
wrapper
glue
different kinds of communication links

Figure 11. The wrapper and glue software (adapted from Luqi et al. 2000).

The glue-and-wrapper approach uses rapid prototyping and automated software synthesis to improve reliability. The approach allows designers to focus on the difficult interoperability problems without worrying of implementation details. Prototyping with engineering decision support can help to identify and resolve requirements conflicts and semantic incompatibilities. (Luqi et al. 2000.)

Glue software controls the orderly execution of components within a subsystem. In addition, it ensures the timely delivery of information between components across a network. An automated generation of glue code depends on automated local and distributed scheduling of actions on heterogeneous computing platforms. Identifying timing constraint conflicts and assessing constraint feasibility are critical in designing and constructing real-time software quickly. (Luqi et al. 2000.)

2.6.3 Methodology for COTS integration and extension

Mehta and Heineman (2000) present a four-step methodology for COTS integration and extension into an existing system. The steps are:

1. Perform buy vs. build analysis,

2. Create extensible model,

3. Compose new component by integrating and extending COTS,

4. Evolve and document communication.

Firstly, a buy vs. build analysis has to be performed as a part of the acquisition process. The purpose of the second phase is to ensure that developers keep the component as highly decoupled from the system as possible when integrating a component into a system. Therefore, an extensible model should be created. In the third phase, the COTS component is wrapped by a new component and is extended by adding new functionality. Mehta & Heineman (2000) regard COTS extension as different from COTS integration and they consider COTS extension to be a complementary technique. The aim of the last phase is to ensure that evolution is supported by communication with the target environment.

The main benefits of this methodology are that it evaluates pre-existing COTS components, it is incremental and it advocates a multi-tier architecture and integration design strategy, which brings benefits such as flexibility, scalability and extensibility. In addition, the methodology results in an explicit, documented and usable component with well-defined interfaces. (Mehta & Heineman 2000.)

2.7 Testing and maintenance

Currently, commercial components are usually not tested systematically. However, components should be tested carefully, because integration of several components could be difficult even if component's functionality is correct. Of course, integration problems can occur when component size and complexity grows. Fortunately, the commercial component’s functionality can be assessed immediately for instance integrator can test a component just by connecting the component to the system and run the desired test cases. Integrator can see if the component works correctly only by regarding the component's performance. (Hardy 2000.)

In general, Open Source software is without guarantee, which means that maintenance and support is not included. Some Open Source software can be used as a black box, which contains all the required functionality. Some other OS software offers functions that contribute to a larger system. In both cases, upgrades, maintenance and support must be considered. In the case where the Open Source software offers a platform it is likely that modifications will be made to the original OS component over time. (White et al. 2002.)

Release schedules for Open Source software are often very frequent and can be as frequent as daily. These releases will contain new features and will likely contain bug fixes. (White et al. 2002.)

Deploying the COTS components, extensive testing must be performed to ensure that the system will behave correctly. Components can be tested via different kinds of testing methods, which analyse the performance of a component in a system. Testing process includes the following methods: glass-box testing, black-box testing and fault-injection. (Hardy 2000.)

2.7.1 Glass-box testing

Glass box is like white box and means a system or component which internal contents or implementation is known. Glass-box testing is structural testing and takes into account the internal mechanism of a system or component. It includes branch testing, path testing and statement testing. (IEEE 1990.)

In COTS-based software development, glass-box testing is rather different. Testing a component with low-level access to source code – is neither effective nor possible with most COTS systems, because necessary insight is not possible. However, glass-box testing can be used during the implementation of custom wrappers around third-party components. Component wrappers have to be tested comprehensively in situations for which they were designed. This helps to ensure buffering of the component from outside environment. (Hardy 2000.)

2.7.2 Black-box testing

A black box means a system or component whose inputs, outputs, and general function are known, but whose contents or implementation are unknown or irrelevant. Black-box testing is same than functional testing and ignores the internal mechanism of a system or component. Black-box testing focuses solely on the outputs generated in response to selected inputs and execution conditions. (IEEE 1990.)

COTS component is usually delivered in a black box, which limits integrator looking component only outside. Naturally, this protects COTS vendor trade secrets and constrains the integrator to focus on the deployment of component instead of the implementation. An access to source code is unnecessary because black-box testing should be done according to the product specification or the specific project requirements. However, the main purpose is that the integrator can test a COTS component. (Hardy 2000.)

2.7.3 Fault-injection

In integration phase components may perform correctly, but misbehave at runtime. Fault injection is often used to see how well a component stand up to unfavourable conditions. Fault-injection is designed to simulate unpredictable errors during runtime performing of a system. Glass-box and black-box testing methods analyse situations for which the component was designed while fault injection simulates situations for which a component was not designed. Strict development procedures can reduce the amount of gaps in a complex system, but not totally avoid them. However, COTS component developers should be more worried with anticipating system errors rather than trying to avert all of them. (Hardy 2000.)

2.8 Summary

The objective of this chapter was to investigate third-party component-based software development. Benefits and risks of third-party components are discussed in Section 2.1. Furthermore, different kinds of COTS processes were presented and phases of third-party component-based development described. Presented processes focus only on the use of COTS components. However, the same process phases can be found when using Open Source components. As a conclusion, the third-party component-based software development process (Figure 12) is outlined based on literature research where the particular phases involving the process were found (see Section 2.2). In addition, a case study is influenced on this process model as well. This model presents tasks and communication flows in the COTS and OS based software development process. Interview questions (see Chapter 3.) are developed based on activities of this process model.

An organisation who use third-party component is called as component integrator and it communicates with a component provider. In this case, component provider means COTS vendor and Open Source community. The component integrator includes organisation and project level approaches. Thus, third-party component-based software development includes communication with project, organisational and provider level. In third-party component-based software development, other product software development and COTS and OS component related activities have to perform in parallel (Figure 12).

Firstly, goals for third-party component-based software development have to be defined, because it will affect the success of COTS and OS projects. The purpose of an organisation COTS/OS responsible person or group is to communicate with the component provider, define requirements for them and make legal and business evaluations. A database for COTS and OS components, criteria, guidelines and experiences is essential to arrange in the organisation.

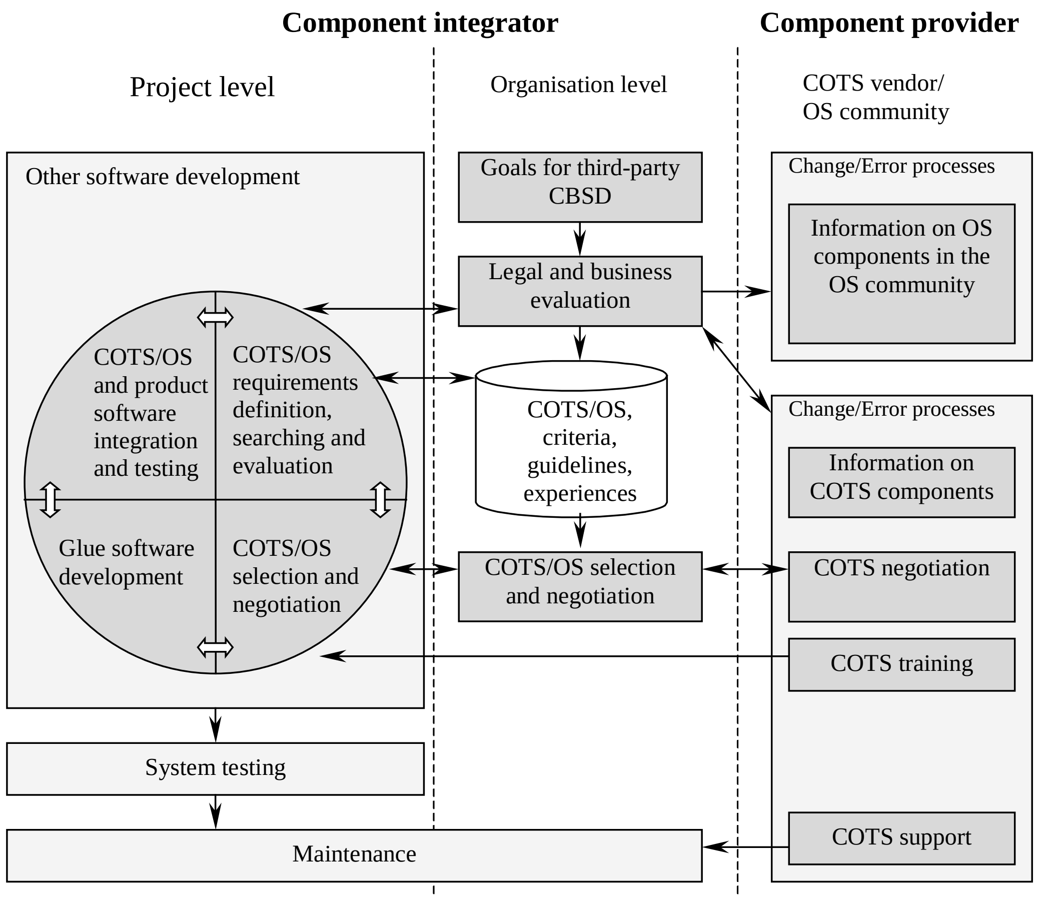

Component integrator
Component provider
Project level
Organisation level
COTS vendor/ OS community
Other software development
COTS/OS and product software integration and testing
COTS/OS requirements definition, searching and evaluation
Glue software development
COTS/OS selection and negotiation
Goals for third-party CBSD
Legal and business evaluation
COTS/OS, criteria, guidelines, experiences
COTS/OS selection and negotiation
Change/Error processes
Information on OS components in the OS community
Change/Error processes
Information on COTS components
COTS negotiation
COTS training
COTS support
System testing
Maintenance

Figure 12. Third-party component-based software development process.

COTS and OS deployment is an iterative process, which starts by defining requirements for components. Then, projects search components from the organisation's database and COTS market places and OS communities. Evaluations are made for the components found. The result of evaluation is to find a few potential component candidates for final selection.

COTS components are selected together at the project and organisational level and with the vendor. Usually negotiations can take a long time, and thus organisation is important to make preliminary contract negotiations with a vendor candidate. The effort of negotiation in preparation depends on the sizes of own and vendor organisations.

OS components acquisition is quite a different process. A project could select a component from the OS community directly or they can use a subcontractor, which searches and selects components. Thus, the OS-based software process is easier, because change and corrections are a subcontractor's responsibility. If project won't use a subcontractor is itself responsible for a component's life span. A project can also communicate with the OS community for example to change requests or correct error. The community may willing to make modifications, but if not, the project has to make them itself or search a new component. (White et al. 2002.)

When components are acquired the glue software development process begins, where glue software is developed between own software and COTS and OS components. Although differences between glue and wrapper software were discussed in Section 2.6.2, in this model (Figure 12) glue software is used to mean all necessary software in integration of in-house and third-party components.

COTS/OS component-based development has to synchronise to the traditional software development, because of architecture and interface issues. Finally, components, glue software and own product software are integrated together and tested. Furthermore, this package has to be tested in system testing. Product maintenance now becomes organisational responsible. The organisation has to negotiate with the COTS vendor for further component support.

3. Case study of third-party CBSD

This chapter presents the results of a current state analysis done at Nokia Networks (NET). The first sections include a short introduction on Nokia Networks and presents objectives and the research process of current state analysis. Also results of current state analysis at NET are presented. The last section describes glue software development process.

3.1 Case organisation in briefly

Nokia is a global company that has become the world's leading supplier of mobile phones, fixed broadband and IP (Internet Protocol) networks. In 2001, the net sales of Nokia totalled EUR 31.2 billion (USD 28.15 billion). Nokia had 18 production facilities in 10 countries around the world and Research and Development units in 15 countries. Nokia employs around 54,000 people worldwide. (Nokia 2002.)

Nokia Networks is a forerunner providing mobile, broadband and IP networks and related services. NET also develops mobile data applications and solutions for operator customers and Internet service providers. Nokia Networks has a strong position in wireless networks and in 2001 it represented approximately 22% of Nokia's net sales. Nokia Networks production is worldwide and units are located in Finland, Brazil, China, Great Britain and Malaysia. (Nokia 2002.)

3.2 Objectives and research process

The purpose of the current state analysis was to clarify how broadly third-party components were used in software development projects on the IP Mobility Networks site at NET in Helsinki and Tampere. The target environment was FlexiServer, which is a product line platform for about ten different products. Some of the products are known at the moment, but continuously new will be developed. Currently, the FlexiServer platform is under development and commercial components are acquired from partners in Oulu, Budabest and the United State of America.

In the case organisation, the third-party component-based software development process was studied and improved using the Pri2mer (Practical process improvement for embedded real-time software) method developed at VTT Electronics. The method is a systematic and practical approach for selecting, integrating and bringing into use the best practices in product development. The Pri2mer method (Figure 13) includes the following four tasks: Analysis of the current practices, Definition of goals for improved practices, Plan for process improvement steps and Piloting and use in product development. (Komi-Sirviö et al. 1998.)

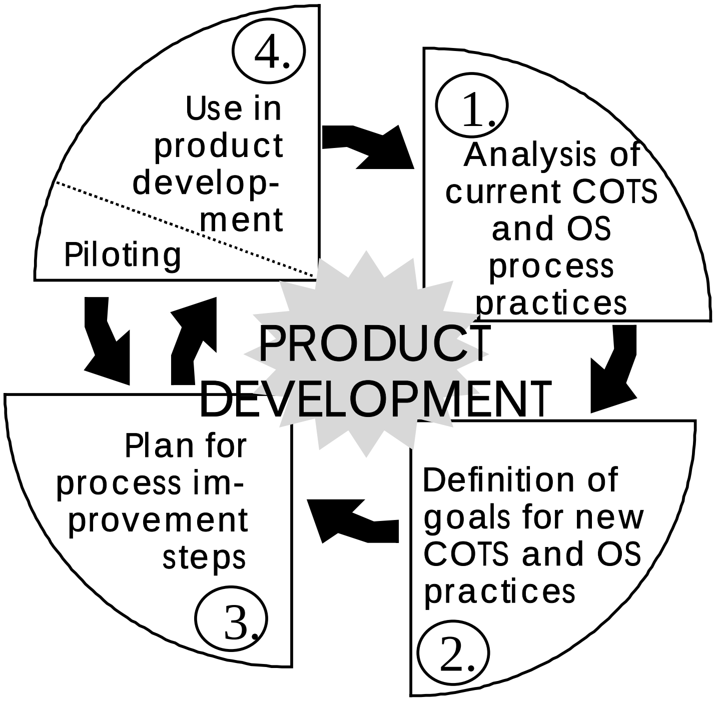

1.
Analysis of current COTS and OS process practices
2.
Definition of goals for new COTS and OS practices
3.
Plan for process im-provement steps
4.
Use in product develop-ment
Piloting
PRODUCT DEVELOPMENT

Figure 13. COTS and OS process improvement according to the Pri2mer method.

At first, a questionnaire (Appendix A) was made based on literature. It concentrated on the third-party component-based software development process, integration and architecture issues. Thirteen interviews were held, interviewees were managers and software architects. Each interview took one to two hours and questions were selected depending on the role of interviewee.

The questionnaire included 174 questions, which were divided according to activities of the third-party component-based software development process (see Section 2.4.). Questions concerned issues of requirements, architecture, the software development process, project planning and training and communication with supplier. Such questions as the following were asked:

- How COTS, OCM and OS components usage affects in-house coding?
- Are technical and non-technical requirements for COTS, OCM and OS components defined? How? What they are?
- How requirements for the component integration were communicated to suppliers?
- Is the architecture documented? How?
- Who does the search and selection of COTS, OCM and OS components? (for reuse projects, with reuse projects?)
- How COTS, OCM and OS components integration was planned?
- How the interface to COTS, OCM and OS components was defined and communicated?
- Did you use some method or technique to integrate COTS, OCM and OS components?
- How detailed a level of architecture should be described so that glue software could be developed?
- How the use of COTS, OCM and OS components affects product testing?
- How the use of COTS, OCM and OS components affects the maintenance?
- What are the most common problems in glue software development?
- How possible integration problems affect the whole product development process?
- How the use of COTS, OCM and OS components affects software development planning?

Secondly, a current state analysis document was created including improvement ideas and good practices. The practices were analysed and goals for new COTS and OS practices were defined. Current state description was delivered to the case organisation. The fourth step is to bring the new process guidelines into use in the case organisation, which is currently ongoing process.

3.3 Results of current state analysis

Based on literature and current state analysis the improvement ideas were defined, analysed and prioritised. The most of the improvement ideas were focused on the COTS acquisition process. However, the process for COTS component acquisition and management has been developed by University of Oulu and the process is presently being piloted. Because that process is not established yet, the current state analysis elicits improvement ideas, which are already solved in the COTS acquisition and management process.

Furthermore, current state analysis illustrated that integration of third-party components and product software is an important and difficult area. Thus, improvement ideas were focused on integration issues and the glue software development process was prioritised as the most important improvement target (see Section 3.4).

3.4 Glue Software development in third-party CBSD

Glue software development is an important process for the success of third-party component-based software development. Thus, it should be well defined and performed. However, COTS acquisition, including evaluation and negotiations, affects, firstly, the glue software development process and, secondly the change and error management processes. Consequently, problems appear mostly in the change and error situations. The glue software process is parallel to other product software development and should be synchronised with product software planning and third-party component acquisition.

The glue software development process (Figure 14) includes several activities that affect the successful integration of in-house and third-party components. Firstly, the needs for glue software have to be defined and its development planned. These activities affect the decision of the third-party components' selection. Therefore, they have to be parallel to the third-party component evaluation process. The next activity is to develop glue software, which is basically the same as traditional software development but is rather better to make incrementally and faster. Finally, all the necessary components are integrated into the integration testing phase.

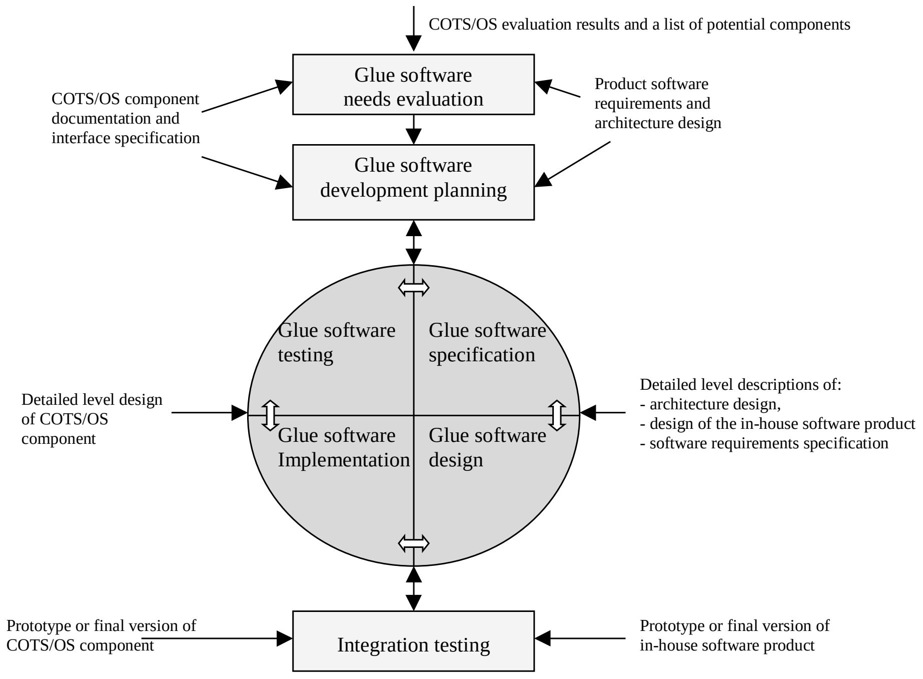

COTS/OS evaluation results and a list of potential components
Glue software needs evaluation
COTS/OS component documentation and interface specification
Product software requirements and architecture design
Glue software development planning
Glue software testing
Glue software specification
Glue software Implementation
Glue software design
Detailed level design of COTS/OS component
Detailed level descriptions of:
- architecture design,
- design of the in-house software product
- software requirements specification
Prototype or final version of COTS/OS component
Integration testing
Prototype or final version of in-house software product

Figure 14. Glue software development.

Glue software development is a part of the third-party component-based software development process (see Figure 12). Before glue software development, the requirements for third-party components are defined and the components are sought and evaluated against the requirements and evaluation criteria. Therefore, glue software needs evaluation also has to be considered in the component evaluation and selection phases. In addition, glue software development planning has to be synchronised with traditional software development from the project management viewpoint.

3.4.1 Glue software needs evaluation

The purpose of this phase is to evaluate the amount of glue software and necessary resources to integrate third-party and in-house code. This phase requires information on the interactions between acquired and own components. Such information could be, for instance, use cases and features of both kinds of components. Parallel with this phase, suitability tests of third-party components are performed. However, the product-level requirement specification and high-level architecture description, including potential third-party components, should exist for the glue software needs evaluation.

The glue software needs evaluation is an incremental process and the glue software has to be ready before the third-party and in-house software components integration. The glue software documentation will help in locating errors and error correction during the integration.

The glue software can be evaluated on technical and non-technical issues. From the technical evaluation perspective, integration may require functions that cover unnecessary features of third-party components (Hardy 2000). Furthermore, additional functions can be developed in the glue software (Dean & Vigder 1997). This is necessary, for instance, when an available third-party component doesn’t fulfil all of the needed requirements. In addition, glue software is needed to patch the difference gaps between the third-party and in-house software. The following differences between components could be elicited (Yakimovich et al. 1999):

- Syntactic differences
- Component internal problems:
- not matching requirements
- an internal fault

- Interaction problems with two or more components
- Conflict with other system components:
- interactions with system software and hardware
- Problems with environment:
- problems with-development or target environment
- Information or message is unsuitable between components
- Components original interface may not fit the architecture.

The non-technical needs of the glue software are related to integration testing, maintainability, interchangeability and traceability issues. Firstly, components' incompatibilities have to be evaluated during the third-party component acquisition and in-house software development phases (Yakimovich et al. 1999). Based on the evaluation results, incompatibilities can be decreased by glue software. Likewise, glue software can improve integration testing by offering the possibility of developing error locating functions in the glue software (Yakimovich et al. 1999). Thus the traceability of errors will be easier. Glue software can test the component outputs and report if the result is incorrect, and track information on component interfaces.

In third-party component-based software development the essential issue is to ensure component maintainability, modifiability and interchangeability (Hardy 2000). When a need to change a component appears, the need for modifying the glue software has to be evaluated as well. The ideal would be that only the glue software has to be modified when a new version of a component must be changed. Sometimes, it would be easier and faster to develop a new glue software because the amount of source code and documentation is much less than in in-house or third-party software development. Furthermore, the aim is to avoid changes to the in-house code or third-party component.

3.4.2 Glue software development planning

The purpose of the glue software development planning is to support the project management. Glue software development tasks should be defined and scheduled in a similar way as tasks in traditional software development. It is important that the tasks and schedule are defined in co-operation between the project managers and glue software developers. The inputs for the glue software development planning are the results of the glue software needs evaluation and the documentation of the third-party components and in-house software. Third-party component activities – for instance, component evaluation, selection and integration activities – are planned as a part of the project planning. The main differences between the traditional and the third-party component-based software development planning are:

- Tasks of third-party component evaluations and glue software development
- More effort for the integration activities and less for the software development
- Tasks of COTS contract negotiation and vendor management
- Tasks of OS community monitoring and OS maintenance activities
- More effort for the change and error management (vendor dependence)
- More effort for the risk management.

From a project management perspective, the main effort is to synchronise the glue and in-house software development. This means defining what documentation and implementations have to be ready for each other – for instance, which parts should be specified from the in-house software so that the glue software needs evaluation can be done. This evaluation should be done for each glue software development task. Glue software development affects the activities of traditional software development. Firstly, glue software definition and development requires effort from the project management. Furthermore, the architecture should be defined at a detailed level in an early phase because the architecture should be suitable for glue software development. Also, COTS locations in the architecture should be defined for the glue software needs evaluation. Likewise, in-house product software specifications have to be ready for the glue software needs evaluation. Also, product software designs should be implemented in an early phase because they have to be ready for the glue software specification and design.

3.4.3 Glue software development

Glue software development is basically similar to traditional software development, but glue software development is an incremental and fast process. It includes the same phases as traditional software development (specification, design, implementation and testing). However, a detailed level architecture design is needed so that the glue software development and the architecture are consistent with each other.

The requirements for the glue software development comes from the glue software needs evaluation. Therefore, the main purpose of the glue software specification is to specify the results of the glue software needs evaluation at a more detailed level. The high quality of the glue software specification is very important for locating errors during the integration and error analysis.

A detailed level design documentation of the third-party components and in-house components is needed for the glue software specification. In addition, information on the third-party components' future features and product architecture is needed for the glue software specification. The glue software specification could be a separate document or included as a part of the in-house product specifications. In COTS-based software development, the glue software could be developed by the integrator or the vendor.

Glue software design, implementation and testing can be performed according to the traditional software design, implementation and testing methods. The implementation phase includes coding, validation and verification of the glue software.

Glue software test needs should be evaluated case by case. If the glue software, third-party components or in-house components are very complex, the testing should be done separately. Otherwise, the integration testing is enough. When the glue software unit testing is done and integration testing begins, the prototype or application of the third-party and in-house software should be available.

3.4.4 Integration testing

The purpose of integration testing is to integrate the third-party components and in-house software piece-by-piece using glue software. When the glue software is being developed, the glue software needs evaluation document should be updated because it will help error location during integration testing.

Integration is a different process in third-party component-based development. Firstly, the third-party and in-house components are integrated using glue software, not by changing the components themselves. Integration means testing in-house software and third-party components in co-operation, which elicits new problems – for instance, error could be located in in-house code, third-party component or glue software. When error or a change request is focused on the third-party component, problems can appear. In this kind of situation error location and correction take a lot of the integrator’s time. Therefore, it is important to negotiate a contract for these kinds of situations.

Glue software will help integration testing and error analysis because the development is divided into smaller pieces. The number of integration incompatibilities will decrease if the differences between the third-party components and the in-house software are specified during the glue software development.

When the integration errors appear, the integration problem classification may help the error location and cause analysis during the integration testing. There could be functional problems, which means errors caused by missing or wrong functionality. Non-functional problems can be a consequence of non-matching requirements, such as reliability, maintainability and usability. In addition, problems could be architectural, which may require changing the whole architectural structure. There could also be conflicts between components in the systems – for instance, deadlocks. Furthermore, there could be interface problems, which are caused by syntax or 2-order semantic software and hardware incompatibilities. Functional and non-functional error analysis requires information on the in-house product software and third-party component functionality. (Voas 1998; Yakimovich et al. 1999.)

3.5 Summary

The purpose of the current state analysis was to clarify how third-party components are used in software development at Nokia NET/IMN. In addition, the aim was to study how components are integrated together and what problems can occur in integration.

According to the literature and the case study, glue software is needed to develop when integrating several components. In this chapter, glue software development process was described (Figure 14). It includes the phases of glue software needs evaluation, glue software development planning and development and integration testing. Glue software development includes the same phases than traditional software development such as specification, design, implementation and testing. The presented process model is validated with co-operation persons from case organisation, but it is not piloted in real third-party component-based environment yet.

4. Conclusions

The purpose of the research was to clarify how the use of the third-party components affects the software development process from the integrator's point of view. In addition, examination includes characterising third-party components and what advantages and disadvantages they will bring. This research area is solved by literature analysis and case study.

Theoretical analysis addressed that the use of third-party components affects the software development for instance emphasising to project management, component selection, integration and maintenance issues. From project management approach, the use of third-party components requires more resources to put in schedule estimation, risks analysis and evaluations of legal and business issues. Component selection requires more resources and well-defined procedures for components' searching, evaluation and negotiation activities. Moreover, integration of components is the key issue of success of third-party component-based software development. Thus, integration issues have to take into account in very early phases of software development. In addition, maintenance of software product will require new activities relating to third-party components and their providers. The process, which takes into account the use of COTS and Open Source software components, is presented in Section 2.8.

The other result of study was to analyse a status of the third-party component-based software development process in a case organisation. A questionnaire is presented as one result of research. Status of the case organisation's process is presented in Chapter 3. The main result of this work is examination of integration issues during third-party component-based software development. Problems in the integration phase may cause additional costs for software development. Thus, integration issues must be taken into account in the early phase of third-party component-based software development. Integration of different components requires the glue software between them. Glue software needs to be developed in parallel with other software, when third-party components are selected. The glue software development process was presented in Section 3.4.

There are many possibilities for further research. Firstly, the presented glue software development process would need to be piloted in the real environment. Currently, piloting of this process was out of the scope of this research because of resource constraints.

As this study concentrates on the examination of third-party component-based development from the integrator's point of view. It would be interesting to research third-party component issues from component provider’s, such as COTS vendors and Open Source developers’, perspective.

In the future, it would be interesting to examine a life cycle of COTS and OS components. This may clarify where software costs come in to play and offer an opportunity to analyse integration issues.

Indeed, the world of software development has changed a lot in the last few decades. Although software developers can use existing methods, languages, tools and even ready-made components for software product development, we can’t honestly say that software development is now much easier and faster. Nevertheless, there are always new different and difficult aspects, which have to be considered one way or the other.

References

Alasuutari, P. 1994. Laadullinen tutkimus. Tampere: Vastapaino. 281 p.

Ambler, S. W. 2000. Reuse through Internal Open Source. Software Development Magazine, December 2000.
Available: http://www.sdmagazine.com/. [Referenced 15.8.2002.]

Chung, L. & Cooper, K. 2001. Towards a Model Based COTS-aware Requirements Engineering Process. Proceedings of the 1st International Workshop on Model-based Requirements Engineering, San Diego, USA, November 2001. Available:
http://www.utdallas.edu/~chung/publication.html. [Referenced 12.3.2002.]

Chung, L., Cooper, K. & Huynh, D. T. 2001. COTS-Aware Requirements Engineering Techniques. Proceedings of the Workshop on Embedded Software Technology, 2001. Available:
http://www.utdallas.edu/~kcooper/research/WEST.pdf. [Referenced 12.3.2002.]

Dean, J. C. 2000. An Evaluation Method For COTS Software Products. Proceedings of the Twelfth Annual Software Technology Conference, Salt Lake City, Utah. April/May 2000. NRC 43660. Available:
http://seg.iit.nrc.ca/papers/ NRC43660.pdf. [Referenced 12.8.2002].

Dean J.C. & Vigder M.R. 1997. System Implementation Using Commercial-Off-The-Shelf (COTS) Software. Proceedings of the Software Technology Conference, Salt Lake City, Utah, April/May 1997. NRC 40173. Available: http://seg.iit.nrc.ca/ papers/NRC40173.pdf. [Referenced 20.4.2002.]

Feller, J., Fitzgerald, B. & Raymond, E. S. (Foreword). 2002. Understanding Open Source Software Development. London: Addison-Wesley. 211 p.

Feller, J. & Fitzgerald, B. 2000. A Framework Analysis of the Open Source Software Development Paradigm. Proceedings of the 21st Annual International Conference on Information Systems, Brisbane, Australia. Available: http://afis.ucc.ie/jfeller/publications/ICIS2000.pdf. [Referenced 25.6.2002.]

Hardy, M. G. 2000. COTS Components in Software Development. Proceedings of the Computer Science Discipline. Seminar Conference (CSCI 3901). University of Minnesota, Morris, Division of Science and Mathematics. Available: http://cda.mrs.umn.edu/~lopezdr/seminar/spring2000/hardy.pdf [Referenced 18.3.2002.]

Heineman, G. T. & Councill, W. T. (eds.) 2001. Component-Based Software Engineering, Putting the Pieces Together. New York: Addison–Wesley. 817 p.

IEEE Std 610.12-1990 1990. IEEE Standard Glossary of Software Engineering Terminology. New York: Institute of Electrical and Electronics Engineers, Inc. USA. 1990.

Jacobson, I., Martin, G. & Jonsson, P. 1997. Software Reuse: Architecture process and organization for business success. New York: Addison-Wesley. 497 p.

Järvinen, P. 2001. On research methods. Tampere: Opinpaja-kirja. 190 p.

Karlsson, E.-A. 1995. Software reuse, a holistic approach. Chichester: John Wiley and Sons. 510 p.

Komi-Sirviö, S., Oivo, M. & Seppänen, V. 1998. Experiences from practical software process improvement. Proceedings of European Software Process Improvement Conference, Göteborg, Sweden, November 1998. Pp. 5.31–5.45.

Kontio, J., Caldiera, G. & Basili, V. R. 1996. Defining Factors, Goals and Criteria for Reusable Component Evaluation. Proceeding of the CAS conference (CASCON), Toronto, Canada, November 1996.

Available: http://www.soberit.hut.fi/~jkontio/ CASCON96_Kontio.pdf. [Referenced 27.2.2002.]

Kontio, J., Chen, S.-F., Limperos, K., Tesoriero, R. Caldiera, G. & Deutsch, M. 1995. A COTS Selection Method and Experiences of Its Use. The 20th Software Engineering Workshop, NASA Software Engineering Laboratory, Greenbelt, MD, 1995. Available:
http://www.soberit.hut.fi/~jkontio/sew20_otso.pdf. [Referenced 27.2.2002.]

Luqi, Berzins, V., Shing, M., Nada, N. & Eagle, C. 2000. Software Evolution Approach for the Development of Command and Control Systems. Command and Control Research and Technology Symposium, Monterey, California, Computer Science Department, Naval Postgraduate School, 11–13 June 2000.
Available: http://www.dodccrp.org/2000CCRTS/cd/html/pdf_papers/Track_8/094.pdf. [Referenced 12.3.2002.]

Mehta, A. & Heineman, G. T. 2000. A Framework for COTS Integration and Extension. Position paper of COTS Workshop, Continuing Collaborations for Successful COTS Development, Limerick, Ireland, June 2000.
Available: http://wwwsel.iit.nrc.ca/projects/cots/icse2000wkshp/Papers/16.pdf. [Referenced 6.3.2002.]

Meyers, B. C. & Oberndorf, P. 2001. Managing Software Acquisition: Open Systems and COTS Products. New York: Addison-Wesley. 360 p.

Morisio, M., Ezran, M. & Tully, C. 2002. Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering, Vol. 28, No. 4, April 2002. Pp. 340–357.

Morisio, M., Seaman, C. B., Parra, A. T., Basili, V. R., Kraft, S. E. & Condon, S. E. 2000. Investigating and Improving a COTS-Based Software Development Process. Software Engineering. Proceedings of the International Conference 2000. Pp. 32–41.

Ncube, C. & Maiden, N. A. M. 1999. Guiding parallel requirements acquisition and COTS software selection. Requirements Engineering. Proceedings of the IEEE International Symposium, 1999. Pp. 133–40.

Nokia 2002. Nokia In Brief. [Web-document] Available: http://www.nokia.com/ aboutnokia/compinfo/index.html. [Referenced 1.8.2002.]

Ochs, M., Pfahl, D., Chrobok-Diening, D. & Nothhelfer-Kolb, B. 2001. A Method for Efficient Measurement-based COTS Assessment and Selection – Method Description and Evaluation Results. Proceedings of the 7th International Software Metrics Symposium, 2001. Pp. 285–96.

OSI 2002: Open Source Initiative. The Open Source Definition. [Web-document] Available:
http://www.opensource.org/docs/definition.php. [Referenced 25.6.2002.]

Peeling, N. & Satchell, J. 2001. Analysis of the Impact of Open Source Software. QINETIQ/KI/SEB/CR10223.
Available: http://www.govtalk.gov.uk/documents/ QinetiQ_OSS_rep.pdf. [Referenced 11.9.2002.]

Reifer, D. J. 1997. Practical Software Reuse. New York: John Wiley & Sons, Inc. 374 p.

Seppänen, V., Helander, N., Niemelä, E. & Komi-Sirviö, S. 2001. Original software component manufacturing: survey of the state-of-the-practice. Proceedings of the 27th EUROMICRO Conference, Los Alamitos, USA, September 2001. Pp. 138–145.

Schmidt, D. & Porter, A. 2001. Leveraging Open-Source Communities To Improve the Quality and Performance of Open-Source Software. Making Sense of the Bazaar: 1st Workshop on Open Source Software Engineering, Toronto, May 2001. Available:
http://opensource.ucc.ie/icse2001/schmidt.pdf. [Referenced 25.5.2002.]

Taweel, A. & Brereton, P. 1999. Evaluating off-the-shelf software components a repository selection case study. Preliminary EASE'99 Programme. Available: http://www.keele.ac.uk/depts/cs/ease/ease1999/abstract/a7.html.
[Referenced 8.4.2002.]

Voas, J. 1998. COTS software: the economical choice? IEEE Software. Vol. 15, No. 2, pp. 16–19.

Wang, H. & Wang, C. 2001. Open Source Software Adoption: A Status Report. IEEE Software. Vol. 18, No. 2.

Weinstock, C., Place, P., Hissam, S. Pickel, J. & Plakosh, D. 2000. A discussion of Open Source Software [Web-document]. Moderated by Bill Thomas. SEI Interactive. Available:

http://interactive.sei.cmu.edu/Features/2000/March/Roundtable/Roundtable.mar00.pdf. [Referenced 15.8.2002.]

White, T., Goldberg, J. & Scharich, L. 2002. The Open Source rEvolution - A Pragmatic Approach to Making the Best of It. Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, Orlando, USA. May 2002.
Available: http://opensource.ucc.ie/icse2002/WhiteGoldbergScharich.pdf [Referenced 10.6.2002.]

Yakimovich, D., Travassos, G. H. & Basili, V. R. 1999. A Classification of Software Components Incompatibilities for COTS Integration. Software Engineering Laboratory, Workshop, 1999.
Available: http://www.cs.umd.edu/users/dyak/ COTS/sel99.pdf. [Referenced 12.5.2002.]

Appendix 1: Questionnaire

Prestudy

1. Who does the search and selection of COTS, OCM and OS components? (*for reuse* projects, *with reuse* projects?)

2. What information was available when you started to search for COTS, OCM and OS components?

   - About coming project.
   - About coming products.
   - About product variants.
   - About the kind of needed components: domain/ product.
   - About the selection criteria.
   - About the intended usage of the components.

3. What information should you have for searching for COTS, OCM and OS components?

4. Did you use some framework/criteria to select COTS, OCM and OS components?

   - What kind of framework/criteria?

5. What kind of process was followed, that is, how were the following activities done or are planned to be done for COTS, OCM and OS components?

   - study of available components
   - selecting suppliers and components (criteria, responsibilities, evaluation)

6. Were there any problems with searching for COTS, OCM and OS components?

7. Were there any problems with selecting COTS, OCM and OS components?

8. What possible problems can occur with using COTS, OCM and OS components?

    - competence gaps,

    - unstable design assumptions,

    - component misfit.

Requirements

9. How use of COTS, OCM and OS components affects requirements?

10. What is the goal of COTS, OCM and OS components usage?

11. Are technical requirements for COTS, OCM and OS components defined?

    - What are the requirements?

    - Are technical requirements for components' integration defined?

    - Is it hard to specify all integration requirements in the early phase?

    - How the requirements were defined?

    - Are viewpoints of different stakeholders considered?

12. Are non-technical requirements for COTS, OCM and OS components defined?

    - What are the requirements?

        - quality,

- documentation,
- method of implementation.
- How the requirements were defined?

13. How big a part of the product is built using COTS, OCM and OS components?
14. What part of product is built using COTS, OCM and OS components?
15. What kinds of components are used (COTS, OCM, OS)?
16. Is the requirements definition process the same for small and simple components as for big and complex components?
17. Is the requirements definition process the same for COTS, OCM as OS components?
18. How rights and responsibilities have been agreed with the supplier?
19. How requirements for the component were communicated to suppliers?
20. What else was or what should have been communicated to suppliers?
21. How interface to COTS, OCM and OS components was defined and communicated?
22. Are specific OCM components developed for product line needs required?
23. How can you assure that COTS, OCM and OS components are compatible for several products?
24. How much you can open product line architecture for OCM suppliers?
25. How is product line architecture opened for OCM suppliers?

26. How do you ensure the security of product line architecture when ordering OCM components?

27. How do you ensure that components can be integrated with product line architecture when ordering COTS and OCM components?

28. How do you ensure the integrity of product line architecture when ordering and using COTS and OCM components?

Architecture

29. Is the architecture documented? How?

30. On what abstraction level is it described? (Conceptual, concrete, system level, module, subsystem etc.)

31. Does the current architecture satisfy those functional and quality requirements that it should satisfy?

32. If not, what are the main problems in the architecture?

33. Do you have solution proposals for the problems?

34. Is there or should there be any variability in existing architecture?

35. What is the role of the COTS, OCM and OS components in variation points?

36. Are any of the mandatory parts (subsystem, components etc.) of the architecture filled with COTS, OCM or OS components?

37. Are there any dependencies (requirements, exclusions) between COTS, OCM or OS components and how is this taken into account?

38. Is it usual that some features of the COTS, OCM or OS damage/modify the existing architecture?

39. Is it easy to find COTS, OCM or OS components adaptable to the existing architecture?

40. How COTS, OCM and OS components are integrated to the existing architecture?

41. How COTS, OCM and OS components integration affects architecture issues?

Design

42. How COTS, OCM and OS components usage affects design?

    - tool requirements

43. Is the high level architecture defined for product development?

44. Are COTS, OCM and OS components possible effects considered on high level architecture?

45. How COTS, OCM and OS components integration is designed?

    - responsibilities
    - glue software

46. Is glue software defined for selected COTS, OCM and OS components?

47. Are components integration issues analysed?

Coding

48. How COTS, OCM and OS components usage affects in-house coding?

    - tailoring
    - experience of employees (choice of project member, possible training)

49. What kind of problems can occur when implementing COTS, OCM and OS components?

50. How COTS, OCM and OS components implementation affects development process?
    - schedule
    - costs
    - training.

Integration of COTS, OCM and OS components

51. How COTS, OCM and OS components usage affects integration?
52. What information was available when you started to use/integrate COTS, OCM and OS components?
53. What information should you have when starting to use/integrate COTS, OCM and OS components?
54. In what phase COTS, OCM and OS components' integration becomes parallel with other development?
55. How did you ensure that the defined glue software is appropriate?
56. How did you estimate or predict possible problems?
57. How do you prepare for and minimize possible problems?
58. Were there any expected or unexpected integration problems?
    - What kind of problems?
59. How was the support with suppliers agreed?
60. Who is responsible for integration?
61. Did you use some method or technique to integrate COTS, OCM and OS components?
    - What kind of method/technique?

62. Are there any problems with integrating COTS, OCM and OS components?
    - What kind of technical problems?
    - What kind of non-technical problems?
    - How do you handle those problems?
63. What worked?
64. Which matters influenced successful integration?
65. What could have been done better?
66. How integration problems affect to the whole product development process?
    - schedule / delays / time to market
    - costs
    - contract

Glue software evaluation

67. How glue software need is evaluated?
68. How differences between COTS features and product software requirements are evaluated?
    - interface
    - component domain
    - service
    - logical behaviour
    - syntax
    - others?

69. How glue software is implemented in the existing architecture?

70. How glue software development affects architecture issues?

    - Were there any problems?

71. How detailed a level of architecture should be described in this phase?

Glue software specification

72. How glue software is specified (used methods and tools)?

73. How glue software should be specified?

74. Are there any problems or weaknesses in the glue software specification?

75. How glue software development is considered in project planning?

76. How glue software development should be considered in the project planning?

77. In which phase other software development should be related to the glue software development?

    - How detailed a level of architecture or other design should there be so that glue software can be specified and designed?

Glue software design

78. How glue software is designed?

    - What methods and tools are used?

79. Is all needed information available?

    - About product software and COTS interfaces

- About COTS domain, service, logical behaviour, syntax, other.

80. How glue software design affects product software development?

81. How did you ensure that the defined glue software is appropriate?
    - methods and tools?

Glue software implementation

82. What methods and tools are used during glue software implementation?

83. What kind of problems can occur in glue software implementation?

84. Which matters influenced successful implementation?

85. How implementation problems affect the whole product development process?
    - schedule / delays / time to market
    - costs
    - contract.

86. How change and error issues are managed related to the other software development?

Glue software testing

87. How glue software testing is performed?
    - methods and tools?

88. Is glue software tested individually or only with other software prototypes?

89. How glue software should be tested?

Integration testing

90. How integration testing is performed?
    - methods and tools?
91. Are there any problems in integration testing?
92. How errors are reported and corrective actions implemented to glue software?
    - responsibilities
    - methods/tools
93. How is it decided who will correct the particular error? (glue software engineers or other software engineers)
94. How side effects of corrections to the other components or interface are analysed?
    - responsibilites
95. How do you ensure that all software pieces are tested? Also corrected parts?
96. Is glue software tested similarly like other product software or does glue software brings some new aspects to integration testing? new tests cases?
97. Is the glue software process measured, what metrics are available?
98. How integration testing should be performed?

Testing

99. How the use of COTS, OCM and OS components affects product testing?
    - unit test
    - integration test

- system test.

100. Does use of COTS, OCM and OS components affect quality management?

    - Are effects considered?

101. Are quality requirements defined for COTS, OCM and OS components?

102. How is quality of COTS, OCM and OS components assured?

103. How problems are reported and corrective actions implemented to COTS, OCM and OS components?

    - responsibilities

104. What methods are used?

105. Which tools are used?

106. Who does (roles and responsibilities)?

107. Are existing SQA activities adequate?

108. Are COTS, OCM and OS processes and product measured, what metrics are available?

109. What do you see as important with respect to COTS, OCM and OS components quality?

    - What does product quality mean?

    - What can it affect?

    - What is the current status?

Maintenance

110. How COTS, OCM and OS components usage affects maintenance?

111. Who is responsible for maintenance?

112. How was the maintenance agreed with the supplier?

113. How was component “guarantee” agreed with the supplier?

- defect corrections
- integration problems.

Experiences of the use of COTS, OCM and OS components

114. Are there any problems with using COTS, OCM and OS components?

115. What worked?

116. What could have been done better?

117. Do you use COTS, OCM, OS components enough in the product line?

Project/software development planning

118. How the use of COTS, OCM and OS components affects software development planning?

- schedule estimates
- effort estimates
- size estimates
- resource (human and technical) allocations
- commitments.

119. Are the activities for managing COTS, OCM and OS subcontractors planned for the project?

120. What information is needed from component supplier for project planning?

121. How OCM components suppliers's schedules are agreed and approved?

122. How an approved plan is changed?

123. What methods are used?

- Risk assessment and prevention methods
- Cost estimation and control.

124. Which tools are used?

125. Who does (roles and responsibilities)?

126. Are measurements used?

127. What kinds of meetings are held? (Participants, agenda, schedule)

128. How COTS, OCM and OS components integration affects people management and training?

129. Is personnel trained systematically? (How project members have been trained?)

130. With what other groups do you need to co-ordinated?

131. How commitments are established and managed with these groups?

132. What meetings are held with the related groups?

133. How projects communicate with each other?

134. How COTS, OCM and OS components mutual compatibility is researched or ensured?

135. How the work to be subcontracted is defined and planned?

136. How changes to OCM subcontractors work are agreed?

137. What kind of contract is established?

138. Are periodic technical interchanges held with OCM subcontractors?

139. How subcontracted work products are accepted?

Project/software development tracking

140. How the use of COTS, OCM and OS components affects project/software development tracking?

141. Is COTS, OCM and OS components usage progress reviewed? (Participants, agenda, schedule)

142. Are plans/estimates compared to actuals (size, schedule, effort, technical targets, etc)?

143. What happens if there are deviations in the plan and actual results?

144. How are identified problems tracked?

145. What kinds of meetings are held? (Participants, agenda, schedule)

146. Which tools are used?

147. Who does (roles and responsibilities)?

People management and Training

148. How COTS, OCM and OS components usage affects people management and training?

149. Are training activities planned for reuse in general and for COTS, OCM and OS components technical issues?

150. Is personnel trained systematically? (How have project members been trained?)

151. Has the project team been trained for the tools that they use in their work?

Inter-group co-ordination and software subcontractor management

152. With what other groups do you need to co-ordinated?

153. How commitments are established and managed with these groups?

154. How dependencies are identified and tracked?

155. What meetings are held with the related groups?

156. How projects communicate with each other?

157. What kind of communication exists between for-reuse and with-reuse groups?

158. Is COTS, OCM and OS components acquisition centralized?

159. How COTS, OCM and OS components mutual compatibility is researched or ensured?

160. How the work to be subcontracted is defined and planned?

161. How changes to OCM subcontractors work are agreed?

162. How OCM subcontractors are selected?

163. What kind of contract is established?

164. Are periodic technical interchanges held with OCM subcontractors?

165. Are the results and performance of the software subcontractor tracked against their commitments?

166. Who does (roles and responsibilities)?

167. How subcontracted work products are accepted?

Other

168. Anything else important related to COTS, OCM and OS components usage?

169. If you could change one thing in COTS, OCM and OS components usage what would it be?

170. What do you see as the strengths of the COTS, OCM and OS components usage at the organisation?

171. Anything else important related to glue software development?

172. What strengths can be found in glue software development?

173. What are the most common problems in glue software development?

174. What are the worst problems in glue software development?

Published by

Series title, number and report code of publication

VTT Publications 489
VTT–PUBS–489

Author(s)
Arhippainen, Leena

Title

Use and integration of third-party components in software development

Abstract

Reuse of software components has been seen as an effective way of decreasing costs and reducing software development cycle-time. Nowadays, reusable components are often acquired from outside organisations. Consequently, the use of third-party software components brings new aspects to software development. One aim of this research was to clarify the activities, roles and possible problems related to the use of third-party software components.

The main purpose of this study was to research the software development process when using third-party components. As a result, the third-party component-based software development process is presented. Furthermore, the use of third-party components is clarified in the particular projects at Nokia Networks. This case study offered an opportunity to research the issues involved in integrating third-party components into a software product. As a result of this study, the glue software development process is described.

Keywords
component-based software development (CBSD), Commercial-Off-The-Shelf (COTS), glue software, Open Source (OS), third-party components

Activity unit
VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN 951–38–6032-9 (soft back ed.) 951–38–6033-7 (URL:http://www.inf.vtt.fi/pdf/)			Project number E2SU00277
Date February 2003	Language English	Pages 68 p. + app. 16 p.	Price B
Name of project Minttu 2		Commissioned by	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)		Sold by VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU	VTT INFORMATIONSTJÄNST	VTT INFORMATION SERVICE
PL 2000	PB 2000	P.O.Box 2000
02044 VTT	02044 VTT	FIN–02044 VTT, Finland
Puh. (09) 456 4404	Tel. (09) 456 4404	Phone internat. +358 9 456 4404
Faksi (09) 456 4374	Fax (09) 456 4374	Fax +358 9 456 4374

ISBN 951–38–6032–9 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6033–7– (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)