

Roles of Developers as Part of a Software Process Model

ETSPI06.PDF

Kari Kivisto
MSG Software
P.O. Box 28
FIN-90101 Oulu, Finland
eMail: kari.kivisto@msg.fi
<http://www.msg.fi>

Abstract

The software development process is the core process of any IT organization. This process has required radical redesign as organizations have adopted object-oriented application development approach and client/server architecture. The article raises an important issue noted when organizations have adopted object-oriented client/server application development process models, namely the roles of the developers. Since software is developed by people, the management of these people should be part of the software development process. This article points out the roles and responsibilities needed in object-oriented client/server application development. It also shows how to integrate the roles easily with the process model. The combination of the role model and the object-oriented client/server development model (Team-Based Object-Oriented Client/Server model, OOCS model) has evolved to its current form over the past five years. This article discusses briefly the theoretical premises underlying the model and reports on the lessons learnt when implementing the roles.

KEYWORDS: Process model, role, team, object orientation, client/server architecture.

1. Introduction

Object orientation and client/server architecture (including emerging technologies, such as the Internet, and distributed and cooperative architectures) have caused IT organizations to redesign their software development processes. Most of the current software process models are object-oriented (for instance, Booch 1994, Graham 1995, Jaaksi 1997, Jacobson et al. 1992, Kivisto 1997, Rumbaugh et al. 1991), but not all of them pay attention to the client/server architecture and the problems inherent in it. In addition, there are only a few models that recognize the roles of the developers in this continually changing situation. The purpose of this paper is to evaluate the object-oriented client/server application development process model and the influence it has on the roles of the developers. We propose that a process model should have three dimensions:

1. Organizational dimension

- structure of the project, management of the project
- splitting of the project into teams
- roles and responsibilities in the teams and in the project

2. Technological dimension

- client/server architecture consisting of three sub architectures: technology architecture, application architecture and data architecture

3. Process dimension

- process model, phases, activities and tasks in each phase, deliverables
- methods and techniques to be used
- roles of the developers

The next chapter will discuss the technological dimension. Chapter 3 will address the organizational dimension, i.e., the roles of the developers and the structure of the project. The process dimension will be considered in Chapter four. One process model will be described that integrates the object-oriented paradigm, the client/server architecture and the roles into one coherent model. The lessons learnt about the roles and their adoption in the organizations will be discussed in Chapter five.

2. The Technological Dimension

The technological dimension has not been studied much in process modeling. However, the development of emerging technologies (client/server, internet/intranet, distributed and cooperative architectures) has been a daily concern for developers during the past few years. One reason for the failures in the projects has been the underestimation of technological risks. Projects have adopted new tools (i.e., development environments) and constructed applications to the new run-time environments practically without any baseline architectural tests. The problems have been identified far too late, often in system tests in the actual run-time environment.

A three-tier architecture should be present in the process model designed for client/server application development. The three-tier architecture means that the application is divided into three tiers, namely presentation, business logic and data management (Figure 1, top part). Not many process models stress this point sufficiently. The reason for this may stem from the early days of object orientation and its usage for embedded system development, where presentation and data management were less frequently needed and considered. But those who started client/server application development (i.e., applications that consisted of graphical user interfaces and relational databases) in the early 1990s felt the contemporary object-oriented process models to be inadequate. Fortunately, there was at least one process model that attacked this problem, namely the OOSE by Jacobson et al. (1992). Their use case-driven approach has been applied afterwards to other models (for instance, Lorenz 1993, Jaaksi 1997, Kivisto 1997). The OOSE also gave one solution to the data modeling problem. The weak point of the model was its notation, which deviated from the *de facto* standard OMT by Rumbaugh et al. 1991, which, in turn, did not include use cases. This problem of competing notations has caused quite a lot of trouble for those modeling systems. The Unified Modeling Language (UML Documentation Set 1997) seems to solve this problem. It facilitates the documentation task by giving a standard notation instead of the previous competing notations. One thing that the UML lacks of, however, is the data modeling diagram(s) and notation.

The emerging technologies should be considered thoroughly in the process model. In particular, the client/server architecture should be seen as a construction of three sub architectures (Figure 1): technological architecture, application architecture and data architecture (Microsoft 1993).

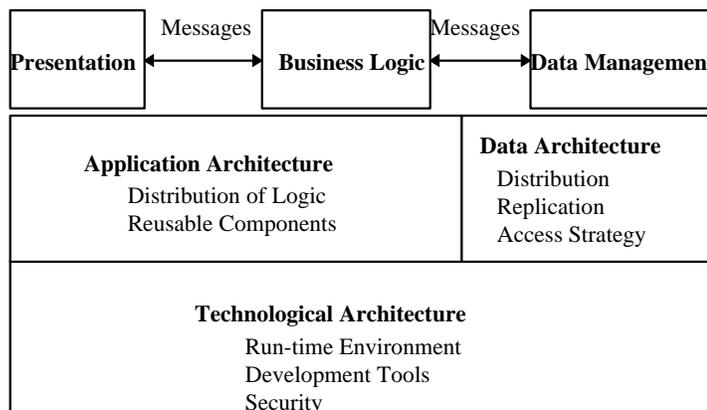


Figure 1. Three-tier client/server architecture.

The Technological Architecture defines both the run-time environment and the development environment.

The Data Architecture deals with such issues as distribution of databases, access strategies and replication strategies. The first client/server applications usually had one relational database, but contemporary applications may have both logically and physically distributed databases. In addition, new object-relational databases (for instance, Oracle 8 from Oracle Co. and DB2 Universal Database from IBM Co.) have object-oriented features, such as user-defined data types, but also new possibilities to incorporate part of the application logic into them. This means that data management currently consists of more than just data modeling and design.

The Application Architecture is based on both the technology and the data architecture, and it defines how the application is built and divided into three parts: presentation (user interfaces), business logic (local and corporate business logic) and data management (object-oriented or relational databases). The Application Architecture is the most interesting and challenging of the three architectures. Usually, the term 'software architecture' (for instance, micro and macro architecture by Booch 1994, 1995) refers to the Application Architecture.

An object-oriented client/server development model should clearly state the commitment to the client/server architecture. All the three sub architectures (technology, data, and application) should be described in the documentation, as the design phase cannot be started if the architectural decisions are missing. The three-tier application architecture (presentation, business logic, and data/object management) must be kept in mind in all phases of the development cycle. This is one conclusion that has emerged from real projects. It has also turned out that this approach may take some time, but it pays back in the lesser need for development (fewer technical problems) and maintenance (well-structured application) time. Furthermore, the overall selection and design of architecture should be allocated a special phase where the architectural decisions are made.

3. The Organizational Dimension

After the huge changes in software development in last few years, it seems amazing how little attention the role issue has attracted. Yet, nearly everything has changed (development paradigm, methods, techniques, notation, tools, complexity of systems, etc.), and one might have expected more interest in the roles of the developers and the roles of the teams and the project organization. This chapter will concentrate on the role issue in more detail.

The roles to be presented next have mainly been derived from the object-oriented paradigm and the three-tier application architecture. The three-tier architecture divides the application in a natural way into three parts: presentation (i.e., user interfaces and reports), business logic (i.e., business components according to business domains), and data/object management (i.e., central/distributed databases). There should be thus three roles in the development team, one for each tier. We name and describe the roles in the following way:

Application Developer (AD)

These developers analyze, design and develop the requested application, using reusable components whenever possible. They write the use cases with the users, and design and implement the interfaces and reports. They may also design and develop new reusable components or drafts for such during the project. Note that application developers are able to work in all phases, avoiding information loss between the phases.

Business Object Developer (BO)

The Business Object Developer designs, develops, maintains and owns reusable business objects. If there is more than one team in the project, each team has at least one Business Object Developer. The BO may be responsible for searching for reusable components from different sources. His/her key activities are to analyze, design and construct business objects, which are the company's key assets.

Database Developer (DB)

The Database Developer is responsible for data modeling and database design, and acts as an expert in data architecture definition. The role is an essential part of the project for two reasons. First, the applications are database-intensive. Second, the new versions of database management systems include object-oriented features providing possibilities to place parts of the functional logic in the databases. This means that the Database Developer has to design and code parts of the application, which is why we speak of ‘developer’.

The fourth essential role is the User:

User/End-user (US)

The Users are an essential part of the development team. User involvement in application development is never overestimated. They take part in system definition, analysis and testing, and, with some training, they are also capable of writing use cases, on-line helps and user manuals.

The Users and Application Developers constitute a very close mini team and cooperate on writing use cases and designing user interfaces.

A leader for the team is also needed:

Team Leader (TL)

The Team Leader is responsible for directing the application development process. He/she is the chief architect of the project. Team Leader works as a peer for the Project Manager. The Team Leader’s skills are tested at two critical points: in the system definition phase, where the team leader is the visionary for the new system, and in the architecture phase, where he/she designs the application architecture. These two activities call for experience and knowledge.

Figure 2 depicts the team organization up till this point.

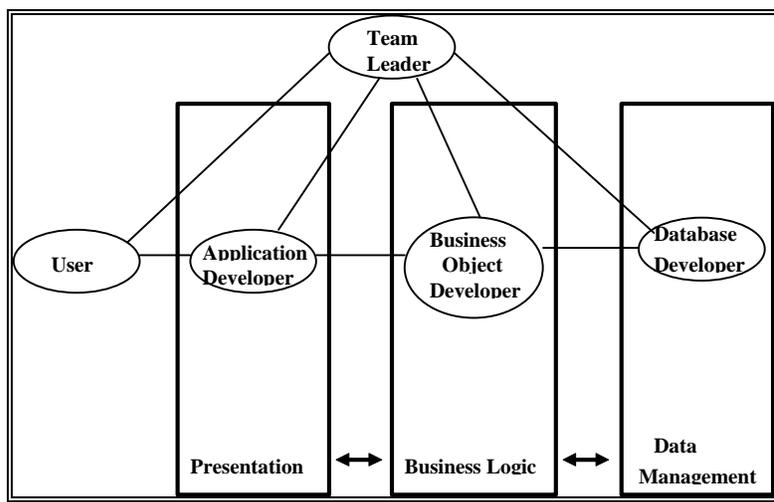


Figure 2. Roles of the team and object-oriented client/server architecture.

These roles make up the basic team, and a project is made up of such small teams. Teams should be established for the business domains, each team being responsible for the portions of the application dealing with one business domain.

The teams make up the project, and the project is managed by the Project Manager:

Project Manager (PM)

The Project Manager is responsible for human resources management, including management of the project’s resources (both human and technical), tasks, deliverables, schedule and planning. He/she controls the project and the teams and determines the rhythm of the project.

There are two additional roles that are needed in the overall organization:

Quality Assurancer/Tester (QA/TE)

The Quality Assurancer is an outsider in the project, who comes from the QA department and reviews, inspects and audits the quality of the project. The Tester is an insider, who is responsible for making test specifications and testing. Application Developers also take part in testing and may write test specifications. This is always the case in small projects.

Expert (EX)

There are at least two categories of experts, namely domain experts and technical experts. Domain Experts work with Business Object Developers and can also help Application Developers, while technical Experts take part in the architectural phase and are interviewed in the system definition phase when architectural issues are under discussion. These Experts may also check the installation instructions, test the installation and perform system tests.

Figure 3 depicts the project organization that consists of two small teams.

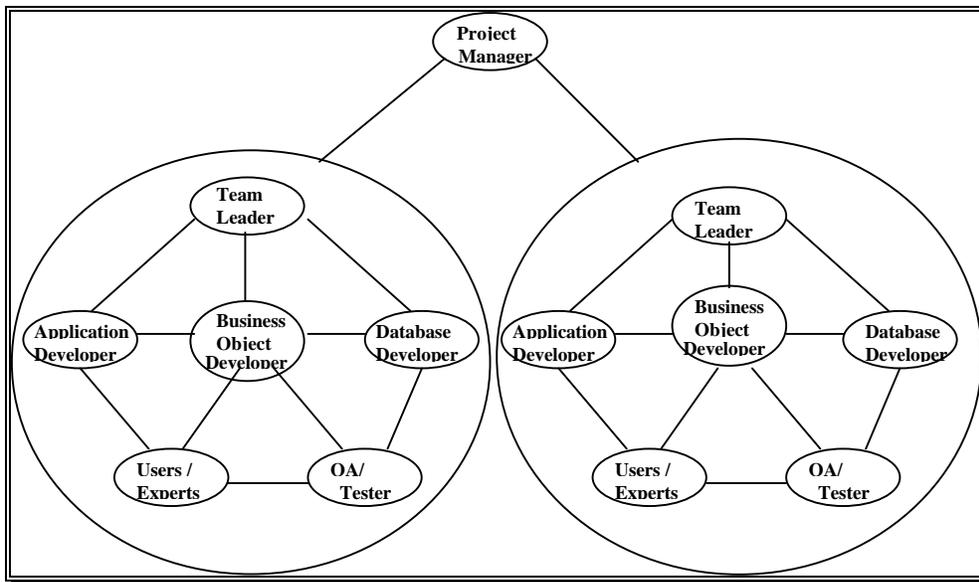


Figure 3. A project consisting of two teams.

We should note that one person may act in several roles and vice versa. However, there are some limitations. For instance, the QA cannot take part in the development process and the Team Leader should avoid coding, in order to be able to supervise the team’s work.

4. The Process Dimension

A process model that combines the elements of the object-oriented client/server architecture (the technological dimension, Chapter 2) and the roles of the developers in an environment (the organizational dimension, Chapter 3) is the Team-Based Object-Oriented Client/Server (OOCS) model. Figure 4 depicts the model. The roles of the developers are also indicated in the figure.

Below, each phase is briefly described in order to give the reader a more detailed of the model. The overall model consists of a documentation set that includes description of the model, instructions on how to carry out each phase, technique-specific instructions and UML-specific document templates for the deliverables. Each phase is described both by a flow diagram and by a table that shows the roles, their activities, the deliverables and the UML diagrams to be used. A more detailed analysis of the UML and its use as a process modeling language can be found from Kivisto (1998).

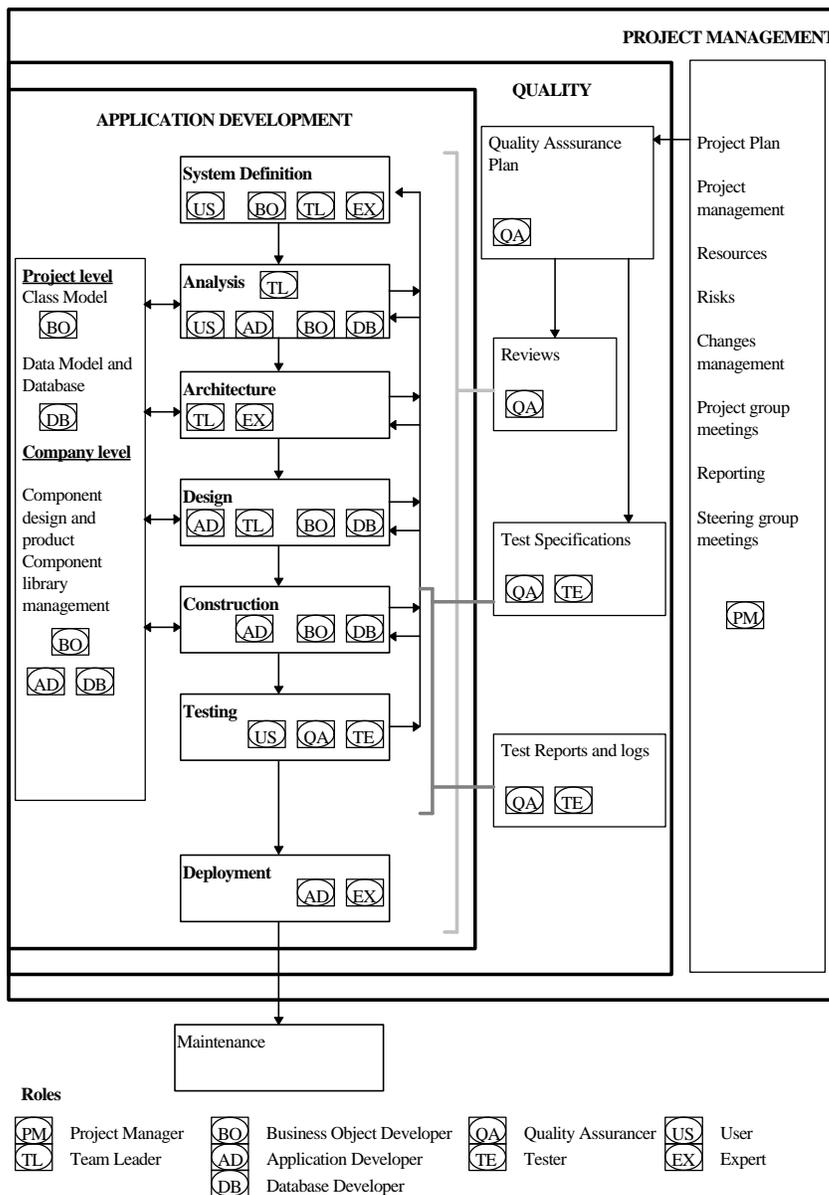


Figure 4. The OOCS model.

4.1 The System Definition Phase

Table 1 summarizes the roles and their activities, the artifacts and the diagrams that can be used. The System Definition document serves as a starting-point for the subsequent activities. The Team Leader and the Users define the roles of the users and their main functions. They are input to a use case analysis in the Analysis phase. The Business Object Developer defines the first draft of the business object model, which is the input to object modeling in the Analysis phase. The Experts and the Team Leader define the architecture draft. Activity diagrams showing the main functions can be used to describe the processes, although they are primarily meant to be used as extensions to the statechart diagrams. The above diagrams can already be used in this phase, if necessary, to give more expressive power to the document.

Table 1. The System Definition Phase

Roles	Activities	Artifacts	UML Diagrams
US & TL	Define the system and the users' main roles and functions	System definition, roles of the users, main functions	Use case diagram Activity diagram
BO & TL	Define business objects	System's business objects	Class diagram
TL & EX	Define Architecture draft	Architecture draft	Deployment diagram
TL & EX	Define interfaces to other systems	System's boundaries and interfaces	Deployment diagram

4.2 The Analysis Phase

Iteration and prototyping are used in the analysis phase as much as possible. Feedback from the users is a critical factor for a successful project, and the best feedback is guaranteed via user interfaces. Thus, the first drafts of the user interfaces should be ready soon after the analysis phase has started. The three-tier client/server approach can be seen in this phase, in which the presentation is described by use cases and interfaces, the business logic by an object model and operations, and the data model by a data model description (Figure 5 and Table 2). The Users and the Application Developers work together on the first tier, the Business Object Developer works on the second tier, and the Database Developer on the third tier. The Team Leader serves as the link between the developers.

The link between the use cases and the object model is a very strong one, and it is essential that the user interfaces are defined after both the use cases and the object model have been designed. This does not mean, however, that the use cases and the object model need to be ready before the process goes on, because the process is iterative in nature. On the other hand, the use cases should be available for user interface design and the object model for data modeling, not vice versa. It must be born in mind that one use case may lead to different user interfaces, depending on the designer.

It is not always necessary to define the data model in this phase, but in projects where the users want executable prototypes, the data model and the database are needed very early. However, the longer the developers can avoid data modeling and concentrate on object modeling, the better the solution they are able to find. This is because developers are used to data modeling, which may dominate over object modeling, causing this to operate in a data-oriented manner.

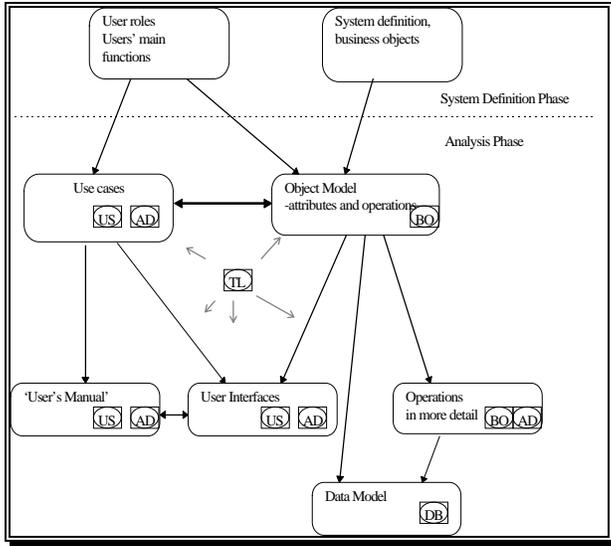


Figure 5. The Analysis Phase

The use case diagrams are definitely something that we can recommend. They have been tested in real life situations and they have shown their power when user needs are collected. Use cases were incorporated in the OOCs soon after Jacobson et al. published their book (Jacobson et al. 1992). There were some practical problems in the beginning, since the OMT was the best known method (in Finland, at least) and it did not include use cases, and, furthermore, the OMT included such scenarios as design time artifacts, which confused the developers (see, Kivisto 1997, pp. 61-62). Practice has shown that most users can adopt use cases after a short training period and are able to write and update use cases whenever necessary. In addition, activity diagrams can be used in the Analysis Phase, displaying how the use cases are linked to each other.

Table 2. The Analysis Phase.

Roles	Activities	Artifacts	UML Diagrams
US & AD	Use case analysis	Use case descriptions	Use Case Diagram Activity Diagram
US & AD	Write user manual draft and on-line help texts	User manual and on-line help draft	
US & AD	Define user interfaces and reports	User interfaces and reports descriptions	
BO	Define Business objects	Class descriptions	Class Diagram
AD & BO	Define operations	Operations descriptions	Class Diagram
DB	Carry out data modeling	Data model description	

The UML lacks a data modeling diagram. Class diagrams can naturally be used for data modeling, but one would expect the presence of data modeling in the UML. Fortunately, tools that implement UML usually include data modeling capabilities. For instance, the mapping of a class model to the data model and a data model to the database model can be done with these tools, too.

4.3 The Architecture Phase

The analysis phase is normally followed by the design phase. The OOCs model emphasizes the architecture phase preceding design for many reasons. First, the technological architecture is changing rapidly, and both the technical infrastructure (workstations, servers, network, operating systems, etc.) and the tools (CASE, I-CASE, application development tools, etc.) are evolving faster and faster. Two things must be ensured: that the application is rational and can be built with the chosen technology, and that the technology will be valid and usable even in the future. Hence, the development and run-time environments are defined in this phase.

The application architecture describes the structure of the application, i.e. the way it is divided into presentation, business logic and data management. It also describes the application by means of executables, dynamic link libraries, and so on. If the business logic is to be divided between the client and the server, this division is decided here. The other relevant parts are the reusable components, which means that one should look for parts in the system that should be constructed with reusability in mind. These parts could later be included into the company's reusable component library. Parts that could be either constructed from the company's own reusable libraries or bought from vendors should also be looked for. The Team Leader's professional skills are weighed at this point.

Table 3. The Architecture Phase.

Roles	Activities	Artifacts	UML Diagrams
TL & EX	Define Technology architecture	Technology architecture description	Deployment diagram
BO & TL & EX	Define Application architecture	Application architecture description	Component diagram
DB & TL & EX	Define Data architecture	Data architecture description	Deployment diagram

The Component and Deployment diagrams are a good addition to the UML. So far, the architectural descriptions have been more or less company-specific, each company having its own description standards. The diagrams mentioned can be used in the OOCs model, as shown in Table 3 with texts, lists and other diagrams fulfilling the document.

4.4 The Design Phase

The goal of the design phase is to make the construction phase as easy as possible. The inputs of the design phase are models from the analysis phase and the architecture descriptions. Table 4 summarizes the design phase.

The technical classes are added to the object model, the business objects are redefined, and the behavior (operations) of the classes is defined exactly. The interfaces are also redefined here, and the aim is to define the finite interfaces. The UML offers several diagrams for the description of logic. There is a sequence diagram and a collaboration diagram for describing the communication between objects and a statechart diagram and an activity diagram for describing the state transitions of a class. The functional logic is defined precisely from the main complex functions. There is no need to define the functional logic of the trivial functions by using the diagrams, as a short textual description of an operation is usually adequate.

Table 4. The Design Phase.

Roles	Activities	Artifacts	UML Diagrams
TL & BO & AD	Design structure of Application		Package diagram
AD	Do detailed class and operation design	Class descriptions (other than business classes)	Class diagram, Statechart diagram, Activity diagram, Sequence

			diagram, Collaboration diagram
BO & AD	Do detailed business class and operation design	Business class descriptions	Class diagram, State-chart diagram, Activity diagram, Sequence diagram, Collaboration diagram
DB	Do database design	Database description	

The main roles of the team work intensively in this phase. They all make detailed specifications that are to be implemented in the next phase.

4.5 The Construction Phase

The object-oriented application development is a bottom-up activity. The entire application is constructed of small parts (components, classes, methods, etc.). This is a very challenging task, as all the pieces must fit together to make the application work smoothly.

A release of the whole system or parts of it (portions) is made available for testing. Programs are made and commented on in accordance with the style guide. On-line helps are ready for use. Scripts for the database are ready to be copied onto installation diskettes. All diagrams are updated in this phase, so that the implementation of the software and its descriptions fit each other. Table 5 attempts to show only that the component and deployment diagrams are in the main role.

Table 5. The Construction Phase.

Roles	Activities	Artifacts	UML Diagrams
US & AD	Finnish User manual and on-line help	User manual and on-line help	
AD	Implement user interfaces and reports	User interfaces and reports	Component diagram, Deployment diagram
AD	Implement classes and methods	Classes and methods (other than business classes)	Component diagram, Deployment diagram
BO & AD	Implement business classes and methods	Business classes and methods	Component diagram, Deployment diagram
DB	Implement database	Database	Deployment diagram

Again, the roles from each tier work to construct the application of the components.

4.6 The Testing Phase

Testing, and especially object-oriented testing, is an art of its own. The use cases are a good input when test cases for a system test are designed. The activity diagrams also facilitate the designing of system tests, since they depict the flows of work of users, which are in the focus of the system tests. The team carries out unit, integration and system testing in such a way that the work of each developer is tested by someone else (Table 6).

Table 6. The Testing Phase.

Roles	Activities	Artifacts	UML Diagrams
TE	Test according to test specifications	Test log and report	

4.7 The Deployment Phase

This is a crucial phase in tailored application development, as also in off-the-shelf product development, involving production of the installation diskettes and the instructions. Deployment planning has been started during the construction phase. The training of the users starts and the training material is made. The application is deployed according to the deployment plan. The deployment diagram can be used in this phase to show how the run-time system is constructed (Table 7).

Table 7. Deployment Phase.

Roles	Activities	Artifacts	UML Diagrams
AD	Produce installation discs and instructions		Deployment diagram
TL & EX	Deploy application, training and education material		

5. Lessons Learnt While Implementing The Roles

This chapter lists some observations made while the roles have been adapted to the organizations.

Organizational status of a developer

Although the roles of the team are fairly easy to 'sell' to IT organizations, they are sometimes hard to adopt. For instance, the status of an Analyst or a Designer in the organizations often implies that the person holding this role does not code anymore. However, one idea behind the object orientation has been to remove the boundaries between the phases, so that less information is lost. On the other hand, the iterative development process strives towards a development style where one person should be able to do all phases (analysis, design and coding). Iteration is actually a way to prevent information loss. Furthermore, Analysts usually start their career by coding, and coding itself should therefore cause no problem. Otherwise, the role of the Application Developer seems to fit nicely to the organizations, as one might expect. Use cases have been easy to learn both for Application Developers and Users. Object-oriented thinking was quite hard to adopt at the beginning of the object-oriented era, but is now accepted as part of the development process.

User involvement

User involvement in the development process has improved during the last few years. One reason is probably the increased teaching of Information Technology, but use cases and object-oriented thinking have also mitigated the communication problems between users and developers. With some training, users seem to be able to write and maintain use cases.

Role of the Business Object Developer

The role of the Business Object Developer has been accepted quite easily in IT organizations. This is quite obvious, since organizations have always had persons that are responsible for some business applications. In object-oriented client/server development, the person with this role is responsible for certain business objects (actually components), and those objects are only accessible via clear interfaces. Especially in cooperative environments (i.e., environments including both workstations in LAN and legacy systems in a main frame) the role has been easy to adopt. In one case, it took only a couple of months to make a repository for reusable objects (both business objects and others) and to organize the developers so that they knew which business objects they are responsible for, who the Business Object Developers are and who work for them (Application Developers).

Role of the Database Developer

The role of the Database Developer was introduced years ago (Kivisto 1992). However, the technology did not develop at the expected rate (Kivisto 1992a). Only recently have the major database vendors (for instance, Oracle and IBM) released databases that make the role of the Database Developer possible. With object-oriented databases this role has naturally been

possible all the time. At any rate, this role will gain more attention in the near future, and the IT organizations should prepare for it.

Development in small teams: control and management

Applications are nowadays developed in small teams. These teams need experienced Team Leaders who know how the application should be developed. Moreover, the Team Leader should be able to analyze, design and even code parts of the application. As pointed out earlier, the Team Leader is the head of the development activity whereas the Project Manager is the head of the whole project. This fact is emphasized in projects where the teams may be located in different places, even in different countries. These 'virtual' teams need a leader for the development process, rather than a project manager. For subcontractors this is quite a normal situation, but IT departments are also organizing themselves into teams.

Development in small teams: communication and conversation

In order to succeed in concurrent development, small teams need a common process model (such as the OOCS) and a common modeling language (such as the UML). Disciplined usage of these two things facilitates application development and conversations between teams.

6. Summary and Conclusions

The role issue is a less intensively studied field, although it is the people who make the applications. There are perhaps hundreds of development process models, but the role issue is usually ignored or discussed only briefly. This article started from the technological dimension and then moved on to the organizational dimension. The role issue was discussed from the object-oriented paradigm and the client/server architecture point of view and the roles were described. The third dimension was incorporated with the first two into a process model called OOCS. The rest of the article was devoted to the lessons learnt by organizations while adopting the role part of the OOCS. The adaptation of the OOCS process model to the needs of organizations has been briefly discussed elsewhere (Kivisto 1997).

The roles introduced in this article have been quite easy to explain to IT organizations. However, there are some obstacles that prevent their adoption. One of them is the prejudice that programming is less valuable than analysis or design. Another is the database technology, which only now begins to support the role of the Database Developer.

Now that the communication possibilities are constantly improving, the project group can be composed of virtual teams. To be successful, this project configuration requires a shared process model (such as the OOCS) and a shared notation (such as the UML). In addition, the roles within the teams and the teams' responsibilities must be generally accepted. With all these three dimensions in mind, the project has all the possibilities to succeed.

7. References

- Booch, G. (1994). Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, Redwood City, CA, 1994.
- Booch, G. (1995). Object Solutions: Managing the Object-Oriented Project. Addison-Wesley, Menlo Park, CA, 1995.
- Graham, I. (1995). Migrating To Object Technology. Addison-Wesley, Wokingham, 1995.
- Jaaksi, A. (1997). Object-Oriented Development of Interactive Systems. Thesis for Doctor of Technology. Tampere University of Technology, 1997.
- Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1992). Object-Oriented Software Engineering - A Use Case Driven Approach. Reading, MA: Addison-Wesley; New York:ACM Press, 1992.
- Kivisto, K. (1992). Object-Oriented Relational Databases - Databases of the Future. In the Precedings of NordData '92, 15.- 18. June 1992, Tampere, Finland.
- Kivistö, K. (1992a). The Database Programmer - A need for a new role in object-oriented application development. Proceedings of The Third International Conference in Information Systems Developers Workbench. September, 22-24, 1992, Gdansk, Poland.
- Kivisto, K. (1997). Team-Based Development of Object-Oriented Client/Server Applications: The Role Perspective. Licentiate thesis. Institute of Information Processing Science, University of Oulu, Finland, 1997.
- Kivisto, K. (1998). Considerations of and Suggestions for a UML-Specific Process Model. Proceedings of the UML98- Beyond the Notation, Mulhouse, France, June 2nd and 3rd, 1998.
- Lorenz, M. (1993). Object-Oriented Software Development: A Practical Guide. Prentice Hall, Englewood Cliffs, NJ, 1993.
- Microsoft Corporation (1993). Analysis and Design of Client/Server Systems. Course Material, 1993.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. (1991). Object-oriented modeling and design. Prentice Hall, Englewood Cliffs, NJ, 1991.
- UML Notation Guide, ver 1.1 (1997). Rational Software Co., 1997.
- UML Semantics, ver 1.1 (1997). Rational Software Co.,1997
- UML Summary, ver 1.1 (1997). Rational Software Co.,1997.