ADOPTING RUP (RATIONAL UNIFIED PROCESS) ON A SOFTWARE DEVELOPMENT PROJECT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

TUFAN TAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2009

Title of the Thesis : **Adopting RUP (Rational Unified Process) on a Software Development Project**

Submitted by **Tufan TAŞ**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

[illegible]

Prof. Dr. Yahya K. BAYKAL
Acting Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Mehmet R. TOLUN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Ziya Aktaş	[illegible]
Prof. Dr. Ziya AKTAŞ Co-Supervisor	Prof. Dr. Mehmet R. TOLUN Supervisor

Examination Date : 03.09.2009

Examining Committee Members

Prof. Dr. Mehmet R. TOLUN	(Çankaya Univ.)	[illegible]
Prof. Dr. Ziya AKTAŞ	(Başkent Univ.)	Ziya Aktaş
Asst. Prof. Dr. Abdül Kadir GÖRÜR	(Çankaya Univ.)	[illegible]
Instructor Dr. Ali Rıza AŞKUN	(Çankaya Univ.)	[illegible]

STATEMENT OF NON PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	: Tufan TAŞ
Signature	: [illegible]
Date	: 03/09/2009

ABSTRACT

ADOPTING RUP (RATIONAL UNIFIED PROCESS)
ON A
SOFTWARE DEVELOPMENT PROJECT

TAŞ, Tufan
M.Sc., Department of Computer Engineering
Supervisor: Prof. Dr. Mehmet R. TOLUN
Co-Supervisor: Prof. Dr. Ziya AKTAŞ

September 2009, 218 pages

This thesis analyzes the process of applying Rational Unified Process (RUP) successfully on a software development project step by step. Many software development projects today have a tendency to fail on some level. Even though they may not fail entirely, they might be completed with schedule delays, budget overrun or with poor quality that do not meet the requirements of customers because of poor management and lack of necessary documentation of the project. Applying RUP avoids these major problems in a project by developing set of work products which depict the essentials of the system from requirements to detailed design before the system could be implemented. However, software development teams have an overall attitude that RUP becomes less agile and too rigid as the size of projects get smaller. The thesis will also try to prove that this opinion is not true by using tools Rational Method Composer (RMC) and Rational Software Modeler (RSM) to successfully complete the project.

ÖZ

RUP YÖNTEMİNİN
BİR YAZILIM GELİŞTİRME PROJESİ
ÜZERİNDE UYGULANMASI

TAŞ, Tufan
Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Prof. Dr. Mehmet R. TOLUN
Ortak Tez Yöneticisi: Prof. Dr. Ziya AKTAŞ

Eylül 2009, 218 sayfa

Bu tez Rational Unified Process (RUP) yönteminin bir yazılım geliştirme projesinde başarıyla uygulanma sürecini adım adım incelemiştir. Günümüzde birçok yazılım geliştirme projesi bazı düzeylerde başarısız olma eğilimindedir. Projeler tamamen başarısız olmasa bile, projenin kötü yönetimi ve gerekli dökümantasyonun eksik olması nedeniyle takvim gecikmeleriyle, bütçe aşımıyla ya da müşterilerin gereksinimlerini karşılamayan düşük kalitede yazılım ürünleri görülmektedir. Bir projede, sistemin uygulanmasından önce gereksinimlerden detaylı tasarımına kadar esaslarını gösteren bir seri iş ürünü geliştirilerek RUP'nin uygulanması ile bu tür sorunların oluşumu engellenir. Bununla birlikte, yazılım geliştirme gruplarının proje boyutları küçüldükçe RUP'nin daha az çevik ve çok katı bir hal aldığına dair genel bir kanı vardır. Bu çalışma, başarıyla bir projeyi tamamlamak için Rational Method Composer (RMC) ve Rational Software Modeler (RSM) araçlarını kullanarak bu görüşün de doğru olmadığını kanıtlamaya çalışmıştır.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Prof. Dr. Mehmet R. TOLUN and co-supervisor Prof. Dr. Ziya AKTAŞ for their guidance, advices, criticism, encouragements, and insight throughout the research. Thanks are also extended to Dr. Alan Brown, Mrs. Jale Akyel and Mr. Erkan Özkan, all IBM members, for their encouragement and support they provided to co-supervisor Prof. Dr. Ziya Aktaş during the supervision of the Thesis. The last, but not the least, support of IBM Turkey by providing Rational Method Composer (RMC), Rational Software Modeler (RSM) and Rational Rose software packages during the thesis is also acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background of the Problem

An important part of a software development project is documentation. Documentation is a process of making a record of information related to the corresponding process being documented. Documentation involves recording information such as functional and technical requirements, standards, design and implementation procedures, testing process and results, operation procedures, support arrangements related to the software and the data used by the software. Documentation is not just a process of recording the information; it also includes the process of making the information available in such a form that the targeted users of the documentation can benefit from it. It is important to remember that documentation is an ongoing process until the software becomes completely available and being used.

Over many years of software development effort, project teams have understood that the most beneficial way to obtain maximum success is about well documentation of the project. We know that each phase in a software development life cycle should have its generic document. Probably the most common problems encountered on a software development project are related to how much and what type of documentation should exist during the development of the required system.

Most programmers tend to forget about the documentation when they need to start with a new software development project. Unless the software managers or customers insist to have the documentation, the documentation will not exist at all. On the other hand we also know that forcing software engineers to produce documents, many times they produce related documents after they finish the coding and make the system available. This kind of attitude collide the structure of a reasonable software development. Especially for programmers, creation in coding and implementation is much more exciting than documenting their systems. As a result, we obtain documentations with low qualities which cause really serious problems during development and maintenance of the system.

On the other hand, documentation standard is important to be enforced in all software projects because its purpose is to communicate only necessary and critical information, not to communicate all information. However, most organizations do not employ any documentation standard. Thus programmers who produce the documentation do not follow any formal guidelines and this causes many different formats in the code to be produced.

Also, comprehension of an existing software system is the most expensive task during the software maintenance process because it includes reading documents, scanning its source codes and understanding the change to be made. Maintenance is one of the areas that software engineers spend their time. However this task becomes most costly and laborious activity because of the lack of needed documentations. Due to the lack of information about the existing system, engineers waste their time to understand the system.

Nowadays, most of the large and medium sized projects are managed and developed successfully by the help of a software engineering process called Rational Unified Process (RUP) that is the widely used methodology on software development. The Rational Unified Process provides a disciplined approach to assigning tasks and responsibilities within a development organization so it ensures the production of high quality software that meets the needs of its end-users, within a predictable schedule and budget. The Rational Unified Process is supported by tools, which automate large parts of the process. They are used to create and maintain the various artifacts of the software engineering process such as visual modeling, programming, testing and many other documentation that are necessary for the project. By using RUP almost all possible problems mentioned in this section are solved for the large sized projects. However, according to a number of software engineers, RUP cannot be agile and it is too rigid for small projects. Because of this, RUP is not considered as an option within teams which causes many management and development problems. So many small projects still have the mentioned problems because of the lack of their necessary documentation work. In these situations some of these projects cannot be concluded or made successfully available. It is dramatic to note here that this misconception is harmful and RUP may be adopted for agile systems as well.

1.2 Statement of the Problem

This research is intended to deal with the problems as discussed in Section 1.1 that are related to software development projects. Project team members cannot find a case study that represents how RUP can be applied on a small project successfully. A sample project that resolves these issues would prove that small projects can also be managed using RUP. So our main goal is to adopt RUP methodology on a small software development project. Case problem to be developed may be defined as the automation of existing manual reservation system of Çankaya University Library.

1.3 Objective of the Study

The objectives of the study may be listed as follows:

- To define a case problem as a small sized project;
- To define project lifecycle on determined small sized software development project by applying RUP methodology;
- To represent all activities step by step performed based on RUP approach;
- To represent all necessary work products and perform a proper documentation;
- To produce working software on time that fits the captured requirements during the development of software project.

1.4 Organization of the Study

This thesis is organized into eleven chapters namely, **Chapter 1:** Introduction, **Chapter 2:** Unified Modeling Language (UML), **Chapter 3:** Rational Unified Process (RUP), **Chapter 4:** IBM Rational Tools, **Chapter 5:** Case Study, **Chapter 6:** Application, **Chapter 7:** Inception Phase, **Chapter 8:** Elaboration Phase, **Chapter 9:** Construction Phase, **Chapter 10:** Transition Phase, **Chapter 11:** Summary and Conclusions.

Chapter 1 is the introduction of the study and consists of the background of the problem, a statement of the problem that is being focused in this study and objective of the study. Chapter 2 offers literature review concerning with Unified Modeling Language emphasizing the importance of its version 2.0. All UML 2.0 diagrams are reviewed with small samples. Chapter 3 offers literature review concerning with Rational Unified Process. RUP lifecycles and disciplines are reviewed emphasizing the importance of iterations. Chapter 4 offers fundamental information about IBM tools that are used in software development project by focusing on their capabilities and features. Also their key concepts are elaborated. Chapter 5 defines a case study problem with its existing problem and solution to that problem which will be used for the software development project. Chapter 6 introduces the preparation of the environment and related IBM tools briefly for software development. Chapter 7 describes the adaptation of inception phase to the software development project step by step. Chapter 8 describes the adaptation of elaboration phase to the software development project step by step. Chapter 9 describes the adaptation of construction phase to the software development project step by step. Chapter 10 describes the adaptation of transition phase to the software development project step by step. Chapter 11 provides the summary of the thesis and includes the conclusions.

CHAPTER 2

UNIFIED MODELING LANGUAGE (UML)

2.1 Modeling Principles

Professionals such as business analysts, engineers, scientists and others who build complex structures or systems are first creating models of what they build [e.g. Cernosek and Naiburg, 2004]. Some of these models are physical and some of them are less tangible. As stated by Booch, Rumbaugh and Jacobson [2005], the use of modeling has a rich history in all the engineering disciplines. That experience suggests four basic principles of modeling:

- The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
- Every model may be expressed at different levels of precision.
- The best models are connected to reality.
- No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.

While developing software, developers need a better understanding of what they are building, and modeling is an effective approach to do that. By modeling software, developers can understand the system better and gain some advantages on development. As stated by Cernosek and Naiburg [2004] some of the benefits and the importance of modeling are listed as follows:

- Create and communicate software designs before committing additional resources.
- Trace the design back to the requirements, helping to ensure that they are building the right system.
- Practice iterative development, in which models and other higher levels of abstraction facilitate quick and frequent changes.

2.2 What is UML?

UML stands for **U**nified **M**odeling **L**anguage which is a family of graphical tools. It is important to understand that UML is a standard language that is used to write software blueprints [Pilone and Pitman, 2005]. Since UML is defined as a language, it has both syntax and semantics as all languages have. When you model a potential problem or a solution in UML, there are some rules regarding how the elements can be put together, how the interaction between them can be created and what it means when they are organized in a certain way.

When UML is applied to software, it attempts to bridge the gap between the original idea for a piece of software and its implementation. The UML may be used for visualization, specification, architecture design, construction, simulation, testing and documentation of the artifacts of software intensive systems. UML was originally developed with the idea of promoting communication and productivity among the developers of object-oriented systems. The most powerful characteristic of UML is that it makes inroads into every type of system and software development.

The most important concept about UML is that it is not a methodology, so it does not require any formal work products. As a relevant methodology one may refer to RUP (Rational Unified Process) as to be described later in the thesis. UML provides several types of diagrams that, when used within a given methodology, increase the ease of understanding of the problem domain and its proposed solution of an application under development [Bell, 2003]. Each model comprises one or more diagrams with supporting documentation and descriptions. The number and size of these diagrams and documentation depends on complexity of systems that is to be developed.

2.3 UML 2.0

The first version of UML, UML 1.0, allowed people to communicate designs unambiguously, convey the essence of a design, and even capture and map functional requirements to their software solutions. Versions of UML 1.x were designed as a unified language for humans. When it became important for models to be shared between machines specifically between Computer Aided Systems Engineering (Case) tools or Computer Aided Software Engineering (CASE) tools, UML 1.x was again found wanting. Underlying notation rules and meta-model of UML 1.x were not formally defined enough to enable machine-to-machine sharing of models [Hamilton and Miles, 2006]. This necessity emerged the revision of present versions. Each revision of UML tried to recover the problems identified within the previous versions. Today UML 2.0 is the most cleanest and compact version.

UML 2.0 is familiar to people who were already using UML 1.x. Many of the original diagrams and associated notations have been retained and extended in UML 2.0. With Version 2.0, UML has evolved to support the new challenges that software and system modelers face today. As stated by Selic [2005], the new developments in UML 2.0 are listed as follows:

- A significantly increased degree of precision in the definition of the language to support the higher levels of automation required for model driven development (MDD).
- An improved language organization which is characterized by a modularity that not only makes the language more approachable to new users, but also facilitates inter working between tools.
- Significant improvements in the ability to model large-scale software systems that new hierarchical capabilities were added to the language to support software modeling at arbitrary levels of complexity.
- Improved support for domain-specific specialization to allow simpler and more precise refinements of the base language.
- Overall consolidation, rationalization, and clarifications of various modeling concepts which is resulted in a simplified and more consistent language.

2.4 UML Diagrams

UML allows people to develop several different types of visual diagrams which represent various aspects of a system. UML categorizes its diagrams into two as structural diagrams and behavioral diagrams:

- **Structural Diagrams:** Used to show the building blocks of your system. Class diagram, object diagram, composite structure diagram, deployment diagram, component diagram and package diagram are structural diagrams.
- **Behavioral Diagrams:** Used to show how your system responds to requests or otherwise evolves over time. Activity diagram, use case diagram and state machine diagram are behavioral diagrams.

In addition to Structural Diagrams and Behavioral Diagrams there is a third group of diagrams that are called Interaction Diagrams.

- **Interaction Diagrams:** Actually a type of behavioral diagram which are used to depict the exchange of messages within collaboration en route to accomplishing its goal. Interaction overview diagram, sequence diagram, communication diagram and timing diagram are interaction diagrams.

This group of diagrams are explained briefly and exemplified by an airline reservation system [e.g. IBM Rational University, 2004] in the next sections.

2.4.1 Class Diagram

Class diagrams are the most common diagrams used in modeling object-oriented systems. A class diagram shows the existence of classes and their relationships between them in the logical design of a system [Fowler, 2003]. It gives you a static picture of the pieces in the system and of the relationships between them. Class diagrams also show the properties and operations of a class and the constraints that apply to the way objects are connected. Developers use class diagrams to actually develop the classes. Analysts use class diagrams to show the details of the system [Boggs and Boggs, 2002].

Class diagrams are the backbone of the UML [Fowler, 2003], so you will use them all the time. The trouble with class diagrams is that they are so rich; they can be overwhelming to use. The biggest danger with class diagrams is that you can focus exclusively on structure and ignore behavior. As an example, Figure 2.1 depicts a class diagram related to flight reservation.

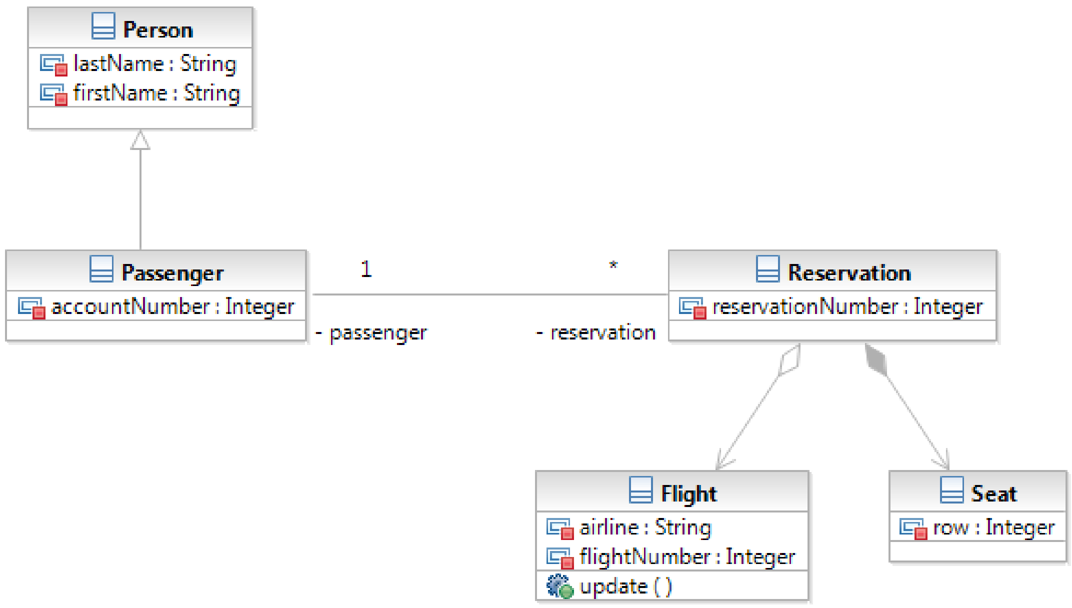


Figure 2.1 Class Diagram

Classes are shown as rectangle boxes with three compartments. First part contains the name of the class; second part contains attributes which are details of class and third part contains operations which are features of classes that specify how to invoke a particular behavior.

Relationships between classes are listed as follows:

- **Dependency:** Weakest relationship between classes. One class uses, or has knowledge of, another class. Read as "…uses a…".
- **Association:** Stronger than dependencies. Specifying objects of one thing are connected to objects of another. Read as "...has a...".
- **Aggregation:** Stronger version of association. A special form of association that models a whole-part relationship between the whole and its parts.
- **Composition:** Strong relationship between classes. Composition is a form of aggregation, with strong ownership and coincident lifetime as part of the whole.
- **Generalization:** The target of the relationship is a general, or less specific, version of the source class or interface. Read as "...is a...".

2.4.2 Object Diagram

Object Diagrams provide a snapshot of system execution at a point in time using objects and links [IBM Rational University, 2004]. It models the instances of things contained in class diagrams. An object diagram is a variant of a class diagram and basically uses the same notation with the difference being that the object diagram shows a set of instances and not actual classes. Object diagrams can be used to show an example configuration of objects. As an example Figure 2.2 depicts an object diagram related to flight reservation.

Object diagrams use notation which is almost identical to class diagrams, but they present the objects and their relationships at a particular point in time. Objects are shown with a rectangle. Within object diagrams, the title is underlined to show that it is an instance of a class. Links between objects on an object diagram show that the two objects can communicate with each other.

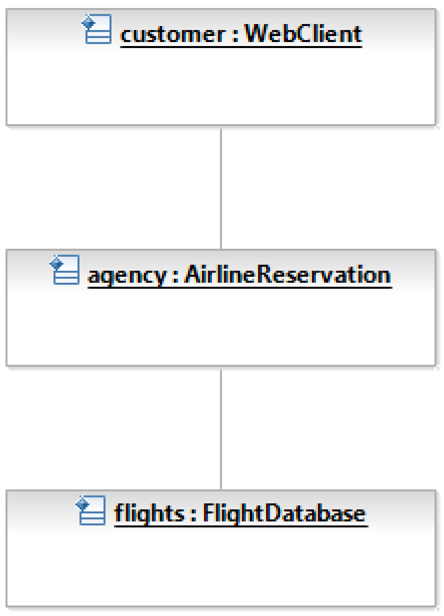


Figure 2.2 Object Diagram

2.4.3 Composite Structure Diagram

One of the most significant new features in UML 2 is the ability to hierarchically decompose a class into an internal structure [Fowler, 2003]. This allows you to take a complex object and break it down into parts that provide to understand and manage complex systems much easier.

Composite structure diagrams are used to depict the internal structure of a classifier such as a class, component, or use case, including the interaction points of the classifier to other parts of the system [Ambler, 2005b]. They are also used to explore how a collection of cooperating instances achieves a specific task or set of tasks and describe a design or architectural pattern or strategy. As an example Figure 2.3 depicts a composite structure diagram related to flight reservation.

Internal structures show the parts contained by a class and the relationships between the parts. Ports show how a class is used on your system with ports. A port may appear either on a contained part representing a port on that part, or on the boundary of the class diagram, representing a port on the represented classifier itself. Collaborations show design patterns in software that is being developed and, more generally, objects cooperating to achieve a goal.

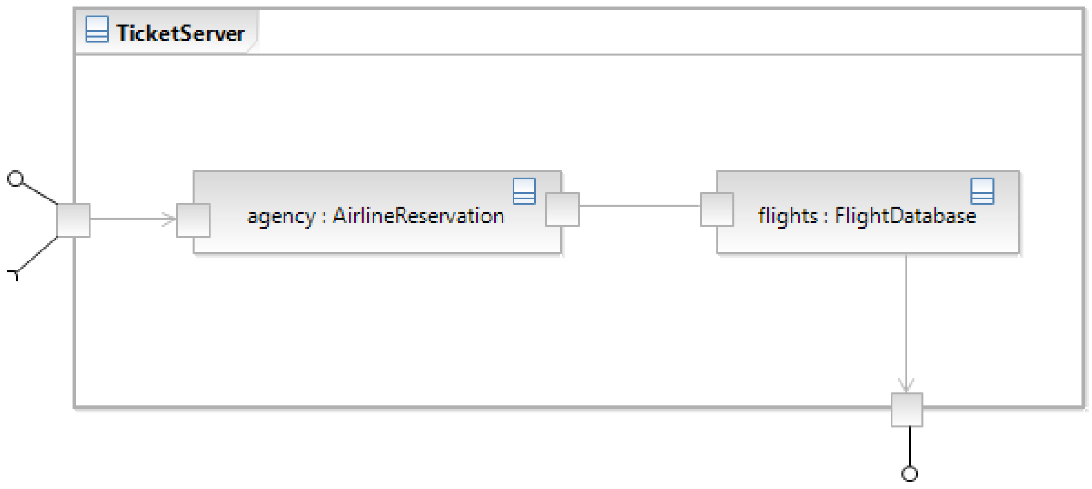


Figure 2.3 Composite Structure Diagram

2.4.4 Deployment Diagram

Deployment diagrams are used to model the physical aspects of an object-oriented system. Deployment diagrams show a system's physical layout, revealing which pieces of software run on what pieces of hardware [Ambler, 2005b]. Software elements are typically manifested using artifacts and are mapped to the hardware or software environment that will host them which are called nodes.

The Deployment diagram is used by the project manager, users, architect, and deployment staff to understand the physical layout of the system and where the various subsystems will reside [Boggs and Boggs, 2002]. As an example Figure 2.4 depicts a deployment diagram related to flight reservation.

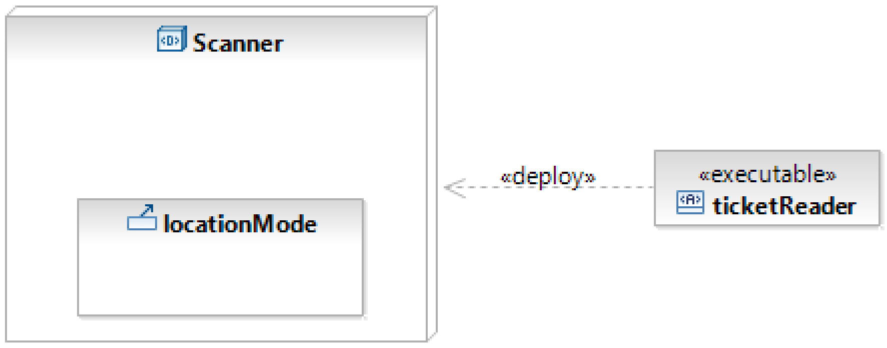


Figure 2.4 Deployment Diagram

Deployment diagrams use nodes to represent hardware in your system. Physical software files are modeled with an artifact. An artifact is deployed to a node, which means that the artifact is installed on the node. An artifact manifests the component if an artifact is the physical actualization of a component. An artifact can manifest not just components but any packageable element, such as packages and classes.

2.4.5 Component Diagram

Component diagrams are used to show a physical view of the model, as well as the software components in the system and the relationships between them [Ambler, 2005b]. Component diagrams are used when the system is divided into components and to show their interrelationships through interfaces or the breakdown of components into a lower-level structure. Development can be changed quickly when a component-based architecture is used. Because of switching components readily, or modified, without compromising overall system integrity.

Component diagrams are used by whoever is responsible for compiling the system. The diagrams will tell this individual in what order the components need to be compiled [Boggs and Boggs, 2002]. The diagrams will also show what run time components will be created as a result of the compilation. As an example Figure 2.5 depicts a component diagram related to flight reservation.

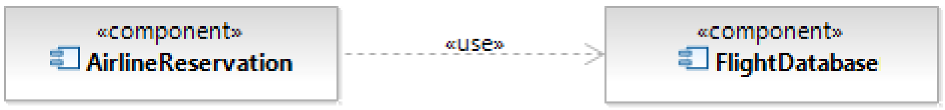


Figure 2.5 Component Diagram

A component is drawn as a rectangle with the <<component>> stereotype and an optional tabbed rectangle icon. A provided interface of a component is an interface that the component realizes. Other components and classes interact with a component through its provided interfaces. A provided interface of a component describes the services provided by the component. A required interface of a component is an interface that the component needs to function. A required interface declares the services that a component will need.

2.4.6 Package Diagram

A package is used to take any construct in the UML and group its elements together into higher level units [Fowler, 2003]. Most common use of a package diagram is to group classes. Nearly all UML elements can be grouped into packages, including packages themselves. Each package has a name that scopes each element in the package. Package diagrams describe the hierarchical organization of model elements. Package diagrams extremely useful on larger scale systems to get a picture of the dependencies between major elements of a system. As an example Figure 2.6 depicts a package diagram related to flight reservation.

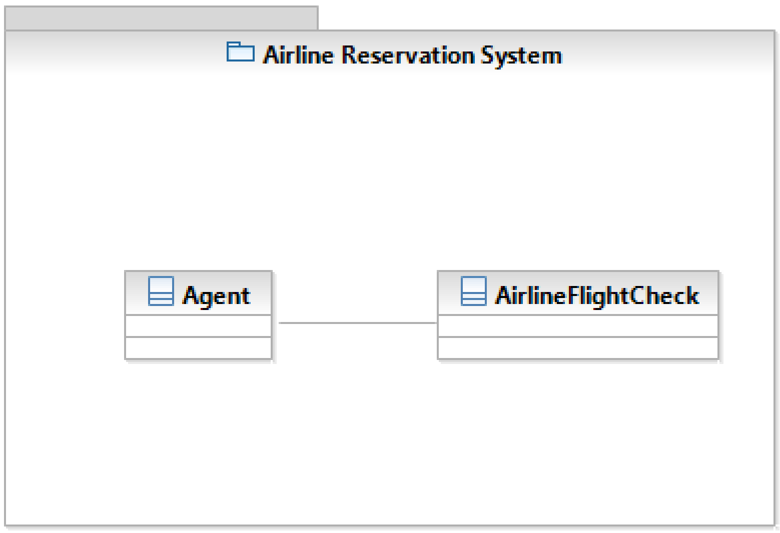


Figure 2.6 Package Diagram

Packages organize UML elements, such as classes, and the contents of a package can be drawn inside the package or outside the package. If the contents of a package are drawn outside of it then they are attached by a line to the package. Elements in a package may have public or private visibility. Elements with public visibility are accessible outside the package. Elements with private visibility are available only to other elements inside the package.

2.4.7 Activity Diagram

Activity diagrams are specialization of state machine diagrams which focus on the execution and flow of the behavior of a system. Activity diagrams apply much more than just software modeling [Pilone and Pitman, 2005] that they play roles in many other areas. They may be used in business modeling to show the business workflows or may be used in requirements gathering to illustrate the flow of events through a use case. These diagrams define where the workflow starts and ends, what activities occur during the workflow, and in what order the activities occur. Activity diagrams capture activities that are made up of smaller actions. Activity diagrams are used to model the dynamic aspects of a system. They also support and encourage parallel behavior that is a critical point to understand the system behavior. Figure 2.7 depicts an activity diagram for part of an airline reservation system.

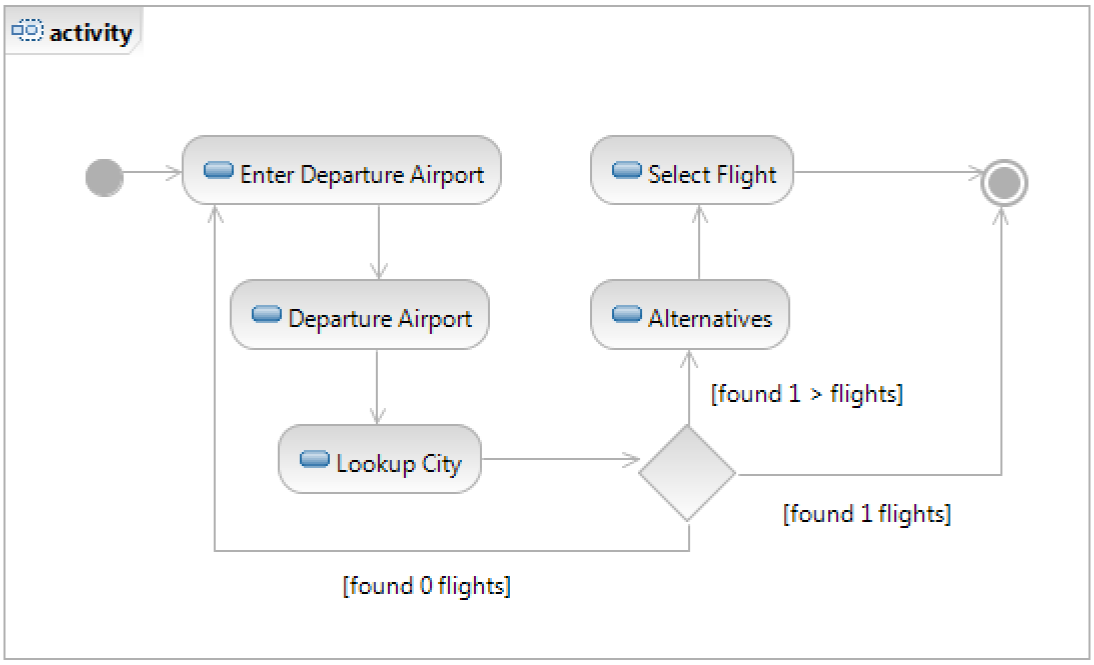


Figure 2.7 Activity Diagram

In activity diagrams there are initial and final nodes. Actions are located between these nodes. Actions are active steps in the completion of a process. Decisions are used when you want to execute a different sequence of actions depending on a condition. Decisions are drawn as diamond-shaped nodes with one incoming edge and multiple outgoing edges.

2.4.8 Use Case Diagram

A use case diagram describes a system's functional requirements in terms of use cases and the persons or things invoking the functionality referred as actors. The most important role of a use case diagram is to communicate the system's behavior to the end user [IBM Rational University, 2004] so the model must be easy to understand. Sometimes business use case diagrams could be used before modeling use case diagrams. Business Use Case diagrams are used to represent the functionality provided by an organization as a whole [Boggs and Boggs, 2002]. Business use case diagrams are not concerned with what is automated, but use case diagrams focus on just the automated processes.

A use case diagram is a valuable tool to help understand the functional requirements of a system. Use case diagrams are important for visualizing, specifying, and documenting the behavior of an element. A big danger of use cases is that people make them too complicated and get stuck. As an example Figure 2.8 depicts a use case diagram to describe the flight reservation for a customer.

Major concepts in use case diagrams are defined as follows:

- **Actor:** Anyone or anything that interacts with the system being built that is external to the system. It represents a coherent set of roles that one plays when interacting with use cases. External entities, actors that are depicted in the figure can also be depicted using simple stick figures.
- **Use case:** Describes a sequence of events, performed by the system that yields an observable result of value to a particular actor. It illustrates how an actor might use the system. An ellipse shape is used.

- **System Boundary:** Includes all use cases and excludes actors. It is useful when determining the scope and assignment of responsibilities when designing a system, subsystem or component.

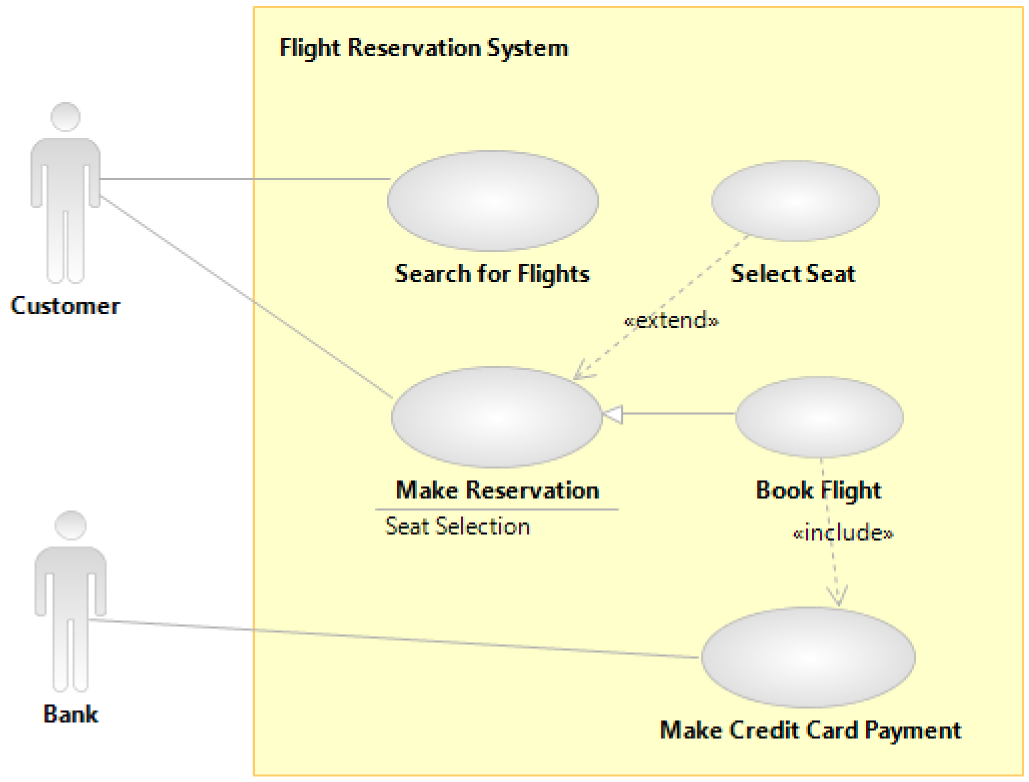


Figure 2.8 Use Case Diagram

A common problem with use cases is that by focusing and the interaction between a user and the system, you can neglect situations in which a change to a business process may be the best way to deal with the problem [Fowler, 2003]. It causes the terms appear as system use case which is an interaction with the software and business use case which discusses how a business responds to an event.

2.4.9 State Machine Diagram

State machine diagrams show the behavior of a system. State machine diagrams provide a way to model the various states in which an object can exist. State machine diagrams can be used to model the behavior of a class, subsystem, or entire application [Pilone and Pitman, 2005]. It is typically used to model the discrete stages of an object's lifetime.

State machine diagrams are not created for every class since they are used only for very complex classes. If an object of the class can exist in several states, and behaves very differently in each of the states, you may want to create a state machine diagram for it to understand the behavior of the class in more details.

State machine diagrams are good at describing the behavior of an object across several use cases. State machine diagrams are not very good at describing behavior that involves a number of objects collaborating [Fowler, 2003]. As an example Figure 2.9 depicts a state machine diagram as part of an airline reservation system.

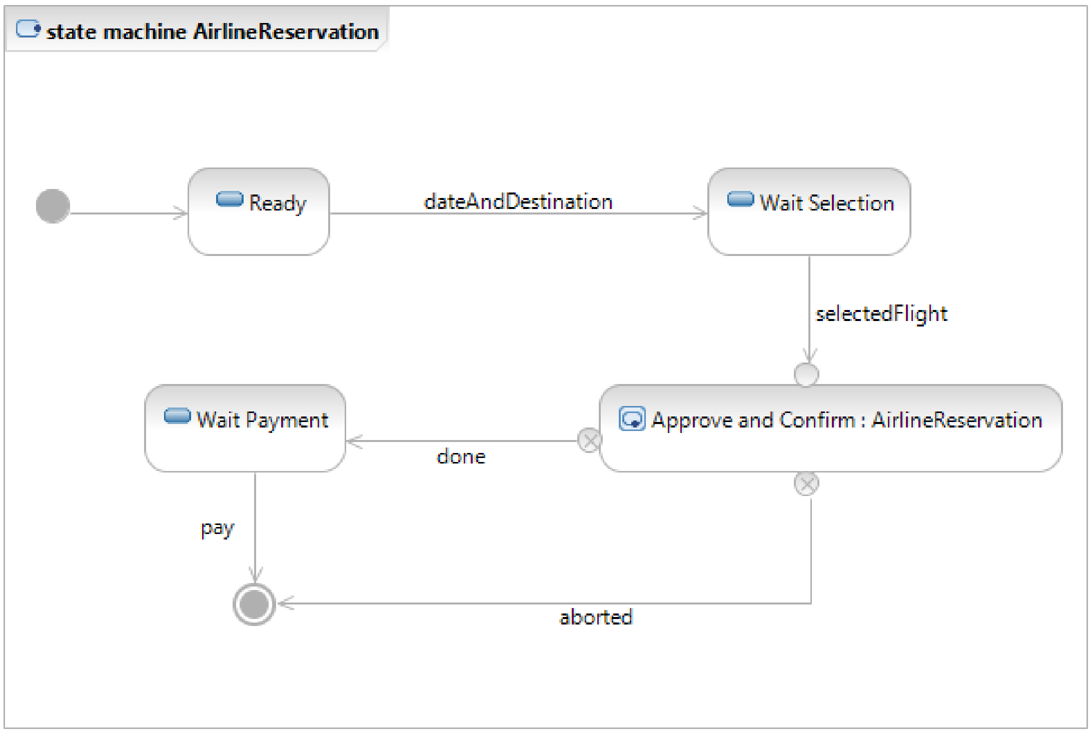


Figure 2.9 State Machine Diagram

A state diagram consists of states, drawn as rounded cornered rectangles, and transitions, drawn as arrows connecting the states. A transition represents a change of state, or how to get from one state to the next. A state is active when entered through a transition, and it becomes inactive when exited through a transition. State diagrams usually have an initial state and a final state, marking the start and end points of the state machine.

2.4.10 Interaction Overview Diagram

The purpose of the interaction overview diagram is to visualize the different options that exist for a given interaction [IBM Rational University, 2004]. Interaction overview diagrams represent interactions using a simplification of the activity diagram notation. Interaction overview diagrams can help you visualize the overall flow of control through a diagram; however, they do not show detailed message information.

Interaction overview diagrams are used to overview the flow of control within a business process, overview the detailed logic of a software process and connect several diagrams together. As an example Figure 2.10 depicts an interaction overview diagram related to flight reservation.

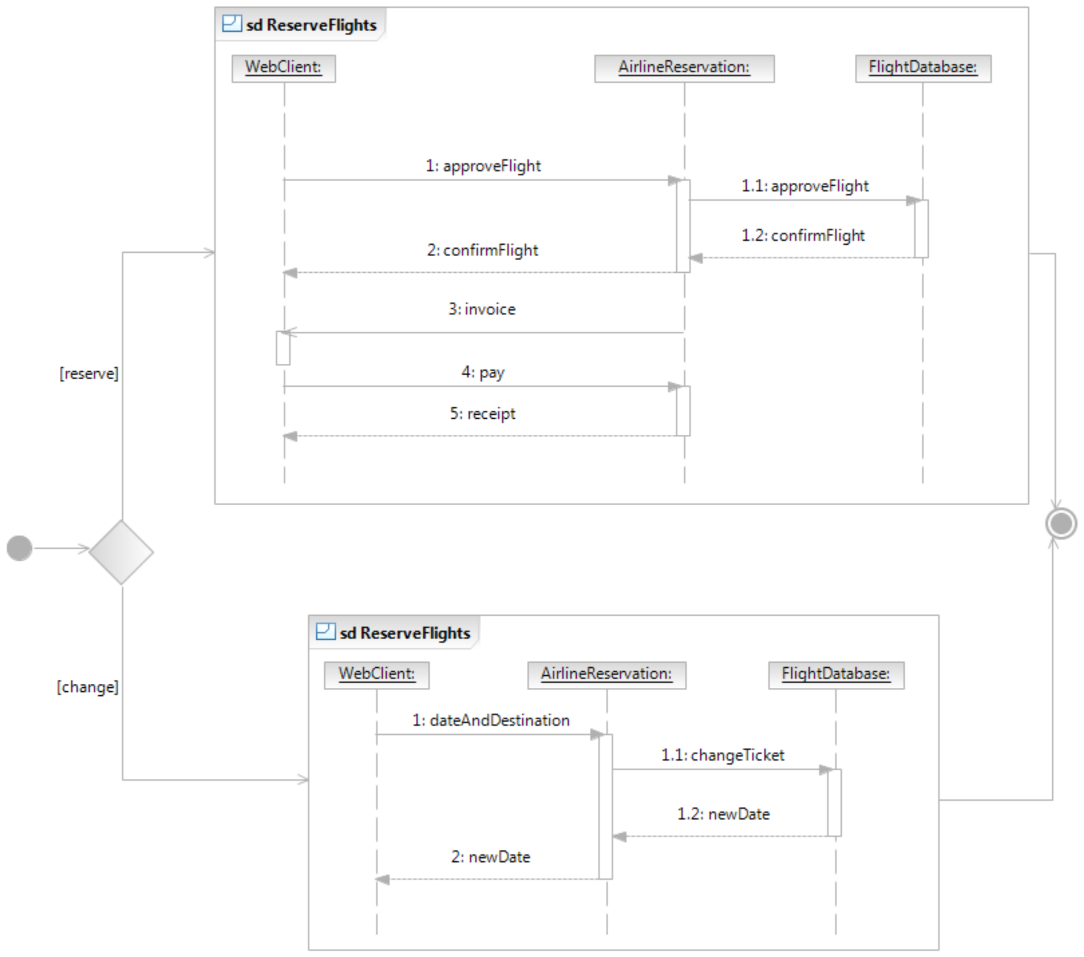


Figure 2.10 Interaction Overview Diagram

Individual interactions are placed on an interaction overview diagram as though they were actions as on an activity diagram (see Section 2.4.7). Similar to an activity diagram, the interaction overview begins with an initial node and ends with a final node. Control flows between these two nodes and passes through each of the interactions in between.

2.4.11 Sequence Diagram

A sequence diagram captures the behavior of a single scenario [Fowler, 2003]. The diagram shows a number of example objects and the messages that are passed between these objects within a use case. It shows the flow of functionality through a use case. Sequence diagrams are used to look at the behavior of several objects within a single use case.

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. Objects are arranged in a chronological timing order. It shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other. The most important characteristic of a sequence diagram is using time ordering between objects.

Sequence diagrams are particularly important to designers because they clarify the roles of objects in a flow and provide basic information for determining class responsibilities and interfaces. As an example Figure 2.11 depicts a sequence diagram related to flight reservation.

A sequence diagram is made up of a collection of participants. Time runs down the page on a sequence diagram in keeping with the participant lifeline. Time on a sequence diagram is all about ordering, not duration. When a message is passed to a participant the receiving participant is said to be active. Also participants do not necessarily live for the entire duration of the interaction of a sequence diagram. Participants can be created and destroyed according to the messages that are being passed. UML 2.0 provides sequence fragments that are used for managing complex interactions.

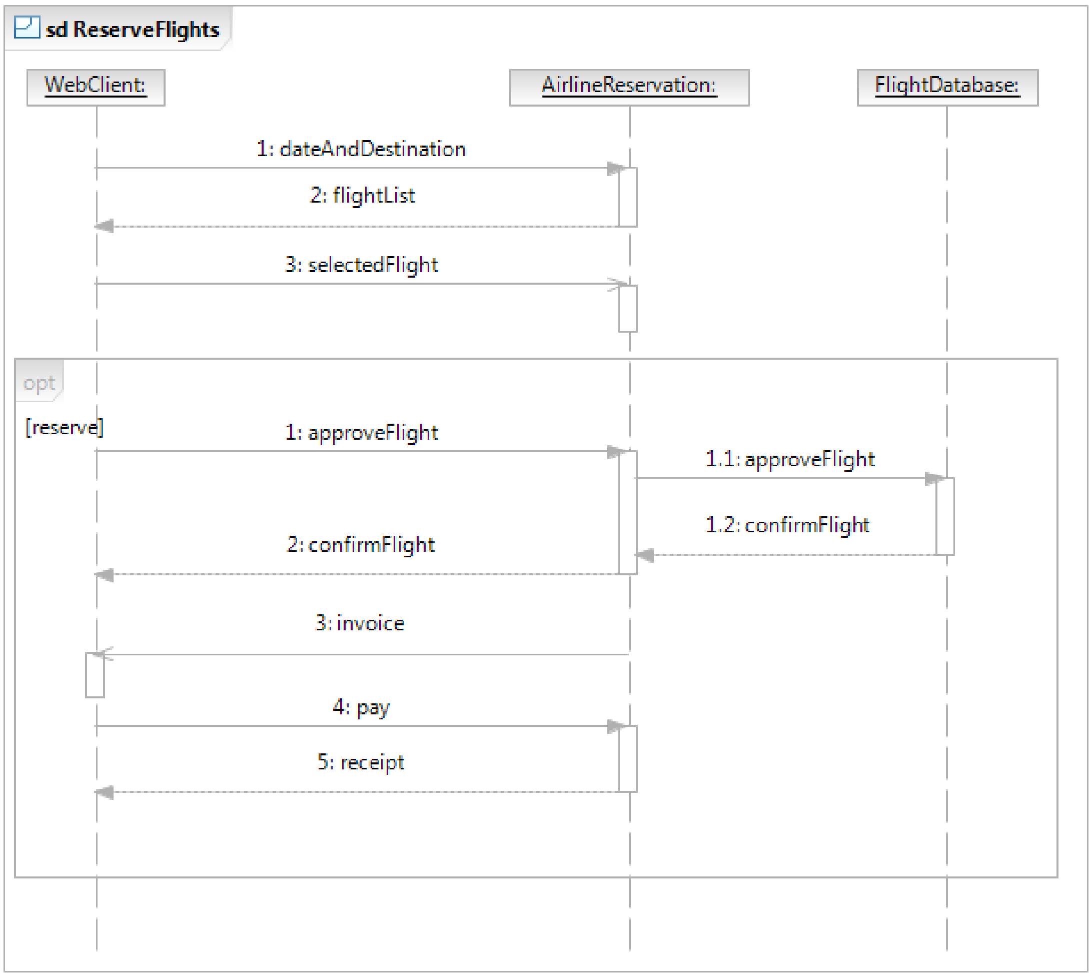


Figure 2.11 Sequence Diagram

2.4.12 Communication Diagram

Communication diagrams show exactly the same information as the sequence diagrams [Boggs and Boggs, 2002] with no time ordering. A communication diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Communication diagrams show the same information as the sequence diagrams however, they are used in different areas as quality assurance engineers and system architects look at these diagrams to see the distribution of processing between objects.

The main question with communication diagrams is when to use them rather than sequence diagrams. A rational approach [Fowler, 2003] says that sequence diagrams are better when you want to emphasize the sequence of calls and that communication diagrams are better when you want to emphasize the links. As an example Figure 2.12 depicts a communication diagram related to flight reservation.

Communication diagrams are much simpler than sequence diagrams that are made up of participants and links. Messages are passed along the link between participants without any time ordering between them.

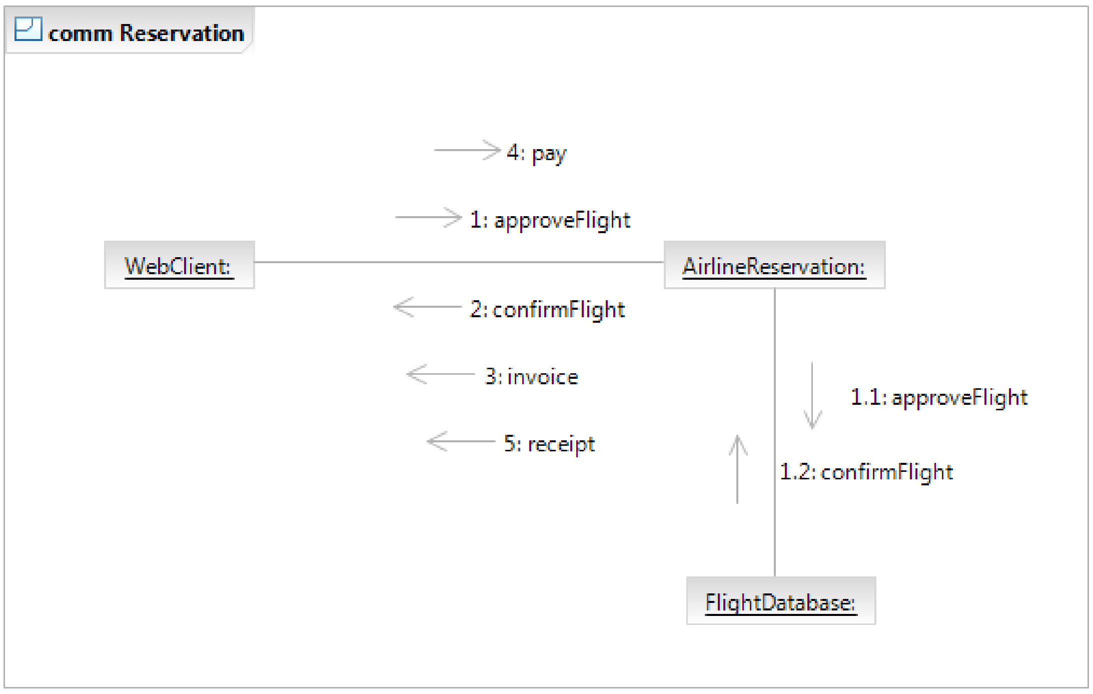


Figure 2.12 Communication Diagram

2.4.13 Timing Diagram

Timing diagram is a new addition to UML. Timing diagrams are a special representation of interactions that focus on the timing of events over the life of objects [Pilone and Pitman, 2005]. Timing diagrams are most often used with real-time or embedded systems. A timing diagram is useful for showing the interaction of objects and the timing constraints between state changes for those objects along a precise timing axis. As an example Figure 2.13 depicts a timing diagram related to flight reservation.

The names of the main participants involved in an interaction are written vertically on the left hand side of a timing diagram. During an interaction, a participant can exist in any number of states. A participant is said to be in a particular state when it receives a message. States are written horizontally on a timing diagram and next to the participant that they are associated with.

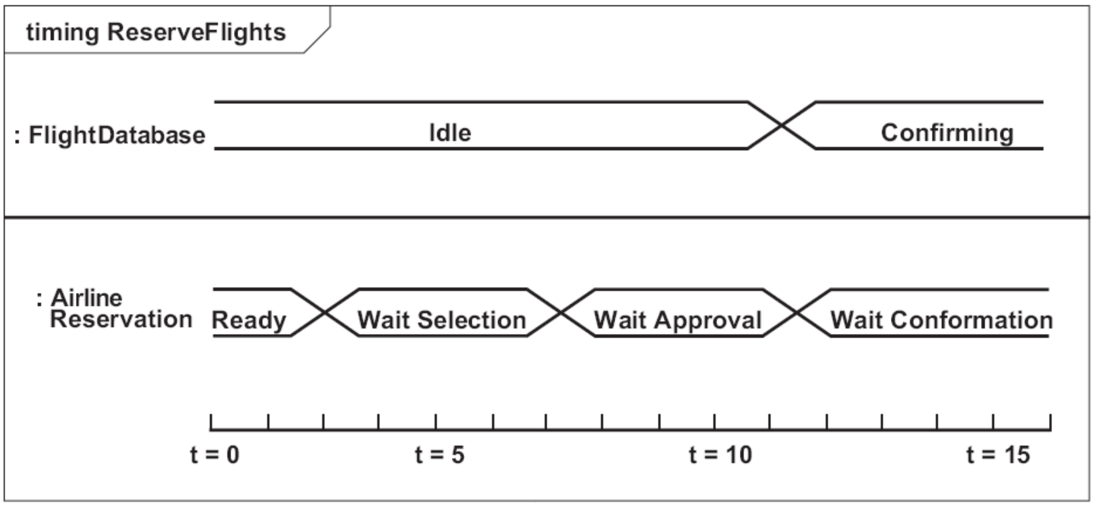


Figure 2.13 Timing Diagram

2.5 The Concept View of a System

The concept view of a system helps modelers to convey the correct information depending on goals. In modeling 4+1 views of a system is used. The 4+1 notation represents four distinct views of a system and one overview of how everything fits together. These four views are design, deployment, implementation and process view. The four distinct views of a system are brought together with a use case view.

The **design view** describes the representation of the problem domain and how the software will be built to address it. The design view typically does not address how the system will be implemented or executed [Pilone and Pitman, 2005].

The **deployment view** captures how a system is configured, installed, and executed. The deployment view captures how the physical layout of the hardware communicates to execute the system, and can be used to show failover, redundancy, and network topology [Pilone and Pitman, 2005].

The **implementation view** emphasizes the components, files, and resources used by a system. Typically the implementation view focuses on the configuration management of a system [Pilone and Pitman, 2005].

The **process view** of a system is intended to capture concurrency, performance, and scalability information [Pilone and Pitman, 2005].

The **use case view** of a system contains the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. This view does not really specify the organization of a software system. Rather, it exists to specify the forces that shape the architecture of the system that is to be developed [Booch, Rumbaugh and Jacobson, 2005].

CHAPTER 3

RATIONAL UNIFIED PROCESS (RUP)

3.1 Overview of the Rational Unified Process

The Rational Unified Process (RUP) is a software engineering process and a process framework for successful iterative-incremental software development [Shuja and Krebs, 2008]. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. The main goal of this disciplined approach is to ensure the production of high quality software that meets the needs of its end users, within a predictable schedule and budget.

The Rational Unified Process captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations [Kruchten, 2003]. It describes how to effectively deploy best practices which are as follows:

- Adapt the process;
- Balance competing stakeholder priorities;
- Collaborate across teams;
- Demonstrate value iteratively;
- Elevate the level of abstraction;
- Focus continually on quality.

The Rational Unified Process recognizes that the traditional waterfall approach can be inefficient. Because the traditional waterfall approach brings key team members idle for extended time periods [Ambler, 2005b]. Many feel that the waterfall approach also introduces a lot of risk because it performs testing and integration activities at the end of the project lifecycle. Problems found at this stage are very expensive to fix that causes team to turn back to development.

By contrast, RUP represents an iterative approach that is superior to the traditional waterfall approach for a number of reasons as follows [Kruchten, 2003]:

- RUP lets you take into account changing requirements which despite the best efforts of all project managers are still a reality on just about every project;
- Instead of performing integration one at a time at the end, elements are integrated progressively;
- Risks are usually discovered or addressed during integration. With the iterative approach, you can mitigate risks earlier which reduce the cost and effort on development;
- Iterative development provides management with a means of making tactical changes to the product. It allows you to release a product within the early iterations with reduced functionality to counter a move by a competitor, or to adopt another vendor for a given technology;
- Iteration facilitates reuse. It is easier to identify common parts as they are partially designed or implemented than to recognize them during planning;
- Errors can be corrected over several iterations which causes a more robust architecture. Performance bottlenecks are discovered at a time when they can still be addressed;
- Developers can learn along the way, and their various abilities and specialties are more fully employed during the entire lifecycle. Testers start testing, technical personnel begin their work early in the product, not at the end of the project lifecycle;
- The development process itself can be improved and refined along the way. There is an assessment at the end of each iteration that examines the status of the project and also analyzes alteration in the organization and in the process to make it perform better in the next iteration.

The Rational Unified Process enhances team productivity, by providing every team member with easy access to a knowledge base with guidelines, templates and tool mentors for all critical development activities. The Rational Unified Process activities create and maintain models. Rather than focusing on the production of large amount of paper documents, the Rational Unified Process emphasizes the development and maintenance of models. The most important advantage of Rational Unified Process is being a configurable process. We know that no single process is suitable for all software development. The Rational Unified Process fits small development teams as well as large development organizations with its highly flexible configurable processes.

Figure 3.1 illustrates the overall architecture of RUP [e.g. IBM, 2007b]. This figure which is also called **hump chart** contains information about phases, iterations, milestones, disciplines, their interrelationships, and the lifecycle concept of RUP.

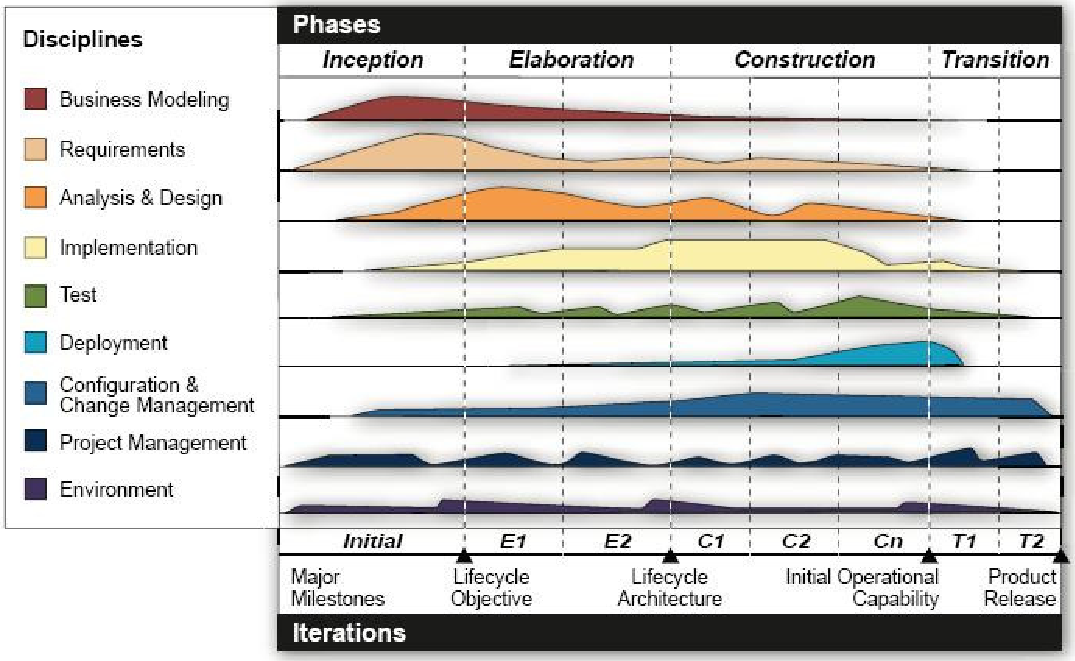


Figure 3.1 Overall Architecture of RUP

The horizontal axis represents time and shows the lifecycle aspects of the process. It represents the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones. The vertical axis represents core process disciplines, which group activities logically by nature. It represents the static aspect of the process as it is expressed in terms of process components, activities, disciplines, artifacts, and roles.

3.2 RUP Lifecycle

The RUP has four sequential phases which are Inception, Elaboration, Construction, and Transition. Each of them plays a central role in managing iterative and incremental development projects using RUP. Each phase concludes with a major milestone, as shown in Figure 3.1. Number of iterations within a phase depends on the size of projects.

3.2.1 Inception Phase

Inception phase defines the scope of the project and develops the business case for the system [Hunt, 2003]. It also establishes the feasibility of the system that is to be built. The overriding goal of the inception phase is to achieve concurrence among all stakeholders on the lifecycle objectives for the project. All high level requirements models are developed within this phase. The inception phase plays the most critical role in the project. Inception phase objectives can be listed as follows:

- Establish the project's scope and boundary conditions;
- Estimate the potential risks which is a critical issue for the project;
- Estimate the overall cost and schedule for the project that is not detailed;
- Prepare the support environment for the project;
- Identify the critical use cases of the system;
- Exhibit one candidate architecture;
- Produce detailed estimates for the Elaboration phase.

The inception phase is concluded by the lifecycle objective milestone. At this point, the lifecycle objectives of the project are examined and decided either to proceed with the project or to cancel it. The evaluation criteria for the inception phase are [Gornik, 2001]:

- Stakeholder concurrence on scope definition, cost and schedule estimates;
- Requirements understanding as evidenced by the fidelity of the primary use cases;
- Credibility of the cost and schedule estimates, priorities, risks, and development process;

- Depth and breadth of any architectural prototype that was developed;
- Actual expenditures versus planned expenditures.

The state of several essential work products at the inception phase milestone are given below [IBM Redbooks, 2007]:

- Vision
- Business Case
- Risk List
- Software Development Plan
- Iteration Plan
- Development Process
- Development Infrastructure
- Glossary
- Use-Case Model

3.2.2 Elaboration Phase

Elaboration phase captures the functional requirements of the system [Hunt, 2003]. It should also specify any non-functional requirements to ensure that they are taken into account. The main goal of the elaboration phase is to baseline the architecture of the system to provide a stable basis for the design and implementation effort in the Construction phase. This is the initial phase that the architecture is proved by creating an architectural prototype. Elaboration phase objectives can be listed as follows:

- Stabilize the architecture and requirements;
- Establish a supportive environment;
- Mitigate risks to determine project cost and schedule;
- Address all architecturally significant risks;
- Establish a baseline architecture that will be used for the entire project;
- Produce an evolutionary prototype;
- Optionally throw away prototypes can be produced to mitigate specific risks such as design tradeoffs component reuse, and product feasibility;
- Demonstrate that the baseline architecture will support the requirements of the system at a reasonable cost and in a reasonable time.

The elaboration phase is concluded by the lifecycle architecture milestone. At this point, the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks are examined. The main evaluation criteria for the elaboration phase involve the answers to following questions [Gornik, 2001]:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the plan for the construction phase sufficiently detailed and accurate? Is it backed up with a credible basis of estimates?
- Do all stakeholders agree that the current vision can be achieved if the current plan is executed to develop the complete system, in the context of the current architecture?
- Is the actual resource expenditure versus planned expenditure acceptable?

The state of several essential work products at the elaboration phase milestone are given below [IBM Redbooks, 2007]:

• Prototypes	• Risk List
• Development Process	• Development Infrastructure
• Development Infrastructure	• Design Model
• Implementation Model	• Vision
• Software Development Plan	• Iteration Plan
• Supplementary Specifications	• Use-Case Model
• Test Suite	• Test Plan
• Test Cases	• Test Scripts

3.2.3 Construction Phase

Construction phase concentrates on completing the analysis of the system, performing the majority of the design and the implementation of the system [Hunt, 2003]. All remaining requirements are identified and the system is developed based on the baselines created in elaboration phase. Construction phase objectives can be listed as follows:

- Minimize development costs through optimization of resource utilization by avoiding unnecessary rework and by achieving a degree of parallelism in the work of development teams;
- Achieve adequate quality as rapidly as is practical;
- Complete the analysis, design, development, and testing of all required functionality;
- Decide if the software and the users are ready for the deployment of the solution;
- Achieve useful executable versions as rapidly as practical;
- Iteratively and incrementally develop a complete product that is ready to transition to its user community.

The construction phase is concluded by the initial operational capability milestone. At this point, we decide if the software, the sites, and the users are ready to go operational, without exposing the project to high risks. The evaluation criteria for the construction phase involve answering the following questions [Gornik, 2001]:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition into the user community?
- Are the actual resource expenditures versus planned expenditures still acceptable?

The state of several essential work products at the construction phase milestone are given below [IBM Redbooks, 2007]:

• Deployment Plan	• Implementation Model
• Test Suit	• User Support Material
• Risk List	• Iteration Plan
• Design Model	• Development Process
• Development Infrastructure	• Data Model
• Test Plan	• Test Cases
• Test Scripts	

3.2.4 Transition Phase

The Transition phase moves the system into the user's environment [Hunt, 2003]. The overall goal of the Transition phase is to ensure that software is available for its users. In some systems testing should be performed within this phase. Transition phase objectives can be listed as follows:

- Validate the new system against user expectations;
- Train the end users and maintainers;
- Roll out the product to marketing, distribution, and sales teams;
- Fine tune the product by engaging in bug fixing and creating performance and usability enhancements;
- Conclude the assessment of the deployment baseline against the complete vision and the acceptance criteria for the product;
- Achieve user self supportability;
- Achieve stakeholder concurrence that deployment baselines are complete and are consistent with the evaluation criteria of the vision.

The transition phase is concluded by the product release milestone. At this point, the objectives are examined to ensure if they were met, and if we should start another development cycle. The evaluation criteria for the transition phase involve the answers to these questions [Gornik, 2001]:

- Is the user satisfied about the system?
- Are the actual resources expenditures versus planned expenditures still acceptable?

The state of several essential work products completed at the transition phase milestone are given below [IBM Redbooks, 2007]:

• The Product Build	• User Support Material
• Implementation Elements	• Deployment Unit

3.3 RUP Disciplines

In RUP, a discipline is defined as a categorization of activities based on similarity of concerns and cooperation of work effort. A discipline is a collection of activities that are related to a major area of concern within the overall project. There are total nine disciplines defined in RUP, six of which are core disciplines and three core supporting disciplines.

The core disciplines in RUP are divided into six:

- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment

Three core supporting disciplines:

- Project Management
- Configuration and Change Management
- Environment

3.3.1 Business Modeling Discipline

One of the major problems with most business engineering efforts is that the software engineering and the business engineering community do not communicate properly with each other [Gornik, 2001]. This problem causes that the output from business engineering is not being used properly as input to the software development effort, and vice versa.

First of all the aim of business modeling is to establish a better understanding and communication channel between business engineering and software engineering. This is required to come up with proper requirements for software systems to be built for the business at hand. Understanding the business is an important aspect which means software engineers must understand the structure and the dynamics of the target organization, the current problems in the organization and possible improvements. A common understanding of the target organization between customers, end users and developers has to be developed. Business modeling explains how to describe a vision of the organization in which the system will be deployed and then how to use this vision as a basis to outline the process, roles and responsibilities.

Activities of the business modeling discipline include [Ambler, 2005b]:

- Assess the current status of the organization;
- Describe the current business processes, roles, and responsibilities;
- Identify and evaluate potential strategies for reengineering the business processes;
- Develop a domain model which reflects the subset of the business.

3.3.2 Requirements Discipline

The goal of the requirements discipline is to describe what the system should do and allows the developers and the customers to agree on that description [Gornik, 2005b]. This discipline explains how to elicit stakeholder requirements and transform them into a set of requirements work products that scope the system to be built and provide detailed requirements for what the system must do. Capturing requirements have a critical importance for the system runs effectively.

The requirements discipline attempts to express the systems requirements in terms of use cases [Hunt, 2003]. The use cases function as a unifying thread throughout the system’s development cycle. During requirements, analysis and design, and test disciplines the same use case model is used.

Activities of the requirements discipline include [Ambler, 2005b]:

- Analyze the problem;
- Work closely with project stakeholders to understand their needs;
- Define and manage the scope of the system;
- Refine the system definition by describing business rules, the user interface, and non-functional requirements via appropriate modeling techniques;
- Manage changing requirements as they are identified throughout a project.

3.3.3 Analysis and Design Discipline

The major goal of the analysis and design discipline is to translate the requirements which are obtained in requirements discipline into a specification describing how to implement the system [Kruchten, 2003]. The architecture and design of the software system is created within this discipline. The goal of analysis and design discipline is to show how the system will be realized. The aim is to build a system that performs tasks and functions specified in the use case descriptions and fulfill all its requirements.

Requirements have to be understood and transformed into a system design by selecting the best implementation strategy. Early in the project a robust architecture has to be established so that we can design a system that is easy to understand, build, and evolve.

Activities of the analysis and design discipline include [Ambler, 2005b]:

- Define a candidate architecture for the system;
- Construct an architectural proof-of-concept to validate the candidate architecture;
- Understand the requirements for the system;
- Analyze the behavior by designing the user interface, and database;
- Design of components, services, and modules.

3.3.4 Implementation Discipline

The implementation discipline is concerned with implementing the design produced by the design discipline [Hunt, 2003]. The system is realized through implementation of components. The Rational Unified Process describes how you reuse existing components, or implement new components with well defined responsibility, making the system easier to maintain, and increasing the possibilities to reuse. Implementation discipline deals with any implementation issues that have been left as too specific during the design discipline. It is important to remember that developer is responsible for unit testing in implementation discipline.

Activities of the implementation discipline include [Ambler, 2005b]:

- Structure the implementation model;
- Understand and evolve the design model;
- Write program source code which is organized into layers;
- Implement components, services, and modules;
- Unit test the source code;
- Integrate the code into subsystems and a deployable build.

3.3.5 Test Discipline

The aim of the test discipline is to ensure that the system provides the required functionality [Hunt, 2003]. Test discipline acts as a service provider to the other disciplines. The Rational Unified Process proposes an iterative approach, which means that you test throughout the project. This allows you to find defects as early as possible, which radically reduces the cost of fixing the defect. Detecting and recovering errors at the early stages of development has a critical importance and one of the main ideas in RUP.

Test discipline has a difference than other disciplines which finds and exposes weaknesses in the software product. Test discipline is performed to find what is missing, incorrect, or inconsistent that not focuses on consistency and completeness as other disciplines does.

Activities of the test discipline include [Ambler, 2005b]:

- Define and plan testing efforts;
- Develop test cases and test scripts;
- Organize test suites to run test cases in a specified order;
- Run tests and evaluate;
- Report defects.

3.3.6 Deployment Discipline

The major goal of the deployment discipline is to successfully produce product releases, and deliver the software to its end users [Kruchten, 2003]. System deployment is a critical aspect of the software development lifecycle because if the software cannot get into the hands of the end users then it has no value even if it is successfully developed. Deployment activities are mostly centered on the transition phase, many of the activities need to be included in earlier phases to prepare for deployment at the end of the construction phase. The effort on these activities mostly depends on the size of the project.

Activities of the deployment discipline include [Ambler, 2005b]:

- Plan the deployment by developing deployment plan;
- Develop support and operations material;
- Create deployment packages;
- Manage acceptance testing efforts;
- Perform alpha/beta testing of the product;
- Deploy software to installation sites;
- Train end users.

3.3.7 Project Management Discipline

Software project management is managing risk and overcoming constraints to deliver a product that meets the needs of the customers and the end users [Kruchten, 2003]. Project planning in the RUP occurs at two levels. There is a coarse-grained phase plan which describes the entire project, and a series of fine-grained or iteration plans which describe the iterations. Project management discipline does not attempt to cover all aspects of project such as managing people and budgets. A difficulty on project management is that it requires specific skills to deal with problems such as risk management, planning and scheduling of the project, motivating and developing the project staff and so on.

Activities of the project management discipline include [Ambler, Nalbone and Vizdos, 2005]:

- Conceive a new project;
- Manage project staff;
- Enhance the relationship with external teams and resources;
- Risk management;
- Estimating, scheduling, and planning of the project;

- Manage an iteration and plan the remainder of iteration;
- Close out a phase or project.

3.3.8 Configuration and Change Management Discipline

The major goal of the configuration and change management discipline is to track and maintain the integrity of evolving project assets [Kruchten, 2003]. Controlling the numerous artifacts produced by the project staff is described within the configuration and change management discipline [Gornik, 2001].

The configuration and change management discipline provides guidelines for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site specific development policies.

Activities of the configuration and change management discipline include [Ambler, Nalbone and Vizdos, 2005]:

- Manage change requests;
- Plan configuration and change control;
- Set up the configuration management environment;
- Monitor and report configuration status;
- Change and deliver configuration items;
- Manage baselines and releases.

3.3.9 Environment Discipline

The purpose of the environment discipline is to provide the software development organization with the software development environment that is needed to support the development team [Gornik, 2001]. The software development organization is supported with both processes and tools. The environment discipline focuses on the activities necessary to configure the process for a project.

Activities of the environment discipline include [Ambler, Nalbone and Vizdos, 2005]:

- Prepare environment for the project by tailoring the process materials for an individual project team;
- Identify and evaluate tools;
- Install and set up tools for the project team;
- Support the tools and process throughout the project.

3.4 Iteration in Rational Unified Process

As we mentioned in the previous sections Rational Unified Process (RUP) projects are iterative. The RUP is an incremental process whereby the overall project is broken down into phases and iterations [West, 2002]. Iterations address only a portion of the entire system that is being developed. Each iteration has a fine-grained plan that defines the steps with a specific goal. Iterative development allows projects to proceed by small steps or increments to gradually build a more complete system [Wessberg, 2005].

The iterative nature of the RUP is reflected in how we approach its disciplines. During each iteration we will alternate back and forth between the activities of the disciplines, performing each task to the extent needed at the time, to achieve the goals of that iteration.

A flexible way to proceed is to go several times through the various development disciplines, building a better understanding of the requirements, engineering a robust architecture, ramping up the development organization, and eventually delivering a series of implementations that are gradually more complete. Each iteration in the RUP is a pass through the disciplines as shown in Figure 3.2 [e.g. Ambler, 2005a] on the next page.

Therefore, from a development perspective the software lifecycle is a succession of iterations, through which the software develops incrementally. With each iteration, the solution is coming closer and closer to the final product.

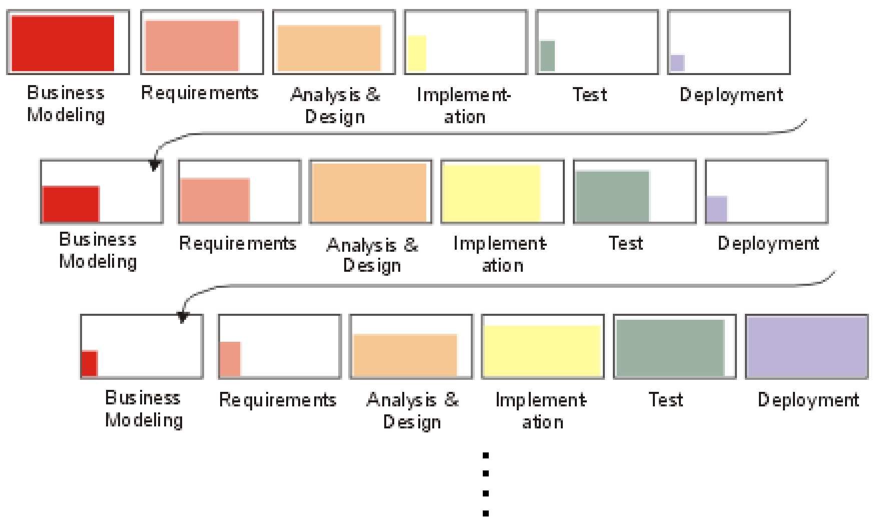


Figure 3.2 The Iterative Development Process of RUP

CHAPTER 4

IBM RATIONAL TOOLS

4.1 General

Over many years of development effort, RUP has evolved into a rich process engineering platform called IBM Rational Method Composer (RMC). RMC is an Eclipse-based tool which enables you to define, maintain, and deploy software process related material by enabling teams to define, configure, tailor, and practice a consistent process.

The RUP describes a set of models such as use-case models, analysis models, and design models that represent well-defined perspectives on the problem and solution domains of systems. The utility of this set of models has been proven in many projects. For applying the modeling guidance found in the RUP, a modeling platform called IBM Rational Software Modeler (RSM) is used that is built on the Eclipse open source software framework. RSM is a robust collaborative platform for visual modeling and design that specifies and communicates software project information from several perspectives to various stakeholders.

As noted above RMC and RSM are IBM Rational Tools relevant to RUP. Additionally, there is a platform called Eclipse that is used to develop such IBM Tools. The Eclipse is a multi-language software development platform that is designed for building Integrated Development Environments (IDEs) known as a software application that provides comprehensive facilities to computer programmers for software development. Eclipse can be used to create diverse end-to-end computing solutions for multiple execution environments [Erickson and McIntyre, 2001]. The platform consists of open source software components that tool vendors use to construct solutions that plug in to integrated software workbenches. The Eclipse is simply a framework and a set of services for building applications from plug-in components. This would be a toolkit for designing toolkits. Not just a set of APIs, the framework will consist of real code designed to do real work [Aniszczyk and Gallardo, 2007].

Thus, RMC and RSM have become the most powerful tools used during the project development. RMC provides a clear path to project team with high efficiency and low risks. On the other hand, RSM provides the communication between the stakeholders and the project team. Their capabilities and advantages are summarized briefly in the next sections.

4.2 IBM Rational Method Composer (RMC)

IBM Rational Method Composer represents a major evolution of IBM's process solutions, which includes and extends the IBM Rational Unified Process (RUP). RMC is a commercial product that is IBM's next generation process management tool platform. Target users who are sanctioned for RMC are process engineers, project leaders, and project and program managers who are responsible for maintaining and implementing processes for development organizations or individual projects. A conceptual framework for authoring, configuring, viewing, and publishing processes are provided to perform each process in flexible manners.

RMC is a complete business driven development solution. Running business driven development projects requires flexible development processes. Such processes not only have to provide concrete support and guidance for modern development practices, such as agile, iterative, architecture centric, risk and quality driven software development [Kroll and Royce, 2005], but also have to be flexible enough to support rapid tailoring and adoption of the process itself as RMC provides. These processes also need to evolve across projects, and the projects being executed must themselves be able to evolve as business needs change mid way to completion.

4.2.1 Purpose and Capabilities

Development leaders and teams face some problems when acquiring and managing their methods and processes. The aim of Rational Method Composer is to provide solutions to these problems as follows:

- Development teams need easy and centralized access to the information repository of the project;
- It is difficult to integrate development processes that convey in their own proprietary format;
- Teams lack an up-to-date knowledge base for educating themselves on methods and best practices;
- Support for right sizing the processes of teams is required;
- Compliance to standardized practices has to be ensured;
- Effective execution of processes in project has to be provided.

Rational Method Composer has two main purposes which are detailed as follows [Haumer, 2005]:

- RMC is a content management system that provides a common management structure and applicable for all process content. All content managed in RMC can be published to HTML and deployed to Web servers for distributed usage in any phase of the management;
- RMC provides capability of selecting, tailoring, and rapidly assembling processes in concrete development projects for process engineers and project managers. RMC provides instructions of predefined processes of RUP for typical project situations that can be adapted to individual needs as size and complexity of the project to be developed. It also provides process building blocks called **capability patterns** that represent best development practices for specific disciplines, technologies, or development styles. Capability patterns form a toolkit for quickly assembling processes based on project specific needs. Finally, the documented processes created with RMC can be published and deployed as Web Sites.

RMC improves team efficiency, responsiveness, productivity and increases project quality [IBM, June 2006]. One of the most important properties of RMC is providing easy to use process tools and innovative technology that reduce the time to customize best practices. It also offers new tools that help automate capturing best practices that other projects across the company can use. These reusable process components are powerful building blocks that help teams to complete projects on time and within budget.

4.2.2 Key Terminology and Concepts

In order to effectively work with Rational Method Composer, a few concepts are needed to be understood that are used to organize the content. First of all when the program is started, a window greets the user as shown in Figure 4.1.

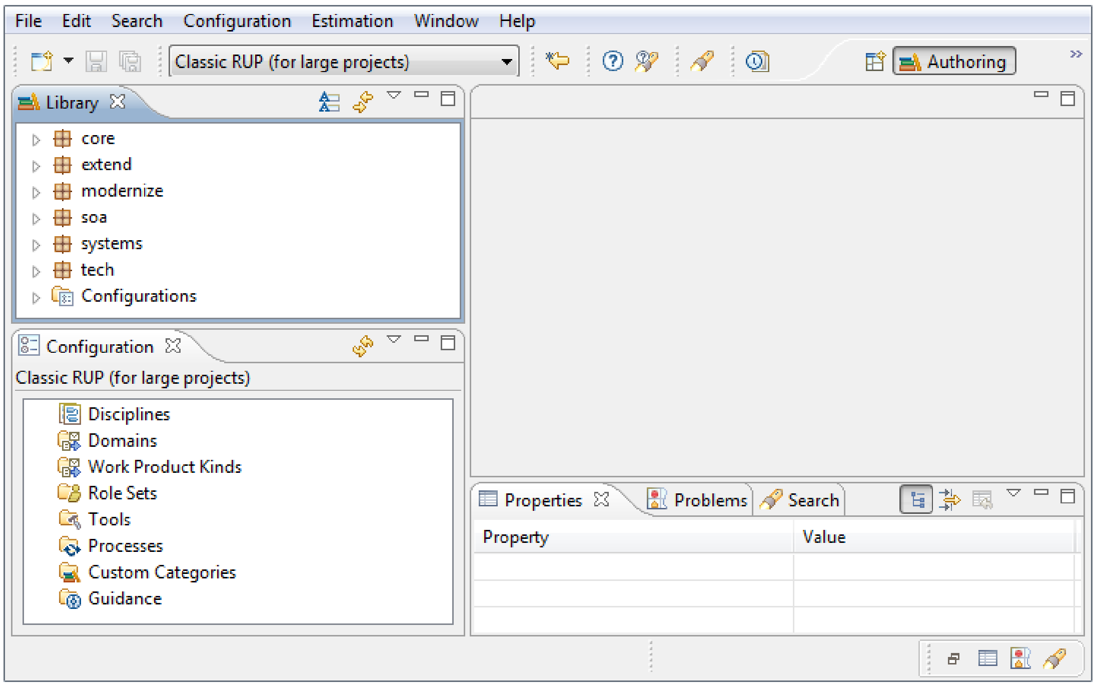


Figure 4.1 RMC Main Window

There are two major views within the main window of RMC which are the Library View and the Configuration View as shown in the left side of the Figure 4.1. The Library View shows all method plug-ins and configurations. In RMC, all of these method plug-ins are classified in six plug-in packages. The plug-in package names are explained as follows:

- **core:** Used for plug-in core to the RUP software development process. Most RUP configurations should contain the core plug-ins;

- **extend:** Extensions to the general RUP software development process that do not fall into the other packages;
- **modernize:** Enterprise modernization;
- **SOA:** Service-oriented architecture;
- **systems:** Systems engineering;
- **tech:** Technology and tool-specific extensions.

The Configuration View shows the content elements in a library filtered by a configuration. A configuration is a subset of the method content. Once a configuration is selected in the Configuration Selection Box, the configuration view is refreshed with the content from the selected configuration. Configuration Selection Box is a simple drop down menu whose initial value is set to “Classic RUP (for large projects)” as default that is depicted in Figure 4.1. There are eight major categories in Configuration View as follows:

- **Disciplines:** A collection of Tasks that are related to a major area of concern within the overall IT environment. Separating these tasks into separate disciplines makes the tasks easier to comprehend.
- **Domains:** A logical, hierarchy of related Work Products grouped together based on timing, resources, or relationship. While a Domain categorizes many work products, a work product belongs to only one Domain. Domains can be further divided into sub-domains.
- **Work Product Kinds:** Used for grouping Work Products. A work product can have many work product kinds.
- **Role Sets:** Used to group Roles with certain commonalities together. Each of these roles work with similar techniques and have overlapping skills, but may be responsible for performing certain tasks and creating certain work products.
- **Tools:** A specific type of guidance that shows how to use a specific tool to accomplish a piece of work.
- **Processes:** Describes how a particular piece of work should be done. A process can reuse method elements and combines them into a structure and sequence for carrying out work.

- **Custom Categories:** Highly customizable and can contain any type of element. Custom Categories allow user to categorize content according to any scheme that he/she wants and can then be used to compose publishable Views, providing a means to organize the method content prior to publishing.
- **Guidance:** General term for additional information related to roles, tasks, and work products.

Rational Method Composer has some key concepts such as Process Content Library, Out-of-the-box Delivery Processes, and Capability Patterns that are to be understood carefully to work with it. High level understanding of these concepts is important for team members to overcome complex development challenges. In the following sections these concepts will be summarized:

A. **Process Content Library** [IBM, 2007a] which is based on the best practices adopted in thousands of RUP projects worldwide. RMC represents process elements in terms of roles, tasks, work products, and guidance as shown in Figure 4.2 that is obtained by clicking core item downward in Figure 4.1.

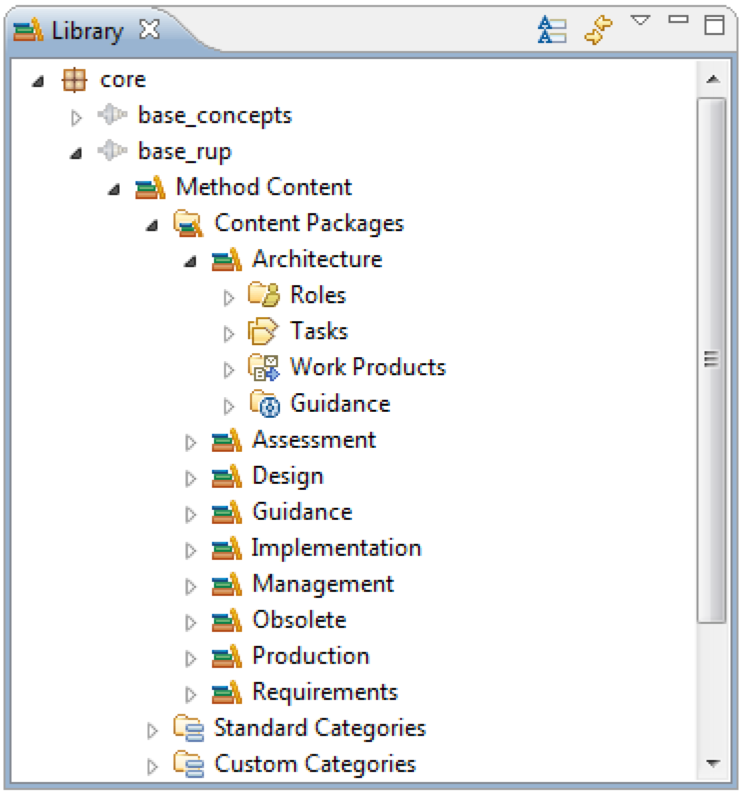

Library
- core
  - base_concepts
  - base_rup
    - Method Content
      - Content Packages
        - Architecture
          - Roles
          - Tasks
          - Work Products
          - Guidance
        - Assessment
        - Design
        - Guidance
        - Implementation
        - Management
        - Obsolete
        - Production
        - Requirements
      - Standard Categories
      - Custom Categories

Figure 4.2 Process Content Library

B. Out-of-the-box Delivery Processes [IBM, June 2006] provides a quick starting point for planning and initiating a project for project manager. This can be achieved by a delivery process by providing an initial project template that identifies the milestones that have to be in the project, work products that have to be delivered by each milestone, and resources that are needed for each phase. RMC includes out-of-the-box delivery processes for COTS, J2EE, Systems Engineering, SOA, etc whose configuration can be selected using Configuration Selection Box as shown in Figure 4.3.

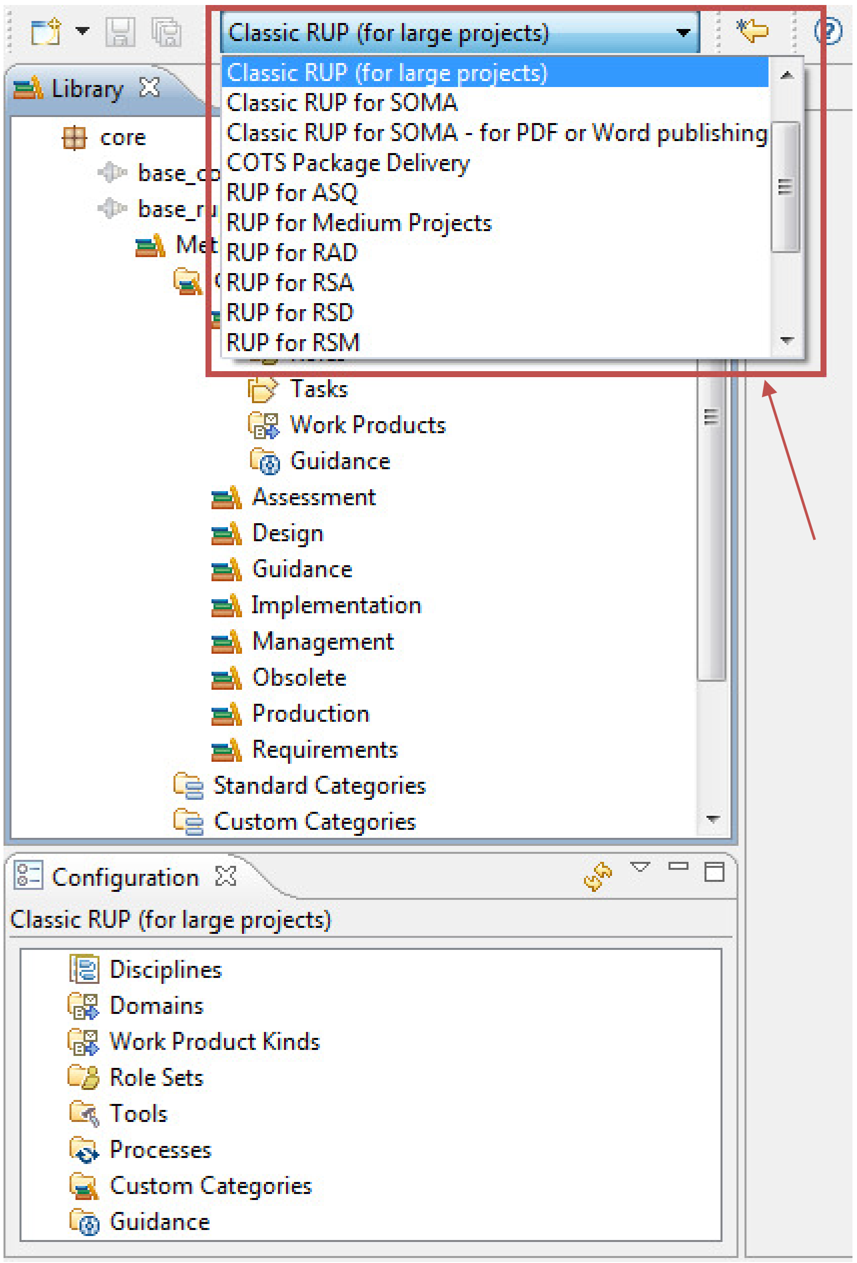


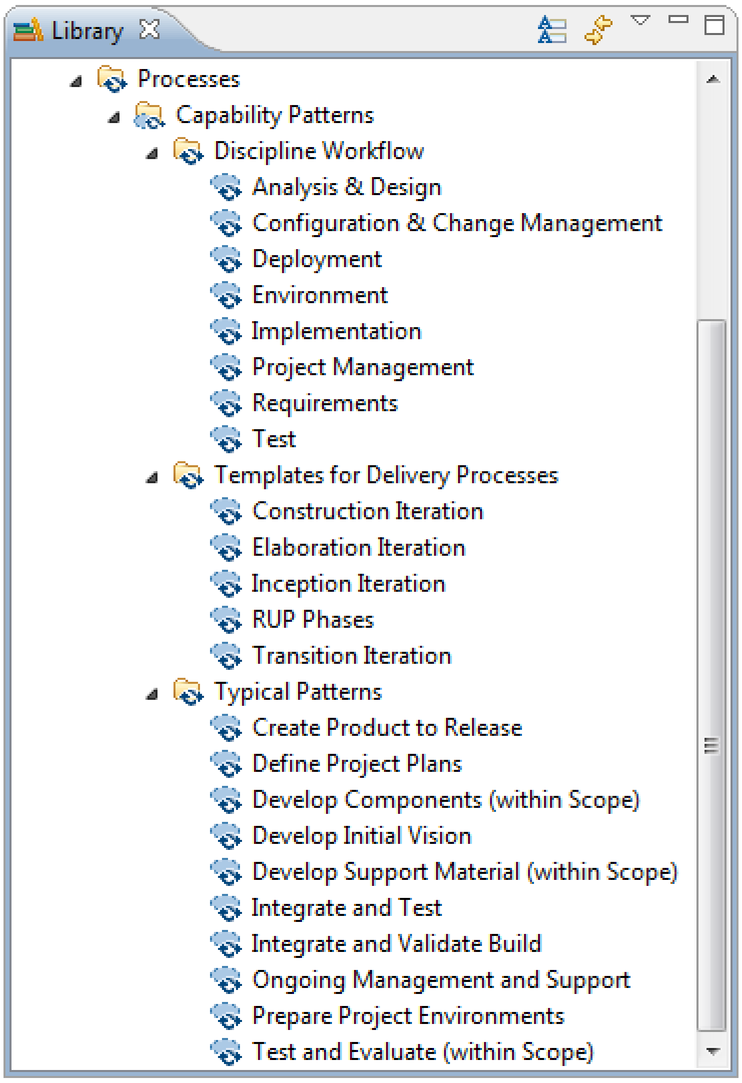
Figure 4.3 Out-of-the-box Delivery Processes

These out of the box delivery processes can be used as a starting point for further customizations. The complete list of processes that is available as follows:

- Asset Based Development
- Classic RUP (for large projects)
- Classic RUP for SOMA
- Classic RUP for SOMA – for PDF or Word publishing
- COTS Package Delivery
- RUP for ASQ
- RUP for Medium Projects
- RUP for RAD
- RUP for RSA
- RUP for RSD
- RUP for RSM
- RUP for Small Projects
- RUP for Small Projects – for PDF or Word publishing
- RUP for System z
- System Engineering
- User-Experienced Modeling

In the above list, Service-Oriented Architecture (SOA) is a business-centric IT architectural approach that provides methods for systems development and integration where systems group functionality around business processes and package them as interoperable services. Nowadays, IBM is interested in closely with the Service-Oriented Modeling Architecture (SOMA) that is the SOA related methodology. SOMA refers to the more general domain of service modeling necessary to design and create SOA. SOMA includes an analysis and design method that extends traditional object-oriented and component-based analysis and design methods to include concerns relevant to and supporting SOA. SOMA is an end-to-end SOA Method for the identification, specification, realization and implementation of services, components, flows.

C. **Capability Patterns** [IBM, June 2006] allow project managers to rapidly add or remove reusable chunks of processes addressing common problems. We know that no two projects are alike, so project managers need to rapidly modify the process to address the specific project needs. Reusable process fragments captured as Capability Patterns as shown in Figure 4.4.



Library

- Processes
  - Capability Patterns
    - Discipline Workflow
      - Analysis & Design
      - Configuration & Change Management
      - Deployment
      - Environment
      - Implementation
      - Project Management
      - Requirements
      - Test
    - Templates for Delivery Processes
      - Construction Iteration
      - Elaboration Iteration
      - Inception Iteration
      - RUP Phases
      - Transition Iteration
    - Typical Patterns
      - Create Product to Release
      - Define Project Plans
      - Develop Components (within Scope)
      - Develop Initial Vision
      - Develop Support Material (within Scope)
      - Integrate and Test
      - Integrate and Validate Build
      - Ongoing Management and Support
      - Prepare Project Environments
      - Test and Evaluate (within Scope)

Figure 4.4 Capability Patterns

The Rational Method Composer addresses two major areas of interest for the process manager. One of them is content reuse and the other is ability to customize the process to the needs of different project types by the help of capability patterns [IBM, 2007a]. It makes easy for organizations to capture their own best practices and make them seamlessly extend to the Rational Method Composer content libraries.

RMC focuses on addressing three critical areas for project managers such as the followings:

- rapid project initiation
- flexibility
- reality based management

The out-of-the-box delivery processes give project managers a quick starting point for planning and initiating a project by using templates related to the project. Through plug-ins and process components, the content around various technologies and domains can be added or removed by project managers. A delivery process is assembled from capability processes that capture recurring process patterns, so it can be instantiated as needed them rather than all at once. Rational Method Composer focuses on **productivity**, **guidance** and **personalization**. Helpful templates, artifacts and tools are included to facilitate increased productivity. Rational Method Composer helps to guide users providing proven concepts and historical best practices in its libraries. The process interface of Rational Method Composer can be personalized to focus on only what matters to the project based on the experience level, role and interest [IBM, June 2006].

RMC helps users to manage their projects with its best practices. These best practices can be achieved by using the Configuration View as we mentioned at the beginning of this section. The Guidance category of the Configuration View provides best practices to users. There are eleven sub-categories for the Guidance that has important supportive effect on projects. Some of these sub-categories could be summarized as follows:

- Guidelines, for techniques and concepts to learn about new concepts and how to effectively leverage key technologies and techniques;
- Examples, about what has worked from other projects;
- Checklists, that provides users to rapidly see how the work can be improved.

The most fundamental principle in RMC is the separation of reusable core method content from processes as shown in Figure 4.5. Separation of method content and processes increases process tailoring ability because method content is reusable when defining processes.

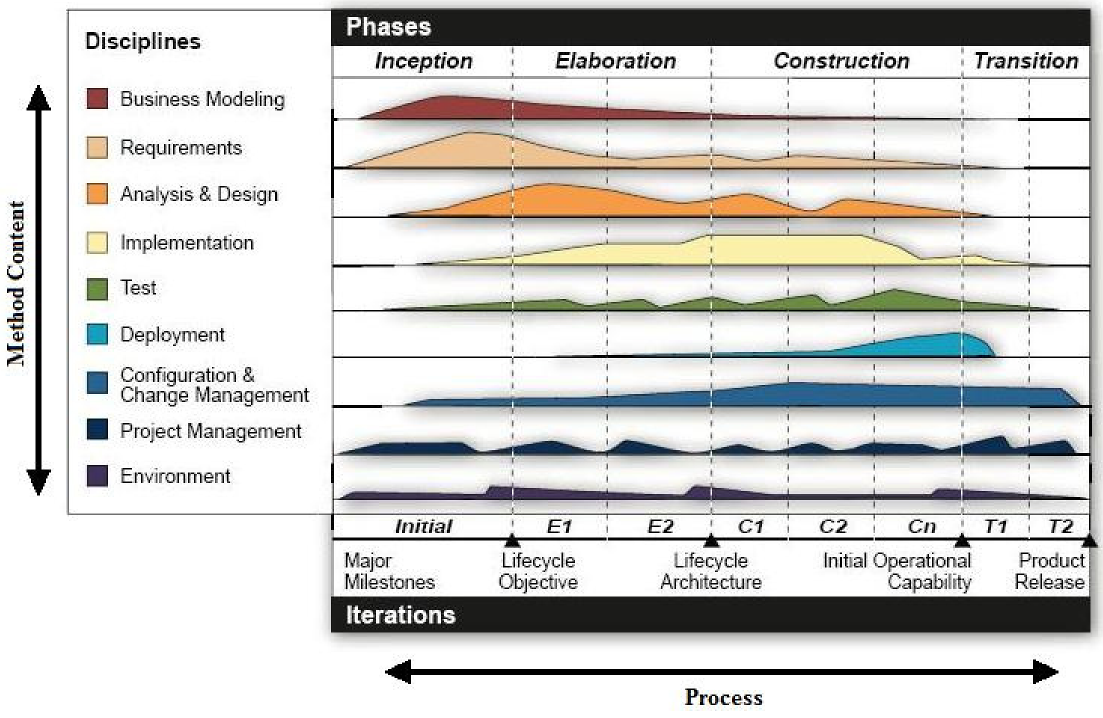


Figure 4.5 Separation of Method Content and Process

A successful development method provides both the descriptions of work and the order of work as shown in Figure 4.6 [IBM, 2007a]. A method is end-to-end and usable on a project. An example of a method is RUP. Method content provides descriptions of work that can be reused as important building blocks. These are the descriptions of tasks, roles, work products, guidelines. Processes provide the order of doing work. They do so by providing the order for the method content.

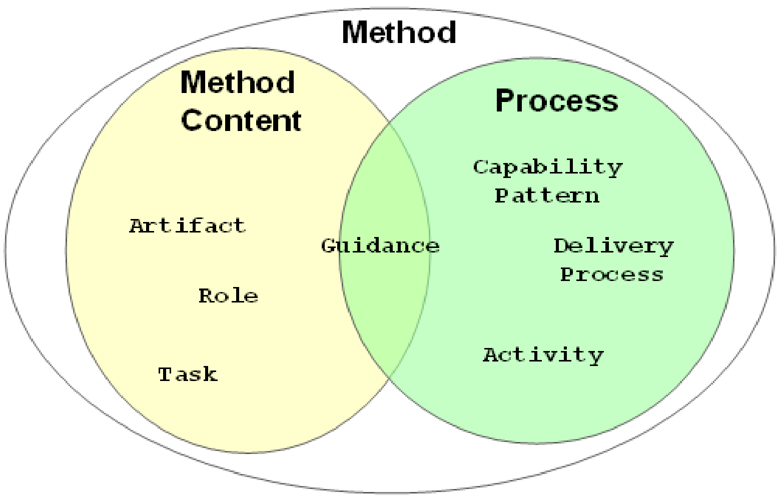


Figure 4.6 The Key Concepts of a Successful Method

Many development methods are described in books, articles, training material, standards and regulations, and other forms of documentation. Rational Method Composer takes such content and structures it in one specific way. Rational Method Composer expresses method content using concepts such as tasks, roles, work products, and guidance [Haumer, 2005]. The relationship between these concepts is shown in Figure 4.7.

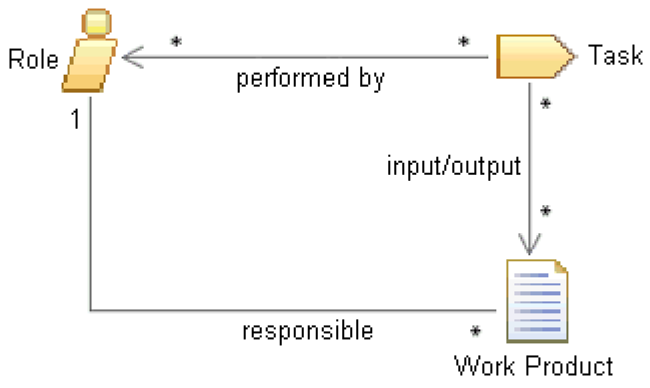


Figure 4.7 Core Method Content Concepts

Roles: Defining development skills and responsibilities for work products.
Tasks: Provide guidance on the work that needs to be done to transform inputs into outputs through a series of steps performed by one or more roles.
Work products: Define the items needed as input or created as output of one or more tasks that are typically the responsibility of a single role.

A development process defines sequences of the work that is being performed by roles and the work products that are being produced and evolved over time [Kroll and MacIsaac, 2006]. Processes can be expressed as workflows or breakdown structures. Rational Method Composer supports processes based on different development approaches and can be used to define different lifecycle models such as waterfall, incremental or iterative lifecycles [Haumer, 2005]. Rational Method Composer can be used to define processes that use a minimal set of method content or no method content to define processes for agile self organizing teams. Figure 4.8 shows such a part of method content. As we mentioned before, Guidance elements let users to add any additional information that makes method content of the project more complete and allows to factor details into separate descriptions.

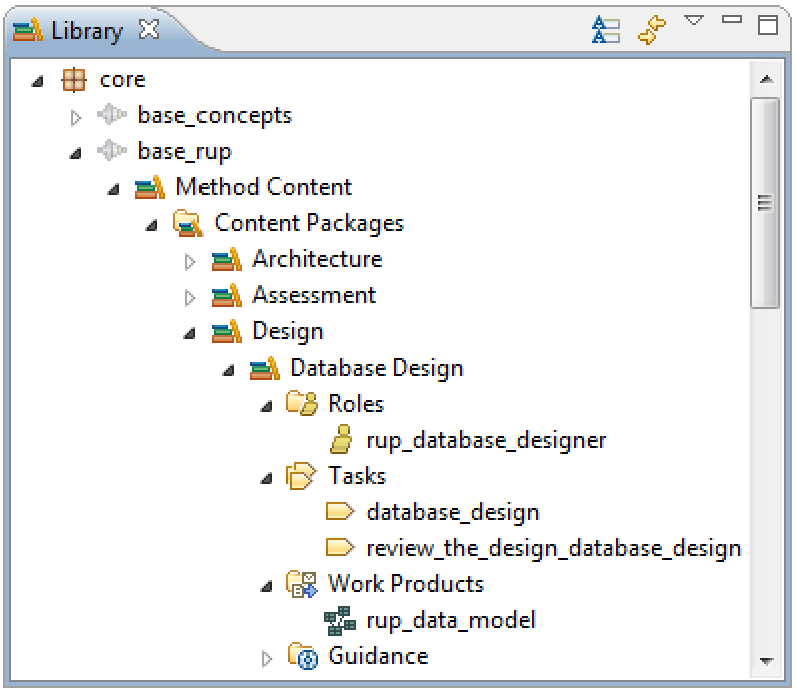


Figure 4.8 Project Specific Method Content

Rational Method Composer expresses process using concepts such as delivery process, capability patterns and activities which are detailed as follows:

- **Delivery Processes:** Defines a complete integrated approach to specific type of project.
- **Capability Patterns:** Special types of process used to define a stereotypical way of performing work related to a particular subject that may be used as a building block for assembling delivery processes or bigger patterns.
- **Activities:** Supports nesting and logical grouping of related breakdown elements.

Guidelines or **Guidance** can be attached to both method and process elements in order to provide additional guidance about those elements. Guidance is supplementary free form documentation such as whitepapers, concept descriptions, guidelines, templates, examples, and so on. In some projects they have critical importance on team members that are not familiar to these concepts.

4.3 IBM Rational Software Modeler (RSM)

Rational Software's first visual modeling and development tool was Rational Rose which was an important step in model driven development. However as noted by [Cernosek, 2004] a problem was identified that developers did not like to leave their own Integrated Development Environment (IDE) that they wanted visual modeling to be integrated inside their own IDE. As a result IBM Rational eXtended Development Environment (XDE) software is created that provides an extended development environment for the next generation of programming technologies. IBM Rational XDE offers software designers and developers a rich set of model-driven development and runtime analysis capabilities for building quality software applications. IBM Rational XDE was characterized as the next generation of IBM Rational Rose. However, as more and more capabilities were added, Rational XDE began to reach the practical limits of this style of tool integration. For the next generation model driven development products, it was only natural to build additional model driven development functions on top of Eclipse to form a more complete model driven development tool. IBM Rational Software Architect (RSA) and IBM Rational Software Modeler (RSM) are the result of these changes.

Rational Software Architect is not the next version of Rational Rose or Rational XDE, but it rather represents a fusion of select capabilities and development paradigms supported by Rational Rose and Rational XDE. Rational Software Architect takes features from these two tools, adds additional Model-Driven Development (MDD) capabilities, and introduces new structural review and control capabilities.

On the other hand, IBM Rational Software Modeler (RSM) is a visual modeling and design tool based on Unified Modeling Language (UML) 2.0 developed by IBM's Rational Software Division. Rational Software Modeler includes capabilities focused on visual modeling and Model-Driven Development (MDD) with the UML for creating resilient, thought out applications and especially **web services**. Using Rational Software Modeler system architects, system analysts, designers and other team members can easily specify software development project information from several perspectives, and communicate with each other and various stakeholders.

RSM automates repeatable activities and help improve the productivity and overall maturity of the development process. This is the reason that system architects, system analysts, designers use the capabilities of the Rational Software Modeler to help visually model and design their systems.

RSM plays a critical role in a software development project. As stated by Brown [2008], successfully performed Model Driven Architecture (MDA) approach provides an integrated business architecture and governance structure that enables project team to respond to business requirements quickly and appropriately.

Recently, Telelogic Rhapsody became a part of IBM Rational Software portfolio in 2008 that is an industry-leading UML based Model Driven Development (MDD) environment for technical, real-time or embedded systems and software engineering.

4.3.1 Features and Benefits

Rational Software Modeler provides UML 2.0 modeling support for analysis and design using use case, class, object, sequence, activity, composite structure, state machine, communication, component, and deployment diagrams. These diagrams allow capturing and communicating all aspects of an application architecture using a standard notation that is recognized by many different stakeholders.

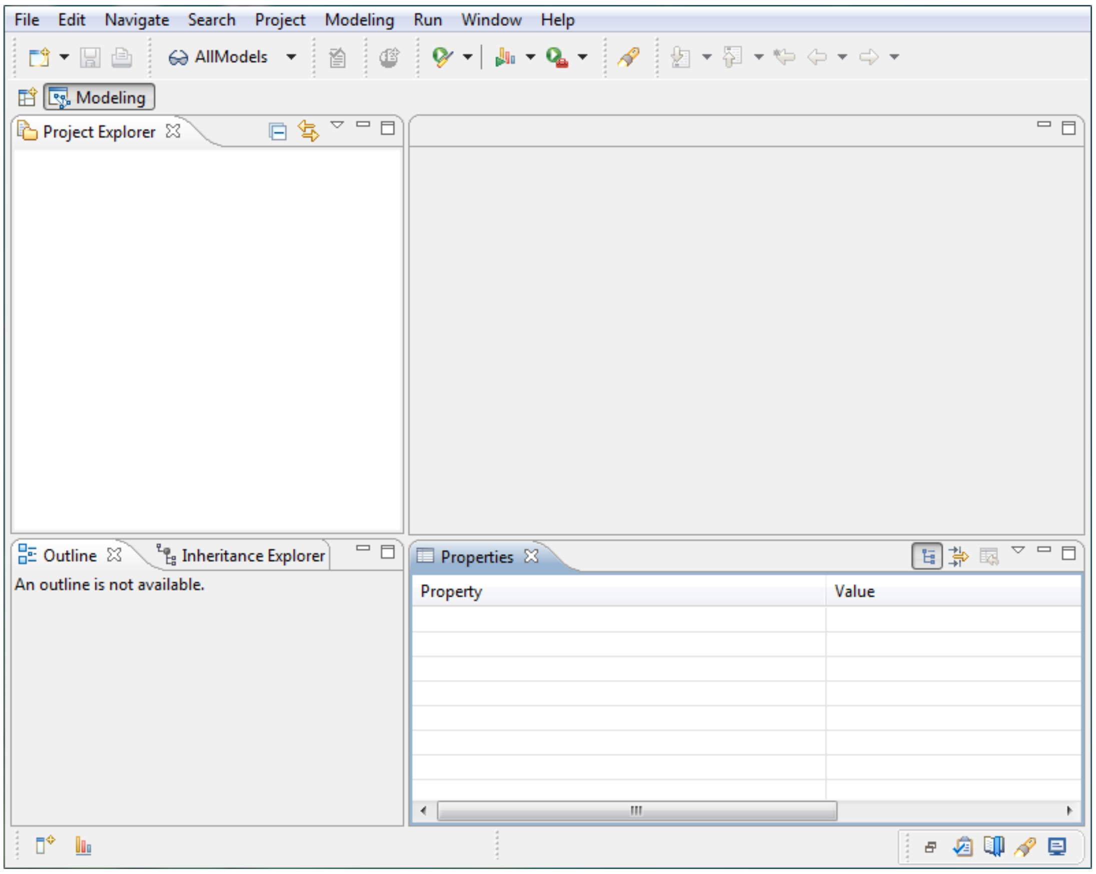
When the program is started, a window greets the user as shown in Figure 4.9. There are four major views within the main window of RSM which are the Project Explorer View, the Properties View, the Outline View, and the Inheritance Explorer View as shown in the Figure 4.9.

The Project Explorer View provides a hierarchical view of the resources in the Workbench.

The Properties View displays property names and values for a selected item such as a resource.

The Outline View displays an outline of a structured file that is currently open, and lists structural elements.

The Inheritance Explorer View provides to view an inheritance hierarchy that is created from the selected classifier such as a class, interface, or use case.

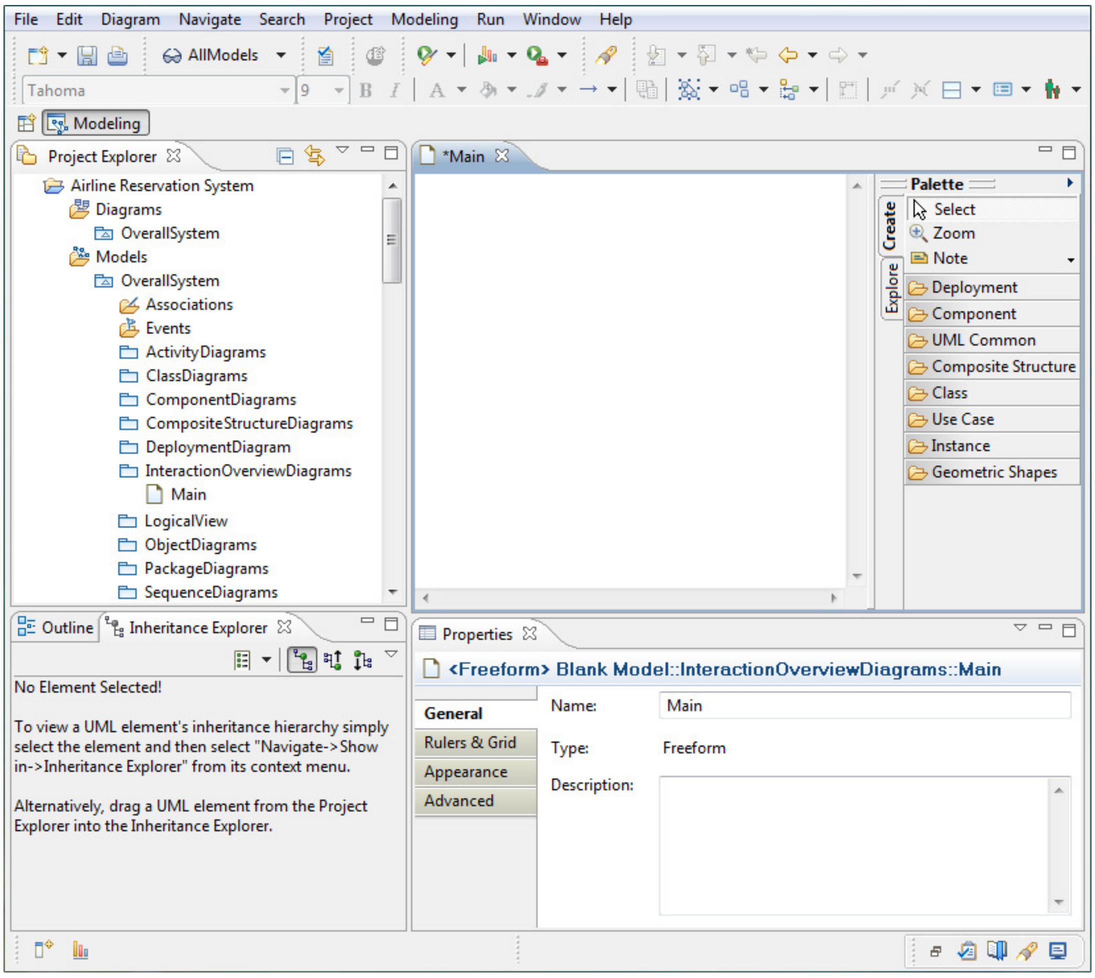


An outline is not available.

Figure 4.9 RSM Main Window

Rational Software Modeler also uses freeform diagrams, topic diagrams and browse diagrams [IBM, December 2006]. This simplifies the usage of UML notation for design, documentation, communication and understanding design elements that are captured in UML models.

RSM provides the option to create a modeling file “Blank Model” that is not based upon a model template. It has no special profiles applied, and no default content other than a single freeform diagram named as “Main”. Blank modeling files can be used as a starting point for any type of model such as use case model, analysis model, design model, deployment model by choosing how to name it, what content to define within it, and what profiles to apply to it. Whenever a new UML package is created in a model, a freeform diagram is automatically created. A Sample freeform diagram is depicted in Figure 4.10 for the airline reservation system that is modeled in Chapter 2. Also the Palette includes the packages of model elements that will be used to design any type of model in freeform diagrams.



No Element Selected!

To view a UML element's inheritance hierarchy simply select the element and then select "Navigate->Show in->Inheritance Explorer" from its context menu.

Alternatively, drag a UML element from the Project Explorer into the Inheritance Explorer.

<Freeform> Blank Model::InteractionOverviewDiagrams::Main

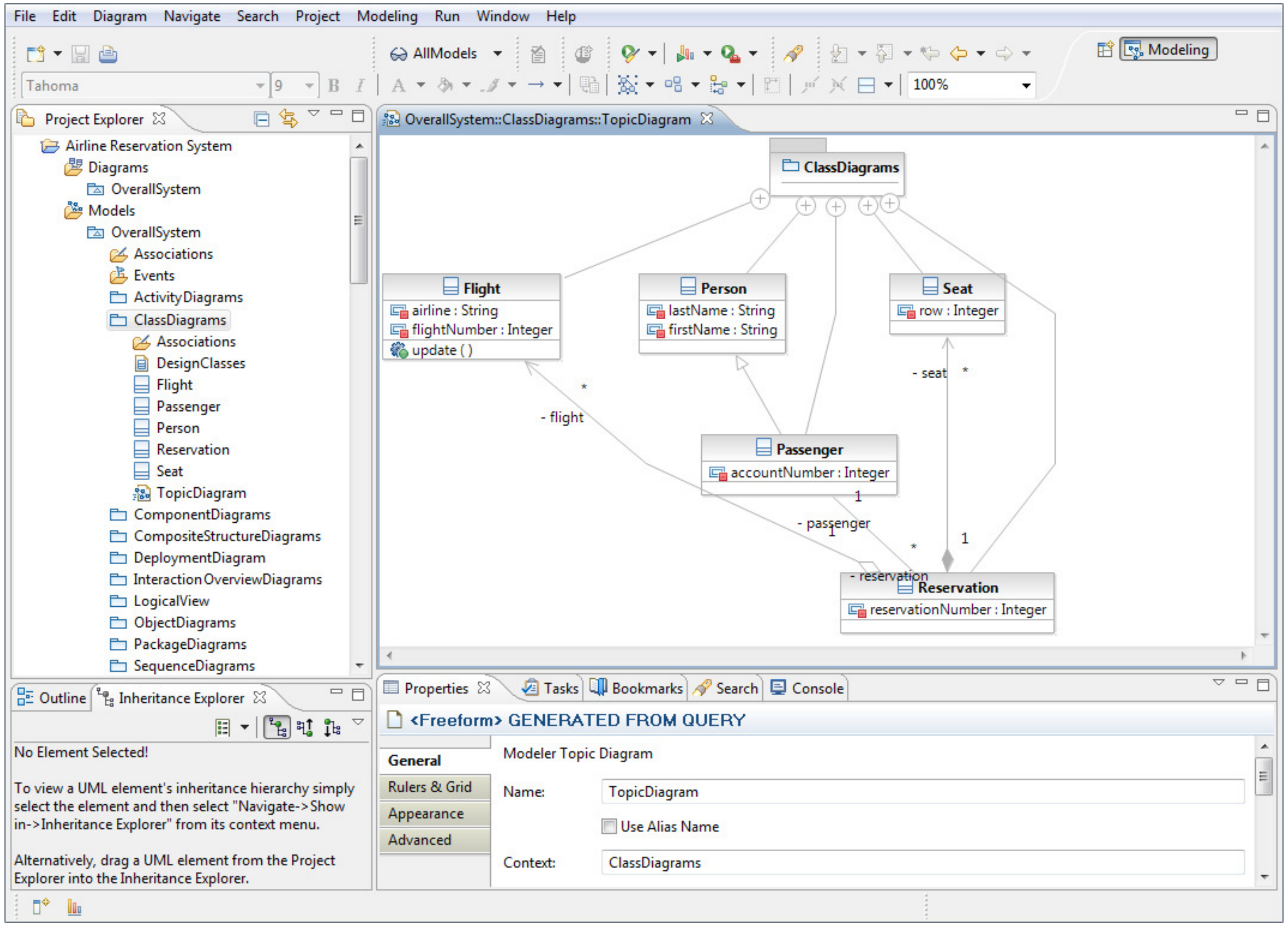
Name: Main

Type: Freeform

Description:

Figure 4.10 Freeform Diagram

In normal diagrams, elements are manually placed that the designer wishes to depict. The contents of a Topic Diagram are determined by a query that is run against existing model contents. To create a Topic Diagram, a topical model element or set of elements will be selected, and then defined what other elements the designer want to show in the diagram, based on the types of relationships that they have to the topical elements [Smith, 2008]. The content of the topical diagram changes according to the changes in the content of models. The definition of a named Topic Diagram can be persisted so that the same query can be rerun at any time. Topic Diagrams can be created simply using the project explorer window by selecting the model content and then follow the steps of context menu as; Visualize > Add to New Diagram in Model File > Topic Diagram. As a result the Topic Diagram is generated automatically. As an example Figure 4.11 depicts a Topic Diagram for the airline reservation system that is sampled in Chapter 2.



No Element Selected!

To view a UML element's inheritance hierarchy simply select the element and then select "Navigate->Show in->Inheritance Explorer" from its context menu.

Alternatively, drag a UML element from the Project Explorer into the Inheritance Explorer.

<Freeform> GENERATED FROM QUERY

Modeler Topic Diagram

Name: TopicDiagram

☐ Use Alias Name

Context: ClassDiagrams

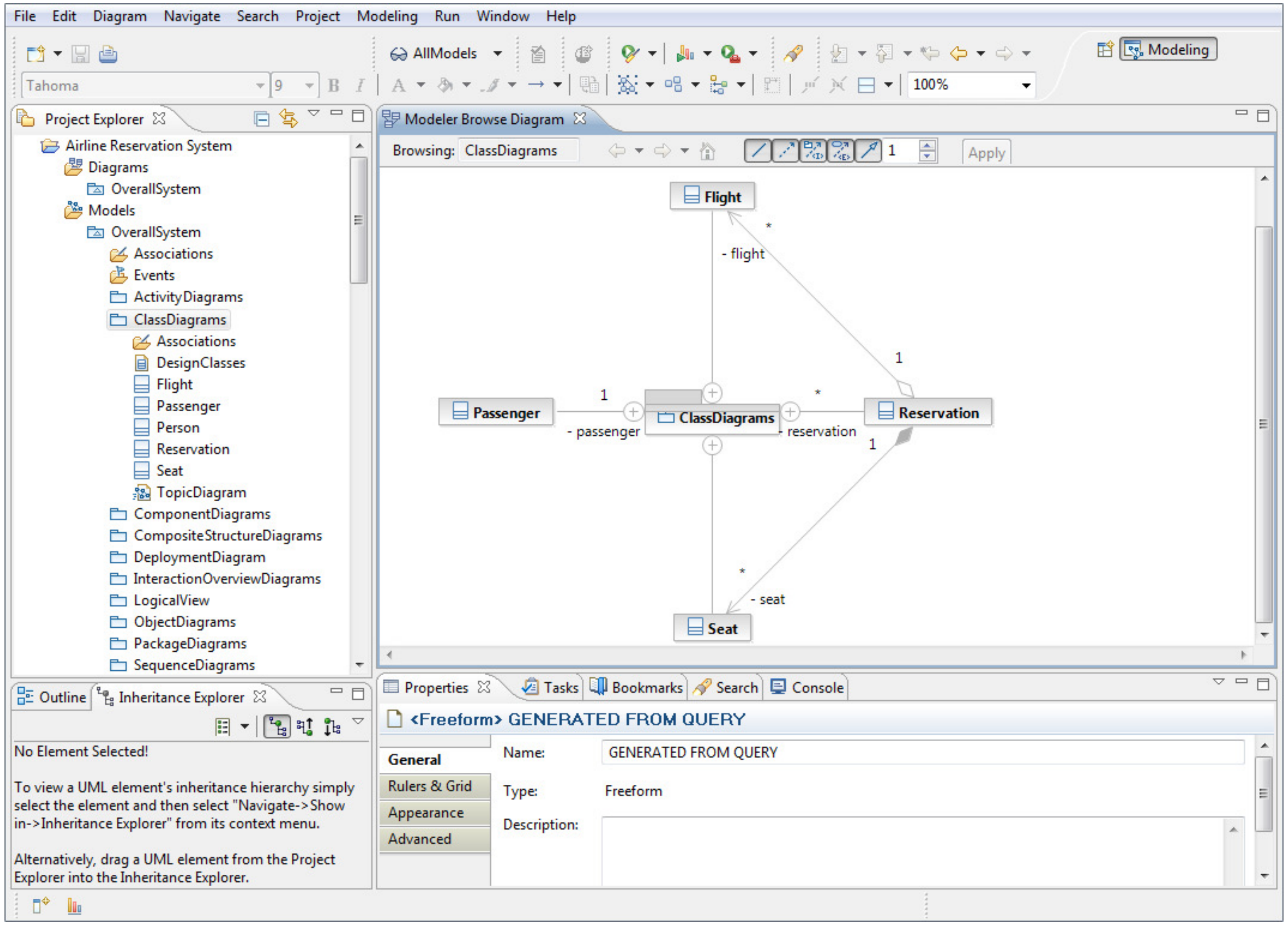
Figure 4.11 Topic Diagram

Browse Diagrams are similar to Topic Diagrams in which it begins by selecting topical elements and then defining filters that govern which kinds of related elements will be depicted. However, Browse Diagrams do not have a persisted definition and they are not specifically for model organization. Their purpose is to facilitate discovery and understanding of model content by enabling graphically navigate through a model without having to manually compose diagrams [Smith, 2008]. Browse Diagrams can be created simply using the project explorer window by selecting the model content and then follow the steps of context menu as; Visualize > Explore in Browse Diagram. Finally the Browse Diagram is generated automatically depicting the selected element as the focal point with related elements presented in a radial layout around the focal point. As an example Figure 4.12 depicts a Browse Diagram for the airline reservation system that is sampled in Chapter 2.

Topic Diagram and Browse Diagram are both created by selecting existing model elements that are created during the development of the projects. RSM puts some restrictions while using these two diagrams. The content of these diagrams changes automatically according to the changes in the content of models. However this action is not bidirectional. It is not allowed to change the contents directly using the Topic Diagram or Browse Diagram. The view of both diagrams is static. So model elements and relationships between these model elements cannot be modified using these two diagrams even their locations in the screen.

Visual modeling with content assistance guides with action bars, connection handles, context sensitive content suggestions, task specific modeling cheat sheets, extensive online help, samples and tutorials [IBM, December 2006] for creating well formed models.

An increased predictability and repeatability of software development can be achieved by pattern and transform authoring. The authoring and apply capabilities support teams in developing artifacts for reuse and developing artifacts with reuse. RSM includes tools for developing custom transformations that might target any type of implementation outputs and transformations between UML models at different levels of abstraction.



No Element Selected!

To view a UML element's inheritance hierarchy simply select the element and then select "Navigate->Show in->Inheritance Explorer" from its context menu.

Alternatively, drag a UML element from the Project Explorer into the Inheritance Explorer.

<Freeform> GENERATED FROM QUERY

Name: GENERATED FROM QUERY

Type: Freeform

Description:

Figure 4.12 Browse Diagram

Customizing and extending the modeling environment is supported by open Application Program Interface (API). Plug-ins can be developed by the organizations. Also the analysis and design tools for environment and process can be configured depending on the organization.

UML designs create reports and documentation that can be in the forms of HTML, Portable Document Format (PDF) and XML [IBM, December 2006]. These reports can be reviewed by team members or other stakeholders.

Team support with multi model support, compare merges and Software Configuration Management (SCM) integrations provides all the capabilities required for distributed teams to design and develop applications.

4.3.2 Capabilities

The way of using UML modeling by RSM can also range from very formal to very informal depending on the organization. Models can be chosen like formal architectural drawings that are to be strictly followed during construction or models can be in the form of sketches that suggest the broad outlines of a design. Rational Software Modeler can provide support at either end of these process and modeling spectrums [Smith, 2006].

Rational Software Modeler provides a software development platform which can be used to take the Model Development Architecture (MDA) approach for developing software applications [Cernosek, 2004]. Among the major functionality supported by Rational Software Modeler is the capability to share and communicate the modeled complexity within team, or across teams through model publishing.

Rational Software Modeler includes perspectives, which enables switching the view to show the toolbars that are needed [Mittal, 2005]. The perspectives and features which most commonly used are:

- **Modeling Perspective**: This view is used to create and manage UML assets. Rational Software Modeler allows building any UML diagram: use case diagrams, class diagrams, object diagrams, sequence diagrams, collaboration diagrams, and so on.

- **Requirements Perspective**: This view is for integration with IBM Rational RequisitePro which is a requirement and use case management tool using familiar document based methods. IBM Rational RequisitePro improves the communication of project goals, reduce project risks and increase the quality of applications before deployment. The result is better communication and management of requirements with the increased likelihood of completing projects on time, within budget and above expectations.
- **Model Publishing**: This view allows publishing model so that other team members can view it.

Raw UML models can be stored or the source code is generated from the models. There is no automated synchronization features between the UML models and the code. It can be easily generated one from the other. This feature is great for developers because an initial version of the code right from the UML diagrams can be viewed by the developer, which saves them up front development time. Developers are more motivated to spend additional time during the design phase when they know that they will get something tangible from it.

CHAPTER 5

CASE STUDY

5.1 Existing Information System

Çankaya University Library was founded at 1997 and began its structuring and functioning. It uses open shelves for the library layout. It is decided to use LC (Library of Congress)'s implementation for the classification technique and AACR II (Anglo American Cataloging Rules) is applied. "Connexion" catalog system is used to generate library collection, make cataloging and classification much faster and reliable within international standards. At 1998 it became a member of OCLC which is an international information sharing and distribution center. Book sharing is performing by the system of Inter Library Loan that cooperates with the surrounding university libraries in Ankara.

Çankaya University Library is using BLISS-PC software package developed and marketed by Bilkent University for library automation. Library has a website that is rearranged to be comprehensive and the library catalog is available to scan over the Internet. Çankaya University Library is a member of various online databases which can be accessed through library website. There is a network system (Library Intranet) within the library that has a single point of entry and exit. There is 16 Internet connection points available within the Çankaya University Library that can be upgraded to 50. Also an announcement list is in use for the purpose of announcing any information about the library to all university members.

Dynamic and successful services are provided by Çankaya University Library. Staff has special skills as in the following:

- Knowledgeable about education and training;
- Experienced in information sources selection, arrangement and use;
- Expert in library automation and information technologies.

Çankaya University Library services are depicted in Figure 5.1 as a business use case diagram.

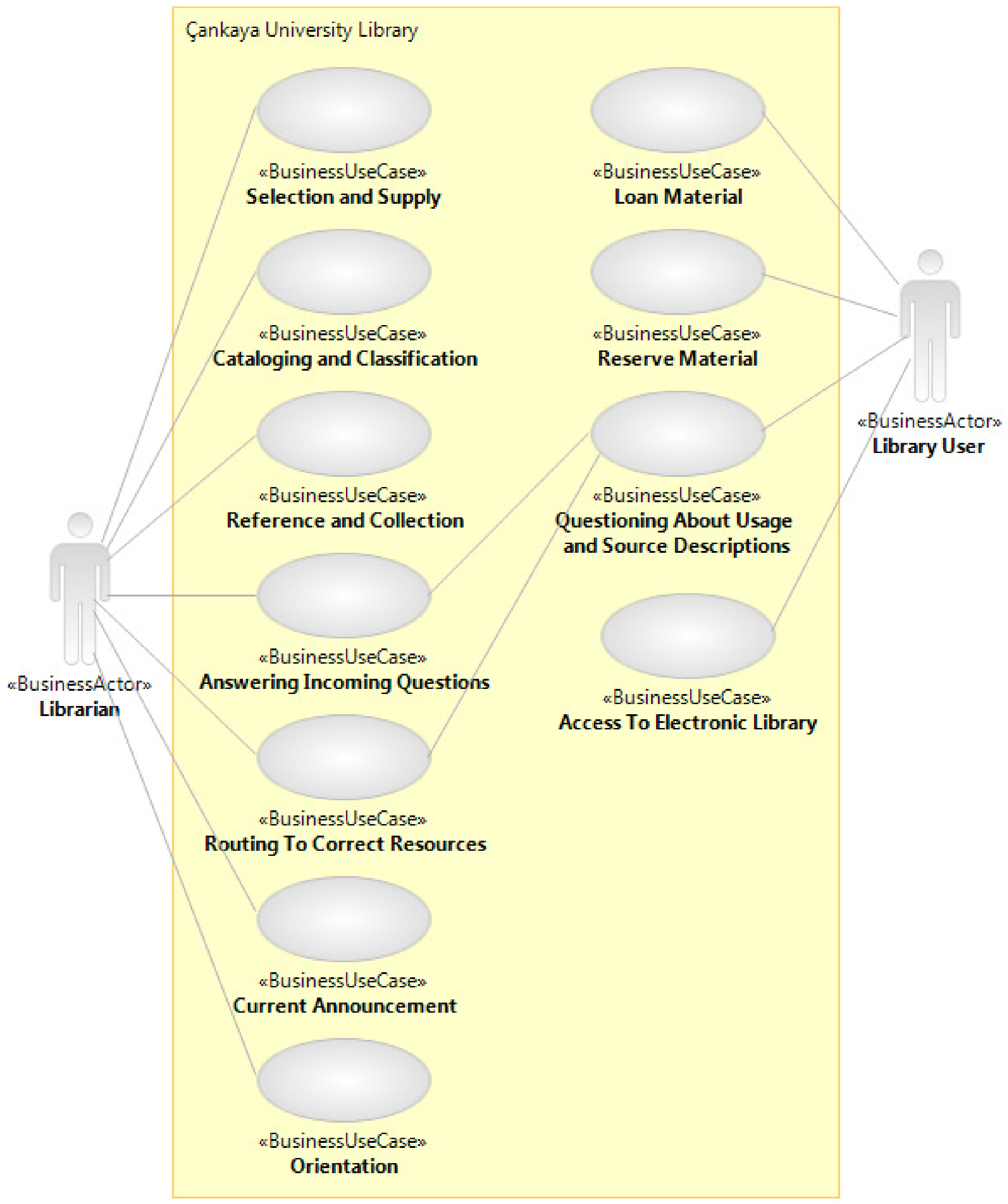

Çankaya University Library
«BusinessUseCase» Selection and Supply
«BusinessUseCase» Loan Material
«BusinessUseCase» Cataloging and Classification
«BusinessUseCase» Reserve Material
«BusinessUseCase» Reference and Collection
«BusinessUseCase» Questioning About Usage and Source Descriptions
«BusinessActor» Library User
«BusinessActor» Librarian
«BusinessUseCase» Answering Incoming Questions
«BusinessUseCase» Access To Electronic Library
«BusinessUseCase» Routing To Correct Resources
«BusinessUseCase» Current Announcement
«BusinessUseCase» Orientation

Figure 5.1 Çankaya University Library Services

The organizational structure of Çankaya University Library is shown in Figure 5.2:

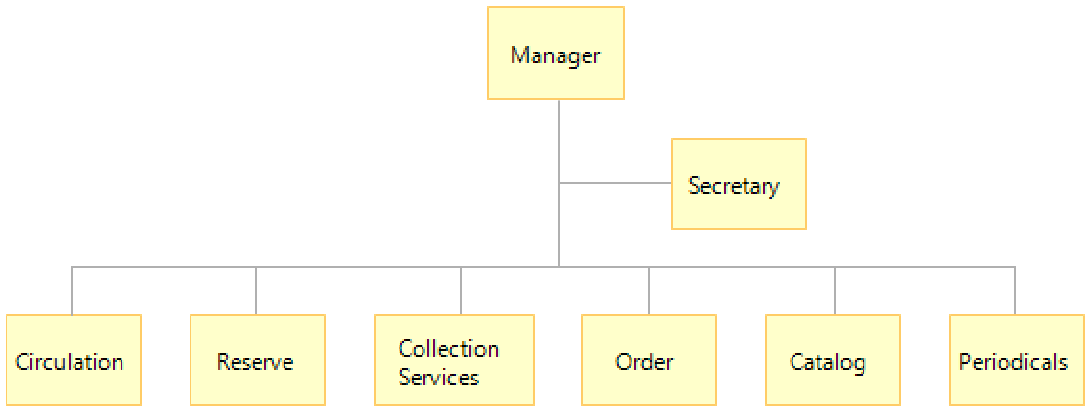


Figure 5.2 Organizational Structure

Some service policies of the Çankaya University Library should be listed as in the following [cankaya.edu.tr]:

- Contribute to training and education programs on users effectively;
- Help academic readers for their professional work;
- Respond to needs about courses and evaluate the free time of students and provide them to interest the constructive, argumentative and aesthetic values;
- To provide any additional information resources, organizing and providing services, to all of our users, in the manner of development of thinking skills and knowledge to meet the requirements;
- Following innovations and migrate them to life;
- Providing positive attitude to readers.

5.2 Existing Problem

Çankaya University Library has a reservation system currently used for sharing books, academic magazines and documents across lecturers and students. At present, reservation operations are performed manually by the staff who is responsible for the reserve operations. Lecturers loan some materials related to their courses to be placed on the reserve shelf of the library for students to utilize them during the term. Limited number of copies of such materials causes some problems. Firstly because of the limitation on the number of copies, each student who reserves the material has to return it in a couple of hours in a day. This situation decreases the efficiency and causes other students to wait for the return of the material. Some other problems that are caused by the borrowers include the loss of material. This is the worst situation, because sometimes there is only one copy of the loaned material. Another problem is that the materials sometimes do not return on time.

5.3 Solution to the Problem

The manual reservation system is going to be automated by developing and implementing an online **e-Reserve** system. The objective of this MS Thesis study is to develop such a system. Existing manual system will be still working for sometime and e-Reserve system is planned to assist the existing system until it is fully operational and reliable. By the help of e-Reserve system current problems on reservation are expected to be solved or at least minimized. Newly proposed system will have three groups of users which are students, instructors and librarian. Student is the main user of the system. Student can download books and documents about courses that are uploaded by either course instructor or by the librarian (by scanning reserve books). An instructor can thus upload books or documents about his/her course(s) so that students can download them for studying or for making researches. Librarian can be considered as the administrator of this project. Librarian can do all the operations on all courses and will be able to view the usage reports and system performance.

Thus, major problems of the existing system such as the limitation on number of loaned copies and delay of return will be avoided and there is no need for one borrower to wait for another for reserving the material, thus there will be simply almost no waiting time.

CHAPTER 6

APPLICATION

6.1 Selecting RUP

In order to apply a new approach to develop the **e-Reserve System** for Çankaya University Library, possible solutions were examined first. We looked for a development methodology that could be easily customized and integrated with existing processes. The goal was to create a robust and practical system based on object-oriented development process tailored as an answer to the needs of Çankaya University Library. RUP appeared to be the perfect answer for such a development project.

6.2 Project Initiation

The need behind the **e-Reserve** project is the necessity of an online reservation system for Çankaya University Library. The problem of the existing system and a suitable solution to the problem is mentioned in the previous Chapter. Our main goal is to produce an adequate executable system by adopting Rational Unified Process (RUP) methodology.

In general, Rational Unified Process is applied on software development projects in two major configurations either for small projects or large projects. When the project is defined we have to decide if it is a small or a large project by estimating the duration of the project, the amount of code to be produced, the amount of money to be spent on the project or the complexity involved. By evaluating these characteristics for our case study, in Chapter 5, it is found that the most suitable configuration will be the "Rational Unified Process for small projects".

The most popular debate between Rational Unified Process and small projects is that many programmers feel that the Rational Unified Process is too rigid and too structured for small development projects. One can, however, configure Rational Unified Process to fit his/her project's needs. Recent changes to Rational Unified Process have made it easier to navigate and configure. We are expecting in this application to show that Rational Unified Process not only applies to large projects but it can also be used effectively on small projects. It shares many of its principles with other methodologies, including agile methods which are advocated for small projects.

6.3 RMC Preparation

After we decided to begin the project using RUP, we first started to use IBM Rational Method Composer (RMC) which is discussed in Chapter 4 to create a method plug-in for e-Reserve System. The method plug-in is a well-formed definition of a component of a method in terms of its method elements and their relationships. In RMC, a method plug-in is a container for method packages which includes method elements. Method elements composed of following elements:

- Content Element: Consists of role, task, work product, and guidance;
- Process Element: Consists of activity, capability pattern, delivery process, and guidance.

All content about the project is organized in that method plug-in. With method plug-ins, the content can be organized at a level of granularity that meets the needs for authoring and reusing content. When a method plug-in is created, we can reference other plug-ins by reusing the content; modifying or extending the content; or adding our own content to those plug-ins that is depicted in the present section. A method plug-in can also be standalone and not reference other plug-ins. Method plug-ins can also perform a supporting role. Supporting method plug-ins provide reusable content for other method plug-ins. The content that is stored in a supporting method plug-in is only visible and published for a method configuration if other content that is not in a supporting plug-in references it.

The RMC main window which is depicted in Figure 4.1 in Chapter 4 contains menu items in which all actions can be performed using these items. So, method plug-ins can be created simply using the menu by following the steps of as; File > New > Method Plug-in as shown in Figure 6.1.

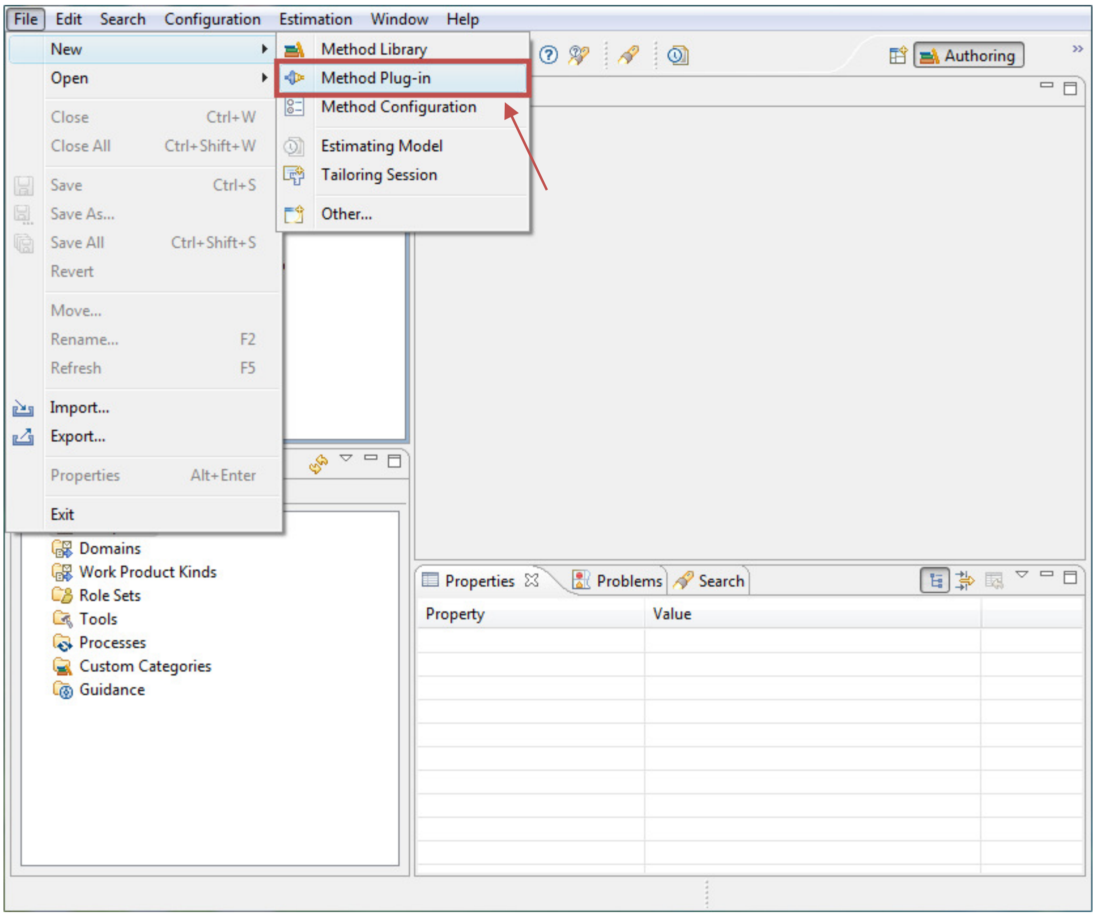
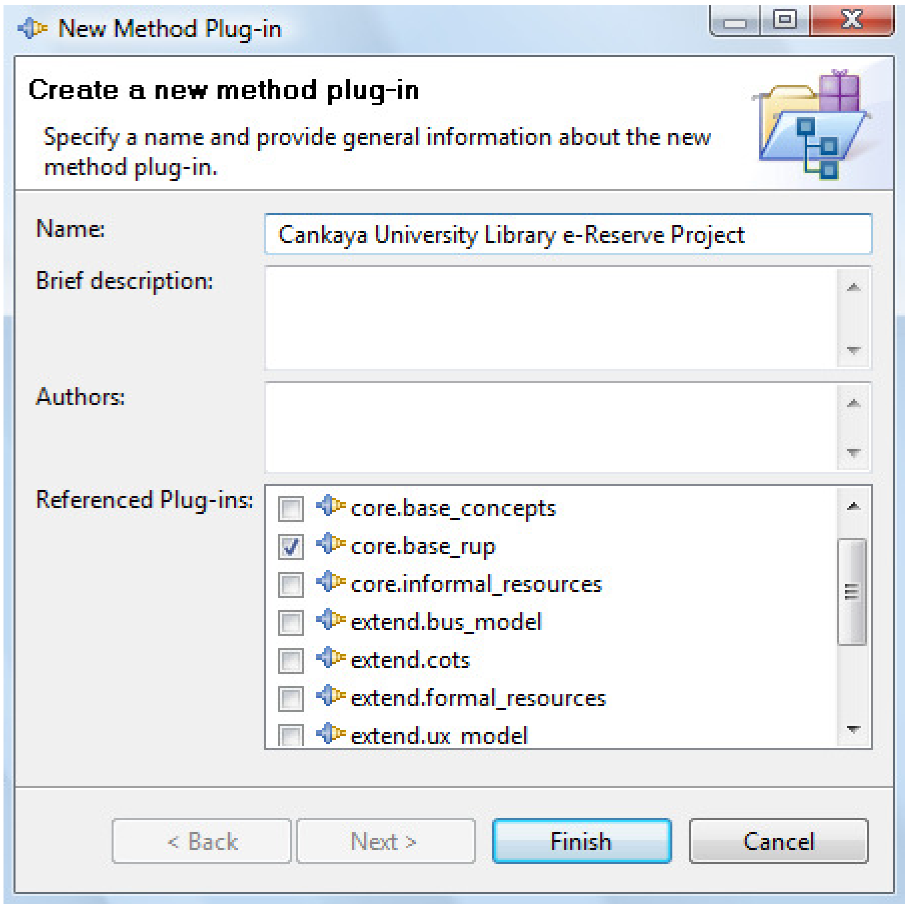


Figure 6.1 Creating Method Plug-in

After completing the steps, the New Method Plug-in wizard is opened that guides you to begin creating your own method plug-in. First of all, user enters the name of the method plug-in such as “Cankaya University Library e-Reserve Project” that we did in our project. In the next step, the Referenced Plug-ins is selected that contains additional plug-ins that is referenced as shown in the Figure 6.2. Referenced Plug-ins, identify plug-ins that will have content contributed to extended or replaced. While selecting the Referenced Plug-ins we have to know what type of project to deal with. As we mentioned before, the size of the project will be small and we decided to apply classical RUP methodology. So, core.base_rup plug-in is selected that includes plug-in for RUP for small projects. Also the wizard for creating the new method plug-in provides an additional area to make a brief description about the method plug-in and extra information about the author(s) of this part.


New Method Plug-in

Create a new method plug-in

Specify a name and provide general information about the new method plug-in.

Name: Cankaya University Library e-Reserve Project

Brief description:

Authors:

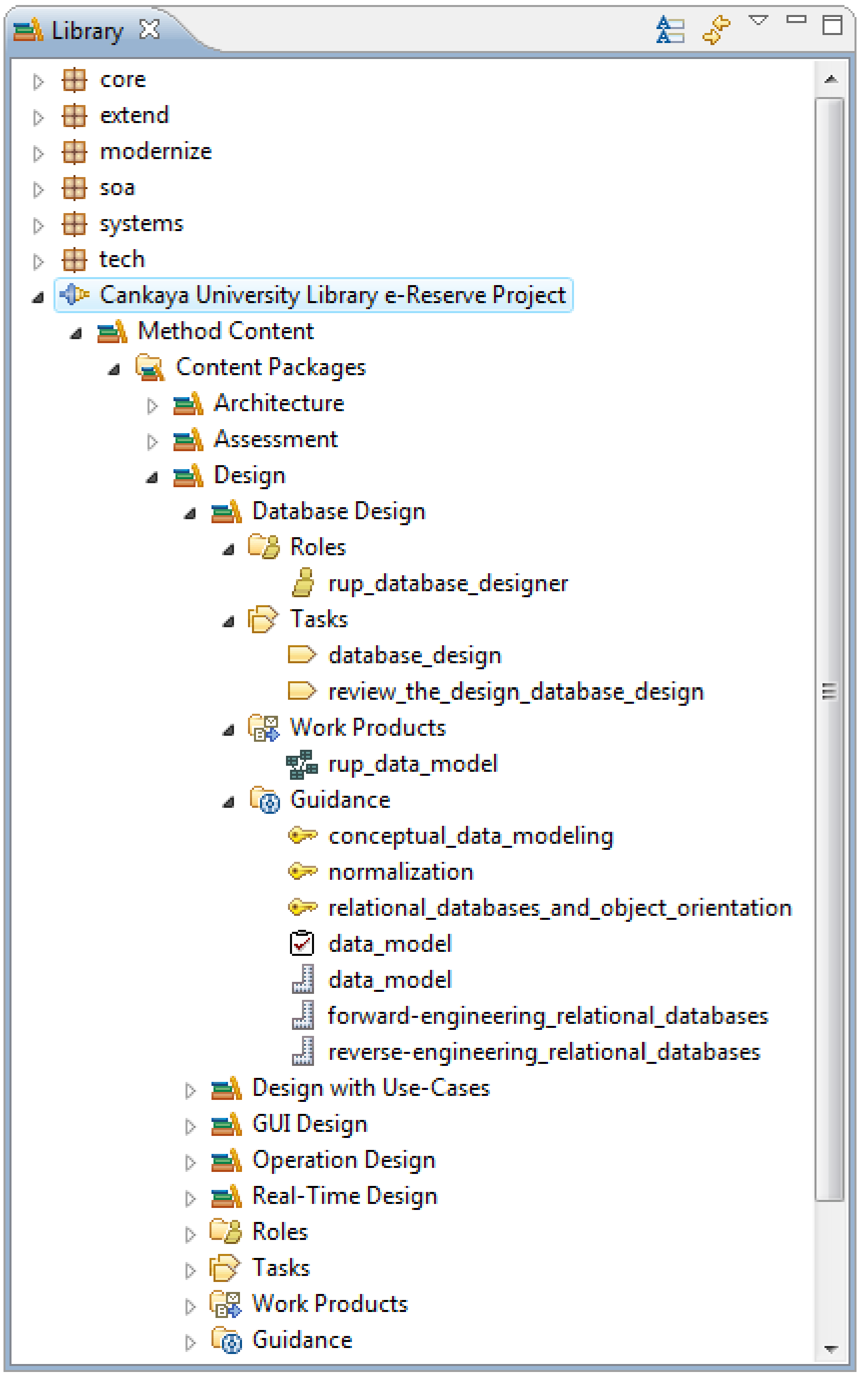
Referenced Plug-ins:
- [ ] core.base_concepts
- [✓] core.base_rup
- [ ] core.informal_resources
- [ ] extend.bus_model
- [ ] extend.cots
- [ ] extend.formal_resources
- [ ] extend.ux_model

Figure 6.2 Method Plug-in Wizard

After creating our method plug-in we have to prepare the method content. Method content provides step-by-step explanations, describing how specific development goals are achieved independent of the placement of these steps within a development lifecycle. Processes take the method elements and relate them into semi-ordered sequences that are customized to specific types of projects.

A method content element can be created manually or it can be referenced by using best practices that is most suitable to the current project by modifying its content. So our method content of “Cankaya University Library e-Reserve Project” is created by simply copying the elements from “base_rup” plug-in method content. Some method contents and their elements are excluded that are not necessary to our project.

Finally some of the remaining elements are modified to adapt our project and some of them are used as in their original form. Part of newly created method content from “Cankaya University Library e-Reserve Project” is shown in the Figure 6.3.



Library

- core
- extend
- modernize
- soa
- systems
- tech
- Cankaya University Library e-Reserve Project
  - Method Content
    - Content Packages
      - Architecture
      - Assessment
      - Design
        - Database Design
          - Roles
            - rup_database_designer
          - Tasks
            - database_design
            - review_the_design_database_design
          - Work Products
            - rup_data_model
          - Guidance
            - conceptual_data_modeling
            - normalization
            - relational_databases_and_object_orientation
            - data_model
            - data_model
            - forward-engineering_relational_databases
            - reverse-engineering_relational_databases
        - Design with Use-Cases
        - GUI Design
        - Operation Design
        - Real-Time Design
        - Roles
        - Tasks
        - Work Products
        - Guidance

Figure 6.3 Method Content

Now we have to create a new method configuration for our new method plug-in. As we mentioned before in Chapter 4, RMC offers a library containing a great deal of reusable content. None of the organizations or projects requires all of this documentation all at once, but would work with a selection of specific subsets. So method configuration allows us to specify working sets of content and processes for a specific context, such as a specific variant of the RUP framework that user wants to publish and deploy for a given software project or as a foundation for a development organization. Simply, method configuration is a configuration that allows us to select or deselect from the method packages available in our library's set of plug-ins.

Method configuration can be created simply in to the Configurations folder using the menu by following the steps of as; In the Library, right-click the Configurations folder and click New > Method Configuration as shown in Figure 6.4.

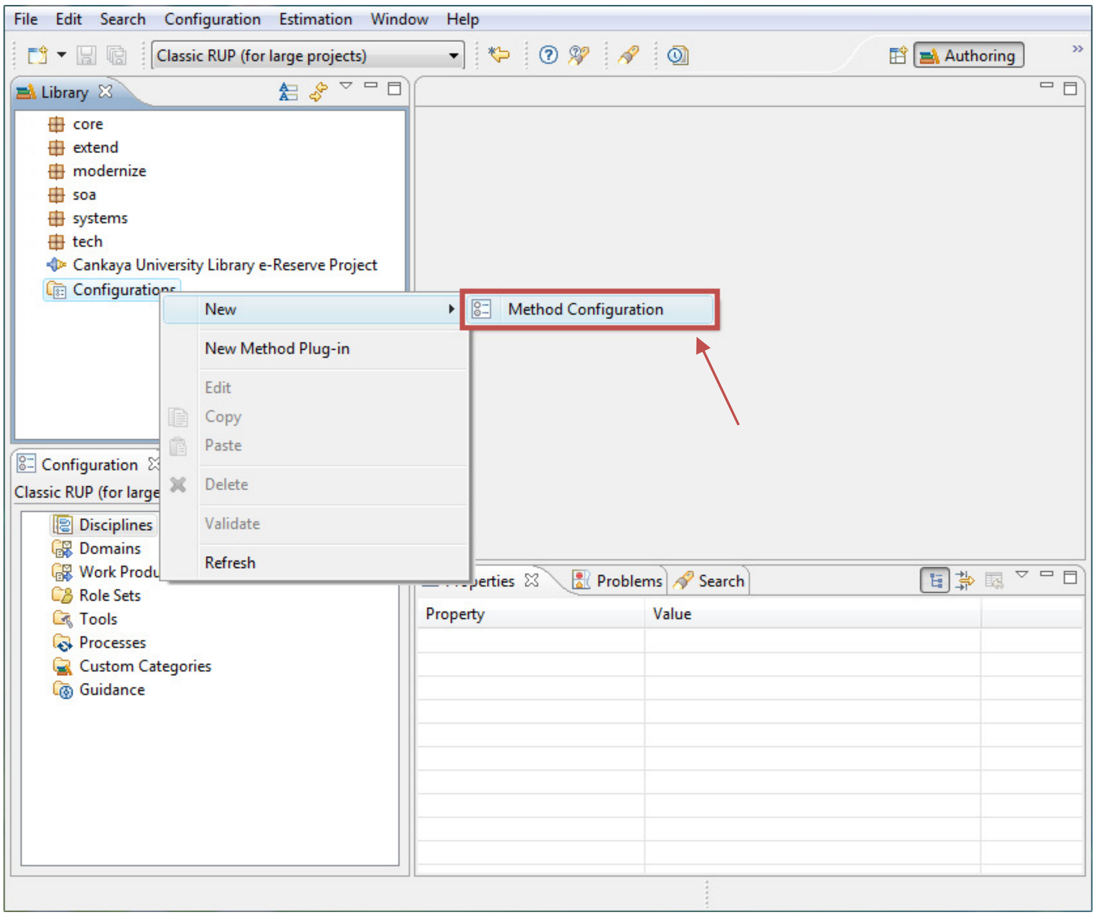
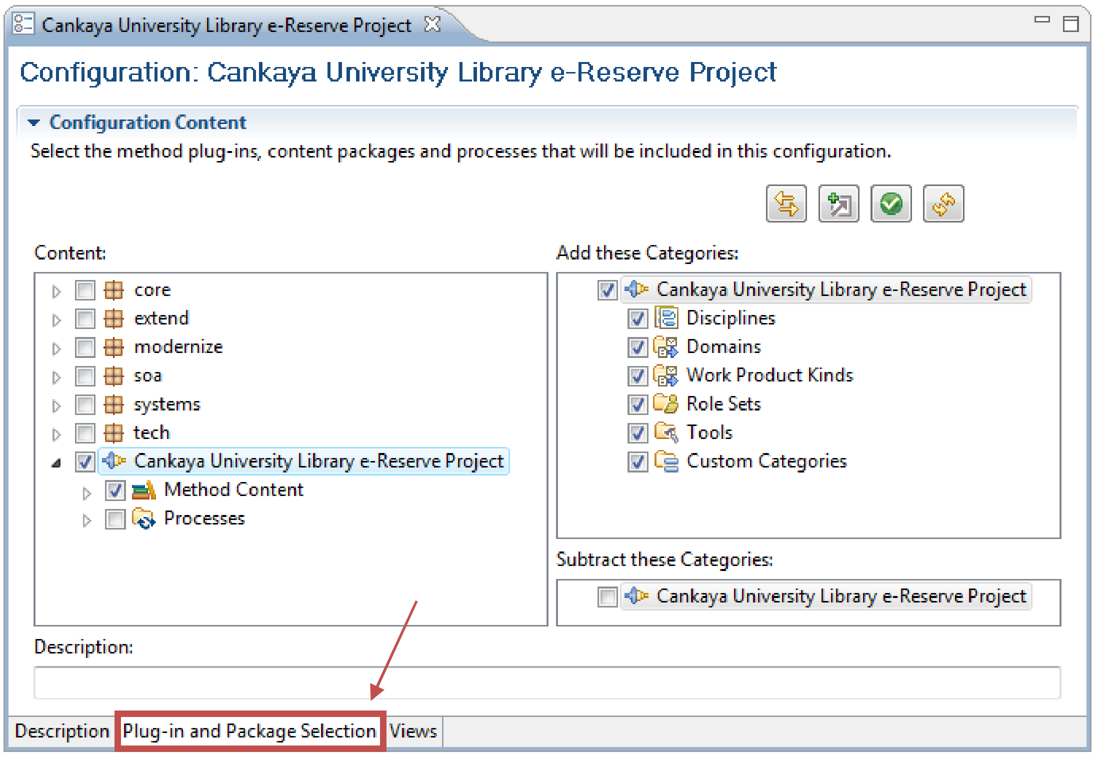


Figure 6.4 Creating Method Configuration

Then the method configuration wizard greets us. This is the initial step before modifying our configuration that we firstly name our configuration as "Cankaya

University Library e-Reserve Project". Then, the configuration editor is opened and name of the configuration is given as "Cankaya University Library e-Reserve Project". As a result our new configuration is created. At the bottom of the editor, we click the **Plug-in and Package Selection** tab to modify configuration in more details as shown in Figure 6.5. List boxes in the right hand side are used to add or remove elements to define our configuration.


Cankaya University Library e-Reserve Project

Configuration: Cankaya University Library e-Reserve Project

Configuration Content

Select the method plug-ins, content packages and processes that will be included in this configuration.

Content:
- [ ] core
- [ ] extend
- [ ] modernize
- [ ] soa
- [ ] systems
- [ ] tech
- [✓] Cankaya University Library e-Reserve Project
  - [✓] Method Content
  - [ ] Processes

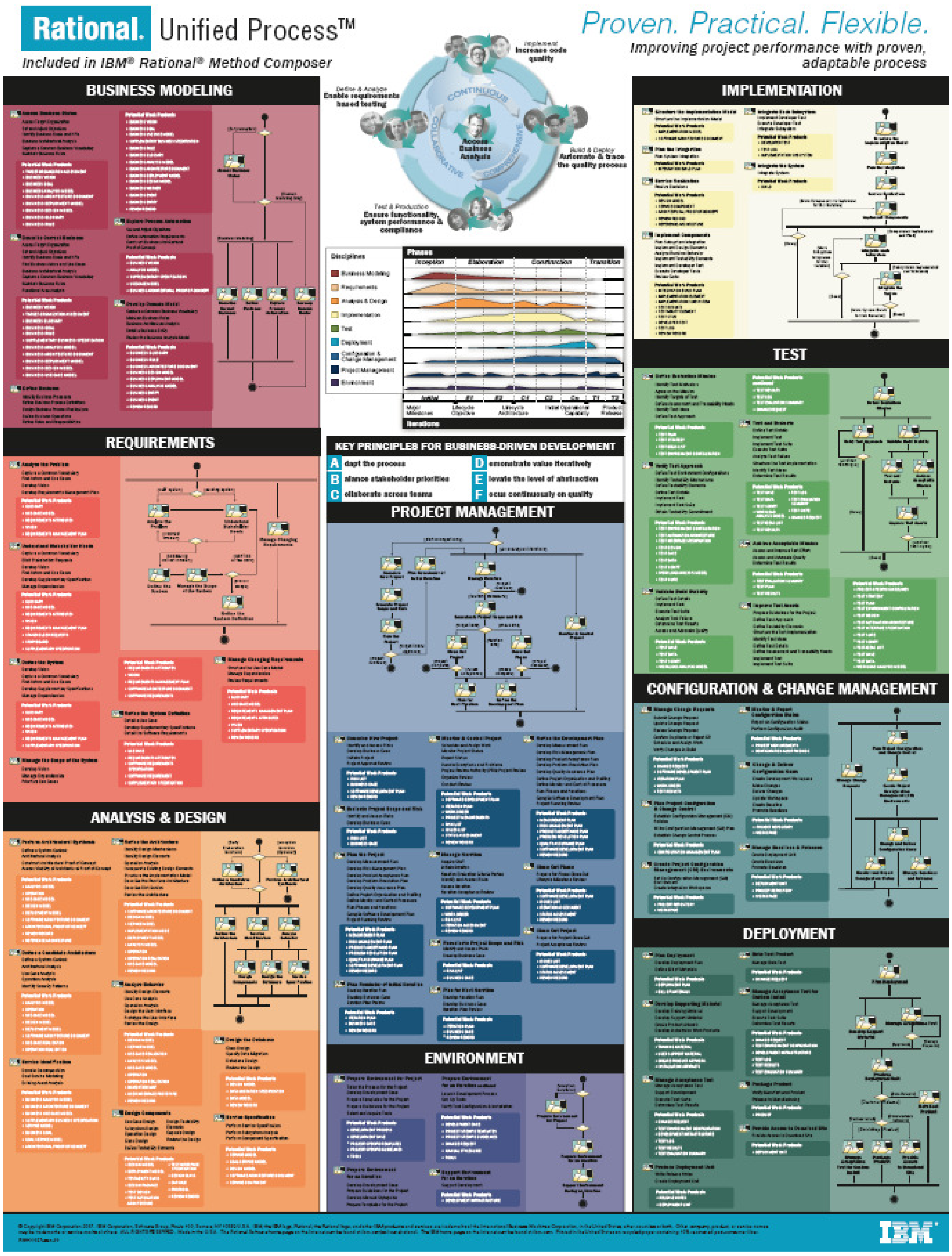
Add these Categories:
- [✓] Cankaya University Library e-Reserve Project
  - [✓] Disciplines
  - [✓] Domains
  - [✓] Work Product Kinds
  - [✓] Role Sets
  - [✓] Tools
  - [✓] Custom Categories

Subtract these Categories:
- [ ] Cankaya University Library e-Reserve Project

Description:

Figure 6.5 Creating New Method Configuration

We mentioned the delivery process and capability patterns already in Chapter 4. For our project we have to define a complete integrated approach that will be used in a given pattern. The capability patterns can be either in discipline workflows that are listed in **Appendix A** regarded using the recent IBM Poster [IBM, 2007b] as given in Figure 6.6 or in templates for delivery processes which will be found in RMC. We will construct our delivery processes by simply choosing the templates for delivery processes and making some modifications on it to adapt our project.



Rational. Unified Process™
Included in IBM® Rational® Method Composer
Proven. Practical. Flexible.
Improving project performance with proven, adaptable process
BUSINESS MODELING
IMPLEMENTATION
TEST
REQUIREMENTS
KEY PRINCIPLES FOR BUSINESS-DRIVEN DEVELOPMENT
A dapt the process
B alance stakeholder priorities
C ollaborate across teams
D emonstrate value iteratively
E levate the level of abstraction
F ocus continuously on quality
PROJECT MANAGEMENT
CONFIGURATION & CHANGE MANAGEMENT
ANALYSIS & DESIGN
ENVIRONMENT
DEPLOYMENT
Disciplines
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Management
Project Management
Environment
Phases
Inception
Elaboration
Construction
Transition
Iterations
IBM

Figure 6.6 RUP Discipline Workflows

Capability patterns express and communicate process knowledge for a key area of interest, such as a discipline, and can be directly used by process practitioners to guide their work. Capability patterns are also used as building blocks to assemble delivery processes or larger capability patterns ensuring optimal reuse and application of the key practices they express. We begin to construct our capability patterns by using "Templates for Delivery Processes" for small projects as a reference from the base_rup plug-in in RMC. So firstly we create four capability patterns that will form the phases of our project. Capability patterns can be created simply using the menu by following the steps of as; In the Library, right-click the Capability Patterns under Processes and click New > Capability Pattern as shown in Figure 6.7. We name capability patterns as Inception Iteration, Elaboration Iteration, Construction Iteration, and Transition Iteration that emphasizes each iteration in its specific phase in our project. One more capability pattern is created which is named as RUP Phases. This capability pattern is used to combine other four capability patterns and provide an overview and order for the project phases.

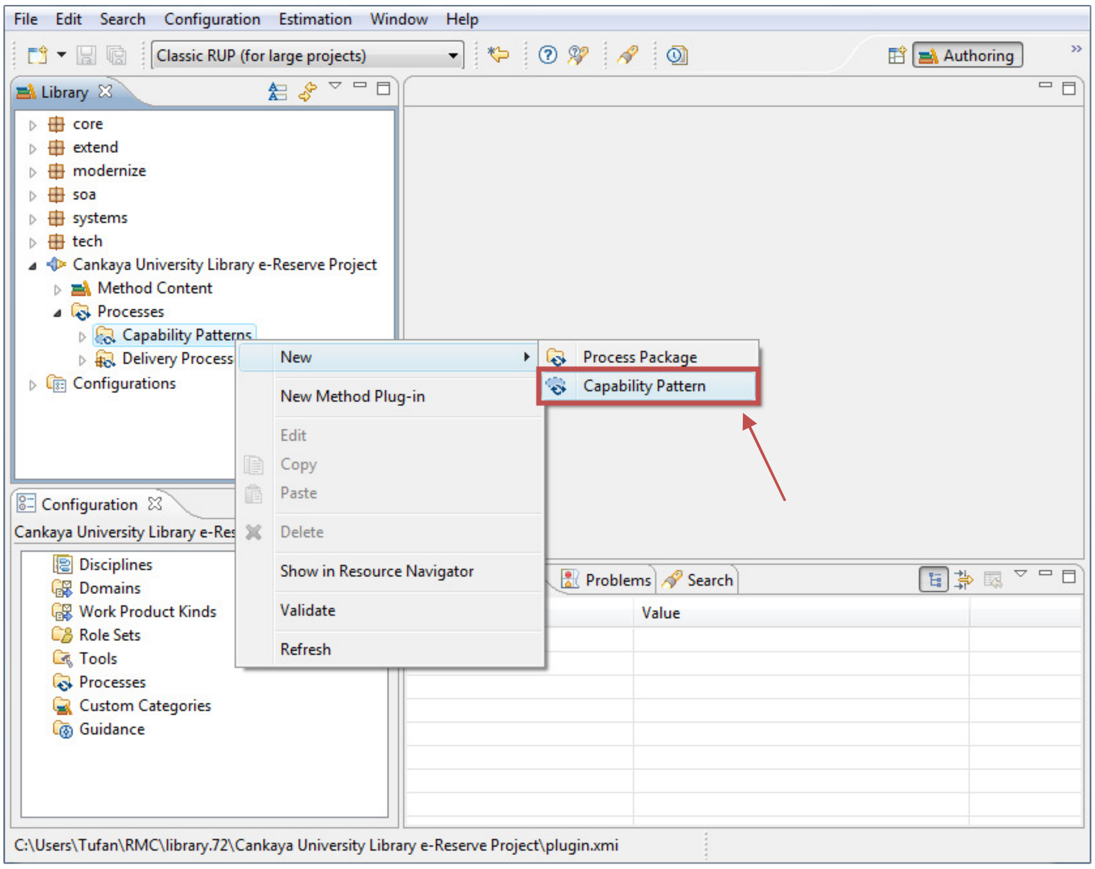
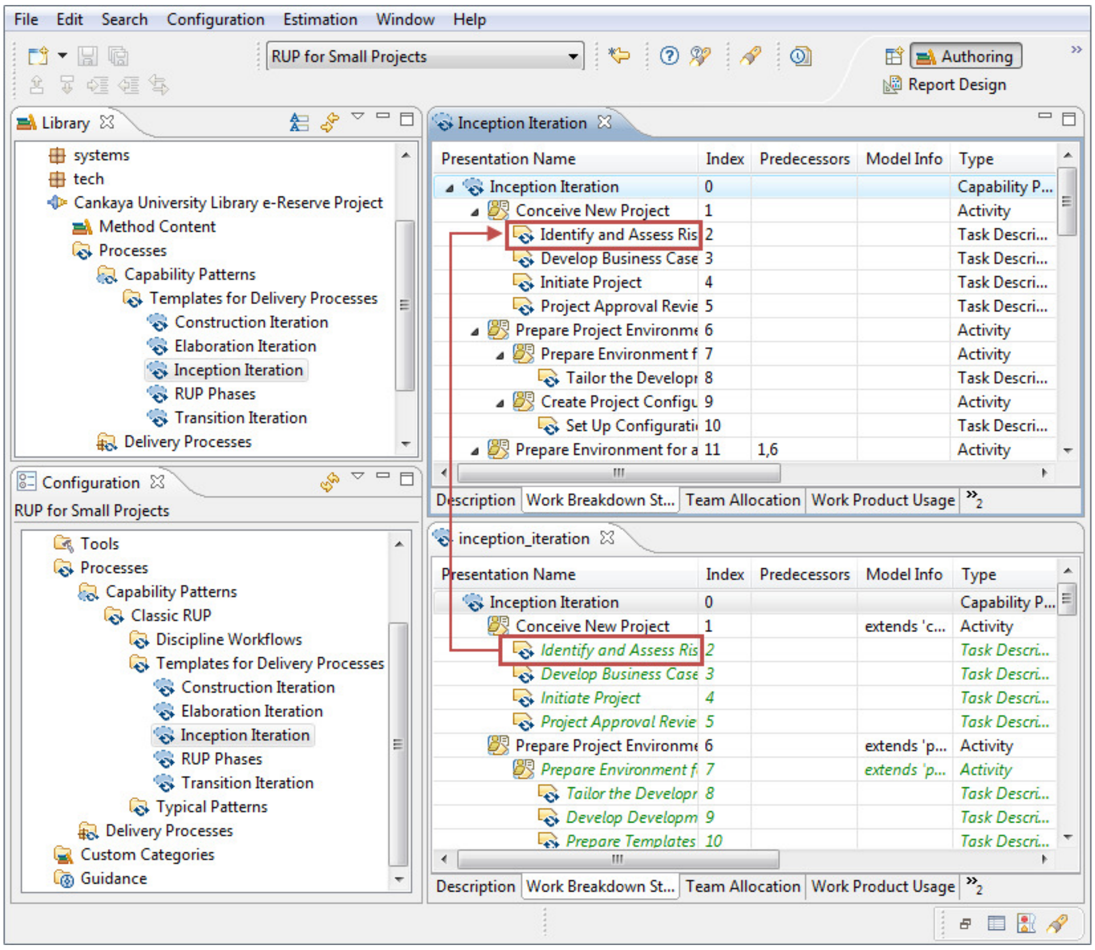


Figure 6.7 Creating Capability Patterns

After creating the capability patterns, we have to define activities and tasks for each of these activities. As we mentioned before, there is no need to define all activities and its tasks from the beginning because we are using “base_rup” in RMC as a reference point for our project. So firstly we choose the configuration “RUP for small projects”. Then in the Library window we open the capability pattern that we want to define and in the configuration window we chose the related capability pattern that will be used as a guide for preparing our capability pattern. Figure 6.8 depicts the view during the configuration of our capability pattern named as “Inception Iteration” in RMC.



Library
- systems
- tech
- Cankaya University Library e-Reserve Project
  - Method Content
  - Processes
    - Capability Patterns
      - Templates for Delivery Processes
        - Construction Iteration
        - Elaboration Iteration
        - Inception Iteration
        - RUP Phases
        - Transition Iteration
    - Delivery Processes

Configuration
RUP for Small Projects
- Tools
- Processes
  - Capability Patterns
    - Classic RUP
      - Discipline Workflows
      - Templates for Delivery Processes
        - Construction Iteration
        - Elaboration Iteration
        - Inception Iteration
        - RUP Phases
        - Transition Iteration
      - Typical Patterns
  - Delivery Processes
- Custom Categories
- Guidance

Inception Iteration

Presentation Name	Index	Predecessors	Model Info	Type
Inception Iteration	0			Capability P...
Conceive New Project	1			Activity
Identify and Assess Ris	2			Task Descri...
Develop Business Case	3			Task Descri...
Initiate Project	4			Task Descri...
Project Approval Revie	5			Task Descri...
Prepare Project Environme	6			Activity
Prepare Environment f	7			Activity
Tailor the Developr	8			Task Descri...
Create Project Configu	9			Activity
Set Up Configurati	10			Task Descri...
Prepare Environment for a	11	1,6		Activity

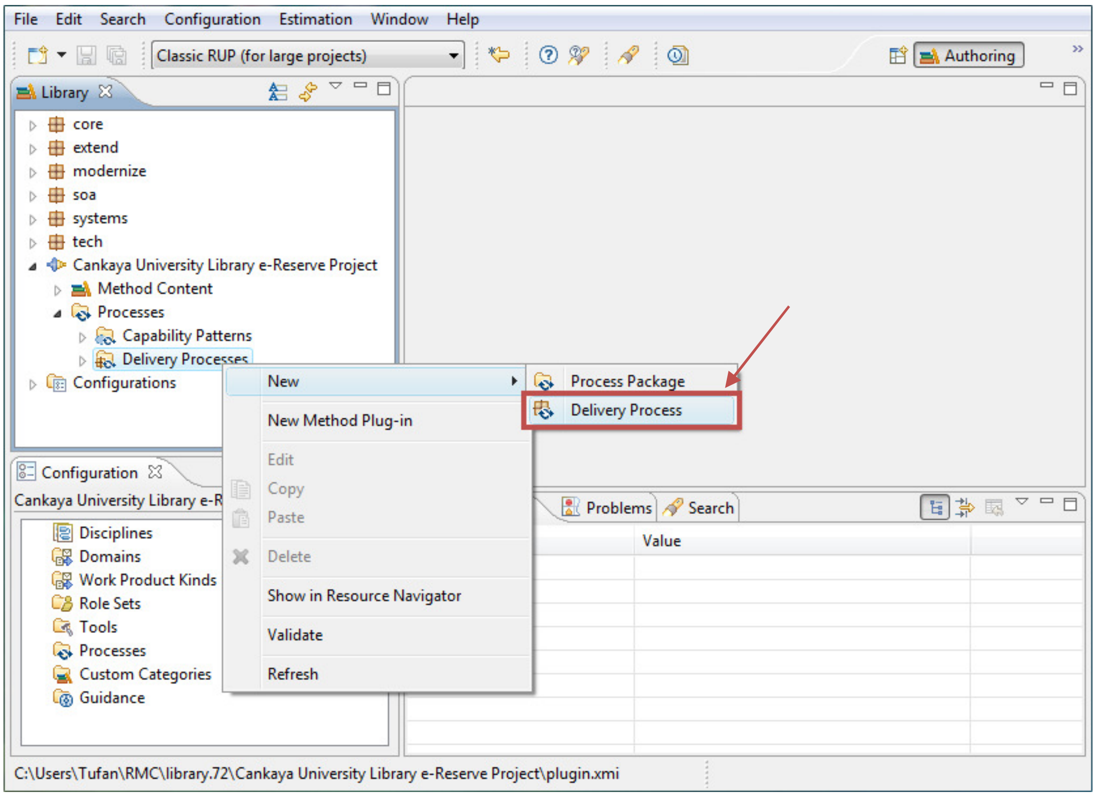
inception_iteration

Presentation Name	Index	Predecessors	Model Info	Type
Inception Iteration	0			Capability P...
Conceive New Project	1		extends 'c...	Activity
Identify and Assess Ris	*2*			*Task Descri...*
Develop Business Case	*3*			*Task Descri...*
Initiate Project	*4*			*Task Descri...*
Project Approval Revie	*5*			*Task Descri...*
Prepare Project Environme	6		extends 'p...	Activity
Prepare Environment f	*7*		*extends 'p...*	*Activity*
Tailor the Developr	*8*			*Task Descri...*
Develop Developm	*9*			*Task Descri...*
Prepare Templates	*10*			*Task Descri...*

Figure 6.8 Defining Capability Patterns

The work breakdown structure of the capability patterns can be examined briefly under the **Work Breakdown Structure** tab of the right bottom and top windows. Right bottom window shows the possible activities and its task that will be used in small projects during an inception iteration named as “inception_iteration”. So simply drag and drop any activity, task or both that are suitable to the current project to the right top window which is our created capability pattern named as “Inception Iteration”. Alternatively new activities and tasks can be created manually using the right click in the window of our newly created capability pattern that is “Inception Iteration”. Team allocation and work product usage of the current iteration can be viewed using the tabs under both window of capability patterns which are “Inception Iteration” and “inception_iteration”. They can be easily prepared using the same way as done for the activities and tasks by taking the advantage of drag and drop property of the RMC.

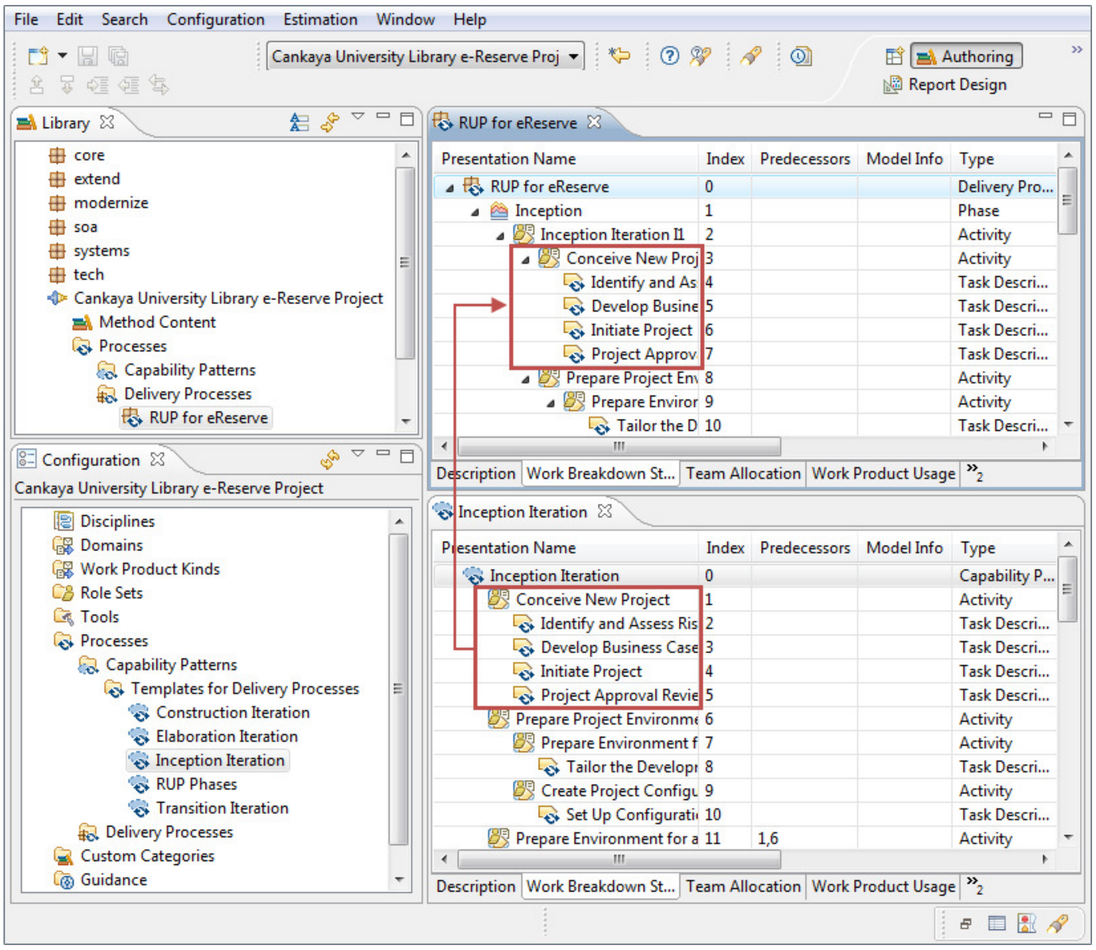
We thus completed defining our capability patterns so that we are ready to define the delivery process. A delivery process describes a complete and integrated approach for performing a specific type of project. A delivery process describes what is produced, how it is produced and the required staffing for the entire project lifecycle. Delivery process can be created simply using the menu by following the steps as follows; In the Library window, right-click the Delivery Processes under Processes and click New > Delivery Process as shown in Figure 6.9.



C:\Users\Tufan\RMC\library.72\Cankaya University Library e-Reserve Project\plugin.xmi

Figure 6.9 Creating Delivery Process

After creating the delivery process we have to define it using the capability patterns that we constructed in the previous step. So firstly we choose the configuration "Cankaya University Library e-Reserve Project" using the configuration selection box. Then in the Library window we open the delivery process that we want to define and in the configuration window we chose the capability patterns in a sequence of order that will form our delivery process. Figure 6.10 depicts the status of RMC during the configuration of our delivery process.



Presentation Name	Index	Predecessors	Model Info	Type
RUP for eReserve	0			Delivery Pro...
Inception	1			Phase
Inception Iteration I1	2			Activity
Conceive New Proj	3			Activity
Identify and As	4			Task Descri...
Develop Busine	5			Task Descri...
Initiate Project	6			Task Descri...
Project Approv	7			Task Descri...
Prepare Project Env	8			Activity
Prepare Enviror	9			Activity
Tailor the D	10			Task Descri...

Presentation Name	Index	Predecessors	Model Info	Type
Inception Iteration	0			Capability P...
Conceive New Project	1			Activity
Identify and Assess Ris	2			Task Descri...
Develop Business Case	3			Task Descri...
Initiate Project	4			Task Descri...
Project Approval Revie	5			Task Descri...
Prepare Project Environme	6			Activity
Prepare Environment f	7			Activity
Tailor the Developr	8			Task Descri...
Create Project Configu	9			Activity
Set Up Configurati	10			Task Descri...
Prepare Environment for a	11	1,6		Activity

Figure 6.10 Defining Delivery Process

Right bottom window shows the capability pattern that is named as "Inception Iteration". This capability pattern contains the activities and tasks which is defined in the previous steps. So simply drag and drop activities to the right top window which is our newly created delivery process. Alternatively new activities and tasks can be created manually using the right click in the window of our newly created delivery process.

As a result we obtain delivery processes for e-Reserve Project including inception, elaboration, construction, and transition phases as shown in the Figure 6.11.

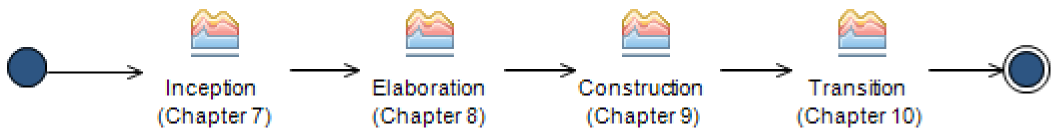


Figure 6.11 RUP Phases for the Project

Application of each phase is explained in detail in the following chapters. Each phase consists of activities which is a breakdown element that supports the nesting and logical grouping of related process elements. Activities can include one or more sub-activities. Each of these activities and sub-activities include tasks which describes a unit of work performed by specific roles. By the nature of Rational Unified Process, parallelisms are provided between activities/sub-activities/tasks. All activities and its tasks are discussed in detail with produced work products. While a task is being performed the emergent endeavor is represented by formal rules designated on work products. A work product is a content element that represents anything used, produced, or modified by a task. These work products can be in the form of document, model or model element whose templates can be found in the RMC work product templates as mentioned in Chapter 4. Work products have to be produced cautiously. When a task is concluded with a work product potentially it triggers another task, activity, or sub-activity.

6.4 RSM Preparation

For UML modeling of the system, IBM Rational Software Modeler (RSM) is used which is also discussed in Chapter 4. A UML model is a model that uses the UML notation to graphically represent a system at various levels of abstraction. UML models can be used to visually represent the system that is to be built, to communicate our vision of system with customers, and for direct code generation. UML models contain model elements, such as actors, use cases, classes, and packages, and one or more diagrams that show a specific perspective of a system. In RSM, models can be created and managed easily by using Model Projects. At the beginning of the e-Reserve project, we create the Model Project. We have to create this Model Project first, because we need to hold our UML 2.0 model artifacts that will be created while the development of the project continues.

The RSM main window which is depicted in Figure 4.9 in Chapter 4 contains menu items in which all actions can be performed using these items. So, the Model Project can be created simply using the menu by following the steps as follows; File > New > Model Project as shown in Figure 6.12.

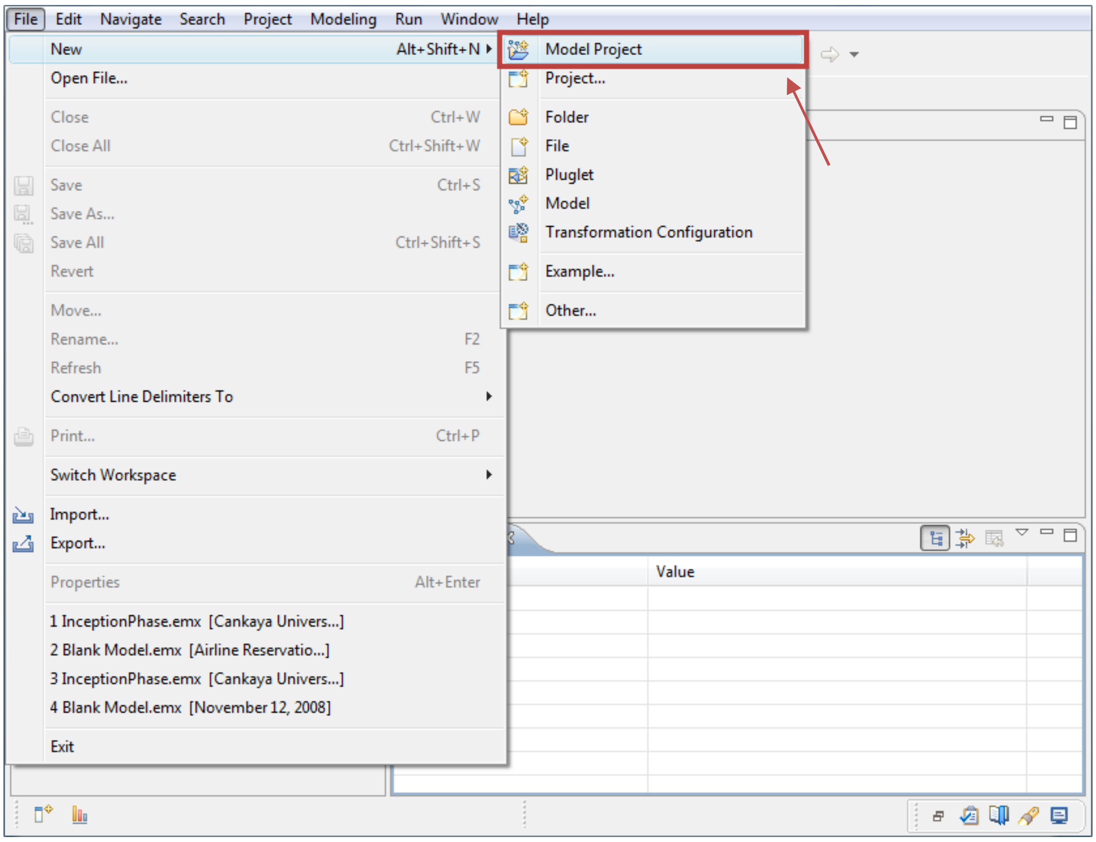
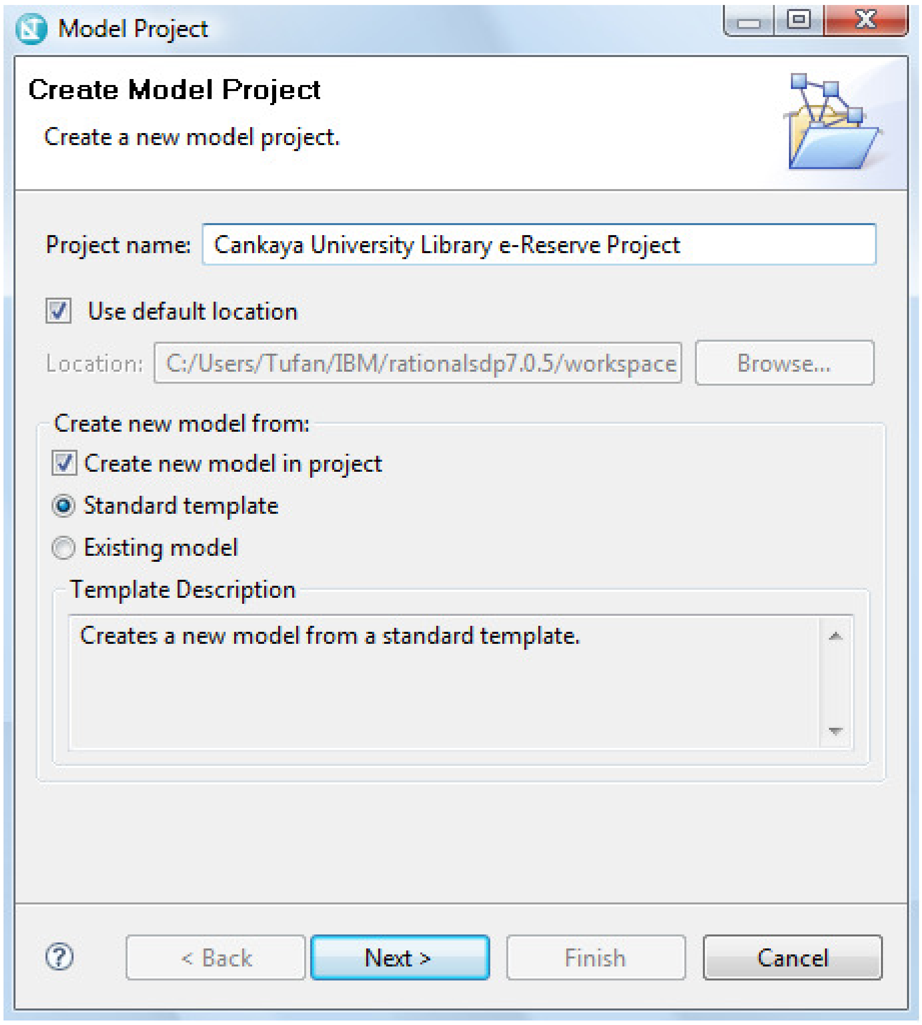


Figure 6.12 Creating Model Project

Then, the Model Project wizard is opened and user enters the name of the Model Project such as “Cankaya University Library e-Reserve Project”. RSM provides several templates that will be used to create models. Each template helps to create content for a particular type of model. So, we choose the Standard Template in the Model Project wizard as shown in Figure 6.13. Also existing models can be chosen if there exists any other past works that are completed in previous projects. Initial step of the Model Project creation is completed by assigning a name to the project and choosing the Standard Template.



Create Model Project

Create a new model project.

Project name: Cankaya University Library e-Reserve Project

[✓] Use default location

Location: C:/Users/Tufan/IBM/rationalsdp7.0.5/workspace

Create new model from:

[✓] Create new model in project

[●] Standard template

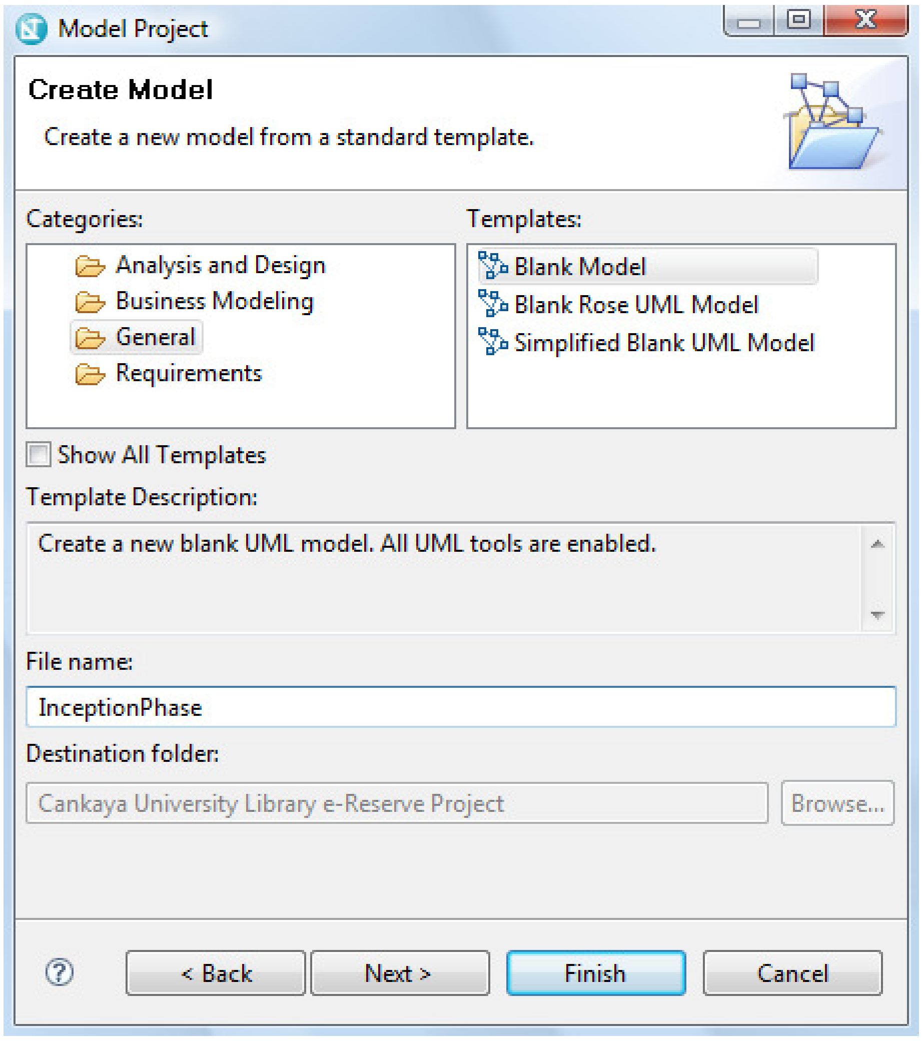
[○] Existing model

Template Description

Creates a new model from a standard template.

Figure 6.13 Model Project Wizard Step 1

In the next step of the wizard, we configure the type of model that will be initially created automatically and form a baseline for the entire of the project which will be placed in “Cankaya University Library e-Reserve Project” Model Project. First of all we assign a name to the model which is “InceptionPhase” as shown in Figure 6.14. This model will contain UML model artifacts that will be created within the Inception Phase of our project. The wizard window has two main views. The view in the left hand side contains four categories of existing Standard Templates. One of the categories has to be chosen depending on the action that will be taken within this model which will be analysis, design, business modeling, requirements gathering, or all of them. So we choose the General that includes all other categories which are Analysis and Design, Business Modeling, and Requirements. The view in the right hand side contains the templates according to the selected category. After choosing the General category, we see three different templates. All of these templates contain blank UML models with different subsets of UML tools. For example, Blank Rose UML Model is created with a simplified subset of UML tools is enabled that emulates constructs available in Rational Rose. Also, Simplified Blank UML Model is created with a simplified subset of UML tools is enabled that emulates constructs available in UML 1.x. So, we choose the Blank Model which will create a blank UML model with access to all UML tools.



Model Project

Create Model

Create a new model from a standard template.

Categories: Analysis and Design, Business Modeling, General, Requirements

Templates: Blank Model, Blank Rose UML Model, Simplified Blank UML Model

Show All Templates

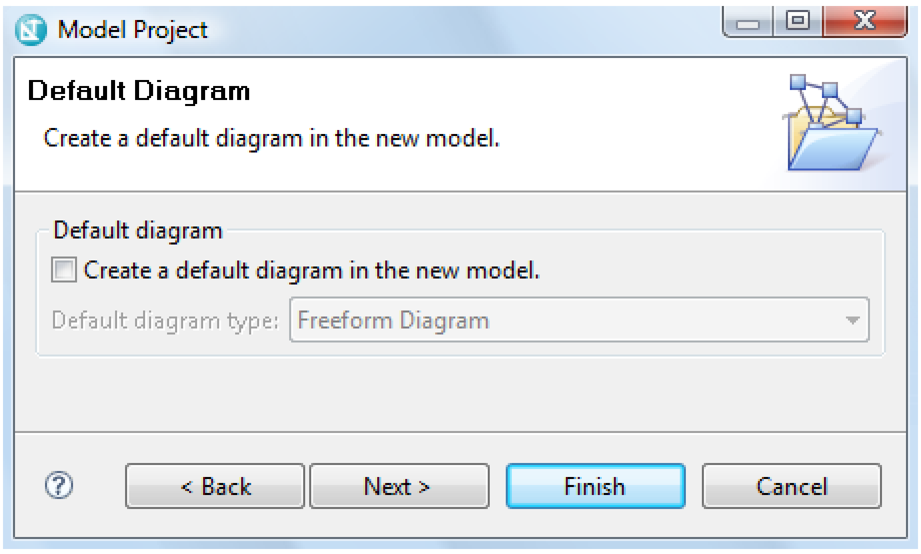
Template Description: Create a new blank UML model. All UML tools are enabled.

File name: InceptionPhase

Destination folder: Cankaya University Library e-Reserve Project

Figure 6.14 Model Project Wizard Step 2

In the last step of the Model Project wizard, we are able to create a freeform diagram as shown in Figure 6.15. If we choose to create a freeform diagram then it will be placed in the previously created model. We do not need to create a freeform diagram for now, because we did not decide which diagram will be created first yet. As a result we conclude the wizard by clicking Finish button.


Model Project

Default Diagram

Create a default diagram in the new model.

Default diagram

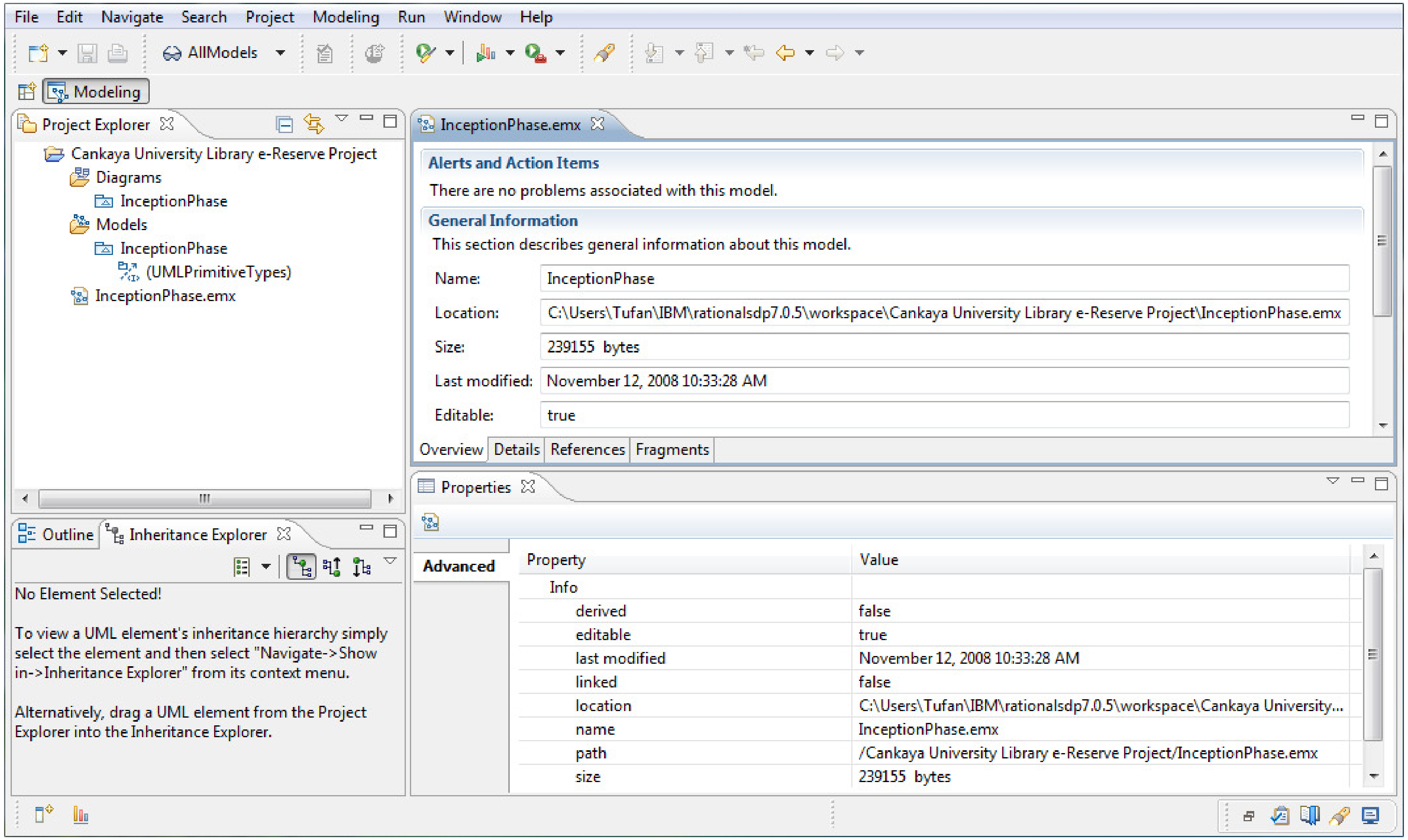
☐ Create a default diagram in the new model.

Default diagram type: Freeform Diagram

Figure 6.15 Model Project Wizard Step 3

After concluding the Model Project wizard the Model Project called "Cankaya University Library e-Reserve Project" and its initial model called "InceptionPhase" are created. These resources are represented to users as shown in Figure 6.16. The window on the left top side in RSM depicts the Project Explorer view. As we mentioned before, in RSM, we can create and manage models using modeling projects in the Project Explorer view. The contents of a modeling project are organized into two types of logical folders: diagrams and models. This structure displays the logical containment of the UML model elements, regardless of where they are stored physically. Models contained in a modeling project are displayed under the Models folder, or node. These nodes are not the physical model (.emx) files, but rather the root model elements of the models. Similarly, the corresponding diagrams in a modeling project are displayed under the Diagrams folder.

The window on the right top side gives some general information about the physical model which is "InceptionPhase.emx" such as size, location, creation date, and so on. Also the window on the right bottom side which is the Properties view gives some descriptive information about the physical model. Properties view displays property names and basic properties of a selected resource from Project Explorer view. The window on the left bottom side is Inheritance Explorer view as we mentioned in the previous sections that provides users to view a UML element's inheritance hierarchy. This property will be used in next sections.



Alerts and Action Items

There are no problems associated with this model.

General Information

This section describes general information about this model.

Name:	InceptionPhase
Location:	C:\Users\Tufan\IBM\rationalsdp7.0.5\workspace\Cankaya University Library e-Reserve Project\InceptionPhase.emx
Size:	239155 bytes
Last modified:	November 12, 2008 10:33:28 AM
Editable:	true

No Element Selected!

To view a UML element's inheritance hierarchy simply select the element and then select "Navigate->Show in->Inheritance Explorer" from its context menu.

Alternatively, drag a UML element from the Project Explorer into the Inheritance Explorer.

Property	Value
Info	
derived	false
editable	true
last modified	November 12, 2008 10:33:28 AM
linked	false
location	C:\Users\Tufan\IBM\rationalsdp7.0.5\workspace\Cankaya University...
name	InceptionPhase.emx
path	/Cankaya University Library e-Reserve Project/InceptionPhase.emx
size	239155 bytes

Figure 6.16 Initial Project Screen

Now our Model Project is ready and we can create new UML model elements using the Project Explorer view. In this step we are going to create a package which is a UML model element to store other UML model elements that will be constructed during the Inception Phase of e-Reserve project. A package can be created simply using the menu by following the steps as follows; In the Project Explorer view, right-click the model called "InceptionPhase" under Models folder and click Add UML > Package as shown in Figure 6.17.

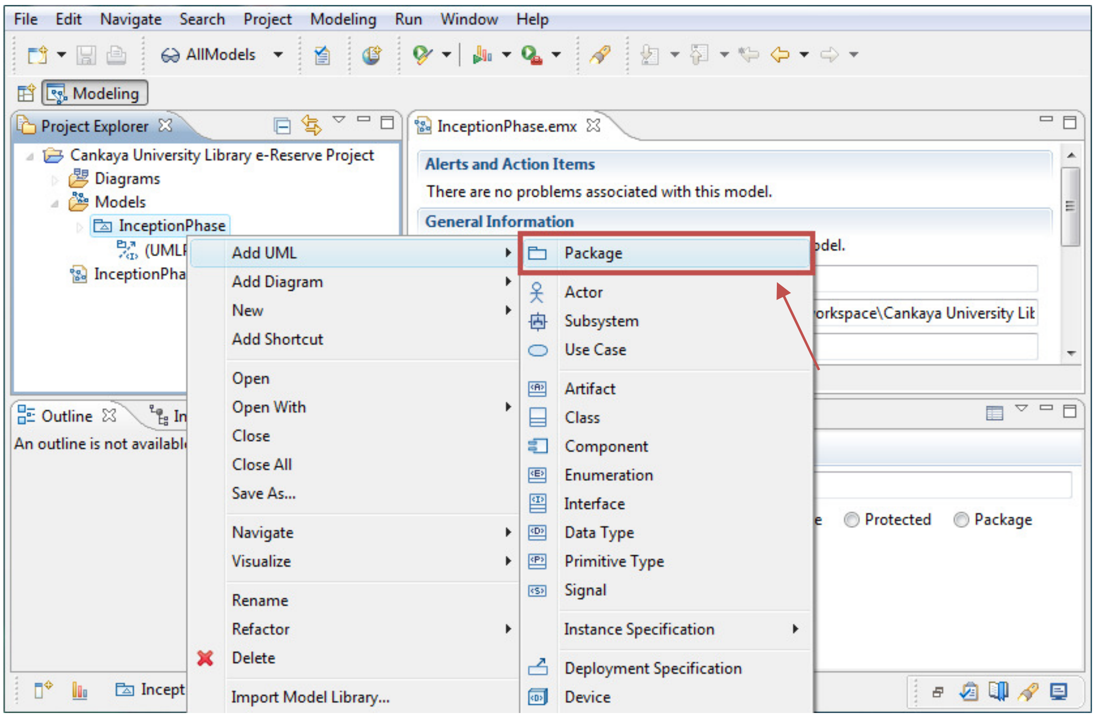


Figure 6.17 Creating UML Elements

We name the package as "IterationI1" which is depicted in Figure 6.18 that emphasizes the first iteration of the Inception Phase of e-Reserve project.

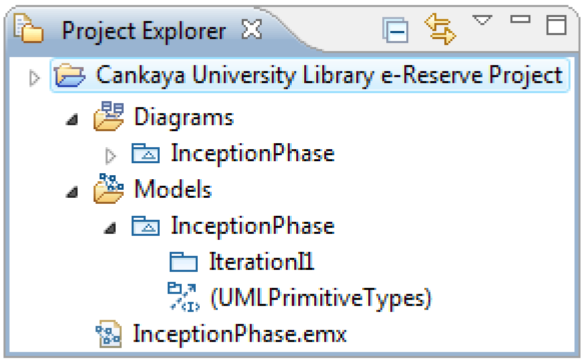


Figure 6.18 Explorer View of Creating UML Elements

CHAPTER 7

INCEPTION PHASE

We begin to structure our project by constructing delivery processes from the templates in RMC and making some modifications on it to adapt our project as mentioned in Chapter 6. Starting point of these delivery processes is the Inception phase that establishes the feasibility of the system and phase plays the most critical role in the project as mentioned in Chapter 3. Inception Phase of our project consists of one iteration which includes several activities and concluded by a milestone as shown in the Figure 7.1.

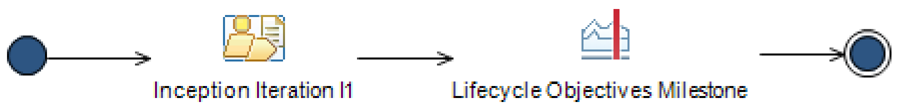


Figure 7.1 Inception Phase

Each activity consisting of one or more tasks performed in Inception Phase Iteration and each task is concluded by a work product that points the critical parts and summarizes the task. Work products are important to complete the project successfully and achieve its objectives. So they have to be prepared correctly by following unique method that will be easily understood by team members and an appropriate format with universal practices. RMC provides templates and examples of various types of work products to apply correct formats. So, the guide for work products can be reached simply using the menu by following the steps as follows; Help > RMC Process Browser as shown in Figure 7.2.

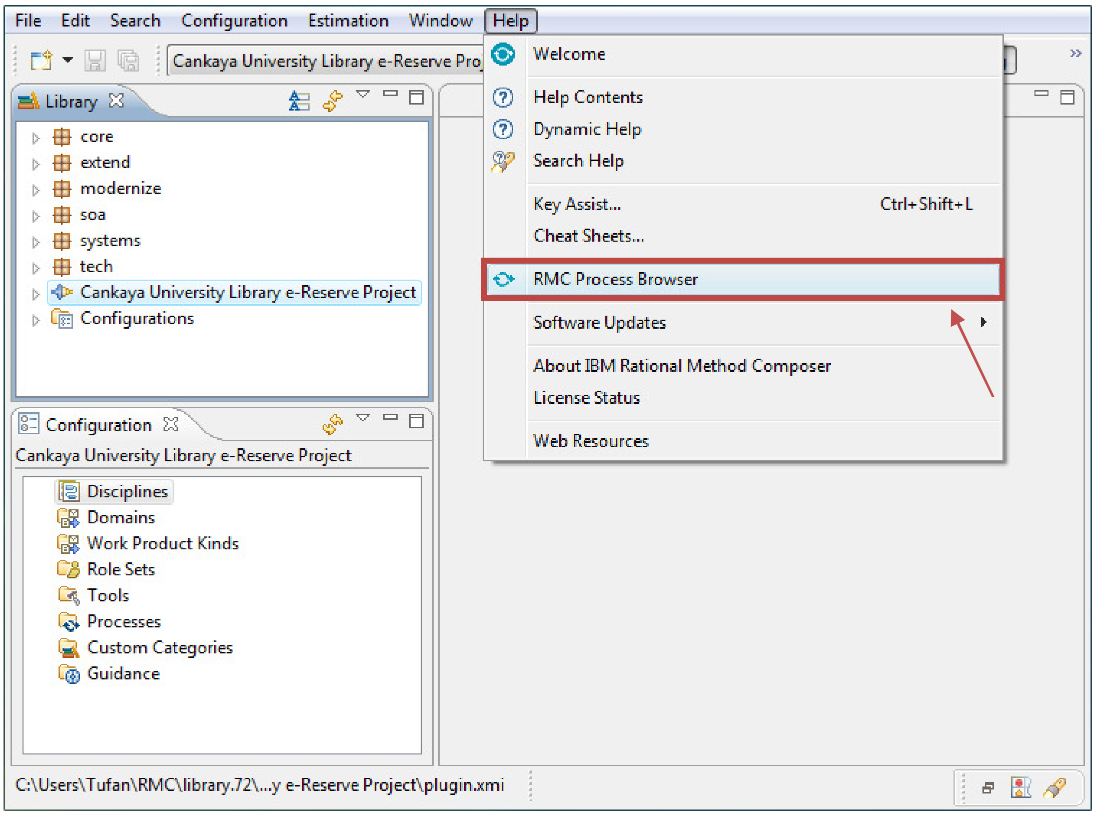
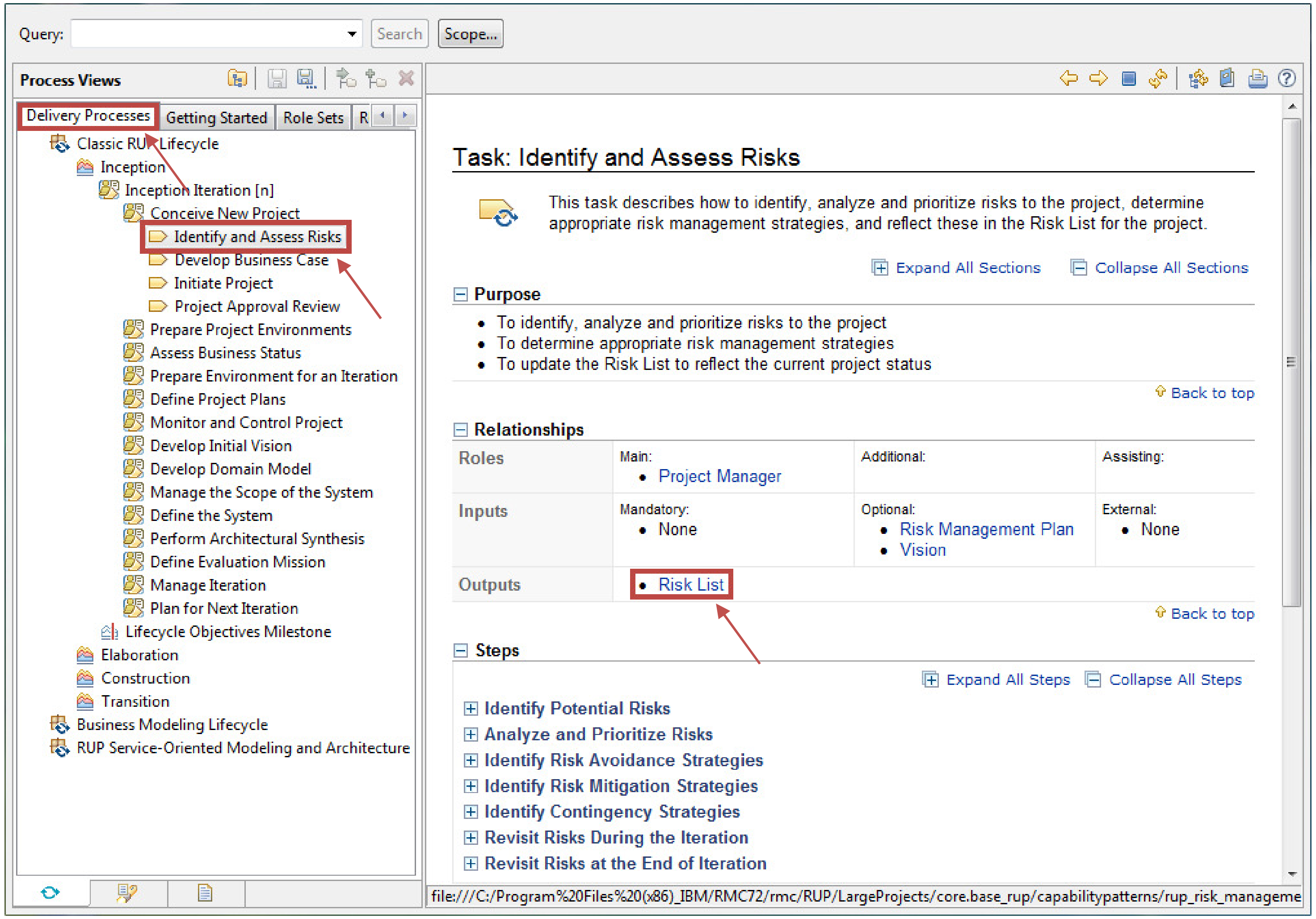


Figure 7.2 Open RMC Process Browser

The RMC Process Browser appears as shown in Figure 7.3 after following the previous steps depicted in Figure 7.2. Process Browser allows developers to reach instantaneously process guidance or policies, including the latest document templates they should use. RMC Process Browser has two views in its window. The window in the left most side contains the process view that provides the guidance and the window in the right most side shows the details of selected items. Our aim is to gain some knowledge about work products that will be produced during the development of the project. So, first of all we have to select the appropriate task under the activities of current iteration performed in that phase under the "Delivery Processes" tab as shown in Figure 7.3. Then the task specific information will appear in the right most side of the browser. This brief information tells us which work product, named as outputs, has to be produced within that selected task. Now we know which work product to produce. Process Browser provides more information about the work product when we click on it as shown in Figure 7.4. As a result, all necessary information can be obtained from templates section to produce work product and some samples from examples section.



Query: Search Scope...

Process Views

Delivery Processes | Getting Started | Role Sets | R

- Classic RUP Lifecycle
  - Inception
    - Inception Iteration [n]
      - Conceive New Project
        - Identify and Assess Risks
        - Develop Business Case
        - Initiate Project
        - Project Approval Review
      - Prepare Project Environments
      - Assess Business Status
      - Prepare Environment for an Iteration
      - Define Project Plans
      - Monitor and Control Project
      - Develop Initial Vision
      - Develop Domain Model
      - Manage the Scope of the System
      - Define the System
      - Perform Architectural Synthesis
      - Define Evaluation Mission
      - Manage Iteration
      - Plan for Next Iteration
    - Lifecycle Objectives Milestone
  - Elaboration
  - Construction
  - Transition
- Business Modeling Lifecycle
- RUP Service-Oriented Modeling and Architecture

Task: Identify and Assess Risks

This task describes how to identify, analyze and prioritize risks to the project, determine appropriate risk management strategies, and reflect these in the Risk List for the project.

Expand All Sections Collapse All Sections

Purpose

- To identify, analyze and prioritize risks to the project
- To determine appropriate risk management strategies
- To update the Risk List to reflect the current project status

Back to top

Relationships

Roles	Main: • Project Manager	Additional:	Assisting:
Inputs	Mandatory: • None	Optional: • Risk Management Plan • Vision	External: • None
Outputs	• Risk List		

Back to top

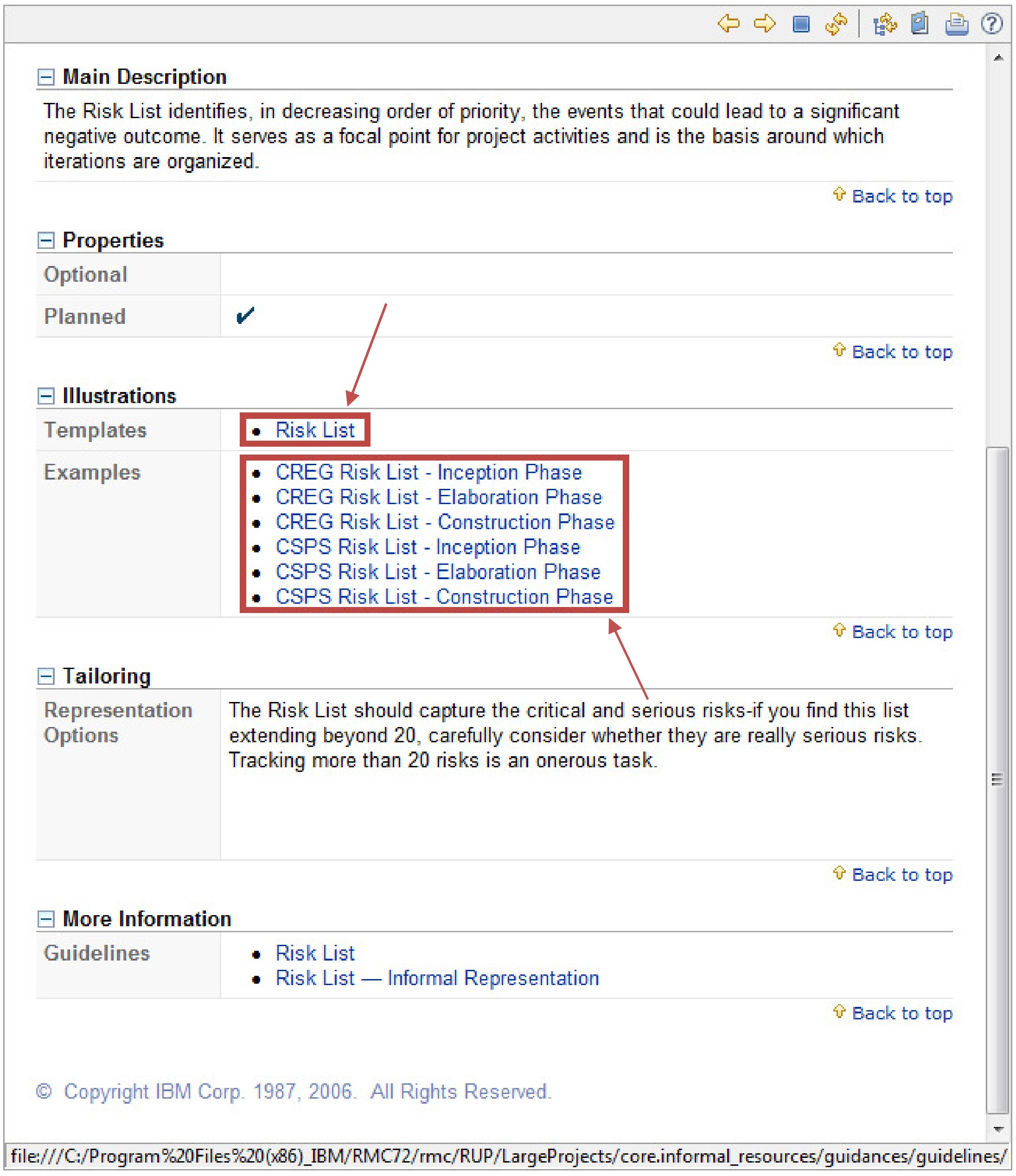
Steps

Expand All Steps Collapse All Steps

- Identify Potential Risks
- Analyze and Prioritize Risks
- Identify Risk Avoidance Strategies
- Identify Risk Mitigation Strategies
- Identify Contingency Strategies
- Revisit Risks During the Iteration
- Revisit Risks at the End of Iteration

file:///C:/Program%20Files%20(x86)_IBM/RMC72/rmc/RUP/LargeProjects/core.base_rup/capabilitypatterns/rup_risk_manageme

Figure 7.3 RMC Process Browser



⊟ Main Description

The Risk List identifies, in decreasing order of priority, the events that could lead to a significant negative outcome. It serves as a focal point for project activities and is the basis around which iterations are organized.

Back to top

⊟ Properties

Optional	
Planned	✓

Back to top

⊟ Illustrations

Templates	• Risk List
Examples	• CREG Risk List - Inception Phase • CREG Risk List - Elaboration Phase • CREG Risk List - Construction Phase • CSPS Risk List - Inception Phase • CSPS Risk List - Elaboration Phase • CSPS Risk List - Construction Phase

Back to top

⊟ Tailoring

Representation Options	The Risk List should capture the critical and serious risks-if you find this list extending beyond 20, carefully consider whether they are really serious risks. Tracking more than 20 risks is an onerous task.

Back to top

⊟ More Information

Guidelines	• Risk List • Risk List — Informal Representation

Back to top

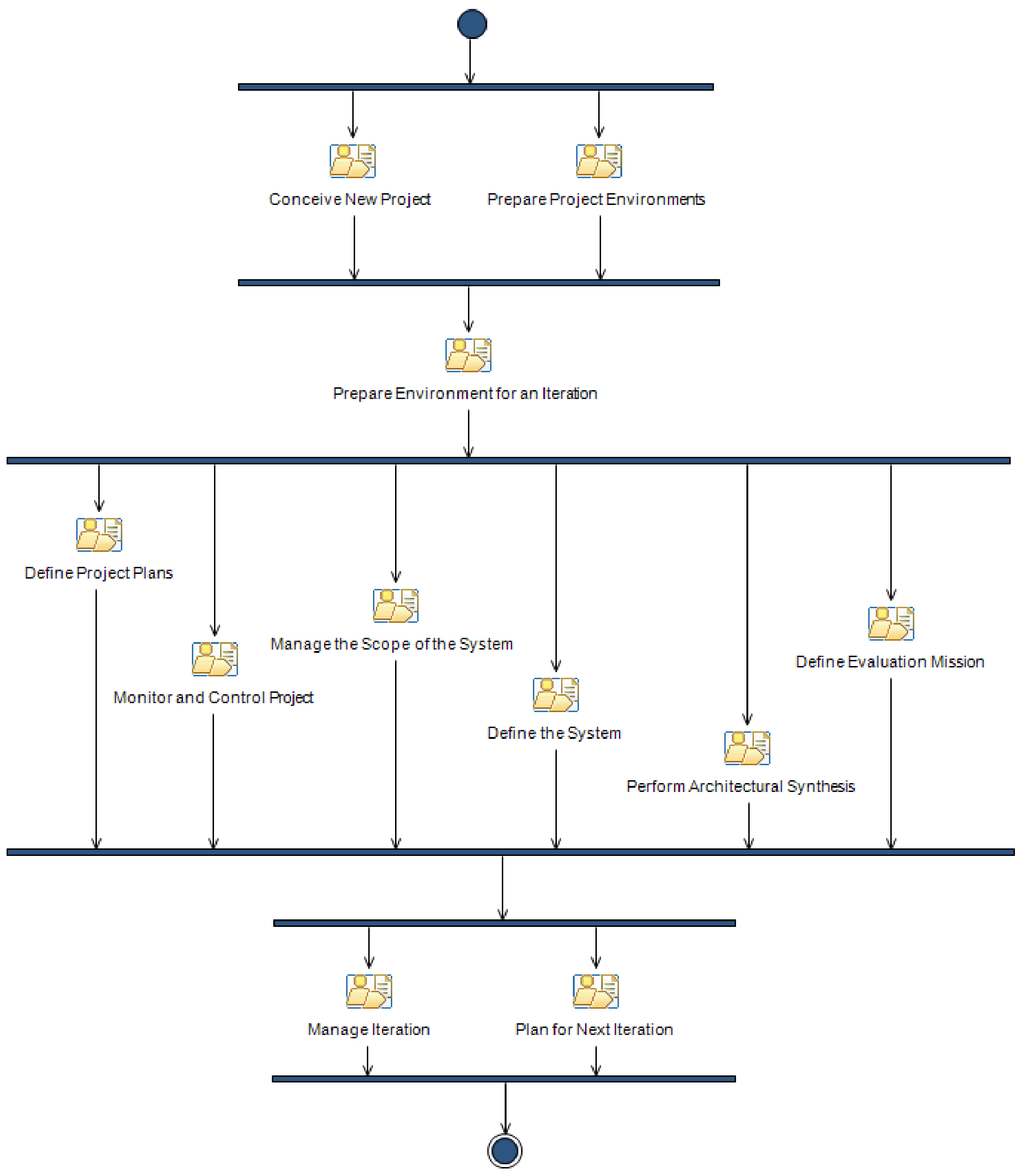
© Copyright IBM Corp. 1987, 2006. All Rights Reserved.

file:///C:/Program%20Files%20(x86)_IBM/RMC72/rmc/RUP/LargeProjects/core.informal_resources/guidances/guidelines/

Figure 7.4 Selected Work Product

7.1 Inception Iteration I1

Inception Iteration I1 is referred as Preliminary Iteration of the Inception Phase. The activities performed in Preliminary Iteration of inception phase are shown in the Figure 7.5. The activity diagram in the Figure 7.5 and all of its activities and tasks can be obtained from RMC and modified to adapt on projects.



Conceive New Project
Prepare Project Environments
Prepare Environment for an Iteration
Define Project Plans
Monitor and Control Project
Manage the Scope of the System
Define the System
Perform Architectural Synthesis
Define Evaluation Mission
Manage Iteration
Plan for Next Iteration

Figure 7.5 Inception Phase Activity Diagram

Conceive New Project activity brings our project from the origin of an idea to a point at which a reasoned decision can be made to continue or abandon the project. It consists of four tasks as shown in the Figure 7.6.

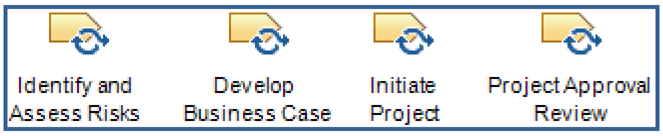


Figure 7.6 Conceive New Project Tasks

The conceive new project activity begins with *Identify and Assess Risks* task which is very important for identifying, analyzing and prioritizing risks at the beginning of the project. Within this task risks should be carefully identified. Rank the risks and group them to avoid long list of similar risks. Identify mitigation plans to reduce the impact of the risks on the project. Revisiting the risks is a critical issue because risks appear dynamically in the project. The task concludes with a Risk List (see Appendix B.I.1) that captures the potential risks of the project. Risks are defined in a decreasing order of priority. Risk List also given below as Table 7.1.

Table 7.1 Risks for Iteration I1

Risk Ranking/ Magnitude	**Risk Description & Impact**	**Mitigation Strategy and/or Contingency Plan**
High	The team is unfamiliar with Web architecture and technology. This risk may impact the ability to deliver a Web application on time.	Train the Team on Web technologies (to be done in the second elaboration iteration, or E2). Allocate time (during E2) for the learning curve and monitor progress weekly in elaboration.
Medium	Volume of users logged on during peak periods (on holidays and special days) may significantly degrade system performance.	Early prototyping and extrapolation of response time data should be done in the elaboration phase.
Low	Incompatibility with internet browsers and specific configurations on client machines.	Address during elaboration (E2).

The second task in Figure 7.6 is *Develop Business Case* and it is used to develop the economic justification for the product and we obtain Business Case (see Appendix B.I.2) that develops the economic plan and gives the economic value of the product. A brief description of the product that is to be developed will be given in this document. Business context is defined which helps stakeholders to understand intended market for the product. Objectives of the product will be given here that provides a support for managing risks. Some constraints specified for the system within this document that will be obeyed from beginning to the end of the project. These constraints could be about standards, technologies or techniques that will be used during the development as depicted in the Figure 7.7.

General design and implementation constraints include:

- Only CU students and staffs can use this system.
- Software system shall be written in .NET 3.5 Technologies
- Student basically can see the courses that has been taken in that semester. But if student use search functions, he/she can see all the courses and can download all materials.
- Instructor basically can add materials only the courses given by him/herself. But if instructor use search functions, he/she can add materials to all courses.
- A material can be deleted only by the owner of material.
- An instructor can only activate materials that are added by him/herself or materials of the courses that are given by him/herself in that semester.
- The documentation shall be in accordance with the IEEE Standards.

Figure 7.7 Constraints

Initiate Project sets up the necessary executive management and project planning teams, and the criteria that will be used for successful project completion. During this task an initial draft of Software Development Plan (see Appendix B.I.3) is produced which defines the process of the project. It gathers all information necessary to manage the project by the managers. By using the information that is gathered in this task project overview is prepared including purpose, scope and objective of the project and documented as shown in Figure 7.8. It also includes the list of possible work products that will be produced during the development.

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

The project will implement an e-Reserve application. The e-Reserve application will be an upload/download document, reserve document application. It will provide the ability to list details of a document and select a quantity of that document to reserve.

A CU library terminal interface will provide interface to enquire and reserve documents.

A Web based interface will allow enquiring and reserving documents using an internet browser.

The e-Reserve application will be a C# application that uses ASP.NET. The e-Reserve application will access a document stored in the main database.

Figure 7.8 Project Phase Plan

Project Approval Review determines whether or not the project is worth investing in and concludes with Review Record (see Appendix B.I.4). This document captures the results of the review activity for the current task in which identifies the possible problems that team members faces and proposes possible solutions to these problems. Key activity to be performed is a meeting in the current task. All decisions about approval of the work products that are produced will be made at this meeting. An example of a problem that arises while performing the task and a possible solution to it is shown in the Figure 7.9.

Problems identified
Use Case Diagrams: The use case for view documents includes books and magazines but not documents.
Recommendations
Use Case Diagrams: The use case for view documents will be changed to view material.

Figure 7.9 Problems and Recommendations

Prepare Project Environments activity of the Inception Phase given in Figure 7.5 is responsible for the preparation of the project environment and composed of two sub-activities as shown in Figure 7.10.

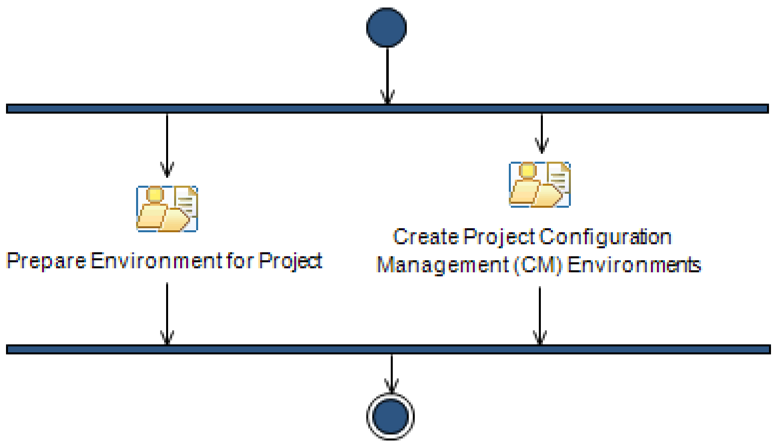

Prepare Environment for Project
Create Project Configuration Management (CM) Environments

Figure 7.10 Prepare Project Environments Sub-Activities

Prepare Environment for Project sub-activity prepares the development environment for the project and has one major task as shown in Figure 7.11.

Figure 7.11 Prepare Environment for Project Task

Tailor the Development Process for the Project task is responsible for customizing the development process. Analyzing the problem fires this task and arise the requirement of a project specific process that concludes by a Development Process. The key work product of this task is a Development Process that describes the process a project has to follow. There are different types of processes that have to be selected for the entirely of the project: Delivery Process or Capability Pattern, which are mentioned in Chapter 4. Development Process is documented formally in Development Case document (see Appendix B.I.5). It is developed to produce guidance to the members of project and adaptations on development process takes place in the Development Case document. At each phase it has to be revised to give the major activities of that phase. The version in inception phase includes the overview of the development process as shown in the Figure 7.12. Development Case document also includes the major phases of inception phase not in details but as a list of the activities to establish a path of work.

This project will consist of a full Inception phase, a two iteration Elaboration phase, a three iteration Construction phase, and a full Transition phase. Design and code reviews will take place at key iteration milestones, and project quality reviews will be conducted at the end of each phase.

Figure 7.12 Overview of the Development Process

Create Project Configuration Management (CM) Environments sub-activity is responsible for the overall product development environment and has one major task as shown in the Figure 7.13.

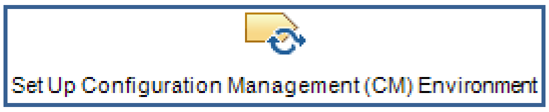


Figure 7.13 Create Project Configuration Management Environments Task

Set Up Configuration Management (CM) Environment task sets up the environment of the project. We begin with setting up a hardware environment which will form our Project Repository. It holds all the documents of the project. It also serves as a guideline to the new members who join to project team and helps to capture main steps of work done. It contains only documentation of the project and may be backups only. Project Repository is documented in Configuration Management Plan (see Appendix B.I.6). This document mainly describes the computing environment and software tools to be used throughout the project. The critical part of Configuration Management Plan is definition of Project Repository and its detailed physical location. All the information about Project Repository is depicted in the Figure 7.14. Also format for the Change Request Form is given in Configuration Management Plan.

2.2 Tools, Environment, and Infrastructure

All environment files will be available at Çankaya University MP Lab. Moreover the configuration manager will save a backup copy on the space allowed to the team in the lab. All environment material files will be available at any time at the following address:

Host: ceng.cankaya.edu.tr/~mp
Port: 21
Login: c0771000
Password: CUeReserve

Figure 7.14 Project Repository

Prepare Environment for an Iteration activity prepares the environment for the iterations in the project. It is examined for the current iteration that is Inception Iteration I1. It consists of one task as shown in the Figure 7.15.

Figure 7.15 Prepare Environment for an Iteration Task

Launch Development Process rollouts the development process. Efforts in this task primarily make the changes public to all members of the project. The selected development process in the previous activities would be a new concept for many team members. So training on newly proposed development process is planned. A Change Request Form is required for this task. In Change Request Form problem that causes a change on the product and suggestion for the solution will be given. Current problem and proposed change is given clearly in this document. Change Request would be prepared if a change to the product is needed. For the inception phase we do not need to prepare it; however it changes from project to project.

Define Project Plans activity shown in Figure 7.5 provides general planning for the release of the project and composed of one sub-activity as shown in the Figure 7.16.

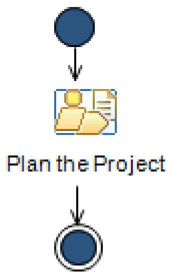


Figure 7.16 Define Project Plans Sub-Activity

Plan the Project sub-activity develops the components and enclosures of the Software Development Plan document. It has three major tasks as shown in the Figure 7.17.

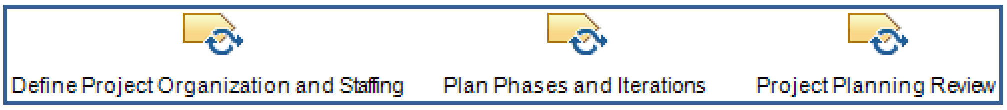

Define Project Organization and Staffing
Plan Phases and Iterations
Project Planning Review

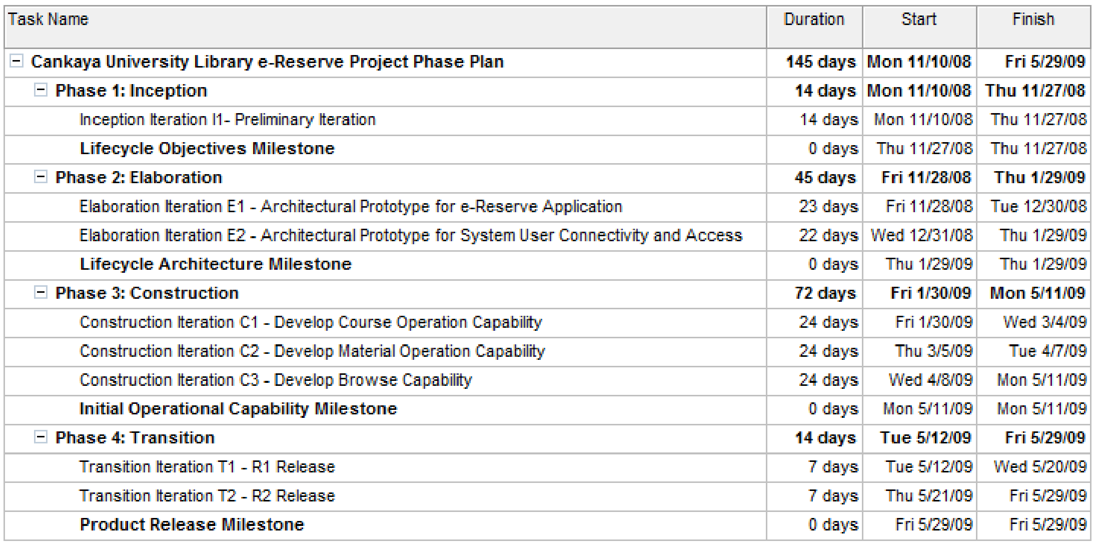
Figure 7.17 Plan the Project Tasks

Define Project Organization and Staffing task defines an organizational structure for the project. Project staff is organized to be managed by the managers. In Software Development Plan organizational structure of the project is defined clearly depending on the characteristics of the project. Key roles are defined especially during the inception phase team should be small. Software Development Plan (see Appendix B.I.7) is revised to capture structuring to manage key roles on the project as depicted in Table 7.2 in addition to Project Overview and Management Process.

Table 7.2 Project Roles

Role	**Inception**	**Elaboration**	**Construction**	**Transition**
Project Manager	Team Member A	Team Member A	Team Member A	Team Member A
System Analyst / Requirement Specifier	Team Member B	Team Member B		
Software Architect	Team Member C	Team Member C	Team Member C	
Designer		Team Member D	Team Member D	
Developer / Integrator		Team Member E Team Member F	Team Member E Team Member F	Team Member E
Tester	Team Member G	Team Member G Team Member H	Team Member G Team Member H	Team Member G
Deployment Manager			Team Member I	Team Member I
Technical Writer			Team Member J	Team Member J
System Administrator	Team Member K	Team Member K	Team Member K	Team Member K

Plan Phases and Iterations describes the project phases and iterations by making estimations. Estimations on the project are made in the current task such as project phase plan is developed. Milestones are defined and their dates are identified if possible. Resources required carrying out the project are defined based on the estimation made in the previous stages of the task. This task is concluded with the revision of Software Development Plan (see Appendix B.I.7). By using the information that is gathered in this task an initial project phase plan is prepared and documented in Software Development Plan as shown in Figure 7.18.



Task Name	Duration	Start	Finish
⊟ Cankaya University Library e-Reserve Project Phase Plan	**145 days**	**Mon 11/10/08**	**Fri 5/29/09**
⊟ Phase 1: Inception	**14 days**	**Mon 11/10/08**	**Thu 11/27/08**
Inception Iteration I1- Preliminary Iteration	14 days	Mon 11/10/08	Thu 11/27/08
Lifecycle Objectives Milestone	0 days	Thu 11/27/08	Thu 11/27/08
⊟ Phase 2: Elaboration	**45 days**	**Fri 11/28/08**	**Thu 1/29/09**
Elaboration Iteration E1 - Architectural Prototype for e-Reserve Application	23 days	Fri 11/28/08	Tue 12/30/08
Elaboration Iteration E2 - Architectural Prototype for System User Connectivity and Access	22 days	Wed 12/31/08	Thu 1/29/09
Lifecycle Architecture Milestone	0 days	Thu 1/29/09	Thu 1/29/09
⊟ Phase 3: Construction	**72 days**	**Fri 1/30/09**	**Mon 5/11/09**
Construction Iteration C1 - Develop Course Operation Capability	24 days	Fri 1/30/09	Wed 3/4/09
Construction Iteration C2 - Develop Material Operation Capability	24 days	Thu 3/5/09	Tue 4/7/09
Construction Iteration C3 - Develop Browse Capability	24 days	Wed 4/8/09	Mon 5/11/09
Initial Operational Capability Milestone	0 days	Mon 5/11/09	Mon 5/11/09
⊟ Phase 4: Transition	**14 days**	**Tue 5/12/09**	**Fri 5/29/09**
Transition Iteration T1 - R1 Release	7 days	Tue 5/12/09	Wed 5/20/09
Transition Iteration T2 - R2 Release	7 days	Thu 5/21/09	Fri 5/29/09
Product Release Milestone	0 days	Fri 5/29/09	Fri 5/29/09

Figure 7.18 Project Phase Plan

Project Planning Review in Figure 7.17 helps describing the review of Software Development Plan. As we mentioned in the previous tasks key activity to be performed is a meeting. All decisions about approval of the work products that are produced will be made at this meeting again. At the end of Define Project Plans sub-activity Review Record is handled again to approve the initial Software Development Plan. If there exists any missing parts or mismatching concepts with the project then Software Development Plan is sent back to the project manager to review it.

Monitor and Control Project activity in Figure 7.5 is used to capture continuing work including monitoring project status, reporting to stakeholders, and dealing with issues. It consists of four major tasks as shown in the Figure 7.19.

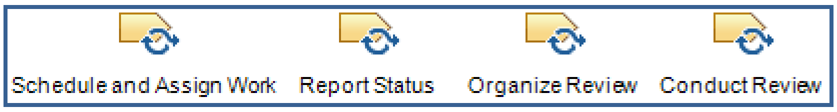


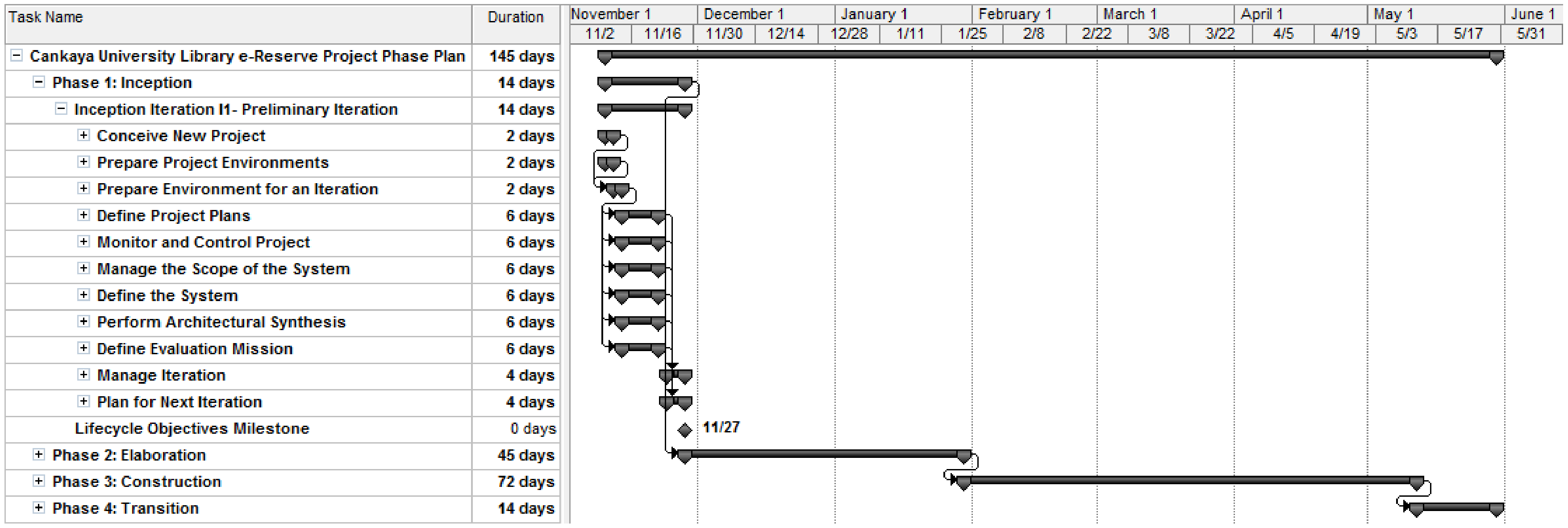
Figure 7.19 Monitor and Control Project Tasks

Schedule and Assign Work task in the activity helps project manager to schedule the work. The critical issue is the state of Change Request. Because we did not need to document a Change Request Form up to this task. So if the Change Request is examined than project manager should fix the iteration. If the Change Request is to be held until a later iteration, the project manager simply re-plans the future iterations. The Iteration Plan (see Appendix B.I.9) is prepared at the beginning of this task that details the work in a fine-grained way. We are now at the early stages of inception phase and the duty of Iteration Plan is to give a fine grained plan of inception phase. It is detailed only as a Gantt Chart for the inception phase as shown in Figure 7.20 and iteration overview of the current phase is reported as in the Table 7.3. Detailed Gantt Chart can be found in Project Phase Plan (see Appendix B.I.8).

Table 7.3 Iteration I1 Overview

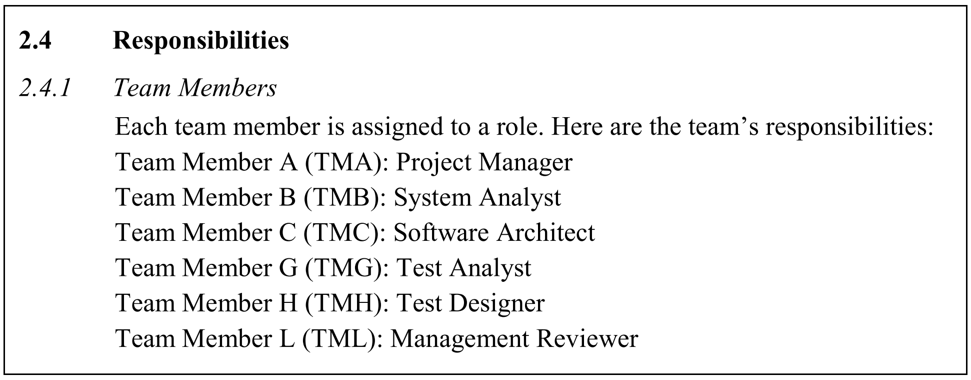
Phase	**Iteration**	**Description**	**Risks Addressed**
Inception Phase	I1 Iteration **Preliminary Iteration**	• **Define and approve Business Case** • **Define high-level product requirements** The Vision document contains key features and constraints. • **Define project scope** A Use Case Diagram includes key Actors and Use Cases. Only a brief description is provided for each Actor and Use Case. • **Plan the overall project and next iteration** A high-level Software Development Plan, a Risk List, and an Iteration Plan for the first elaboration iteration are created. • **Create a very first draft of the Test Plan** • **Define application-specific terminology** Important terms are defined in the Glossary.	Clarifies user requirements up front. Develops realistic Software Development Plans and scope. Determines feasibility of project from a business point of view.

Another work product produced during current task is a Work Order (see Appendix B.I.10) that provides communication between the project manager and the project team. Each team member with a specific role has its own responsibilities which are detailed in Work Order. The roles of team members are identified in the work product as shown in the Figure 7.21. These responsibilities are more detailed in the work order reports and schedules.



Task Name	Duration
Cankaya University Library e-Reserve Project Phase Plan	145 days
Phase 1: Inception	14 days
Inception Iteration I1- Preliminary Iteration	14 days
Conceive New Project	2 days
Prepare Project Environments	2 days
Prepare Environment for an Iteration	2 days
Define Project Plans	6 days
Monitor and Control Project	6 days
Manage the Scope of the System	6 days
Define the System	6 days
Perform Architectural Synthesis	6 days
Define Evaluation Mission	6 days
Manage Iteration	4 days
Plan for Next Iteration	4 days
Lifecycle Objectives Milestone	0 days
Phase 2: Elaboration	45 days
Phase 3: Construction	72 days
Phase 4: Transition	14 days

Figure 7.20 Gantt Chart for Inception Phase



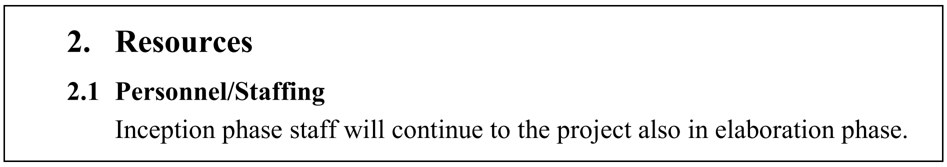
2.4 Responsibilities

2.4.1 Team Members

Each team member is assigned to a role. Here are the team's responsibilities:
Team Member A (TMA): Project Manager
Team Member B (TMB): System Analyst
Team Member C (TMC): Software Architect
Team Member G (TMG): Test Analyst
Team Member H (TMH): Test Designer
Team Member L (TML): Management Reviewer

Figure 7.21 Responsibilities for Iteration I1

Report Status task in Figure 7.19 describes when and how the periodic updates on the project will be done. An important work product in this task is the Status Assessment (see Appendix B.I.11) that has the responsibility to ensure the expectations of all parties are consistent. A lot of the information in the Status Assessment is copied from other sources to provide a comprehensive source of information for the people assessing the project. For example, within the Status Assessment document risks are examined that are already listed in the Risk List. Status of the team members are also given in the Status Assessment as shown in the Figure 7.22.



2. Resources

2.1 Personnel/Staffing

Inception phase staff will continue to the project also in elaboration phase.

Figure 7.22 Status of Team Members

The next task in Figure 7.19, *Organize Review*, describes how to facilitate the review process and ensure the review is undertaken appropriately. Within this task review coordinator has to ensure that required review tasks are appropriately planned and organized. There are various approaches to planning review tasks which are based on factors such as team size, team culture and so on. Organize Review task is concluded with revision of Review Records.

The last task in Figure 7.19, *Conduct Review* and it describes how to facilitate a review so as to maximize the productivity of the reviewers and meet defined quality requirements. This task is under the control of reviewer of the project. Conduct Review task is concluded with revision of Review Records.

Manage the Scope of the System activity in Figure 7.5 is used to ensure that requirements are obtained clearly for that iteration. It consists of two major tasks as shown in the Figure 7.23.

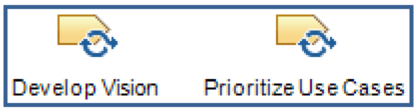


Figure 7.23 Manage the Scope of the System Tasks

Develop Vision task develops a vision for the system especially including the stakeholders, systems key features and constraints that conclude with a Vision document (see Appendix B.I.12). By the help of Vision document the problem being solved simply by asking the problem. Vision document provides to understand stakeholders view of the product to be developed. It is closely related with the work product Business Case. Vision document provides a statement that is summarizing the problem being solved by this project as shown in the Table 7.4.

Table 7.4 Problem Statement

The problem of	A slow and less efficient manual reserving process is currently in use; also students have to come into the library to realize reserve operation.
Affects	Reserve process efficiency, thereby affecting librarian productivity and student satisfaction.
the impact of which is	Negative librarian productivity and student satisfaction, which in turn impacts on CU library efficiency.
A successful solution would be	To improve the inquiry and reserving process.

Stakeholders are identified clearly. It also provides an overall statement that summarizes at the highest level, the unique position product intends to fill the marketplace depicted in the Table 7.5.

Table 7.5 Product Position Statement

For	Librarian, instructor and student
Who	Enquire and reserve books/documents
The e-Reserve application	Is an online reserving system
That	Will enable a real-time online reserving facility
Unlike	The current manual reservation process
Our product	Will vastly improve the reservation process, thereby creating student satisfaction and improving CU library efficiency.

Based on the benefits listed in problem statement, a list of features is developed that we want in the system. The vision document is written from the customers perspective and provides the contract between the funding authority and the development organization.

Prioritize Use Cases task in Figure 7.23 identifies the significant use cases. Use cases are prioritized and significant use cases are listed. They are identified within a Software Architecture document (see Appendix B.I.13). Inception phase version of this document contains only a list of use cases as shown in the Figure 7.24. Use cases in bold are significant to the architecture.

- **Login**
- Logout
- **Insert User**
- Update User
- Delete User
- **Activate Course**
- **Update Course**
- **Deactivate Course**
- Search Corse
- **Insert Material**
- Update Material
- Delete Material
- Search Material
- View Material
- **Download Material**

Figure 7.24 Use Case List

Software Architecture document contains different architectural views. Deployment view of the system is also given in this work product where the system begins to arise step by step however this view is detailed in the next tasks now it is only conceptually formed.

Another important work product of the task is Software Requirement that contains specifications for a condition to which a system must conform. Software Requirements are documented in Software Requirements Specifications (see Appendix B.I.14) document in which provides a complete definition of the software requirements; both functional and non-functional. It includes general factors that affect the product and its requirements and provides a background for those requirements. As an example product functions for the project is depicted in the Figure 7.25.

2. Product Functions

The functionality supported by e-Reserve project can be described in a number of major functional areas.

Students can login to this page and display their courses. Then student can download books and document about his/her courses. Also student can search for another courses and books.

Basically, an instructor can do all the tasks that a student can do. Also an instructor can add, update and delete book/document. Instructor can activate/deactivate course accessibility and update a course.

Librarian can access all the courses categorized by course ids. Can perform all operations on all courses. (Insert/update/delete) Can make bulk activation or deactivation operation. Can view usage reports.

Figure 7.25 Product Functions

Referring to Figure 7.5, **Define the System** activity is responsible on sketching key requirements. It consists of four major tasks as shown in the Figure 7.26.

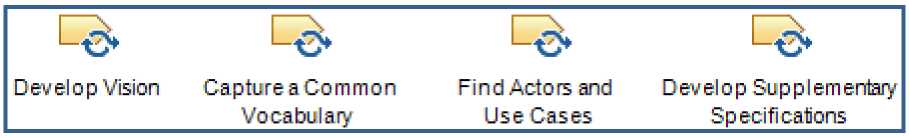

Develop Vision
Capture a Common Vocabulary
Find Actors and Use Cases
Develop Supplementary Specifications

Figure 7.26 Define the System Tasks

Develop Vision task has the same functionality as in the Manage the Scope of the System activity. Vision documented is developed during the previous activity and its development continuous in current activity. Its final changes made on this task if needed and the Vision document (see Appendix B.I.15) is completed at the end of task. Revision of Vision document is completed.

Capture a Common Vocabulary task determines how project specific terms are organized and defined during the development process. A common vocabulary, using the most common terms in the problem domain, is constructed. All team members should use these terms in order to define problem domain. Using a common vocabulary between the team members improve the efficiency and provide increased understanding of the concepts. It is concluded by developing a Glossary (see Appendix B.I.16) that all terms are identified clearly as depicted in the Figure 7.27.

2.1 ERS
e-Reserve system

2.2 ERIS
e-Reserve Information System

2.3 HTML
Hyper Text Markup Language is a text base programming language using many symbols and codes interpreted statically by a web browser.

2.4 IM
Instructor Module

2.5 LM
Librarian Module

2.6 RUP
IBM Rational Unified Process is a software engineering process and a process framework for successful iterative-incremental software development.

2.7 RMC
IBM Rational Method Composer is a commercial product for authoring, configuring, viewing, and publishing processes.

2.8 RSM
IBM Rational Software Modeler is a robust collaborative platform for visual modeling and design.

2.9 SM
Student Module

2.10 SRM
Server Module

Figure 7.27 Definitions for Iteration I1

Find Actors and Use Cases task in Figure 7.26 identifies the actors and use cases for the system that supports the requirements. During the evaluation of task it produces the Use Case Model (see Appendix B.I.17) that documents the systems functionality. It helps to communicate with the stakeholder in a manner of standardized notations and diagrams. First of all actors of the system are identified. Actors that we found are briefly described to understand who or what interacts with the system. All actors are identified and briefly explained as shown in the Figure 7.28. Explaining the roles of each actor provides a better understanding of the system while reviewing the Use Case Model. After we have found the actors, the system's use cases should be found. As we did in the actors, we name and briefly describe the use cases of the system. There are many informal ways to find use cases however most effective approach is to ask that what the actor requires of the system. Most of the time it will be difficult to find the suitable actors and use cases for the system however working with use case gives a better understanding of the system.

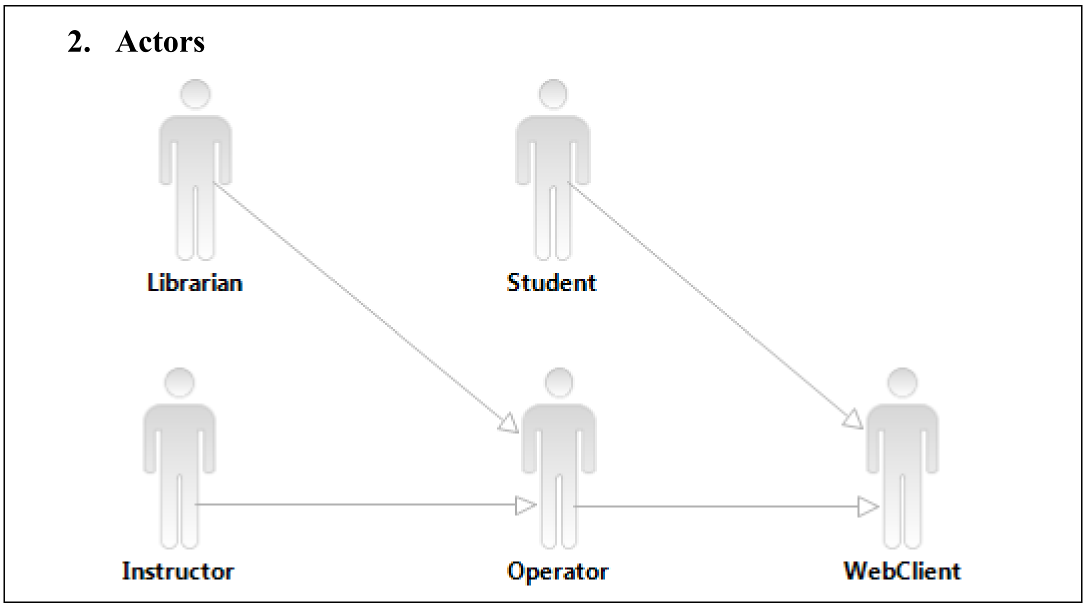


Figure 7.28 Actors of the System

These actors and uses cases are represented in the use case diagrams by showing the relationships between them that how they interact with each other. A use case diagram is depicted in the Figure 7.29.

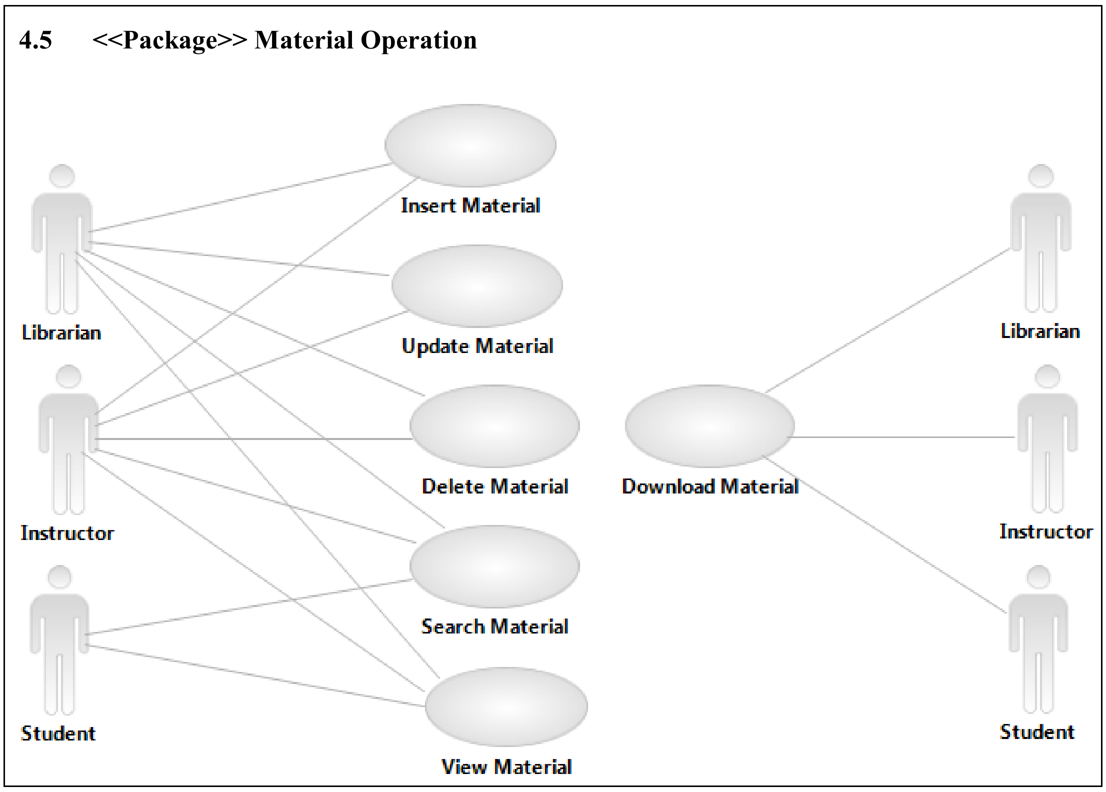


Figure 7.29 Use Case Diagram

If the number of actors and use cases becomes too large then they could be divided into use case packages. The Figure 7.29 also shows how the use case diagrams are packaged. This approach simplifies the view of more complex systems.

The last task of Figure 7.26 is *Develop Supplementary Specification* task. It helps to capture requirements that are not readily captured in use cases. Many functional requirements can be documented in Use Case Model however some cannot. Task is performed and concluded with Supplementary Specification (see Appendix B.I.18) that captures the system requirements which are not applicable to specific use cases. An example of requirements that affect supportability is depicted in the Figure 7.30. Task is performed especially for capturing system qualities and constraints. Supplementary Specification is an important complement to Use Case Model in which they are used to capture all software requirements.

3. Usability

3.1 Multichannel Access
The system will be accessed via both a web browser and CU library terminals.

3.2 Ease of Use
The system will not require user training beyond that of using a web browser.

3.3 Browser Compatibility
The web client application can run under Mozilla Firefox, Microsoft Internet Explorer or Opera.

3.4 Online Help
The web client application provides an online help for the user.

Figure 7.30 Usability Requirements

On Figure 7.5, **Perform Architectural Synthesis** is an activity showing the system is feasible and demonstrates it. It consists of three major tasks as shown in the Figure 7.31.

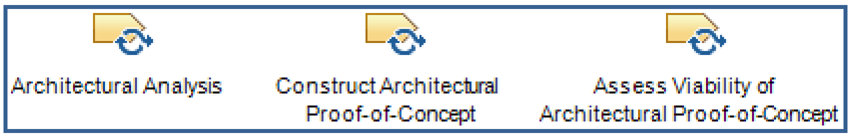

Architectural Analysis
Construct Architectural Proof-of-Concept
Assess Viability of Architectural Proof-of-Concept

Figure 7.31 Perform Architectural Synthesis Tasks

Architectural Analysis task tries to define architecture of the system based on similar systems or similar problem domains. An architecture overview is developed at the early stages of the project. This architecture provides an early understanding of the high-level structure of the intended system to the stakeholders. In inception phase we only deal with the Deployment Model (see Appendix B.I.19) of the system that shows the nodes, devices, and connections between them. Deployment Model represents a high-level overview of the system. We acquire an understanding of the geographical distribution and operational complexity of the system. Deployment Model is documented in the Software Architecture Document (see Appendix B.I.20) that is depicted in the Figure 7.32.

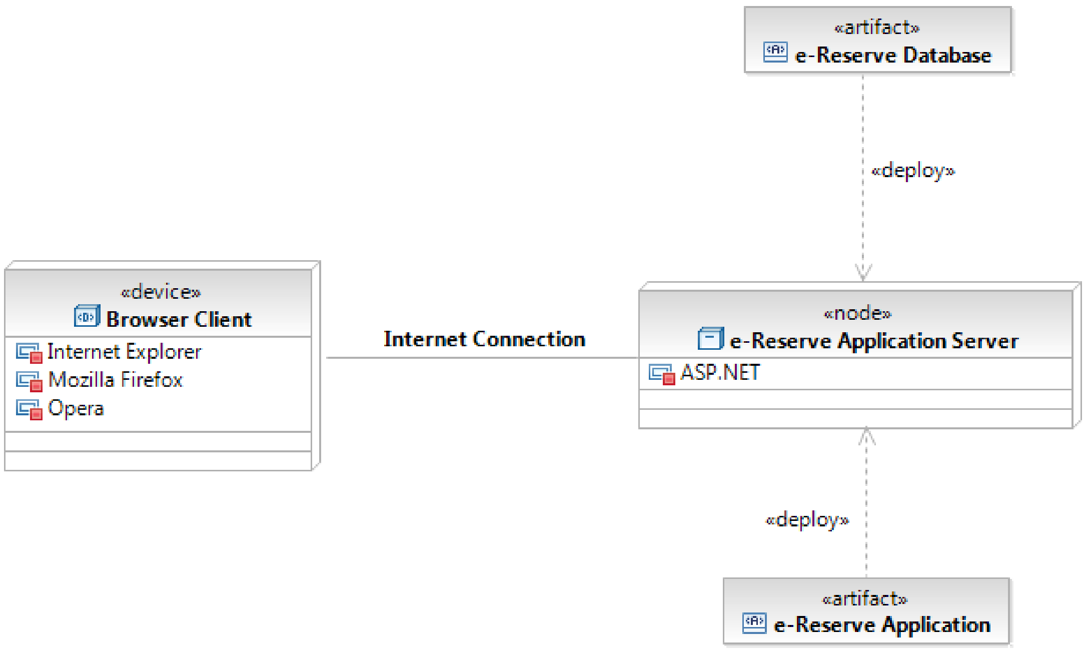


Figure 7.32 Deployment Model

Referring to Figure 7.31, *Construct Architectural Proof-of-Concept* task is the critical point of the management that defines how to develop an Architectural Proof-of-Concept for the system. Architectural Proof-of-Concept can be documented as a list of known technologies which seem appropriate to the solution, a sketch of a conceptual model of a solution using a notation such as UML, or an executable prototype. Using the technique that is listed above Architectural Proof-of-Concept is constructed. In this case our Deployment model is defined as the Architectural Proof-of-Concept for the system that shows a solution for the problem domain.

Assess Viability of Architectural Proof-of-Concept task is required for evaluating defined Architectural Proof-of-Concept. The key work product for the task is

Reference Architecture (see Appendix B.I.21) whose purpose is to form a starting point for the architectural development. Reference Architecture should be defined through different viewpoints and these views map to the 4+1 Views of software architecture. From that 4+1 Views, logical view has four functional layers which are defined in the Reference Architecture document. These layers are; interface layer, business layer, middleware layer, and system software layer. Table 7.6 depicts the middleware layer defined in the document.

Table 7.6 Middleware Layer

Area	Products/Services/Components
Application Servers	Microsoft COM+ BEA Weblogic V7
Messaging Services	Microsoft MSMQ/MQ Series Interface
Directory Services	Active Directory/LDAP
Data Distribution Strategies	No distributed two-phase commit processing will be supported. Centralized data access is encouraged at all times.
Data Access APIs	.NET Framework ADO.NET

On Figure 7.5, **Define Evaluation Mission** activity identifies the test efforts to be taken for the iteration**.** It consists of one major task as shown in the Figure 7.33.

Figure 7.33 Define Evaluation Mission Tasks

Define Test Approach task focuses on test strategies for the desired testing. Effect of the software architecture is considered for the test approach by gaining information from Software Architecture Document. After completing all test approaches, existing test techniques are identified to improve test approaches. In the case of insufficient existing test techniques, new test techniques could be identified briefly. For the inception phase this task is performed shallowly. A Test Strategy document (see Appendix B.I.22) is created at the end of task that defines the strategic plan for how the test effort will be conducted against one or more aspects of the target system. For the inception phase only test strategies are given as a list that will be performed at next phases as shown in the Figure 7.34.

2. Test Strategy

- Function Testing
- User Interface Testing
- Data and Database Integrity Testing
- Performance Profiling
- Load Testing
- Stress Testing
- Volume Testing
- Security and Access Control Testing
- Failover/Recovery Testing
- Configuration Testing
- Installation Testing

Figure 7.34 Test Strategies

On Figure 7.5, **Manage Iteration** activity contains the activities that begin, end and review the iteration. It consists of five major tasks as shown in the Figure 7.35.

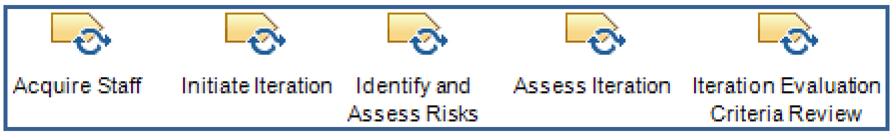

Acquire Staff
Initiate Iteration
Identify and Assess Risks
Assess Iteration
Iteration Evaluation Criteria Review

Figure 7.35 Manage Iteration Tasks

Acquire Staff task is used to organize members into teams. Teams are formed with required skills for the problem domain. Project manager assigns each role to a specific member in the team. This action was already documented in Software Development Plan (see Appendix B.I.7) as in Table 7.2. It is now summarized in Table 7.7. In some cases team members will need a training to develop skills in the project.

Table 7.7 Team Member Roles

Role	Resource
Project Manager	Team Member A
System Analyst / Requirement Specifier	Team Member B
Software Architect	Team Member C
Tester	Team Member G
System Administrator	Team Member K

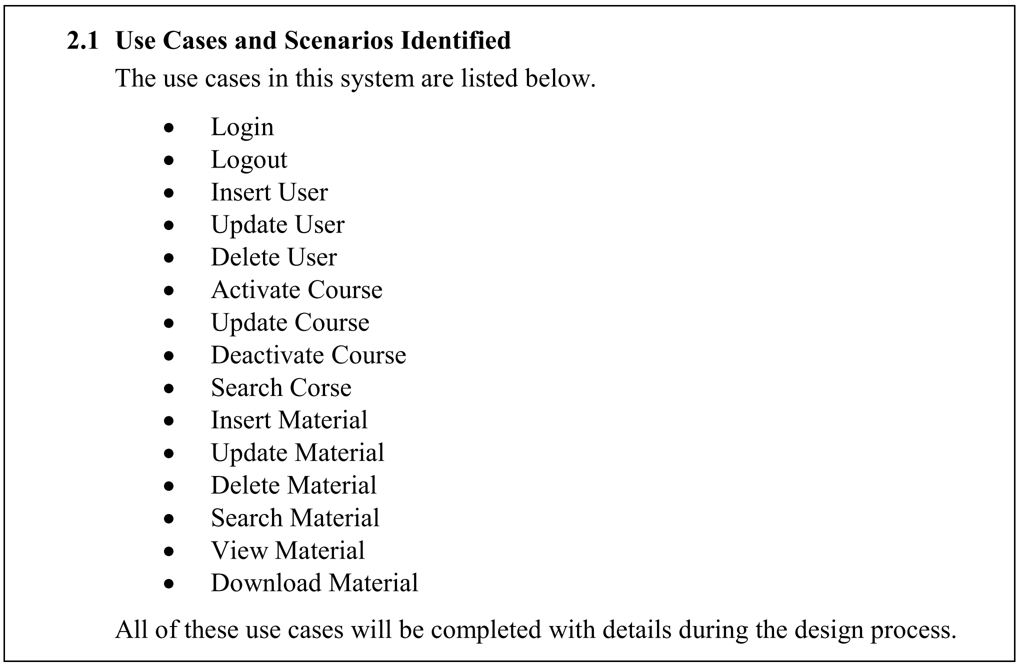
Initiate Iteration task allocates team members to the activities identified for the current iteration. In the previous tasks members are assigned into specific roles. Now activities are assigned to team members that they have to perform. For that operation Work Order (see Appendix B.I.23) is revised and work order reports are prepared for each team member. As an example of the description of the works is given in these reports as shown in the Figure 7.36.

Work Order for e-Reserve		**Created on: <11/14/08>**
Identification		
Work Order ID: <TMA-I1-W02-01>	**WBS ID: <1.10.1>**	
Responsibility (Holder)		
Team Member A		
Associated Change Reports		
None		
Schedule		
Start: < 11/24/08>	**Completion:** < 11/24/08>	**Critical Path:** <NA>
Efforts and Other Resources		
Staff Hours: 1 Hour	**Other Resources:** None	
Description		
Work Description: Acquire Staff		
Expected Output(s): Revise Software Development Plan		
Signature Agreement		
Project Manager: Team Member A **Signed on:** <dd/mm/yy>	**Work Order Holder:** Team Member A **Signed on:** <dd/mm/yy>	

Figure 7.36 Work Order Reports for Iteration I1

Identify and Assess Risks task is primarily performed in the beginning of the project at Conceive New Project activity. As we mentioned before, risks are dynamic elements that could arise at every stage of the project. We will reduce the effects of risks by updating the Risk List to reflect the current project status periodically. So risks could be avoided before they happen or handled rapidly as they appear with a minimum effect on the project.

Next task, *Assess Iteration* evaluates the results of an iteration assess related project information. Main purpose of this task is to determine success or failure of the iteration. Within this task we compare the actual and expected results of the iteration. Also we have to ensure that evaluation criteria for the current iteration are realistic. Based on the results of the assessment Change Requests for any work product could be generated. An Iteration Assessment (see Appendix B.I.24) is created for evaluation of this task that captures the results of the iteration. Objectives that are reached within the iteration are given in Iteration Assessment. An example for the use cases are shown as in the Figure 7.37.



2.1 Use Cases and Scenarios Identified

The use cases in this system are listed below.

- Login
- Logout
- Insert User
- Update User
- Delete User
- Activate Course
- Update Course
- Deactivate Course
- Search Corse
- Insert Material
- Update Material
- Delete Material
- Search Material
- View Material
- Download Material

All of these use cases will be completed with details during the design process.

Figure 7.37 Objectives Reached for Iteration I1

The last task of Figure 7.35 is *Iteration Evaluation Criteria Review* task. It determines how to approve the criteria if the iteration is completed with its meeting objectives. As in the previous review tasks a meeting is planned and all materials about the activity are distributed across related team members to perform review. At the end of the meeting a decision is made to approve or reject the criteria. If the criteria are approved then the next activity will be performed otherwise the project team should address the identified deficiencies and re-submit revised iteration evaluation criteria for a follow-up review. Review Record is completed at the end of meeting that captures the results of the current review activity.

Plan for Next Iteration activity is the last activity of Inception Phase given in Figure 7.5. It guides project team to the next iteration. It consists of two major tasks as shown in the Figure 7.38.

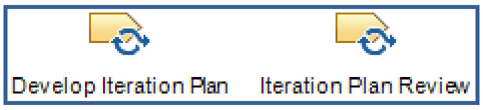


Figure 7.38 Plan for Next Iteration Tasks

Develop Iteration Plan task composes an iteration plan. At the early stages of inception phase we created an Iteration Plan (see Appendix B.I.9) for the Inception Phase Iteration I1 which focuses on proving the concept of the product. Now we are creating an Iteration Plan for the next iteration which is Elaboration Phase Iteration E1 (see Appendix B.I.26). The scope of the next iteration is determined. Again we define the iteration evaluation criteria for Iteration E1 as we did for the Iteration I1. In elaboration phase we will focus on creating a stable architecture. With this conscious we have to select a set of tasks to be performed within Iteration E1. Typically some use cases will be fully developed in Iteration E1 so they are documented in an Iteration Plan that is specially created for Elaboration Phase Iteration E1. The iteration overview for this iteration is depicted in the Table 7.8. Detailed Gantt Chart can be found in Project Phase Plan (see Appendix B.I.25).

Table 7.8 Iteration E1 Overview

Phase	**Iteration**	**Description**	**Risks Addressed**
Elaboration Phase	E1 Iteration **Architectural Prototype for e-Reserve Application**	• **Complete analysis & design for high risk requirements** - Create Use Case Specification for each of the *Login, Logout* use cases, derive an Analysis Model, and refine it into a Design Model. - Document the architecture (high-level design) in the Software Architecture Document. • **Develops the architectural prototype for e-Reserve application** - Code the part of the application implementing the *Login, Logout* use cases. • **Demonstrate feasibility and performance through testing**	Architectural issues related to ERIS clarified. Technical risks related to ERIS mitigated. Early prototype for user review. Performance risks related to high volume of requests mitigated on the ERIS side.

Iteration Plan Review task determines to approve the proposed work plan for the current iteration or not. It is held after the current iteration has been developed. For this review operation again a meeting is planned and all materials about the activity are distributed across related team members to perform review. There is a consideration with this task is that at the end of the review we determine to begin next iteration or not. So the Review Record for the current task has to be created carefully.

7.2 Lifecycle Objectives Milestone

Lifecycle Objectives Milestone given in Figure 7.1, marks the end of the inception phase. It is the first major milestone of the project that is reached at the end of Inception Phase. Now we are standing on a critical region that we have to decide either to proceed with the project or cancel it. At this point a tentative architecture should be established which we will develop further during the next phase.

Evaluation criteria for the Inception Phase can be listed as follows:

- Stakeholder concurrence on scope definition and cost and schedule estimates.
- Agreement that requirements are understood.
- All risks and mitigation strategies have been identified.

The project may be aborted or considerably re-thought if it fails to reach this milestone.

Inception Phase Iteration I1 work products are tabulated in Table 7.9. All the work products are given in the Appendix B of the thesis as B.I.1-26, also on CD to be reached by:

- ~/Appendices/AppendixB/InceptionIterationI1

Table 7.9 Inception Iteration I1 Work Products (APPENDIX B)

eReserve_RiskList_1.0
eReserve_BusinessCase_1.0
eReserve_SoftwareDevelopmentPlan_1.0
eReserve_ReviewRecord_11_11_08_1.0
eReserve_DevelopmentCase_1.0
eReserve_ConfigurationManagementPlan_1.0
eReserve_SoftwareDevelopmentPlan_1.1
eReserve_ProjectPhasePlan_1.0
eReserve_IterationPlanI1_1.0
eReserve_WorkOrder_1.0
eReserve_StatusAssessment_1.0
eReserve_Vision_1.0
eReserve_SoftwareArchitectureDocument_1.0
eReserve_SoftwareRequirementsSpecifications_1.0
eReserve_Vision_1.1
eReserve_Glossary_1.0
eReserve_UseCaseModel_1.0
eReserve_SupplementarySpecification_1.0
eReserve_DeploymentModel_1.0
eReserve_SoftwareArchitectureDocument_1.1
eReserve_ReferenceArchitecture_1.0
eReserve_TestStrategy_1.0
eReserve_WorkOrder_1.1
eReserve_IterationAssessment_1.0
eReserve_ProjectPhasePlan_1.1
eReserve_IterationPlanE1_1.0

CHAPTER 8

ELABORATION PHASE

We had already completed the Inception Phase of our project and ready for the next phase namely Elaboration Phase. The next point in delivery processes is the Elaboration Phase that baselines the architecture of the system. Elaboration Phase of our project consists of two iterations in which each of them includes several activities and concluded by a milestone as shown in the Figure 8.1.

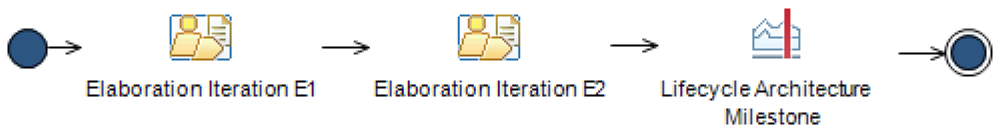


Figure 8.1 Elaboration Phase

As we mentioned in Chapter 7, RMC provides templates and examples of various types of work products. Again, the guide for work products can be reached in the same way as explained in Chapter 7.

8.1 Elaboration Iteration E1

Elaboration Iteration E1 forms the basis for developing an architectural prototype for e-Reserve application in our project. The activities performed in Elaboration Iteration E1 of the elaboration phase are shown in the Figure 8.2. The activity diagram in the Figure 8.2 and all of its activities and tasks can be obtained from RMC and modified to adapt on projects.

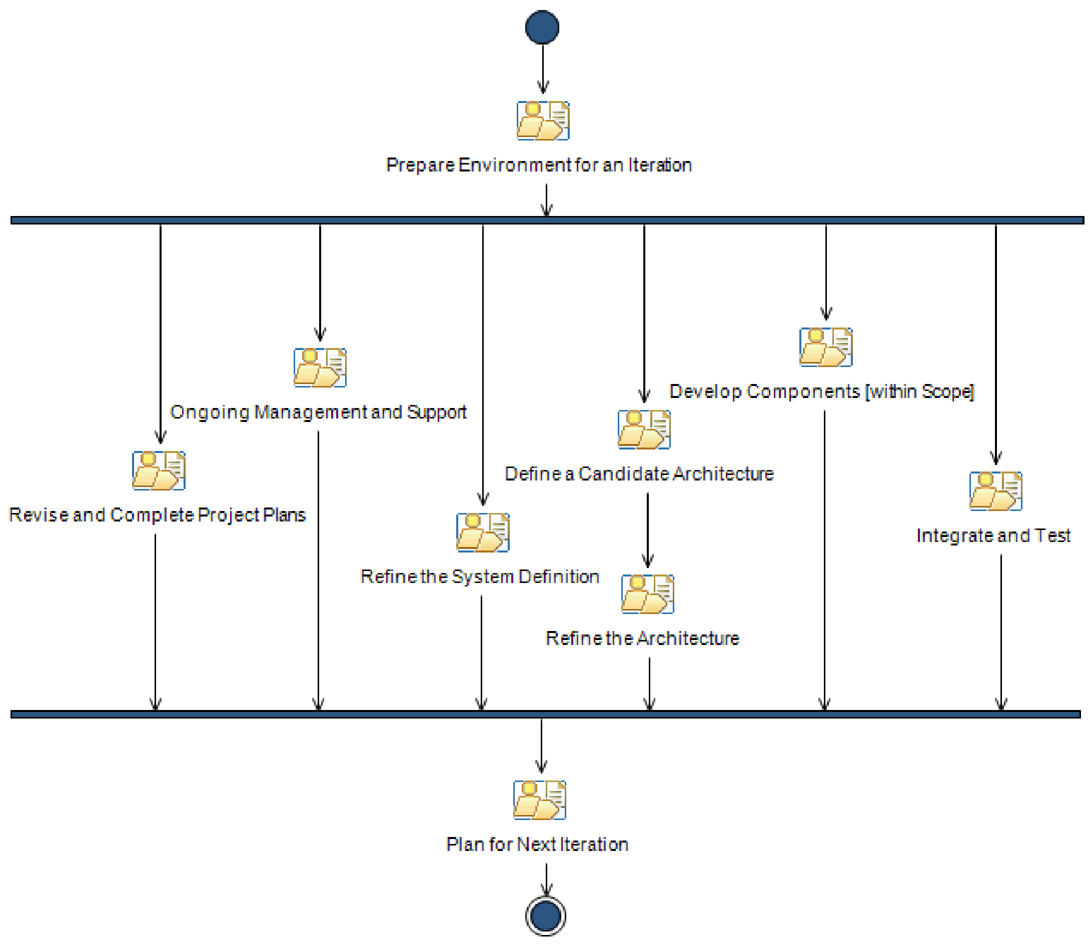

Prepare Environment for an Iteration
Ongoing Management and Support
Revise and Complete Project Plans
Refine the System Definition
Define a Candidate Architecture
Develop Components [within Scope]
Refine the Architecture
Integrate and Test
Plan for Next Iteration

Figure 8.2 Elaboration Phase Activity Diagram

Prepare Environment for an Iteration activity prepares the development environment for the project. It is examined for the current iteration that is Elaboration Iteration E1. It consists of one task as shown in the Figure 8.3.

Figure 8.3 Prepare Environment for an Iteration Task

Launch Development Process task is initially performed in the previous phase which is the Inception Phase. This task is performed again at the beginning of the first iteration of the Elaboration Phase to make necessary changes on the project, if needed, before further development in the project. It is important to make necessary changes in the early stages of development that reduces cost and time by defining the malfunctioning or missing parts. If a change is made then the related team members are trained by a short seminar. If training on proposed development process is planned in the Inception Phase then trainings will begin. Changes are written formally on Change Request Forms that briefly describe the problem and possible solution to the problem. This document simply formulates the changes within a format as shown in Figure 8.4. Currently we do not need any change on the project for now.

Change Request for <Project Name>		**Created on: <dd/mm/yy>**
Identification		
Title:	**Priority:**	**Status:**
	Submitted on:	
	Change Request ID: <>	
Submitter:	**Type: <>**	
Current Problem		
Description:	**Critical Failure:**	
	Nuisance:	
	Source of the Problem:	
Enhancement/New Requirement/Other:		
Observation conditions:		
Proposed Change (Submitter)		
Description:		
Proposed Change (Review Team)		
Approval:	**Reviewed Description:**	
Affected Configuration Items	**Category**	**Enhancement/New Requirements/Other**
Resolution		
Estimated effort (staff hours):		
Change Review Team Disposition		
Changes approved and accepted on:		**By:**
Changes implemented on:		**By:**

Figure 8.4 Change Request Form

A change request form is a document containing a call for an adjustment of a system and it is of great importance in the change management process.This template format is used for enhancements, new requirements or other requests that will be reported during the development of project.

Revise and Complete Project Plans activity in Figure 8.2 gives a general planning for the release and it consists of two sub-activities as shown in the Figure 8.5.

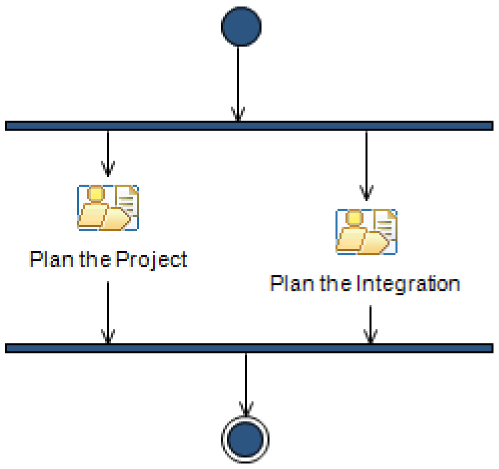


Figure 8.5 Revise and Complete Project Plans Sub-Activities

Plan the Project sub-activity is performed primarily within the Inception Iteration I1 of the Inception Phase that is performed for developing initial plans. This time we are going to complete these plans. We have two iterations in Elaboration Phase so we will complete this task in the second iteration of Elaboration Phase. Now plans are revised for only the current iteration that is Elaboration Iteration E1. This sub-activity has three major tasks as shown in the Figure 8.6.

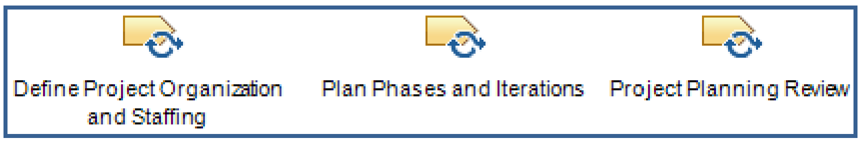

Define Project Organization and Staffing
Plan Phases and Iterations
Project Planning Review

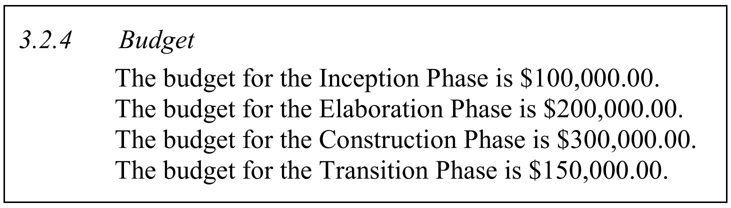
Figure 8.6 Plan the Project Tasks

Define Project Organization and Staffing task has the same responsibility as performed in the Inception Iteration I1 of the inception phase. In this task again project team members and their roles are defined. However during Elaboration Phase, the focus is primarily on the architecture and the architectural prototype. So, most of the effort comes from your architecture team and a designated prototyping team. Because of this necessity team members have to be chosen carefully. The size of the project team will vary across phases, and the Software Development Plan will be updated to reflect these changes. The Software Development Plan (see Appendix C.I.1) is revised to describe the responsibilities of defined roles in the project as shown in the Table 8.1.

Table 8.1 Roles

Role	**Description**
Project Manager	Allocates resources, shapes priorities, coordinates interactions with the customers and users and generally tries to keep the project team focused on the right goal. The project manager establishes a set of practices to ensure the integrity and quality of project artifacts.
System Analyst / Requirement Specifier	Leads and coordinates requirements elicitation and use-case modeling by outlining the system's functionality and delimiting the system. Details the specification of a part of the system's functionality by describing the Requirements aspect of one or several use cases and other supporting software requirements. The requirements specifier may also be responsible for a use-case package, and maintains the integrity of that package.
Software Architect	Leads and coordinates technical activities and artifacts throughout the project. The architect establishes the overall structure for each architectural view: the decomposition of the view, the grouping of elements and the interfaces between these major groupings.
Designer	Defines the responsibilities, operations, attributes, and relationships of one or several classes, and determines how they will be adjusted to the implementation environment. In addition, the designer role may have responsibility for one or more design packages, or design subsystems, including any classes owned by the packages or subsystems.
Developer / Integrator	Responsible for developing and testing components, in accordance with the project's adopted standards. Additionally, the Developer / Integrator integrates components into the system.
Tester	Responsible for the core activities of the test effort, which involves conducting the necessary tests and logging the outcomes of that testing.
Deployment Manager	Provides the overall Configuration Management (CM) infrastructure and environment to the product development team.
Technical Writer	Responsible for writing a meeting minutes document after each team-wide meeting and making it available to all team members.
System Administrator	Responsible for maintaining the project web site, which contains project news, general project information and project documentation.

Plan Phases and Iterations task has the same responsibility as performed in the Inception Iteration I1 of the Inception Phase. We know that this task has an importance for the project that the most common estimations about the project are made here. From the previous iteration we obtained an initial overview of the overall duration of the project. By the help of this view we complete the estimations about the budget of the project within this task. This task is concluded with the revision of the Software Development Plan, and estimations about budget of the project as shown in the Figure 8.7.



3.2.4 Budget

The budget for the Inception Phase is $100,000.00.
The budget for the Elaboration Phase is $200,000.00.
The budget for the Construction Phase is $300,000.00.
The budget for the Transition Phase is $150,000.00.

Figure 8.7 Budget

Project Planning Review task of Figure 8.6 helps describing the review of Software Development Plan. As we mentioned in the tasks of the previous iteration, key activity to be performed is a meeting. All decisions about approval of the work products that are produced will be made at this meeting again. At the end of Plan the Project sub-activity Review Record is handled again to approve the revised Software Development Plan. If there exists any missing parts or mismatching concepts with the project then Software Development Plan is sent back to the project manager to review it.

Plan the Integration sub-activity in Figure 8.5 plans the integration of the system and composed of one task as shown in the Figure 8.8.

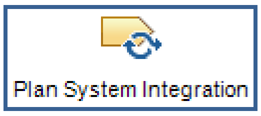


Figure 8.8 Plan the Integration Task

Plan System Integration task identifies the plan of integration. The first step of the task is defining the subsystems, if available. In complex systems we have to define build sets to manage the complexity. The purpose of defining these build sets is to make it easier to do the integration planning. The task is concluded with an Integration Build Plan (see Appendix C.I.2) that defines the details of the integration for the current iteration. Build for the project is given in the Integration Build Plan as shown in the Table 8.2.

Table 8.2 Build Set

Build 1	This build will include the use cases Login, Logout.
Build 2	This build will add the use cases Insert User, Update User, Delete User.

Ongoing Management and Support on Figure 8.2 activity covers the various management and support activities that are repeated on an ongoing basis throughout the project. It is composed of five sub-activities as shown in the Figure 8.9.

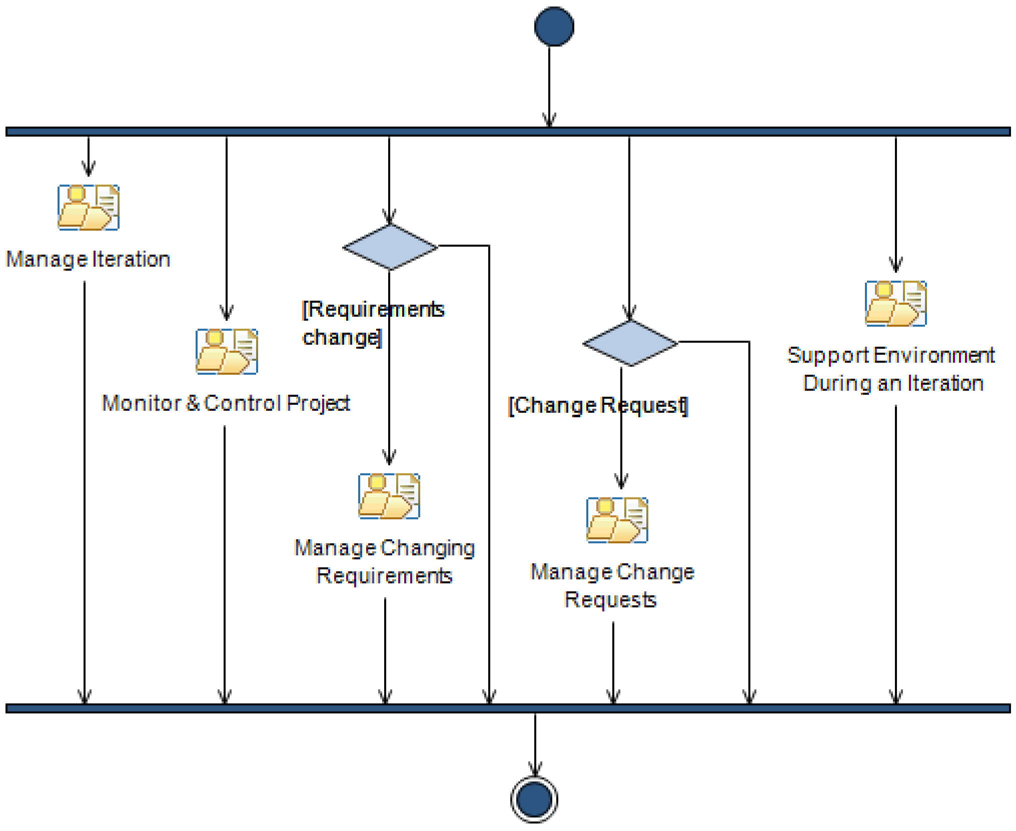

Manage Iteration
Monitor & Control Project
[Requirements change]
Manage Changing Requirements
[Change Request]
Manage Change Requests
Support Environment During an Iteration

Figure 8.9 Ongoing Management and Support Sub-Activities

Manage Iteration sub-activity contains the activities that begin, end and review the iteration. It consists of five major tasks as shown in the Figure 8.10.

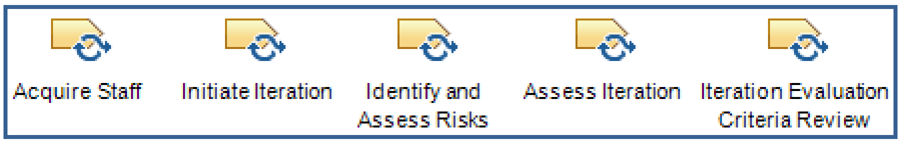

Acquire Staff
Initiate Iteration
Identify and Assess Risks
Assess Iteration
Iteration Evaluation Criteria Review

Figure 8.10 Manage Iteration Tasks

Acquire Staff task has the same responsibility as primarily performed in the Inception Iteration I1 of the inception phase. Within this task teams are formed with required skills for the Elaboration Iteration E1. Role to specific members are assigned in the team for the Elaboration Iteration E1 by the project manager. If any changes occurs on team members then Software Development Plan is revised to represent that changes and assign a role to new members.

Initiate Iteration task allocates team members to the activities identified for the current iteration. In the previous tasks members are assigned into specific roles. Now activities are assigned to team members that they have to perform. For that operation Work Order (see Appendix C.I.3) is revised and work order reports are prepared for each team member for the Elaboration Iteration E1. Description of the works is given in these reports as shown in the Figure 8.11.

Work Order for e-Reserve		**Created on: <12/02/08>**
Identification		
Work Order ID: <TMQ-E1-W01-01>	**WBS ID: <8.3.1>**	
Responsibility (Holder)		
Team Member Q		
Associated Change Reports		
None		
Schedule		
Start: < 12/22/08>	**Completion:** < 12/22/08>	**Critical Path:** <NA>
Efforts and Other Resources		
Staff Hours: 2 Hours	**Other Resources:** None	
Description		
Work Description: Implement Test Suit		
Expected Output(s): Test Suit		
Signature Agreement		
Project Manager: Team Member A **Signed on:** <dd/mm/yy>	**Work Order Holder:** Team Member Q **Signed on:** <dd/mm/yy>	

Figure 8.11 Work Order Reports for Iteration E1

Identify and Assess Risks task has the same responsibility as primarily performed in the beginning of the Inception Iteration I1 of the Inception Phase. As we mentioned before, risks are dynamic elements and we have to update the Risk List to reflect the current project status periodically for avoiding risks before they happen or handle them rapidly as they appear with a minimum effect on the project. So the Risk List (see Appendix C.I.4) is updated to capture the potential risks of the project. Within the Elaboration Iteration E1 some other potential risks are identified and defined in a decreasing order of priority. Some priorities of the existing risks are changed within the new identified list and new risks are added as shown in the Table 8.3.

Table 8.3 Risks for Iteration E2

Risk Ranking/ Magnitude	**Risk Description & Impact**	**Mitigation Strategy and/or Contingency Plan**
Medium	R1 and R2 Releases may slip and not be available by 05/12/2009.	Monitor progress against the schedule and milestones. Update effort to complete and time to complete on a regular basis.
Medium	Interfaces to the old legacy Library System may introduce performance and response time issues.	Continue to develop prototype. Monitor this issue at weekly progress meetings.
Low	The team is unfamiliar with Web architecture and technology. This risk may impact the ability to deliver a Web application on time.	Train the Team on Web technologies (to be done in the second elaboration iteration, or E2). Allocate time (during E2) for the learning curve and monitor progress weekly in elaboration.
Low	Volume of users logged on during peak periods (on holidays and special days) may significantly degrade system performance.	Early prototyping and extrapolation of response time data should be done in the elaboration phase.
Mitigated	Incompatibility with internet browsers and specific configurations on client machines.	Address during elaboration (E2).

On Figure 8.10, *Assess Iteration* task evaluates the results of an iteration assess related project information as in the Inception Iteration I1. The success or failure of the current iteration is determined. The actual and expected results of the current iteration are compared. Also we have to ensure that evaluation criteria for the current iteration are realistic. Test results are also given roughly, not in details. At the end of this task Iteration Assessment (see Appendix C.I.5) document is revised for the Elaboration Iteration E2. Objectives that are reached within the Elaboration Iteration E2 are given in Iteration Assessment. An example for the use cases that are implemented are shown as in the Figure 8.12.

2.2 Use Cases and Scenarios Implemented

The following use cases were completed with details uncovered during the design process:

- Login
- Logout

Figure 8.12 Objectives Reached for Iteration E2

The last task of Figure 8.10, *Iteration Evaluation Criteria Review* task determines how to approve the criteria if the iteration is completed with its meeting objectives. As we did in all previous review tasks, a meeting is planned and all materials about the activity are distributed across related team members to perform review. At the end of the meeting a decision is made to approve or reject the criteria for the Elaboration Iteration E2. If the criteria are approved then the next activity will be performed otherwise the project team should address the identified deficiencies and re-submit revised iteration evaluation criteria for a follow-up review. Review Record is completed at the end of meeting that captures the results of the current review activity.

Monitor and Control Project activity of Figure 8.9 is used to capture continuing work including monitoring project status, reporting to stakeholders, and dealing with issues. It consists of four major tasks as shown in the Figure 8.13.

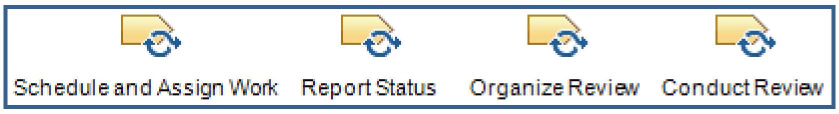


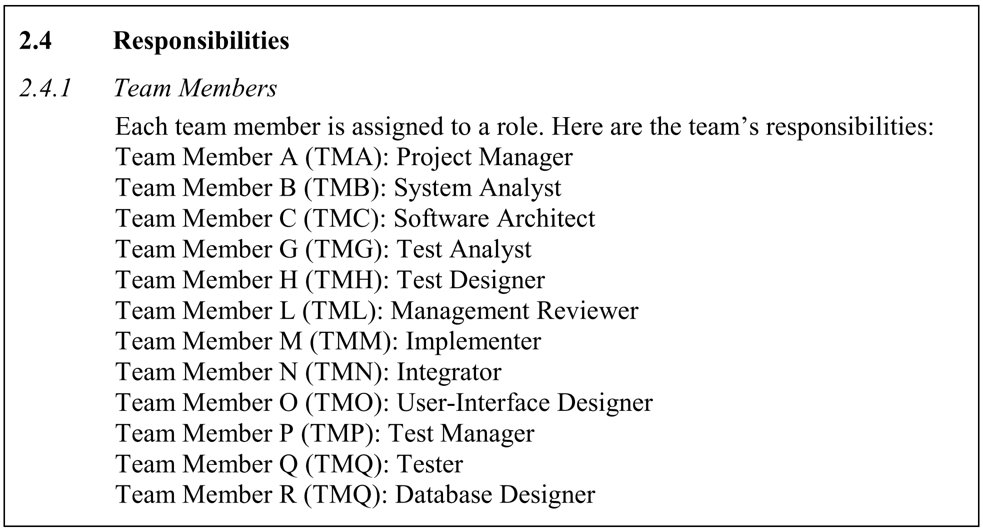
Figure 8.13 Monitor and Control Project Tasks

Schedule and Assign Work task helps project manager to schedule the work at any iteration. As we mentioned in the previous iteration Change Requests are critical for this task. Also for this iteration Change Requests have critical importance in this task for project manager to fix the iteration. If necessary, the Iteration Plan E1 is revised, and any impact on future iterations should be reflected in the Software Development Plan. The iteration overview of the Elaboration Iteration E1 that is reported in the Software Development Plan is shown in the Table 8.4.

Table 8.4 Iteration E1 Overview

Iteration	**Description**	**Risks Addressed**
E1 Iteration **Architectural Prototype for e-Reserve Application**	• **Complete analysis & design for high risk requirements** - Create Use Case Specification for each of the *Login, Logout* use cases, derive an Analysis Model, and refine it into a Design Model. - Document the architecture (high-level design) in the Software Architecture Document. • **Develops the architectural prototype for e-Reserve application** - Code the part of the application implementing the *Login, Logout* use cases. • **Demonstrate feasibility and performance through testing**	Architectural issues related to ERIS clarified. Technical risks related to ERIS mitigated. Early prototype for user review. Performance risks related to high volume of requests mitigated on the ERIS side.

During this task Work Order is revised to ensure that responsibilities that are given in the work order reports and schedules are satisfied. Each team member with a specific role has its own responsibilities. If any changing occurs on these roles or on the development team members of the project then this action have to be reported in the Work Order document. The roles of team members are identified for the Elaboration Iteration E1 is shown in the Figure 8.14.



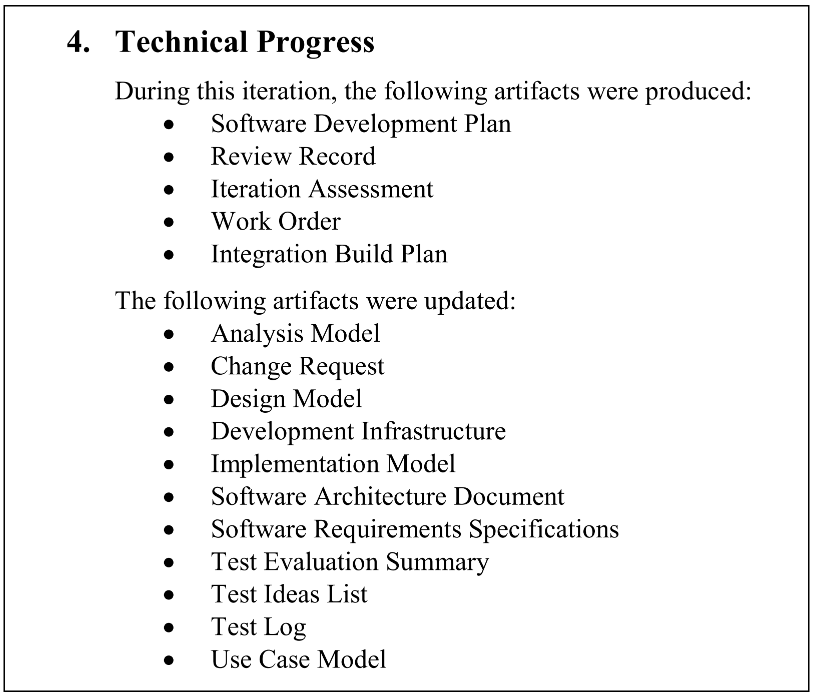
2.4 Responsibilities

2.4.1 Team Members

Each team member is assigned to a role. Here are the team's responsibilities:
Team Member A (TMA): Project Manager
Team Member B (TMB): System Analyst
Team Member C (TMC): Software Architect
Team Member G (TMG): Test Analyst
Team Member H (TMH): Test Designer
Team Member L (TML): Management Reviewer
Team Member M (TMM): Implementer
Team Member N (TMN): Integrator
Team Member O (TMO): User-Interface Designer
Team Member P (TMP): Test Manager
Team Member Q (TMQ): Tester
Team Member R (TMQ): Database Designer

Figure 8.14 Responsibilities for Iteration E2

Report Status task of Figure 8.13 is performed to identify when and how the periodic updates on the project will be done as performed in the previous iteration. An important work product in this task is the Status Assessment (see Appendix C.I.6). Many work products are produced until this task and many of them are still in progress. So status of each work product is given in Status Assessment Document for the current iteration. Risks are again given in the Status Assessment Document to point them out. The technical progress during the Elaboration Iteration E1 is reported in the Status Assessment as shown in the Figure 8.15.



4. Technical Progress

During this iteration, the following artifacts were produced:

- Software Development Plan
- Review Record
- Iteration Assessment
- Work Order
- Integration Build Plan

The following artifacts were updated:

- Analysis Model
- Change Request
- Design Model
- Development Infrastructure
- Implementation Model
- Software Architecture Document
- Software Requirements Specifications
- Test Evaluation Summary
- Test Ideas List
- Test Log
- Use Case Model

Figure 8.15 Technical Progress

Organize Review task of Figure 8.13 describes how to facilitate the review process and ensure the review is undertaken appropriately. Within this task review coordinator has to ensure that required review tasks are appropriately planned and organized. There are various approaches to planning review tasks which are based on factors such as team size, team culture and so on. Organize Review task is concluded with revision of Review Records.

The last task of Figure 8.13, *Conduct Review* describes how to facilitate a review so as to maximize the productivity of the reviewers and meet defined quality requirements. This task is under the control of reviewer of the project. Conduct Review task is concluded with revision of Review Records.

Referring back to Figure 8.9, ***Manage Changing Requirements*** sub-activity used to manage changes to requirements and assesses their overall impact. It consists of two major tasks as shown in the Figure 8.16.

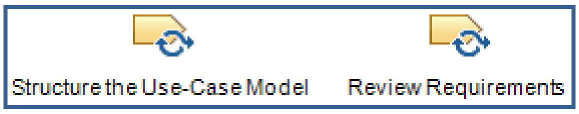


Figure 8.16 Manage Changing Requirements Tasks

Structure the Use-Case Model task is performed to make the requirements easier to understand and to maintain by structuring use case models. First of all review the use cases that are modeled in the previous iteration to form an understanding for the requirements. In this task use cases are analyzed in more details. Include and extend relationships are formed between use cases. An include relationship between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base. Only the base use case knows of the relationship between the two use cases. An extend relationship between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case. Only the extension use case knows of the relationship between the two use cases. Also generalization between use cases, and between actors can be defined. As we mentioned in the previous iteration if the number of actors and use cases becomes too large then they could be divided into use case packages. At the end of this task the Use Case Model (see Appendix C.I.7) is revised for the Elaboration Iteration E1 that is created in the Inception Iteration I1. Include and extend relationships are added to the use case diagrams as shown in the Figure 8.17.

Another work product of this task is the Glossary (see Appendix C.I.8) in which a common vocabulary, using the most common terms in the problem domain, is constructed. In some cases Glossary is the primary artifact that is used to capture information about the project's business domain. So it is important to update terms that are defined in the Glossary.

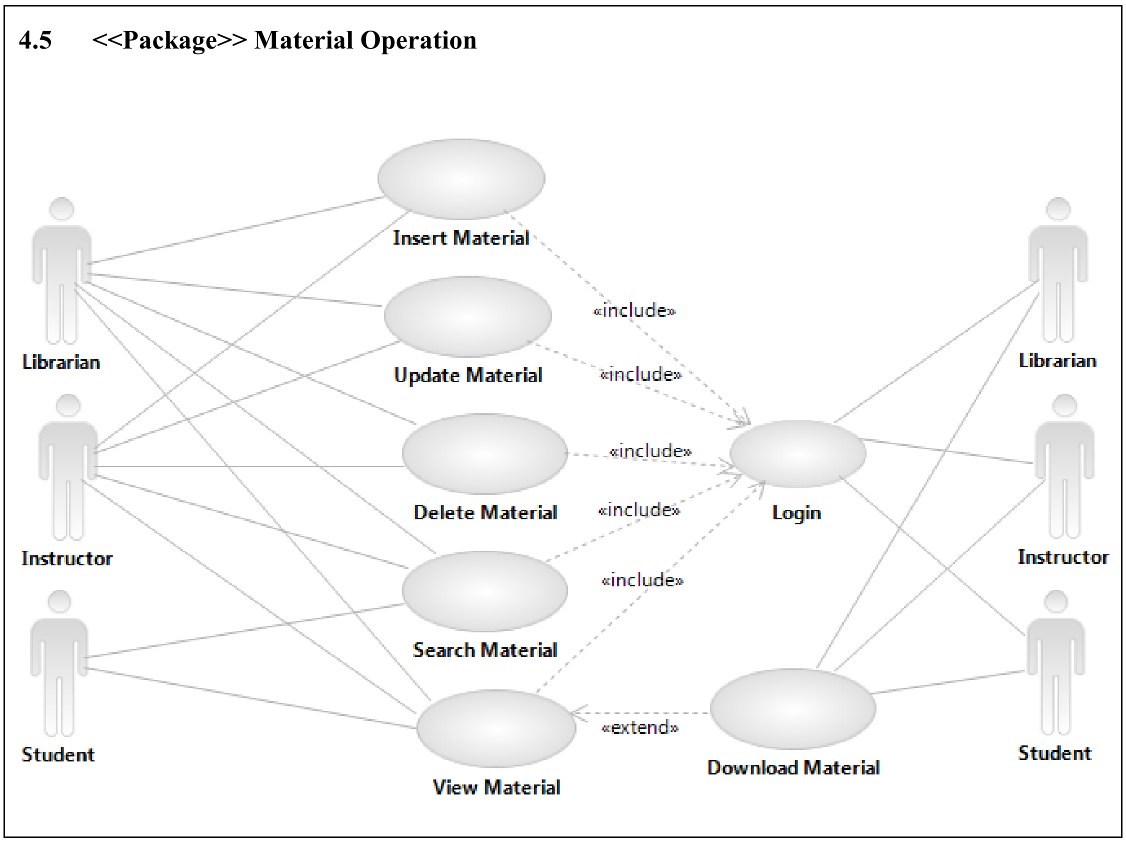


Figure 8.17 Use Case Diagram

Review Requirements task identifies how to review the requirements work products. The main purpose of this task is to review the results of the tasks that are related with requirements. Each review should include a recommendation. These recommendations are discussed in review meetings as done in the previous review tasks. At the end of meetings a Review Record is created that documents the review results.

Manage Change Requests sub-activity of Figure 8.9 is used to manage Change Request reports. The sub-activity ensures that due consideration is given to the impact of change on the project and that approved changes are made within a project in a consistent manner. It consists of four major tasks as shown in the Figure 8.18.

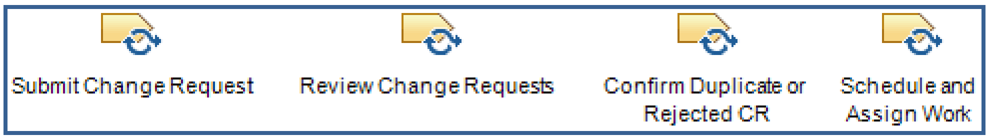


Figure 8.18 Manage Change Requests Tasks

Submit Change Request task identifies to how to create a Change Request. An advantage of this task is that it can be performed by any role who submits a change request throughout the project lifecycle. The work product of the task is Change Request (see Appendix C.I.9) that documents and tracks requests for a change to the product. Change Request can include new features, enhancement requests, defects, changed requirements, and so on. After we completed the Change Request it is submitted to its destination. Also a stakeholder on the project can submit a Change Request. Current problem and a proposed solution for that problem are shown in the Figure 8.19 from the Change Request.

Current Problem	
Description: Currently system is accessible by Student, Lecturer, and Librarian. However system cannot recognize which type of user currently uses the system. That causes security problems.	**Critical Failure:** N/A
	Nuisance: When accessing the system privileges cannot be assigned.
	Source of the Problem: Misunderstanding of the Software Requirements Specification
Enhancement/New Requirement/Other: User types will be received first before displaying its own form and a new attribute will be added to the user information tables.	
Observation conditions: During a review of prototype of the e-Reserve System.	
Proposed Change (Submitter)	
Description: Add a new attribute to the user information table in the database to determine the type of the user. Retrieve the user type from the user information tables when a student, instructor or librarian logins the system. Access the proper functions from the form depending on the received user type.	

Figure 8.19 Change Request Sections

Review Change Requests task of Figure 8.18 performs prioritizing Change Requests. This task determines to accept or reject a Change Request. Submitted Change Requests are viewed by a meeting to determine if it is a valid request or not. Change Requests include all state changes along with dates and reasons for the change. This information will be available for any repeat reviews. At the end of this task submitted Change Requests are reviewed. At the beginning of the meeting a Change Request is determined that it is valid or not. If it is valid but out of scope for the current release then it will be put in the Postponed state and will be held and reconsidered for future releases. If a Change Request is believed to be a duplicate of another Change Request that has already been submitted then it will be put in the

Duplicate state. If a Change Request has been determined to be in scope for the current release then it will be put in the Opened state and is awaiting resolution. If it is invalid then it will be put in the Closed state.

Confirm Duplicate or Rejected CR task of Figure 8.18 determines that the Change Request be rejected or labeled as Duplicate. First of all, change control manager retrieves the Change Request which will be labeled. In this task Change Requests are examined to decide whether it is a Duplicate or it must be Rejected. In some cases reports will return to the submitter to provide more information about the related Change Requests. At the end of the task Change Request is updated to represent its current status.

The last task of Figure 8.18, *Schedule and Assign Work* task describes all the things that must be accomplished for an approved Change Request to be incorporated in the development schedule. The Change Requests are examined and essential modifications are made on them in the previous tasks within this activity. As a result if the Change Request is to be held until a later iteration then the future iterations have to be re-planed by updating Software Development Plan, Iteration Plan, and Work Order documents.

Support Environment During an Iteration sub-activity of Figure 8.9 supports the development environment for a project. It consists of one major task as shown in the Figure 8.20.

Figure 8.20 Support Environment During an Iteration Task

Support Development task is performed to support development with hardware and software. It regroups a large range of technical services such as, maintaining the development infrastructure, backup, document creation and reproduction, and so on. The task is concluded with Development Infrastructure (see Appendix C.I.10) that includes hardware and software. As an example Project Management discipline in the RUP and the tools that are used to perform the activities and produce the artifacts necessary for the project development process are shown in the Figure 8.21 and reported in Development Infrastructure.

3. Project Management

3.1 Hardware

This discipline is focused on the project management activities. The tools for this include Microsoft Excel, Word and Microsoft Project. These tools are located on each user's desktop.

3.2 Software

The details on the minimum required software include:

- Version 2000 or higher for all three tools.
- MS Project recommended ideal version is 2007.

Figure 8.21 Project Management Discipline

Refine the System Definition activity of Figure 8.2 identifies the requirements for the current iteration. It consists of three major tasks as shown in the Figure 8.22.

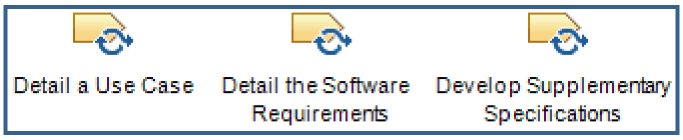

Detail a Use Case
Detail the Software Requirements
Develop Supplementary Specifications

Figure 8.22 Refine the System Definition Task

Detail a Use Case task examines use cases in more details. The task begins with reviewing the scenarios for the current iteration which is previously defined in the Inception Iteration I1. After reviewing use cases, flow of events are detailed for each of them. While defining flow of events for use cases it is important to be careful about what is done in the use case. The flow of event simply defines how and when the use case starts, how it interacts with the actors, what action does it takes, and how it ends. In some cases one action can be divided into several sub-actions. Each sub-action has to be defined in details as a main action. Preconditions and postconditions for a use case are defined within the flow of events. A precondition on a use case explains the state the system must be in order for the use case to be possible to start. A postcondition on a use case lists possible states the system can be in at the end of the use case. If a use case is extended by another use case it is specified in the flow of events. At the end of the task use cases are reviewed with the stakeholders to confirm that the required system is well understood. The Use Case Model (see Appendix C.I.11) is reviewed that is created in the previous iteration and flow of events is added for the use cases which are specified for the Elaboration Iteration E1. A flow of event for the use case "Login" is shown in the Table 8.5.

Table 8.5 Flow of Events

Use Case Name	**Login**	
Actor	Student, Instructor, Librarian	
Description	Student, Instructor or Librarian can access the system.	
Preconditions		
Post conditions	Actor accessed the system successfully.	
Priority	High	
Normal Course	Login	
	Actor Actions 1) Enter the username and password. 3) - 5) If Actor reached wanted result then press view button, if not reached go to state 1.	**System Responses** 2) Check validation control. 4) If not valid go to Alternative Course 1. If valid apply the criteria and display the result. 6) View the account information.
Alternative Course 1		
	Actor Actions 1) - 3) Correct the errors and resubmit. 5) If Actor reached wanted result then press view button, if not reached go to Normal Course state 1.	**System Responses** 2) Give the message that the submitted password criterion is not valid. 4) If not valid go to state 1. If valid apply the criteria and display the result. 6) View the account information.
Exceptions		
	Actor Actions	**System Responses**
Includes		
Special requirements		

Detail the Software Requirements task of Figure 8.22 identifies the requirements for the system that is to be developed. The task is initiated by detailing the software requirements clearly. These requirements can be managed by using special tools for graphical or textual documentation. At the end of the task all the requirements are packaged in the Software Requirements Specifications (see Appendix C.I.12) document that captures the software requirements for the system. Some specific functional requirements are defined in the Software Requirements Specifications for the Elaboration Iteration E1 is shown in the Figure 8.23.

3.3 Librarian Module

- The user shall be able to load the LM within Web Browser.
- The initial window of the LM shall contain a label for enter user id and password, and a button for login.
- When Librarian selects login. SRM checks the user and returns the successful message. After successful message a new page is loaded which is LM specific.

3.4 Server Module

- The SRM shall be the only intermediate between SM, IM, LM, and the database.
- The SRM shall receive all the requests and format the pages.
- The SRM shall accept all connections from the SM, IM, and LM.
- The SRM shall validate and execute all requests coming from LM.
- An error of execution, communication or else shall be identified and appropriate display.
- The SRM shall try recovery from most common errors.

Figure 8.23 Librarian and Server Modules for E1

Develop Supplementary Specifications task of Figure 8.22 is used to capture requirements that do not apply to specific use cases. As we mentioned in the previous iteration this task helps to capture requirements that are not readily captured in use cases. System wide functional requirements are identified in this task. Some constraints and compliance requirements such as licensing of the system are given in the Inception Iteration I1 of the inception phase. In Elaboration Iteration E1 Supplementary Specification (see Appendix C.I.13) document is revised to capture the interfaces of the system that must be supported by the application. An example of interfaces is depicted in the Figure 8.24.

10.2 Hardware Interfaces

All components must be able to execute on a personal computer.

10.3 Software Interfaces

Student Module, Instructor Module, and Librarian Module must be ASP running within browser. The server Module must integrate within a DBMS through Microsoft SQL Database Connectivity. The Server must run within a Web Server available for Windows NT.

10.4 Communications Interfaces

Student Module, Instructor Module, and Librarian Module must communicate within the server over a TCP/IP connection. The Server and Database components should be located on the same host.

Figure 8.24 Interfaces

Define a Candidate Architecture activity of Figure 8.2 establishes an initial sketch of the architecture. It consists of two major tasks as shown in the Figure 8.25.

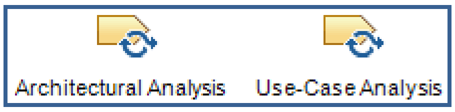


Figure 8.25 Define a Candidate Architecture Tasks

Architectural Analysis task defines the architecture of the system. An architecture overview is developed at the Inception Iteration I1 of the Inception Phase. During architectural analysis of Elaboration Iteration E1, we focus on the high-level layers. Within the current iteration we define the analysis mechanisms and services used by designers. A logical view is defined for the system and it is documented in the Software Architecture Document (see Appendix C.I.14). Logical view of a system illustrates the key use-case realizations, subsystems, packages and classes that encompass architecturally significant behavior. The most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers are described. The logical view of the system is comprised of three main packages as shown in the Figure 8.26.

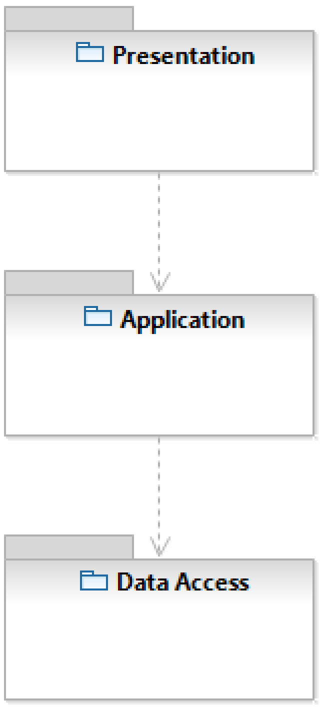


Figure 8.26 Logical View of the System

Use-Case Analysis task develops use case realization using predefined use cases. A work product is created for this task which is an Analysis Model (see Appendix C.I.15). It is the document that is created to identify set of analysis classes. Analysis Model is used to identify the behavior of the system that illustrates how it works. Responsibilities of analyses classes have to be identified that each analysis class should have several responsibilities. After finding responsibilities, we have to find associations between analysis classes that help us to understand class coupling. It is important to focus only on associations needed to realize the use cases. Use case realizations are reconciled that two different analysis use case realizations might include an analysis class that is conceptually the same. So the duplications can be reduced within this stage. Analyses classes that are identified for the Elaboration Iteration E1 are depicted in the Figure 8.27.

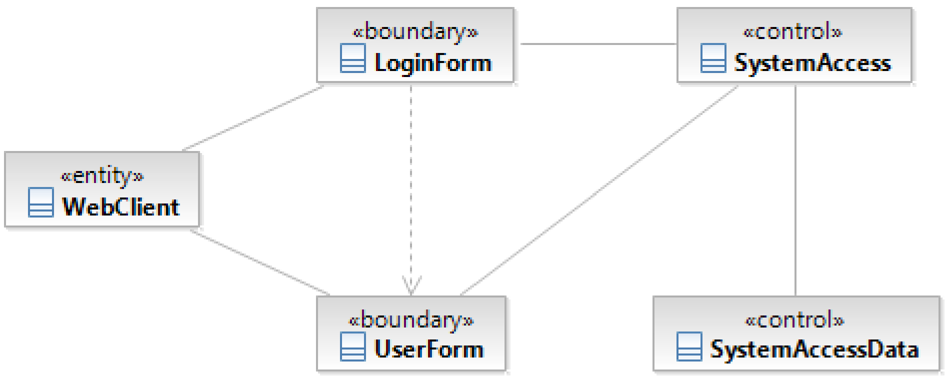


Figure 8.27 Analysis Classes for Iteration E1

Behavior of use cases are defined using a work product Use Case Realization Specification Document (see Appendix C.I.16). There should be a use case realization for each use case which needs to be expressed in the design model. This document enables the transition between requirements, and analysis and design tasks. For each use case realization within the Use Case Realization Specification Document there is one or more interaction diagrams depicting its participating objects and their interactions. We know that there are two types of interaction diagrams: communication diagrams and sequence diagrams. They express similar information, but show it in different ways. Communication diagrams show the communication links, whereas sequence diagrams show the explicit sequence of messages. Within the Use Case Realization Specification document sequence diagrams and communication diagrams are created for the use cases Login and Logout that are planned to be developed in Elaboration Iteration E1. Use Case Diagram, Communication Diagram and Sequence Diagram for the use case "Login" is given in Figures 8.28 – 8.30 respectively as example.

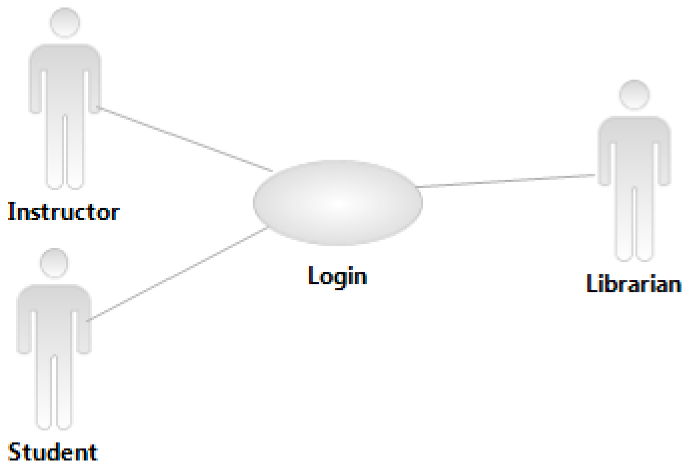


Figure 8.28 Use Case Diagram for Login

At the end of current task a review is performed to verify that the analysis objects meet the functional requirements and if the analysis objects are consistent.

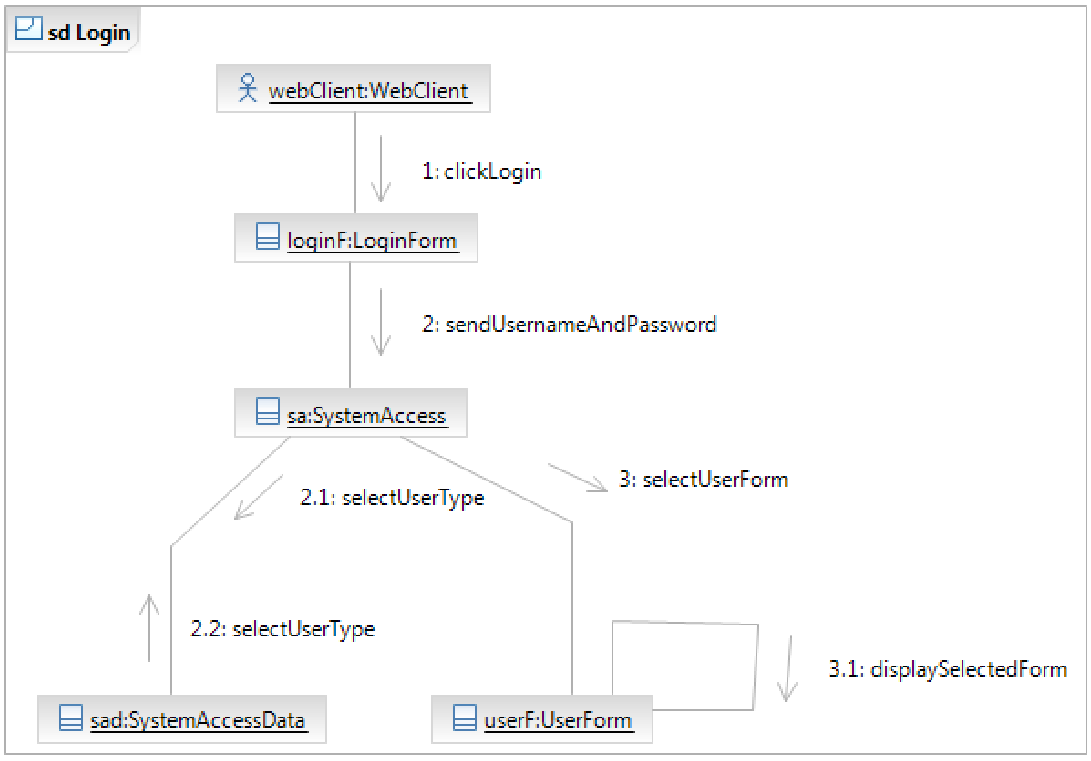


Figure 8.29 Communication Diagram for Login

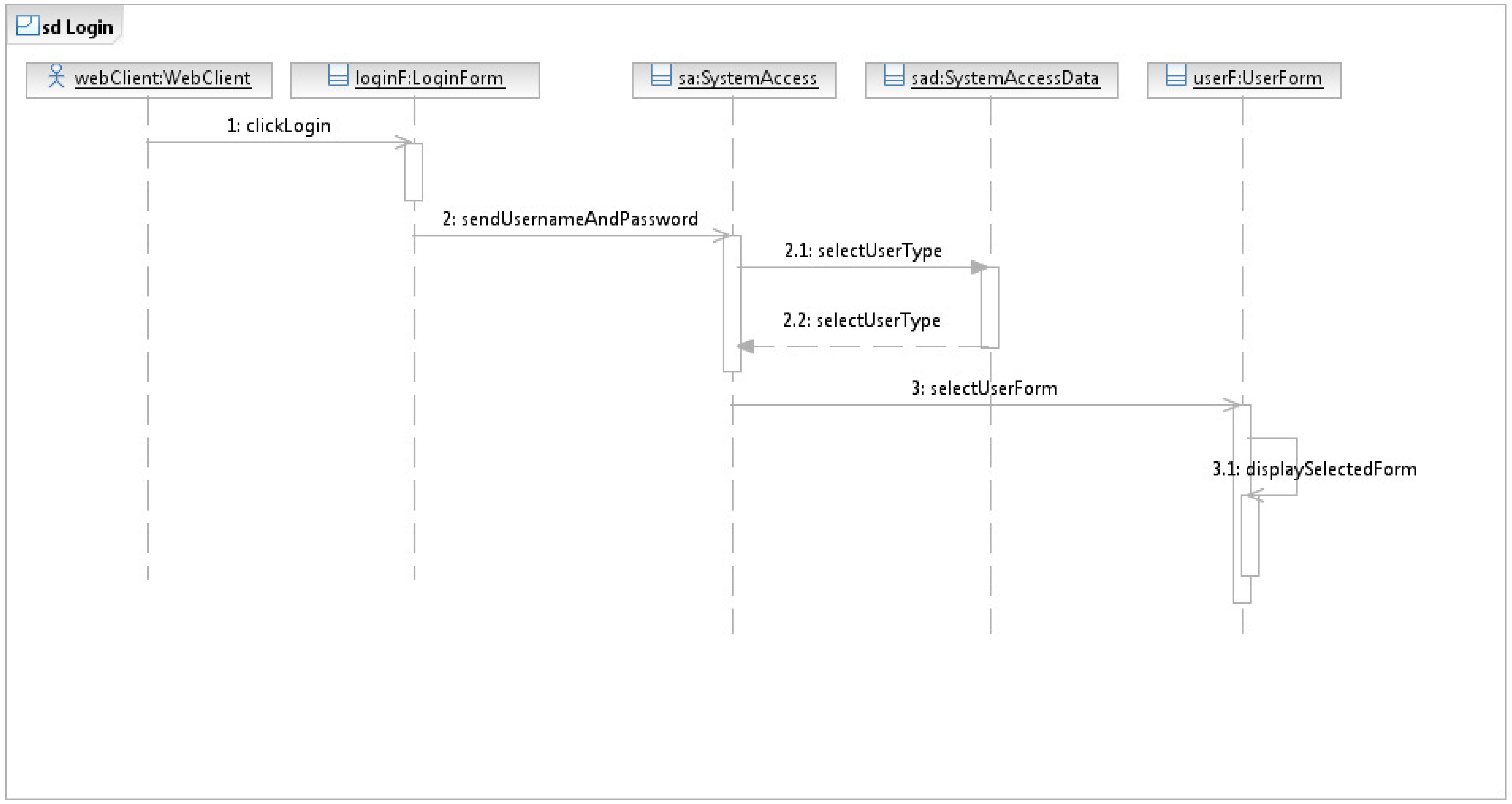


Figure 8.30 Sequence Diagram for Login

Refine the Architecture activity of Figure 8.2 completes the architecture for the current iteration. It consists of seven major tasks as shown in the Figure 8.31.

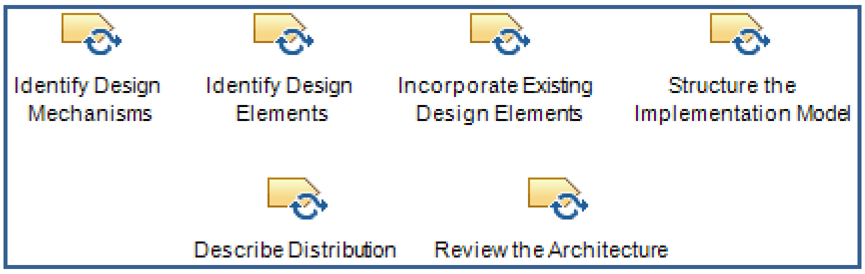

Identify Design Mechanisms
Identify Design Elements
Incorporate Existing Design Elements
Structure the Implementation Model
Describe Distribution
Review the Architecture

Figure 8.31 Refine the Architecture Tasks

Identify Design Mechanisms task adjusts analysis mechanisms into design mechanisms. First of all clients of analysis mechanisms are identified by looking at the characteristics they require for that mechanism. Characteristic profiles are also identified. They can be performance, footprint, security, economic cost and so on. All clients of analysis mechanisms are grouped according the characteristic profiles. This grouping is defined in Software Architecture Document that is also produced by Design Model (see Appendix C.I.17). As a result three layers are used as shown already in Figure 8.26.

Identify Design Elements task identifies subsystems, classes, interfaces, events and signals. In the first step of this task analysis classes are refined into appropriate design model elements. Analysis classes are very simple and they can be directly mapped to the design classes. Design classes are packaged in order to maintain configuration management processes. In some cases analysis classes become too complex to depict the behavior of a single class alone. In such a case analysis classes mapped into design subsystems which is modeled as a UML component having only interfaces as public elements. If a subsystem is defined within the system then we have to identify interface for each subsystem. For our project we do not need a subsystem so we do not deal with subsystem concept. At the end of the task Design Model (see Appendix C.I.18) the resulting document is reviewed. In the previous task we defined only the packages that will hold the design classes. Now for each package design classes are identified and placed under appropriate layers. The current version of Design Model Document contains the design classes for the Elaboration Iteration E1 as depicted in the Figure 8.32.

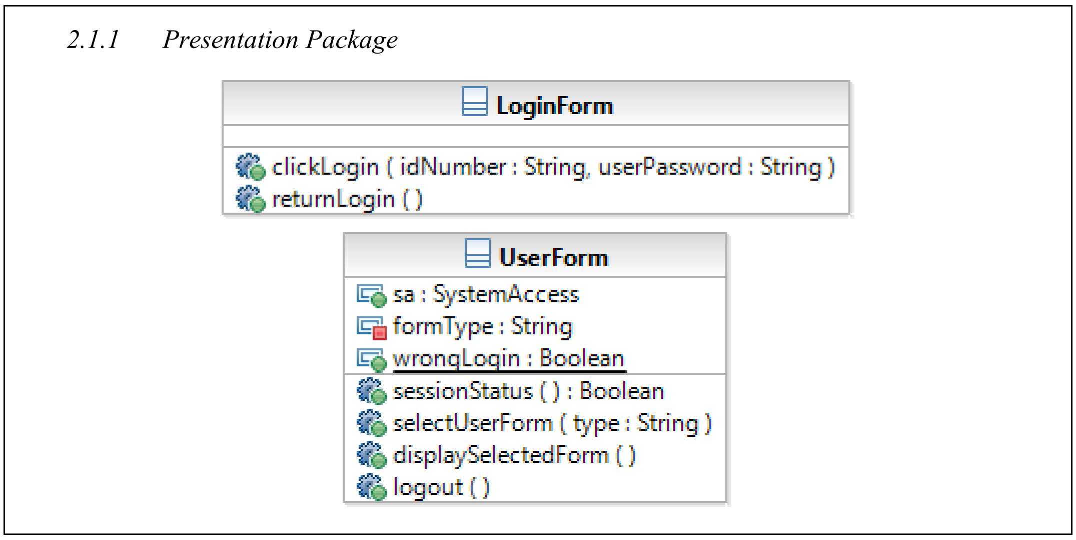

2.1.1 Presentation Package
LoginForm
clickLogin (idNumber : String, userPassword : String)
returnLogin ( )
UserForm
sa : SystemAccess
formType : String
wrongLogin : Boolean
sessionStatus ( ) : Boolean
selectUserForm (type : String)
displaySelectedForm ( )
logout ( )

Figure 8.32 Presentation Layer Design Classes for Iteration E1

Incorporate Existing Design Elements task of Figure 8.31 refines the Design Model. Reusable model elements from other projects or marketplace are incorporated. In the previous task design classes are identified and placed into appropriate packages and now all design classes are examined to identify relationships between them. The Design Model (see Appendix C.I.19) Document is revised to give the detailed relationships between design classes as shown in the Figure 8.33 on the next page. Some of the common components to other projects can provide many of the architectural mechanism needed for current project. In the next step of the task the logical view of the system is revised where new elements will be added to the Design Model and it requires updating logical view. For our project no updating is required that all elements that are added to Design Model adapts the current logical view of the system. Newly added elements in Design Model are represented in Software Architecture Document (see Appendix C.I.20) as in the Design Model document that is shown in Figure 8.32 because the design classes and packages are important from an architectural perspective. This task performs the identification of interactions between analysis classes.

Structure the Implementation Model task of Figure 8.31 establishes the structure of the implementation elements. Main goal in this task is to construct first version of the Implementation Model (see Appendix C.I.21) document. Implementation model structures are represented in the work product Implementation Model by packages and component diagrams. Design Packages will have corresponding Implementation

Subsystems and dependencies between these subsystems are identified carefully. For each subsystem it is defined which other subsystems it imports. The Build package is created that is the topmost level of the hierarchy of the implementation subsystems and represents a collection of executable programs produced by a build process. For the e-Reserve project packages from the Implementation Model are shown in the Figure 8.34 on the next page. At the end of the task the Software Architecture Document (see Appendix C.I.22) is revised and the implementation view is added. This section of the document contains component diagrams that show the layers and the allocation of implementation subsystems to layers, as well as import dependencies between subsystems as shown in the Figure 8.35 on the next page.

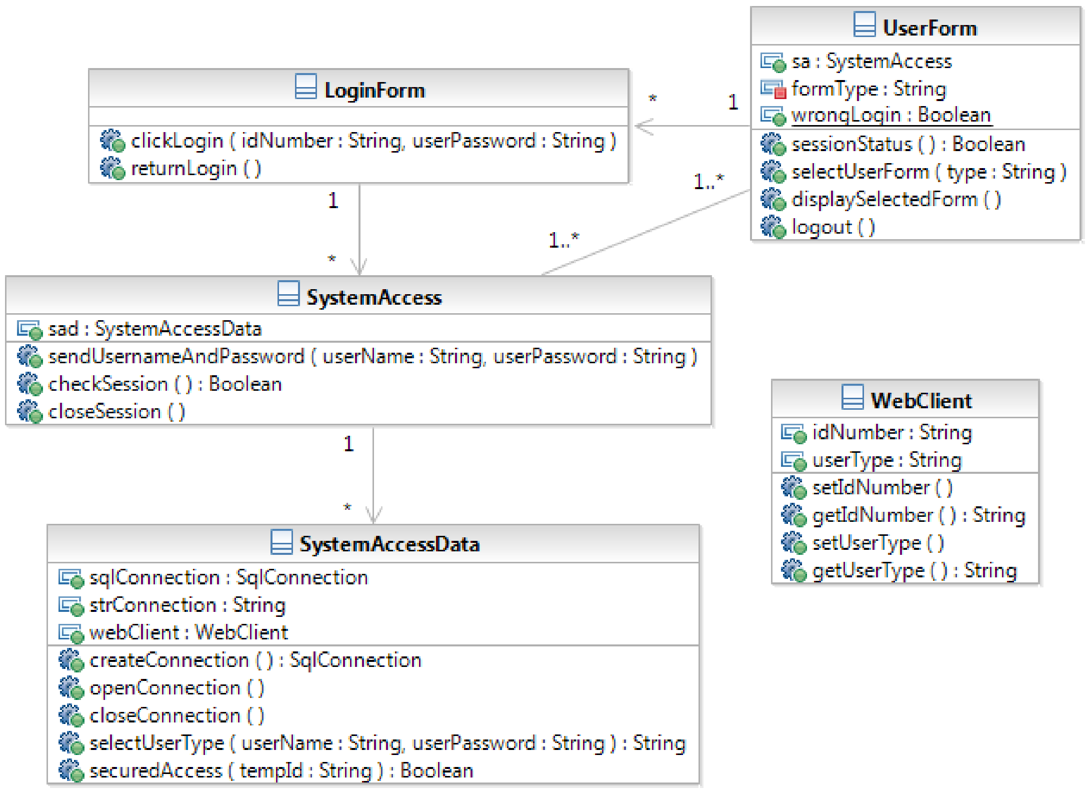


Figure 8.33 Design Class View for Iteration E1

Describe Distribution task of Figure 8.31 is used to describe how the functionality of the system is distributed. The distribution requirements and the network configuration are analyzed within this task. The initial design of the deployment is made in the previous iteration. Any changes on deployment view have to be reported in Deployment Model that is previously created.

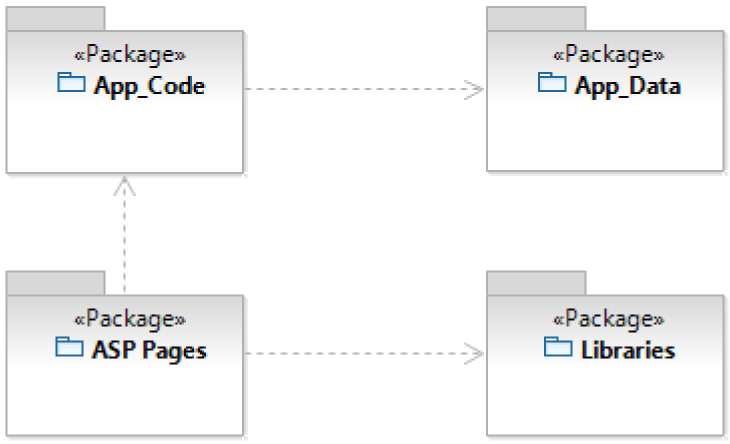
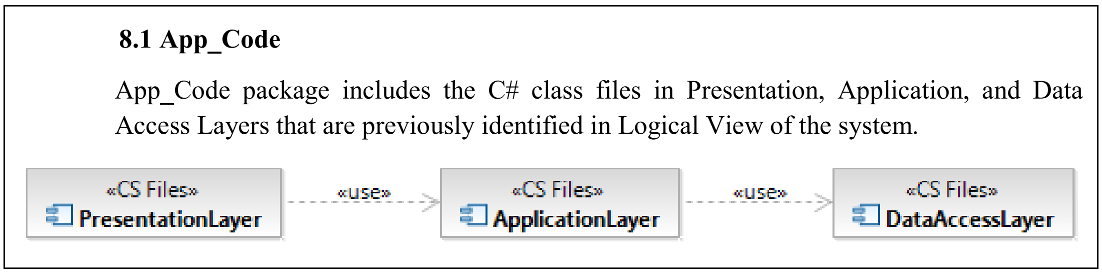


Figure 8.34 System Access – Overview (Level 1)


8.1 App_Code

App_Code package includes the C# class files in Presentation, Application, and Data Access Layers that are previously identified in Logical View of the system.

Figure 8.35 Implementation Model Package

Review the Architecture task is the last task of Figure 8.31 and it performs the review of the architecture. At the end of the Inception Phase, there is usually little of a concrete architecture in place. The most natural place for a software architecture assessment is at the end of the elaboration phase. For this iteration of the elaboration phase the scope and the goals of the review is defined. The review can be done in three different approaches as representation driven, information driven, or scenario driven. In representation driven review, a representation of the architecture is build and questions are asked on this representation. In information driven review, a list of information data is produced that is necessary for reasoning and then this information is compared to the requirements. In scenario driven review, general questions are transformed and asked in a set of scenarios. At the end of the review of the architecture defects are identified and detailed again in Review Record as done in the previous tasks.

Referring to Figure 8.2, **Develop Components [within Scope]** activity performs a group of sub-activities that are required to develop components within the scope identified in the iteration plan for Elaboration Iteration E1. It is composed of four sub-activities as shown in the Figure 8.36.

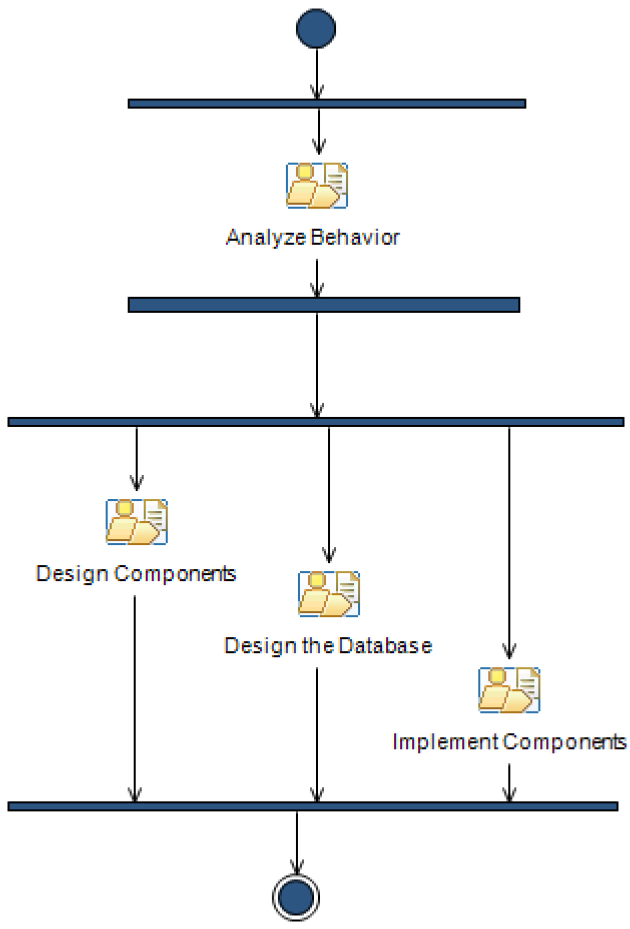


Figure 8.36 Develop Components Sub-Activities

Analyze Behavior sub-activity transforms behavioral descriptions into a set of elements upon which the design can be based. It consists of four major tasks as shown in the Figure 8.37.

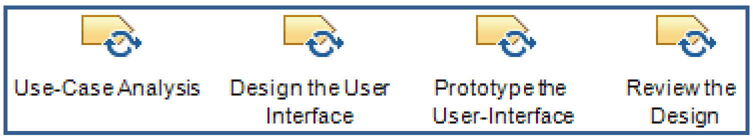


Figure 8.37 Analyze Behavior Tasks

Use-Case Analysis task develops use case realization using predefined use cases. This task is performed one more time while developing components. Use Case Realization Specification document is created in the previous tasks with the use case diagrams, sequence diagrams and communication diagrams. Now object diagrams are added to Use Case Realization Specification (see Appendix C.I.23) document to express the behavior of the use case in more details. These object diagrams show the relations and constraints between classes and objects involved in the use case. The object diagram for the use case Login is created as shown in the Figure 8.38.

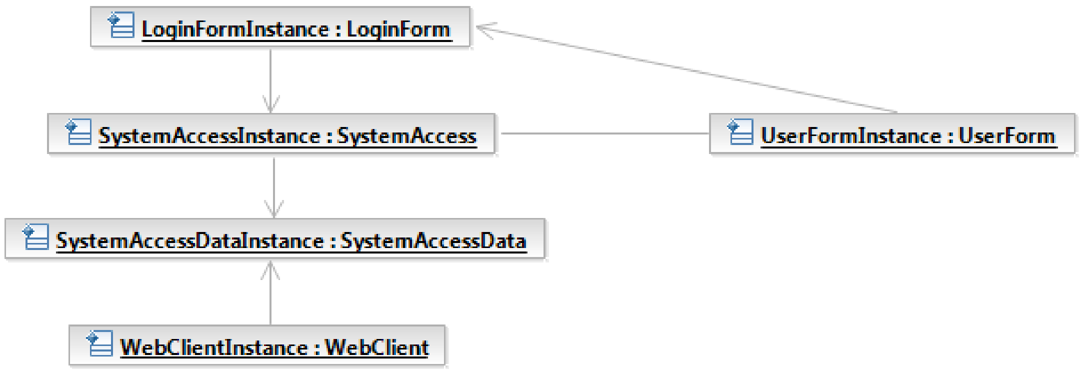

LoginFormInstance : LoginForm
SystemAccessInstance : SystemAccess
UserFormInstance : UserForm
SystemAccessDataInstance : SystemAccessData
WebClientInstance : WebClient

Figure 8.38 Object Diagram for Use Case Login

Design the User Interface task conducts graphical user interface design. The task is initiated by describing the characteristics of the users of the system. Task continues with identifying primary requirements that are captured in the previous activities. Primary windows for the system are created which are the essential windows when the user accesses the system. An important point is to always minimize the number of primary windows. Based on defined primary windows a Navigation Map (see Appendix C.I.24) is created which describes the structure of the user interface elements in the system and their pathways. This document contains only the main pathways, not a detailed view of the path. Potential paths are identified for the Elaboration Iteration E1 as shown in the Figure 8.39.

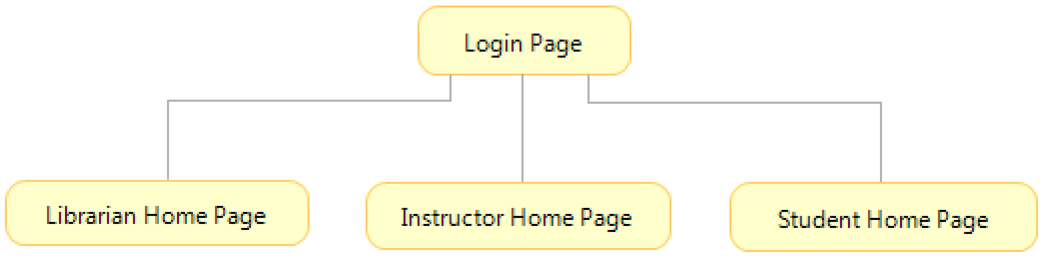
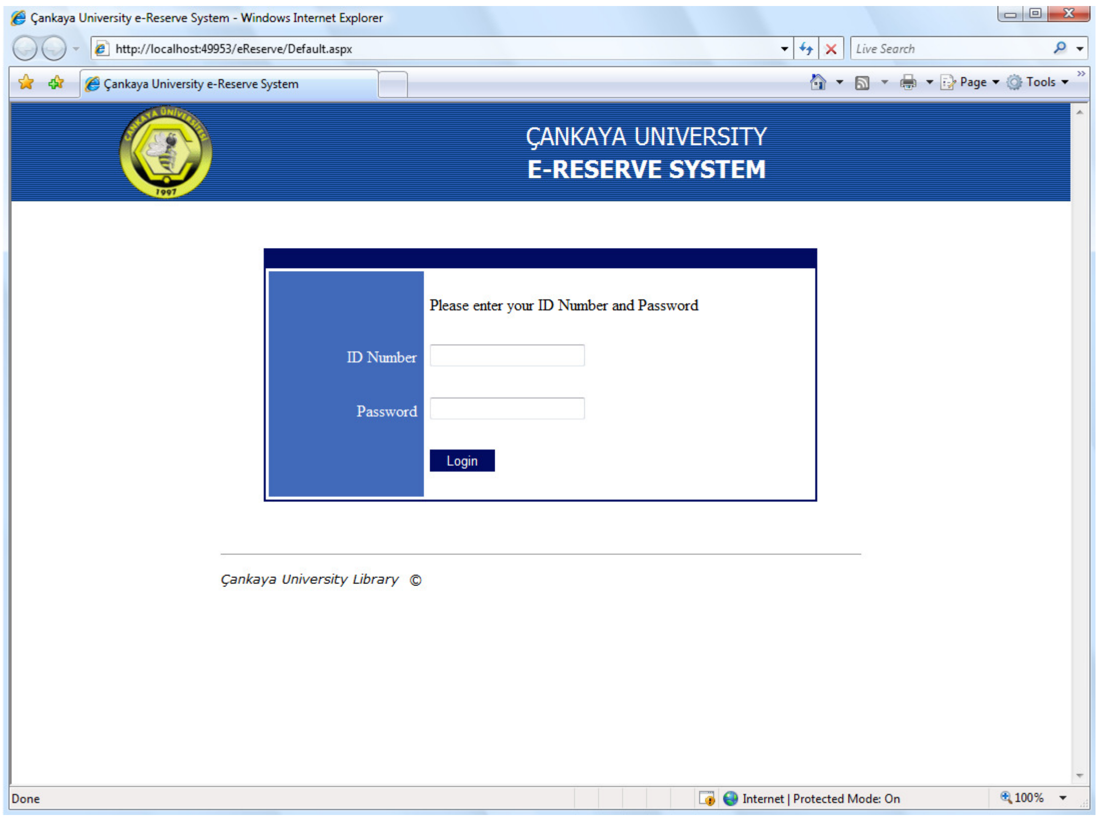


Figure 8.39 Navigation Map for Iteration E1

By the help of navigation maps users can simply figure out how many steps they require to reach their target page in the system.

Prototype the User-Interface task of Figure 8.37 explains how to develop a graphical user interface. A prototype for the user interface is created for the Elaboration Iteration E1 and documented in User Interface Prototype (see Appendix C.I.25) document that shows an example of user interface. In this task it is important to work closely with potential users of the system when prototyping the user interface that helps designers to discover any uncovered requirements. These prototypes provide a baseline for the system's user interface. Prototyping focuses on visualizing the significant aspects of the user interface instead of achieving a good structure and modularization of the source code. Prototyping is much cheaper than developing real interfaces at the early stages of the development. Because several changes on user interfaces and code are needed that causes waste of budget. A user interface prototype is created for the Elaboration Iteration E1 that is documented in the User Interface Prototype document as shown in the Figure 8.40.


Çankaya University e-Reserve System - Windows Internet Explorer

http://localhost:49953/eReserve/Default.aspx

ÇANKAYA UNIVERSITY
E-RESERVE SYSTEM

Please enter your ID Number and Password

ID Number

Password

Login

Çankaya University Library ©

Done

Internet | Protected Mode: On

Figure 8.40 User Interface Prototype for Iteration E1

The last task of Figure 8.37 is *Review the Design* task and it defines a review of the design up to current task. The Design Model is reviewed to ensure that it is well formed. Within the Elaboration Iteration E1 the review on Design Model is focused on the overall structure of it. Also the Use Case Realizations are reviewed to ensure that the behavior of the system matches the required behavior. Results and defects are reported in Review Records as in all review meetings.

Design Components sub-activity of Figure 8.36 refines the design of the system. It consists of three major tasks as shown in the Figure 8.41.

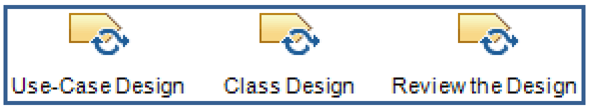


Figure 8.41 Design Components Tasks

Use-Case Design task refines use case realizations that are defined in the previous tasks. Analysis mechanisms are identified in the previous tasks within this task any applicable design mechanisms are incorporated into the use case realizations. For each use case realization the interaction between design objects are shown by using sequence diagrams. The interaction between an object and an actor is represented briefly in sequence diagrams. Also each flow variant can be defined in a separate sequence diagram. Flow of events is refined within this task in which they are defined in the previous tasks and may need to be added additional description to the sequence diagrams using annotations or notes. At the end of the task design model is checked, but not in detail, to verify the work is in the right direction.

Class Design task is used to design the class structure of a component in the system. The main goal in this task is to ensure that classes provide the behavior required by the use case realizations. Classes are created firstly while producing analysis classes as boundary, control, and entity classes. Boundary classes represent the interface so the design of them depends on the user interface development tools that are available to the project. Entity classes represent manipulated units of information and they are detailed in when designing the database. Control classes are responsible for managing the flow of a use case. In the next step of the task, visibilities are defined for each class. Operations are defined for each class that is required for message passing in a sequence diagram. After defining operations briefly their visibilities are identified. Attributes are identified that are needed by the class to carry out its operations. Name, type, and visibilities of attributes are identified. Finally associations and generalizations between classes are identified. At the end of the task design model is checked again as in the previous task, but not in detail, to verify the work is in the right direction.

Review the Design task defines a review of the design for the current activity. It is primarily performed in the previous sub-activity. The Design Model and Use Case Realizations are reviewed again but including current changes to these documents. Results and defects are reported in Review Records as in all review meetings.

Design the Database sub-activity of Figure 8.36 designs the corresponding database structures. It consists of two major tasks as shown in the Figure 8.42.

Database Design
Review the Design

Figure 8.42 Design the Database Tasks

Database Design task defines the way of designing a database for the current iteration. Before beginning the task optionally a logical data model can be created. It is not a necessity but provides an idealized view of the key logical data entities and their relationships. It is in the third normalized form that minimizes the redundancy. The main goal of this task is to develop a physical design of the database. The physical database design represents the physical structure of the database. This physical data model is represented in Data Model (see Appendix C.I.26) that describes the logical and physical representations of persistent data used by the application. The system access operations database tables created for the Elaboration Iteration E1 is depicted in Figure 8.43.

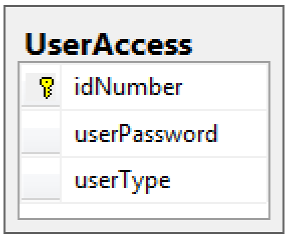


Figure 8.43 Database Tables for Iteration E1

For developing physical database designs, first of all domains are defined. Then the physical data model elements are designed using tables and columns in tables. One or more columns are selected as a primary key to uniquely identify the row of tables.

For Elaboration Iteration E1 it is not necessary to detail the design because there is only one database table in our application.

Review the Design task of Figure 8.42 defines a review of the design for the current activity. This task is performed to verify that design model fulfills the requirements. Now Data Model is reviewed in addition to Design Model to ensure that it fulfills the objectives. As in all previous review tasks, a meeting is performed then the results and defects are reported in Review Records for that task.

Referring to Figure 8.36, ***Implement Components*** sub-activity completes a part of implementation. It consists of two major tasks as shown in the Figure 8.44.

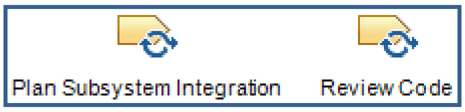


Figure 8.44 Implement Components Tasks

Plan Subsystem Integration task plans the order in which the elements contained in an implementation subsystem should be integrated. First of all the build is defined by selecting the use cases and scenarios for the Elaboration Iteration E1. In the next step of the task, classes are identified that will be participated in the selected scenario from use case realizations. These scenarios are described in the use case realizations by using sequence and communication diagrams. The task concludes by revising the Integration Build Plan (see Appendix C.I.27) that is added integration build one for the Elaboration Iteration E1 as shown in the Figure 8.45.

Integration Build One includes the following Subsystem and Components:

Subsystem	Components
System Access	• LoginForm CS File • UserForm CS File • SystemAccess CS File • WebClient CS File • SystemAccessData CS File • UserAccessTable DBO File • back GIF File • logo GIF File • Default ASPX File (index file) • Student ASPX File • Instructor ASPX File • Librarian ASPX File

Figure 8.45 Build for Iteration E1

Review Code task of Figure 8.44 is performed to review the code to verify the implementation. There three different techniques for reviewing the code. By inspection technique, the implementation is examined in detail. By walkthrough technique, the author of the implementation leads one or more reviewers through the implementation. By code reading technique, one or two persons read the code. At the end of the task, a meeting is performed then the results and defects are reported in Review Records for that task, as in all previous review tasks.

Integrate and Test activity of Figure 8.2 includes tasks to fully integrate and test the product. It is composed of three sub-activities as shown in the Figure 8.46.

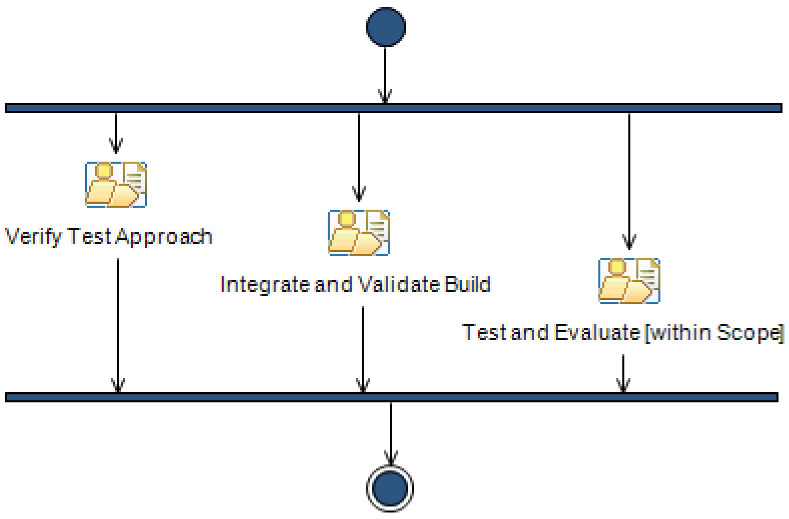

Verify Test Approach
Integrate and Validate Build
Test and Evaluate [within Scope]

Figure 8.46 Integrate and Test Sub-Activities

Verify Test Approach sub-activity represents that the techniques outlined in the Test Approach will facilitate the planned test effort. It consists of one major task as shown in the Figure 8.47.

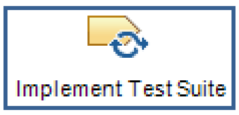


Figure 8.47 Verify Test Approach Task

Implement Test Suite task identifies the grouping of tests to be executed. Test Suite is a collection of related test cases. Test cases can be grouped together to perform different types of activities, such as unit test, integration test, system test, or acceptance test. First of all some Test Suite candidates are selected to be implemented by using test ideas list. Dependencies between tests are identified. If there exists any dependencies then the correct sequence of execution of the tests have to be defined. On the other hand, identifying opportunities for reuse improves the Test Suite maintainability. Test Suite is stabilized to resolve any dependency problems both in terms of system state and test execution sequences. Errors can occur when tests are executed together within a given Test Suite. So it is better to run the Test Suite regularly as new tests are added. Unit tests are the initial tests that will be performed for our project so a Test Suite is created for the unit tests to be run in a given order. Every method of every class will be tested in the project so a Test Suite (see Appendix C.I.28) is created for each class. Test Suites will be documented in the work product that will be created in the next task.

Integrate and Validate Build sub-activity of Figure 8.46 includes activities that are required to integrate, build and validate the build for the entire system. It is composed of three more sub-activities as shown in the Figure 8.48.

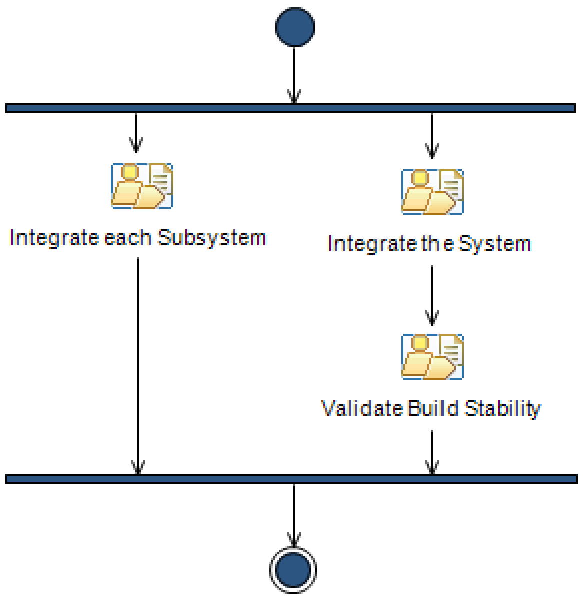


Figure 8.48 Integrate and Validate Build Sub-Activities

Integrate each Subsystem sub-activity is used to create a consistent implementation subsystem. It consists of three major tasks as shown in the Figure 8.49.

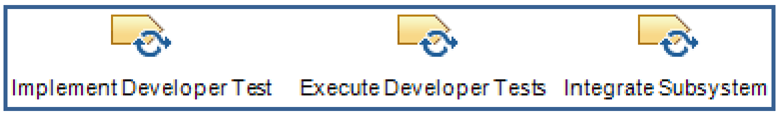
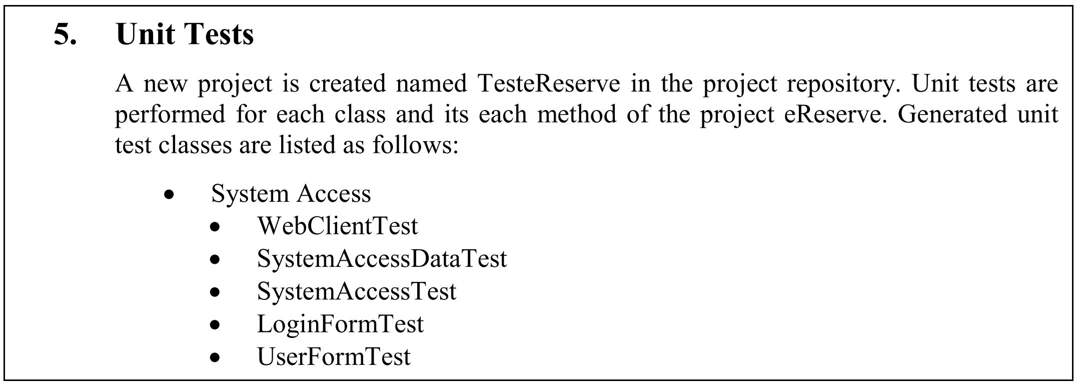


Figure 8.49 Integrate each Subsystem Tasks

Implement Developer Test task is used to create a set of test to validate components before other tests are performed. The task begins by identifying the components that are to be tested. While identifying components, its scope and test type is also defined. After defining fundamental concepts, an appropriate technique is determined to implement the tests in terms of manual and automated testing. Now it is the time to implement the tests that are defined in the previous steps of the task. External data sets can also be created that allows other tests to use current test results. Finally tests are verified to ensure that they work correctly. If any defects occur then discover it during debugging and fix it. At the end of the task a Developer Test (see Appendix C.I.29) is produced. Developer Test defines types of testing that are used in the project, such as Function Testing, User Interface Testing, Data and Database Integrity Testing, Performance Profiling, Load Testing, Volume Testing, Configuration Testing. Testing of the project is begun with unit testing that is reported in the Developer Test as shown in the Figure 8.50.



5. Unit Tests

A new project is created named TesteReserve in the project repository. Unit tests are performed for each class and its each method of the project eReserve. Generated unit test classes are listed as follows:

- System Access
  - WebClientTest
  - SystemAccessDataTest
  - SystemAccessTest
  - LoginFormTest
  - UserFormTest

Figure 8.50 Unit Test for Iteration E1

Execute Developer Tests task of Figure 8.49 runs and evaluates the tests designed in the previous task before more formal tests are performed. For each unit in the test suit a sequence of operations are performed. First of all the test environment is set up and initialized then each unit test is executed. Execution of each unit test is examined to ensure that they complete its execution successfully. If any test is halted for any reason, after determining and correcting the problem test is executed again from the beginning. When testing is completed, the test results are reviewed to ensure the test results are reliable. At the end of the task the Test Log (see Appendix C.I.30) is produced which is the raw output captured during a unique execution of the tests. The Test Log represents the output resulting from the execution of each

Test Suite that is defined in the previous tasks. As an example, the execution of a Test Suit is resulted by a Test Log in which a small partition of it is shown in the Figure 8.51.

```
<UnitTestResult
   executionId="9413f140-8368-4c25-a08a-513b345b791d"
   parentExecutionId="8853e85e-024f-4ebe-bf84-35ab80c98ec5"
   testId="1d7266db-328e-128e-410f-7be4f56c85f9"
   testName="checkSessionTest"
   computerName="TUFAN-PC"
   duration="00:00:01.3569185"
   startTime="2008-12-29T11:59:55.5772166+02:00"
   endTime="2008-12-29T11:59:56.9990916+02:00"
   testType="13cdc9d9-ddb5-4fa4-a97d-d965ccfc6d4b"
   outcome="Passed"
   testListId="8c84fa94-04c1-424b-9868-57a2d4851a1d">
</UnitTestResult>
```

Figure 8.51 Unit Test Result of checkSessionTest

Integrate Subsystem task of Figure 8.49 integrates the elements in an implementation subsystem, then deliver the implementation subsystem for system integration which will be performed in the next task. The order of implementation elements and integration subsystem is previously defined. So the implementation elements are integrated in a bottom up fashion as in their defined order. At each increment only one element is added to the system. After performing all increments, the implementation subsystem is delivered into the system integration.

Integrate the System sub-activity of Figure 8.48 integrates implementation subsystems to create a new consistent version of the overall system in the current iteration. It consists of one major task as shown in the Figure 8.52.

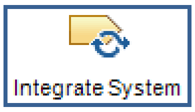


Figure 8.52 Integrate the System Task

Integrate System task integrates the implementation subsystems into a build. Depending on the complexity and the number of subsystem to be integrated, the build is produced in a number of steps. The sequence of components that is to be integrated is previously defined in the Integration Build Plan (see Appendix C.I.27) and the details about the build are documented in Implementation Model (see Appendix C.I.21). After performing all steps, the build becomes ready for system testing. When the build testing is finished, associated baselines are promoted by marking baselines as having passed or failed a certain level of testing.

Validate Build Stability sub-activity of Figure 8.48 performs validation operations that if the build is stable enough for detailed test and evaluation effort to begin. It consists of one major task as shown in the Figure 8.53.

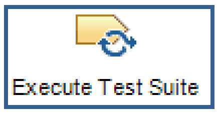


Figure 8.53 Validate Build Stability Task

Execute Test Suite task executes the appropriate collections of tests required to evaluate product quality. First of all the test environment is established to execute the Test Suite. Tools are configured that is used in the Test Suite execution. After setting the environment, the appropriate time is selected and the Test Suite is executed. The Test Suite is previously executed in the previous tasks by the implementer, but now it is executed by the tester and a new version of Test Log is obtained at the end of task. The steps for executing a Test Suite are same as done in the previous tasks.

Test and Evaluate [within Scope] sub-activity of Figure 8.46 includes the activities required for testing within a particular scope. It is composed of three more sub-activities as shown in the Figure 8.54.

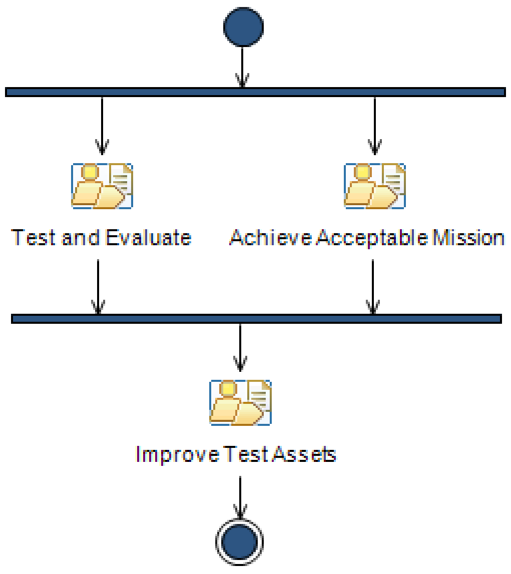


Figure 8.54 Test and Evaluate Sub-Activities

Test and Evaluate sub-activity achieves appropriate breadth and depth of the test effort to enable a sufficient evaluation of the items being targeted by the tests. It consists of five major tasks as shown in the Figure 8.55.

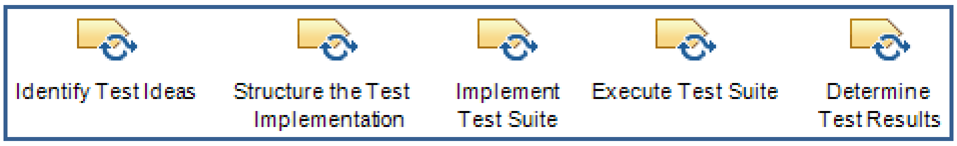


Figure 8.55 Test and Evaluate Tasks

Identify Test Ideas task is used to identify test cases. The task begins with identifying the test motivators that driving the test effort for the Elaboration Iteration E1. Team members brainstorm to create potential test ideas and the most appropriate ones are documented in the Test Case (see Appendix C.I.31) document for each use case included within the current iteration. At the next step all test ideas are refined to make further revisions and improvements. At that moment it is very important to collect many test ideas as much as possible. Finally the task concludes by verifying it to ensure that has been completed appropriately. The Test Case created for the Login use case is depicted in the Table 8.6 (V: Valid, I: Invalid).

Table 8.6 Test Case for Use Case Login

TC ID #	Scenario Name	Page Info	User ID	Password	User Type	Expected Result
Li1	Successful Login	V	V	V	V	Successful Login.
Li2	Invalid Redirection	I	n/a	n/a	n/a	Error Message. Home Page is not available.
Li3	Incorrect Validation (No User ID)	V	n/a	V	n/a	Error Message. Return to Basic Flow 2.
Li4	Incorrect Validation (No Password)	V	V	n/a	n/a	Error Message. Return to Basic Flow 2.
Li5	Incorrect Validation (No User ID and Password)	V	n/a	n/a	n/a	Error Message. Return to Basic Flow 2.
Li6	Incorrect Access	V	I	I	n/a	Redirected to home page. Return to Basic Flow 2.
Li7	Invalid User Form	V	V	V	I	Error Message. Return to Basic Flow 1.

Structure the Test Implementation task of Figure 8.55 defines the overall structure for the test suite implementation. Structuring the test implementation begins with examining the test approach, target test items and assessment needs to understand how the testing will be assessed. After performing this step, an initial Test Suite is structured before its implementation. Test cases are produced in the previous task so the only remaining work is assembling them with the Test Suit. The Test Suite structure is adapted to reflect team organization and tool constraints to work with the team responsibility assignments. Within Test Suites, each Test Case is called in a prescribed order so the test designer has to be ensuring the correct system state is passed through from one Test Case to the next. Some initial dependencies between Test Suite elements are also defined at current task. Finally Test Suite structure is refined to make necessary adjustments to maintain the integrity of the test implementation.

Implement Test Suite task identifies which tests should be executed together. This task is primarily performed in the previous activities but for the unit testing. Now the Test Suite has to be implemented for the Test Cases. Firstly the candidate Test Suite is defined by reviewing Test Cases that are related with each other. All steps are performed again same as done for the unit testing to the workspace that is previously reported in the Developer Test. There are only two Test Cases for the Elaboration Iteration E1 and they are placed in a Test Suite (see Appendix C.I.32) System Access Operations that holds Login and Logout Test Cases.

Execute Test Suite task of Figure 8.55 executes the appropriate collections of tests required to evaluate product quality. This task is again primarily performed in the previous activities but for the unit testing. Now the Test Suite has to be executed for the Test Cases. The test environment is established to execute the Test Suite and the tools are configured. After all configurations are finished, the Test Suite is executed. Execution of each Test Case is examined to ensure that they complete its execution successfully. If any test is halted for any reason, after determining and correcting the problem Test Suite is executed again from the beginning. At the end of the execution of Test Suite a Test Log (see Appendix C.I.33) is obtained that contains the raw output from Test Cases. A Sample Test Log from the execution of the Test Suite is shown in the Table 8.7.

Table 8.7 Test Case Result of Login

Date	Test Module	Test Case	Input	Expected Output	Actual Output	Recommendation
12/25/2008	Login	Successful Login	Valid Page Info Valid User ID Valid Password Valid User Type	Selected user form is displayed.	Display the expected output correctly.	
		Invalid Redirection	Invalid Page Info	Error message is displayed. Home page cannot be shown.	Display the expected output correctly.	
		Incorrect Validation (No User ID)	Valid Page Info No User ID Valid Password	Error message is displayed. Wait on home page to enter User ID.	Display the expected output correctly.	
		Incorrect Validation (No Password)	Valid Page Info Valid User ID No Password	Error message is displayed. Wait on home page to enter Password.	Display the expected output correctly.	
		Incorrect Validation (No User ID and Password)	Valid Page Info No User ID No Password	Error message is displayed. Wait on home page to enter both User ID and Password.	Display the expected output correctly.	
		Incorrect Access	Valid Page Info Invalid User ID Invalid Password	Error message is displayed. Wait on home page to enter valid User ID and Password.	Display the expected output correctly.	
		Invalid User Form	Valid Page Info Valid User ID Valid Password Invalid User Type	Error message is displayed. Directed to home page.	Display the expected output correctly.	
12/25/2008	Logout	Successful Logout	Valid Page Info Valid Session State	Logout from the system. Redirected to home page.	Display the expected output correctly.	
		Session Error	Valid Page Info Invalid Session State	Error message is displayed.	Error message cannot be displayed. System does not response.	Hardware solution is required.
		Invalid Redirection	Invalid Page Info Valid Session State	Error message is displayed. Home page and user form cannot be shown.	Display the expected output correctly.	

The last task of Figure 8.55, *Determine Test Results* task reports and summarizes the test findings. The task begins by forming an understanding on resulting problems. Test Logs are analyzed and the Test Results (see Appendix C.I.34) document is created that provides a detailed assessment of the quality of the target test items and the status of the test effort. Test Results document summarizes the status of Test Cases after their execution as shown in the Table 8.8.

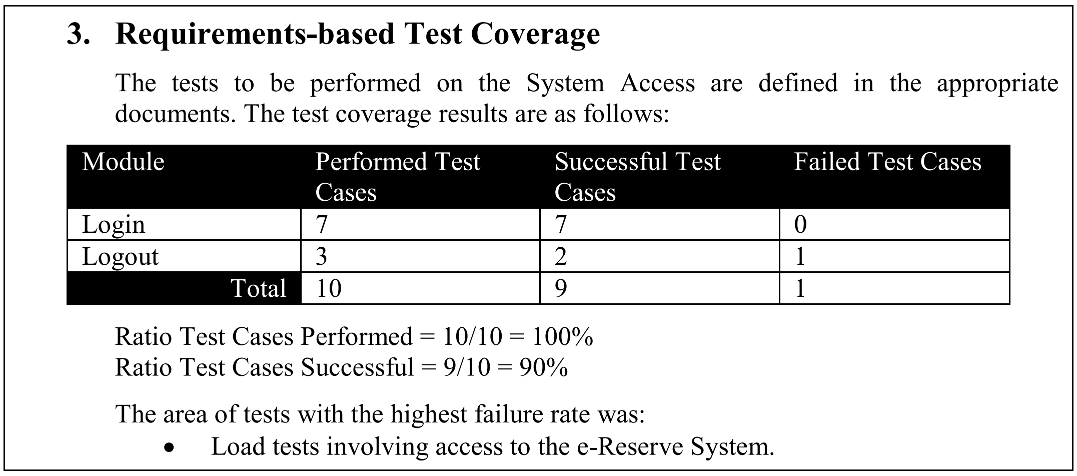
Table 8.8 Test Case Login Execution Results

Test Results – Login Test Case	Status
Li1	Succeeded
Li2	Succeeded
Li3	Succeeded
Li4	Succeeded
Li5	Succeeded
Li6	Succeeded
Li7	Succeeded

By using the summary in Test Results document a Defect Report (see Appendix C.I.35) is created for each failed test. It is important for Defect Report to be understandable and unambiguous. A sample defect report is given in Figure 8.56. This report identifies the problem occurred in testing effort and provides candidate solutions, as many as practical to that problem. It provides an indication to the management and development staff of the severity of the problem. After specifying the problems and their solutions task continues by giving a feedback on the current perceived quality in the software product. An assessment is made to identify the areas that have not yet been addressed in terms of quality risk. At the end of the task a Test Evaluation Summary (see Appendix C.I.36) is created that organizes and represents a summary analysis of the Test Results and key measures of test for review and assessment. In the Test Evaluation Summary document a summary assessment of the test coverage analysis is performed. To evaluate test execution coverage, Test Logs are reviewed. So the ratio between how many test cases has been performed in this test cycle and a total number of tests for all intended target test items, and the ratio of successfully performed test cases are determined. Also the defect sources and their status are reported. The requirements-based test coverage is reported in the Test Evaluation Summary as shown in the Figure 8.57.

Defect Report for <Project Name>		**Created on:** <dd/mm/yy>
Identification		
Title:	**Priority:**	**Status:**
	Submitted on:	
	Defect Report ID: <>	
Submitter:	**Type:** <>	
Current Defect		
Description:	**Critical Failure:**	
	Nuisance:	
	Source of the Problem:	
Observation conditions:		
Proposed Change (Submitter)		
Description:		
Proposed Change (Review Team)		
Approval:	**Reviewed Description:**	
Affected Configuration Items	**Category**	**Error Fix**
Resolution		
Affected number lines of code:		
Estimated effort (staff hours):		
Assessment		
Test Methods:		
Test Cases:		

Figure 8.56 Defect Report



3. Requirements-based Test Coverage

The tests to be performed on the System Access are defined in the appropriate documents. The test coverage results are as follows:

Module	Performed Test Cases	Successful Test Cases	Failed Test Cases
Login	7	7	0
Logout	3	2	1
Total	10	9	1

Ratio Test Cases Performed = 10/10 = 100%
Ratio Test Cases Successful = 9/10 = 90%

The area of tests with the highest failure rate was:

- Load tests involving access to the e-Reserve System.

Figure 8.57 Requirements-based Test Coverage for Iteration E1

Achieve Acceptable Mission sub-activity of Figure 8.54 delivers a useful evaluation result of the test efforts to the stakeholders. It consists of one major task as shown in the Figure 8.58.

Figure 8.58 Achieve Acceptable Mission Task

Assess and Advocate Quality task identifies quality gaps, assesses their risks and finds acceptable solutions. Test Evaluation Summary is examined to perform this task efficiently. This step deals with assessing the software quality by gathering information. Test Results are examined based on the Test Evaluation Summary. Also the Change Requests are examined to gain more information about the possible risks and their solutions. Each gap in quality is identified and the associated impact and risk of each issue is assessed that creates the gap. Potential mitigation and contingency strategies are considered for each gap. The initial findings are formulated to discuss them with the team members. It is an important work to validate performed thoughts. Work priority is negotiated to advocate for an appropriate solution that does not reduce the quality of the product. Monitoring the work progress is an important issue in this task that provides to remain supportive on the resolution of the issue. At the end of the current task the resolutions for key issues are confirmed that should improve the quality.

Improve Test Assets sub-activity of Figure 8.54 is used to maintain and improve test assets. It consists of one major task as shown in the Figure 8.59.

Figure 8.59 Improve Test Assets Task

Define Test Approach task defines the test strategies. After performing the desired tests, this task is handled to improve test assets by examining the Test Strategy document that is previously created in the Iteration I1. All test motivators are examined and for each of them it is considered that what test approach and associated techniques might be required to address them. Completeness of the test approaches is considered. After completing all tests, existing test techniques are identified to improve test approaches. In the case of insufficient existing test techniques, new test techniques could be identified briefly.

Recall that the last activity of Figure 8.2 was **Plan for Next Iteration** activity that guides project team to the next iteration which is Elaboration Iteration E2. It consists of two major tasks as shown in the Figure 8.60.

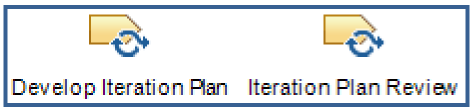


Figure 8.60 Plan for Next Iteration Tasks

Develop Iteration Plan task composes an iteration plan. The scope of the next iteration is determined. In the elaboration iterations scope is defined by risks, criticality, and coverage. Risks have to be mitigated as early as possible. For criticality, project manager have to be sure that the most fundamental function or services provided by the system are included. For coverage, project manager have to be sure that the architecture addresses all aspects of the software to be developed. The iteration evaluation criteria for Iteration E2 are defined. In elaboration phase we are focusing on creating a stable architecture so the evaluation criteria are focused on assessing the stability of the architecture. Finally Elaboration Iteration E2 activities are defined based on the goals. At the end of the task an Iteration Plan (see Appendix C.I.38) is created for the Elaboration Iteration E2. The iteration overview for this iteration is depicted in the Table 8.9.

Table 8.9 Iteration E2 Overview

Phase	**Iteration**	**Description**	**Risks Addressed**
Elaboration Phase	E2 Iteration **Architectural Prototype for System User Connectivity and Access**	• **Train the team on Web Technologies.** • **Complete analysis & design for high risk requirements** - Create Use Case Specification for each of the *Insert User, Update User, Delete User* use cases, derive analysis elements and refine the Design Model. - Refine the architecture (high-level design) in the Software Architecture Document. • **Refine the architectural prototype for system user connectivity and access, so it establishes the connectivity between ERIS and System Users** - Code the elements related to the *Insert User, Update User, Delete User* use cases. • **Demonstrate feasibility through testing (integrate as necessary)**	Risks of low skills related to Web technologies and unknown technology mitigated. Architectural issues related to Web technologies partially clarified. Technical risks related to Web technologies partially mitigated.

Iteration Plan Review task determines to approve the proposed work plan for the current iteration or not. It is held after the current iteration has been developed. For this review operation again a meeting is planned and all materials about the activity are distributed across related team members to perform review. It is important to provide sufficient lead time to allow the participants to review the project materials that will be used as the basis for the approval decision. There is a consideration with this task is that at the end of the review we determine to begin next iteration or not. So the Review Record for the current task has to be created carefully by capturing any important discussions or action items, and recording the result of the Iteration Plan Review.

After we have performed the Iteration Plan Review task of Elaboration Iteration E1 we decided to approve proposed work plan for the current iteration which means that we have completed work products of the current iteration successfully. Elaboration Phase Iteration E1 work products are tabulated in Table 8.10.

Table 8.10 Elaboration Iteration E1 Work Products (APPENDIX C.I)

eReserve_SoftwareDevelopmentPlan_2.0
eReserve_IntegrationBuildPlan_1.0
eReserve_WorkOrder_2.0
eReserve_RiskList_2.0
eReserve_IterationAssessment_2.0
eReserve_StatusAssessment_2.0
eReserve_UseCaseModel_2.0
eReserve_Glossary_2.0
eReserve_ChangeRequestCR_01_1.0
eReserve_DevelopmentInfrastructure_1.0
eReserve_UseCaseModel_2.1
eReserve_SoftwareRequirementsSpecifications_2.0
eReserve_SupplementarySpecification_2.0
eReserve_SoftwareArchitectureDocument_2.0
eReserve_AnalysisModel_1.0
eReserve_UseCaseRealizationSpecification_1.0
eReserve_DesignModel_1.0
eReserve_DesignModel_1.1
eReserve_DesignModel_1.2
eReserve_SoftwareArchitectureDocument_2.1
eReserve_ImplementationModel_1.0
eReserve_SoftwareArchitectureDocument_2.2
eReserve_UseCaseRealizationSpecification_1.1
eReserve_NavigationMap_1.0
eReserve_UserInterfacePrototype_1.0
eReserve_DataModel_1.0
eReserve_IntegrationBuildPlan_1.1
eReserve_TestSuite_1.0
eReserve_DeveloperTest_1.0
eReserve_TestLog_1.0
eReserve_TestCase_1.0
eReserve_TestSuite_1.1
eReserve_TestLog_1.1
eReserve_TestResults_1.0
eReserve_DefectReportDF_01_1.0
eReserve_TestEvaluationSummary_1.0
eReserve_ProjectPhasePlan_2.0
eReserve_IterationPlanE2_1.0
eReserve_Build_1.0

8.2 Elaboration Iteration E2

We have successfully completed Iteration E1 of the Elaboration Phase. Now we are ready to begin development of Elaboration Iteration E2 that is the final iteration of the Elaboration Phase. In this iteration our purpose is to complete analysis and design for all remaining high risk requirements of the e-Reserve project which is the User Operations. The activities performed in Elaboration Iteration E2 of the Elaboration Phase are the same as done in the Elaboration Iteration E1 (see Figure 8.2 on page 115). Tasks for all activities in Elaboration Iteration E2 are also same as in Elaboration Iteration E1 and can be found in the previous section. It is depicted in the Iteration Plan (see Appendix C.I.38) that is created for the Elaboration Iteration E2 at the end of the previous iteration. This Iteration Plan defines the activities and tasks of the current iteration briefly within a schedule.

In the previous iteration, we implemented the system access capability of the e-Reserve System. Now it makes sense to give system the capability of user operations that includes the use cases are implemented within Elaboration Iteration E2 as follows:

- Insert User
- Update User
- Delete User

They are the core use cases and have a great importance for further operations that will be implemented in construction phase iterations. So by implementing these use cases early in the project, the risks would be mitigated by the testing that would occur during the rest of the project.

Some of the work products that were previously created in Elaboration Iteration E1 are updated with additional requirements and changes. Remaining work products are created newly to suit the specific objectives of Elaboration Iteration E2 that are listed in the Iteration Plan for the Elaboration Iteration E2. Complete set of work products that are created and updated within the Elaboration Iteration E2 can be found in Appendix C.II.

The current iteration of the Elaboration Phase again includes the analysis of the problem domain. As mentioned before, analysis means finding the right things to do, after it is clarified, the probability that the system fulfills its goals is increased and the amount of just-in-case programming minimized. Creation of the Use Case Model is one part of this analysis which includes the uses cases and their flow of events. So the flow of events for the related use cases are added in Use Case Model (see Appendix C.II.8) within the current iteration. As an example Table 8.11 depicts the flow of events created for the use case "Insert User".

Table 8.11 Flow of Events for Insert User

Use Case Name	**Insert User**	
Actor	Librarian	
Description	Librarian adds new user to the system.	
Preconditions	Login must be satisfied and Librarian must have enough permission.	
Post conditions	User successfully created.	
Priority	Low	
Normal Course	Insert User	
	Actor Actions 1) - 3) - 5) Fill the form and submit. 7) -	**System Responses** 2) Checks permission of Librarian. 4) If Librarian has enough permission then display page. If Librarian has not enough permission then go to alternative course 1. 6) Create user and integrate it with system. 8) Give message to Librarian that user added to system successfully.
Alternative Course 1		
	Actor Actions 1) -	**System Responses** 2) Give message that you have not enough permission to the Librarian.
Exceptions		
	Actor Actions	**System Responses**
Includes	Login	
Special requirements		

The iteration continues with analyzing the iteration specific use cases that are defined in the Use Case Model. Now we have to identify a preliminary mapping of required behavior onto modeling elements in the system for the Elaboration Iteration E2. So the Analysis Model is updated (see Appendix C.II.11) that was previously created during the analysis of Elaboration Iteration E1 elements. This model includes the analysis classes for the current iteration as done in the previous iteration that is depicted in the Figure 8.61. Iteration specific analysis class diagrams have an advantage that the designer will only deal with analysis classes that are related with the current iteration. It reduces complexity of the view that the designer is capable to see analysis classes created within the current iteration. Such an analysis technique, which contains iteration specific analysis class diagrams, also provides developers to concentrate on part of a whole much more easily during the development of the system for the current iteration.

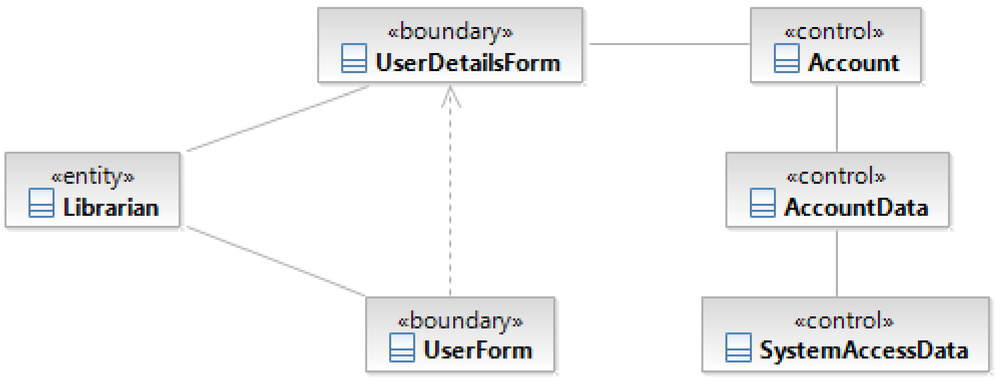


Figure 8.61 Analysis Classes for Iteration E2

However, the Analysis Model has an additional field for this iteration which includes the analysis classes for overall system (also further iterations will include) as shown in Figure 8.62. This kind of analysis classes gives the overall view of the system. It does not have an additional property that is just the union of all analysis classes of all iterations. Such an analysis class diagram helps us to view the whole structure of the system that we are modeling. They also give rise to the major abstractions of the system design. As an example Figure 8.62 shows the analysis classes for overall system that includes the analysis classes from Elaboration Iteration E1 and Elaboration Iteration E2.

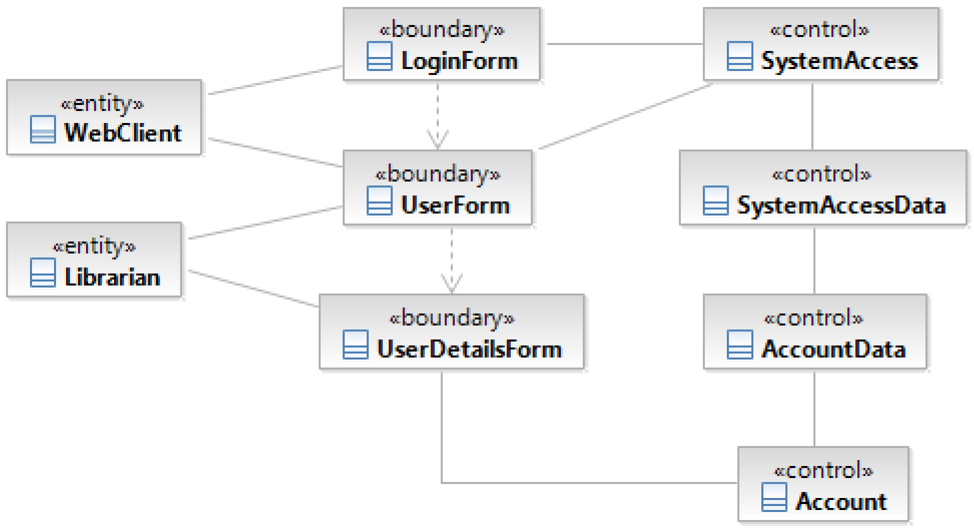


Figure 8.62 Analysis Classes for Overall System

Object design is very much about assigning responsibilities, which are basically of two types: knowing and doing. At the design the responsibility choices are usually considered in the process of creating interaction diagrams; remember that the UML has two diagram types for them: sequence diagrams and collaboration diagrams that were already created in the previous iteration of the project.

The development team then created sequence diagrams and communication diagrams based on the iteration specific use cases. Since Elaboration Iteration E2 includes very complex use cases, the sequence diagrams and communication diagrams are also more complex than the diagrams that were created in Elaboration Iteration E1. Within this iteration the development team spent quite a bit of time analyzing the user operations.

The sequence diagram is created for the Insert User use case as shown in the Figure 8.63 to understand which objects will be needed and how to interact with those objects. Then the communication diagram is created for the same use case as shown in the Figure 8.64 to understand all of the effects on a given object and for algorithm design. The Insert User sequence and communication diagrams are one set of the interaction diagrams that are to be handled for the user operations within the Elaboration Iteration E2 and the other interaction diagrams can be found in Use Case Realization Specification (see Appendix C.II.12) document.

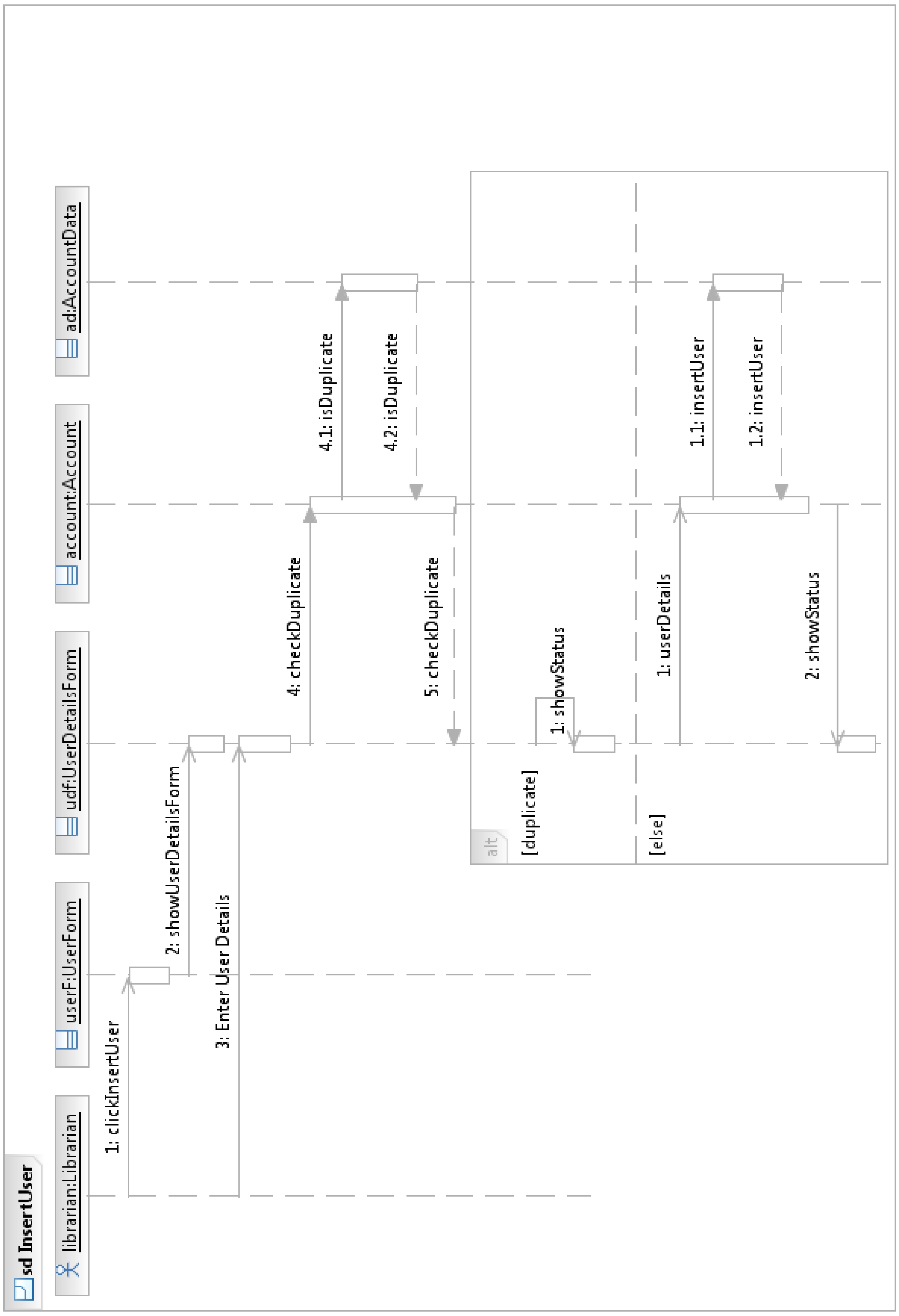


Figure 8.63 Sequence Diagram for Insert User

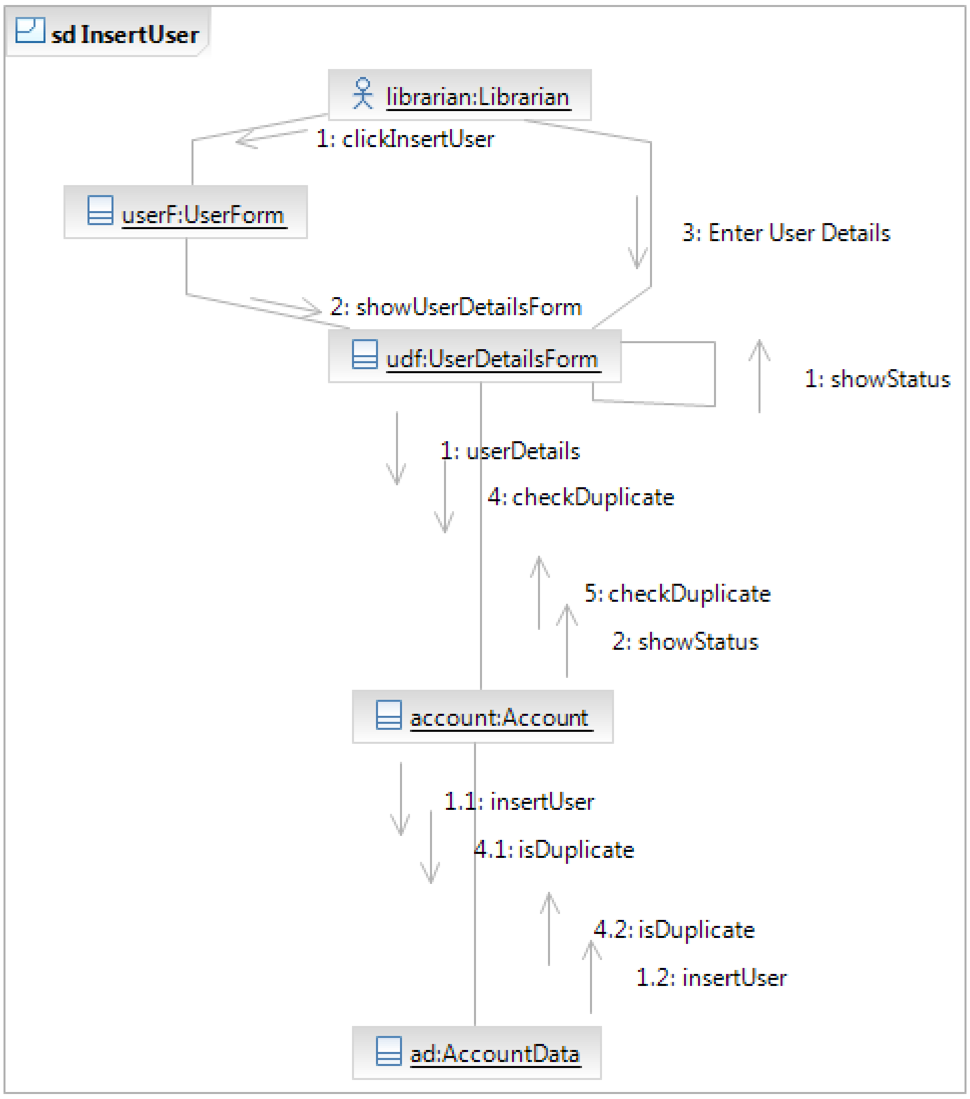


Figure 8.64 Communication Diagram for Insert User

In parallel with creation of the interaction diagrams, the design classes for the e-Reserve system started taking form. Many new classes and associations were added to the design class diagram within the Elaboration Iteration E2. After each sequence diagram was created and refined the design classes are updated, so the Design Model is updated (see Appendix C.II.14) for the Elaboration Iteration E2. This model includes the design classes for the current iteration as done in the previous iteration that is depicted in the Figure 8.65. Iteration specific design class diagrams provide the same advantage that is mentioned for the iteration specific analysis class diagrams. Also, the Design Model has an additional field for this iteration which includes the design classes for overall system (also further iterations will include) which gives the overall view of the system as shown in Figure 8.66.

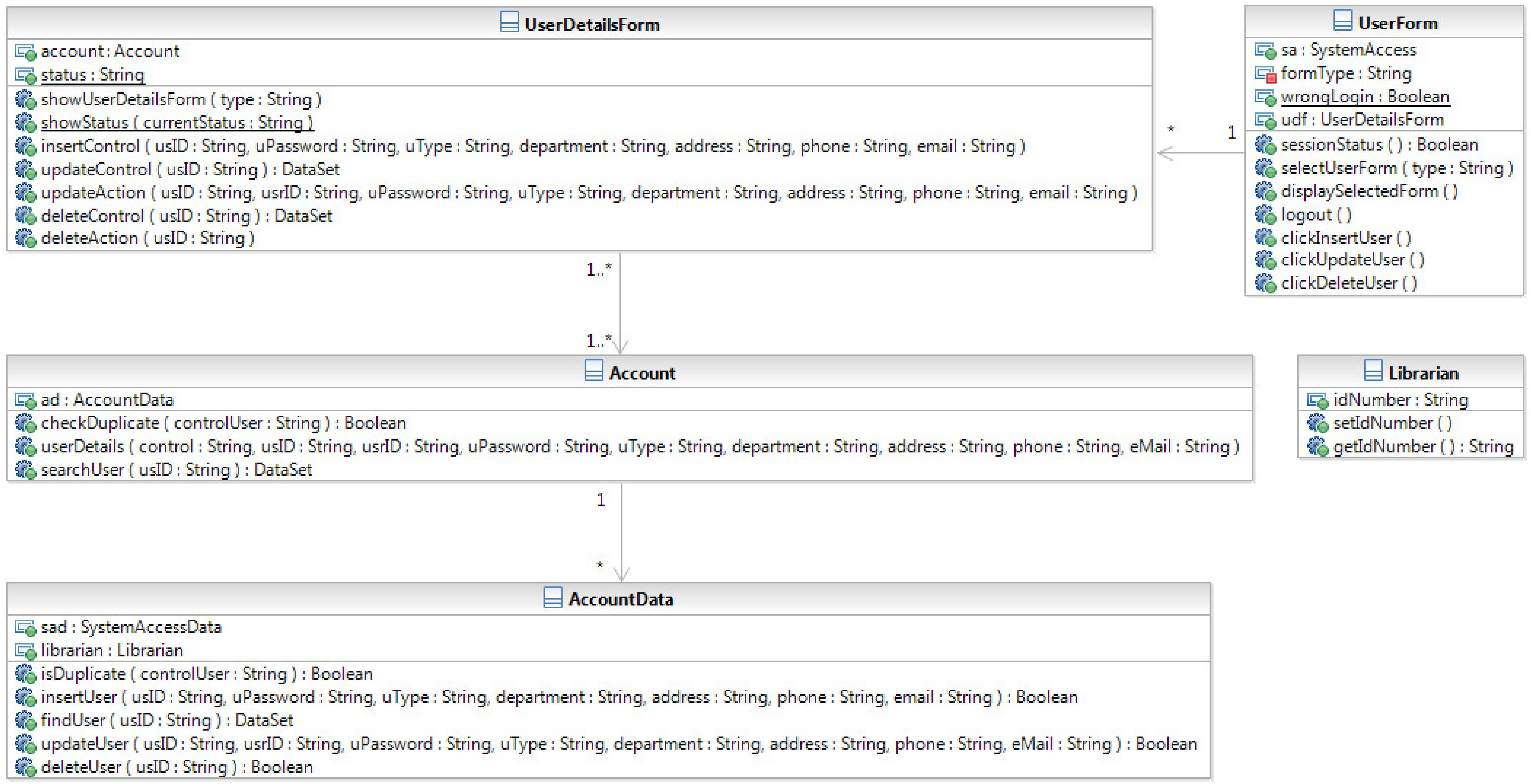

UserDetailsForm
account : Account
status : String
showUserDetailsForm (type : String)
showStatus (currentStatus : String)
insertControl (usID : String, uPassword : String, uType : String, department : String, address : String, phone : String, email : String)
updateControl (usID : String) : DataSet
updateAction (usID : String, usrID : String, uPassword : String, uType : String, department : String, address : String, phone : String, email : String)
deleteControl (usID : String) : DataSet
deleteAction (usID : String)
UserForm
sa : SystemAccess
formType : String
wrongLogin : Boolean
udf : UserDetailsForm
sessionStatus ( ) : Boolean
selectUserForm (type : String)
displaySelectedForm ( )
logout ( )
clickInsertUser ( )
clickUpdateUser ( )
clickDeleteUser ( )
*
1
1..*
1..*
Account
ad : AccountData
checkDuplicate (controlUser : String) : Boolean
userDetails (control : String, usID : String, usrID : String, uPassword : String, uType : String, department : String, address : String, phone : String, eMail : String)
searchUser (usID : String) : DataSet
Librarian
idNumber : String
setIdNumber ( )
getIdNumber ( ) : String
1
*
AccountData
sad : SystemAccessData
librarian : Librarian
isDuplicate (controlUser : String) : Boolean
insertUser (usID : String, uPassword : String, uType : String, department : String, address : String, phone : String, email : String) : Boolean
findUser (usID : String) : DataSet
updateUser (usID : String, usrID : String, uPassword : String, uType : String, department : String, address : String, phone : String, eMail : String) : Boolean
deleteUser (usID : String) : Boolean

Figure 8.65 Design Classes for Iteration E2

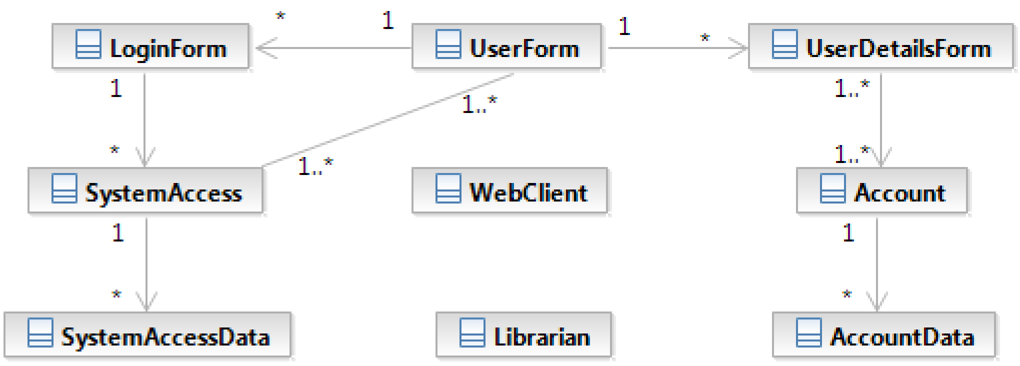


Figure 8.66 Design Classes for Overall System

The iteration continues with graphical user interface design. We describe the characteristics of the user operations for the system. By identifying primary requirements, primary windows for the system are created which are the essential windows when the user operations to be performed. Based on defined primary windows the Navigation Map (see Appendix C.II.19) is updated which was created in the previous iteration of Elaboration Phase. We know that by using the Navigation Map, users can simply figure out how many steps they require to reach their target page in the system. Potential paths that are identified for the Elaboration Iteration E2 are added into existing paths as shown in the Figure 8.67.

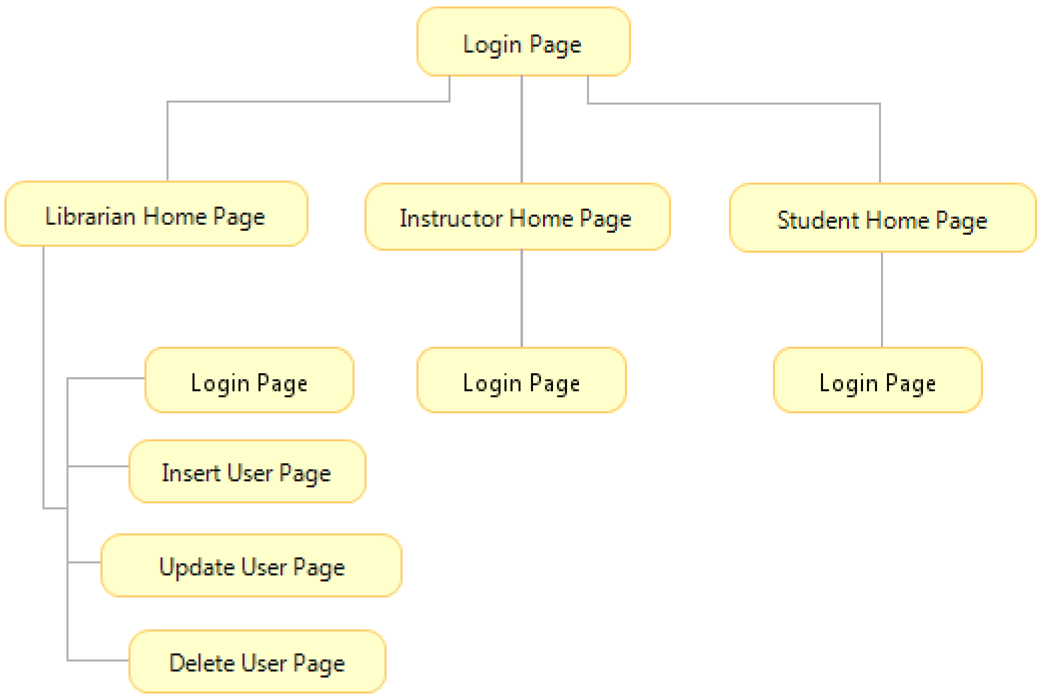
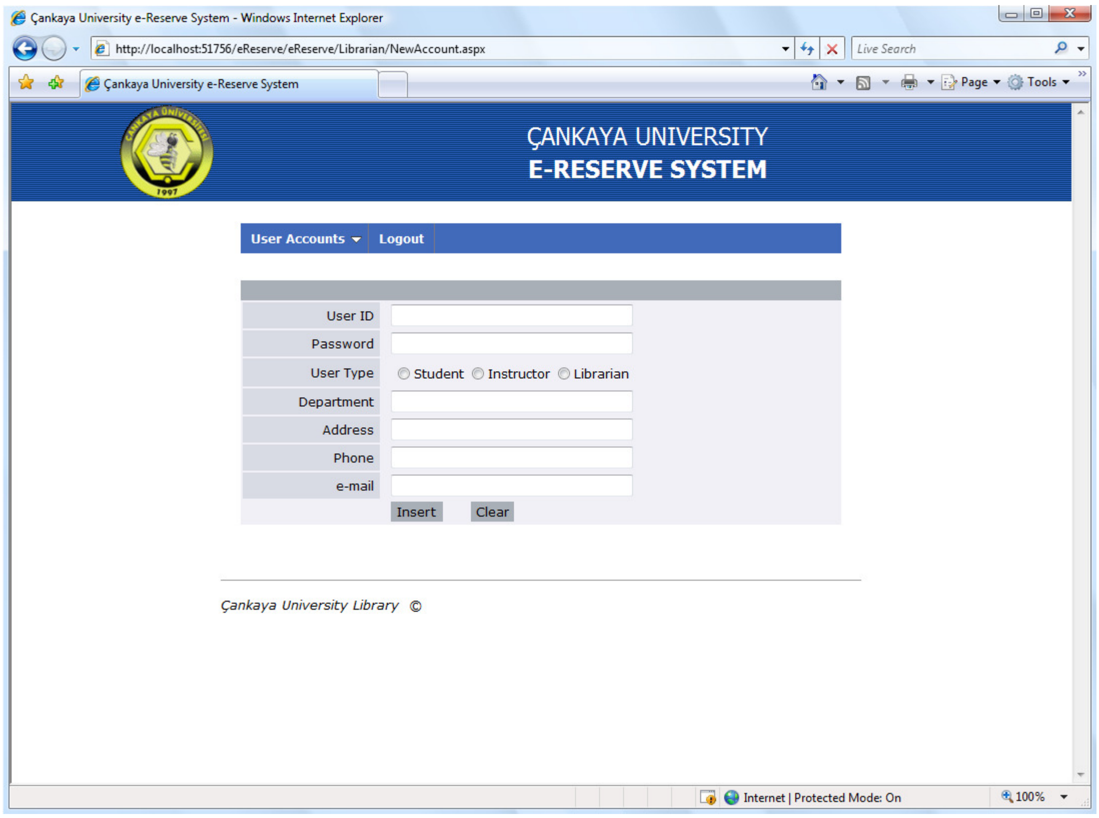


Figure 8.67 Navigation Map for Iteration E2

The Elaboration Iteration E2 activities continue with the construction of related user interface prototypes. So that we are able to test out the user-interface design, including its usability before the real development starts. Insert User page prototype is depicted in the Figure 8.68. All other user interface prototypes that are related to the Elaboration Iteration E2 can be found in the User Interface Prototype (see Appendix C.II.20) document.


Çankaya University e-Reserve System - Windows Internet Explorer
http://localhost:51756/eReserve/eReserve/Librarian/NewAccount.aspx
Live Search
Çankaya University e-Reserve System
Page
Tools
ÇANKAYA UNIVERSITY
E-RESERVE SYSTEM
User Accounts
Logout
Internet | Protected Mode: On
100%

User ID	
Password	
User Type	○ Student ○ Instructor ○ Librarian
Department	
Address	
Phone	
e-mail	

Çankaya University Library ©

Figure 8.68 User Interface Prototype for Iteration E2

Additionally, database tables and relations are formed as shown in the Figure 8.69 that is detailed in Data Model (see Appendix C.II.21) document.

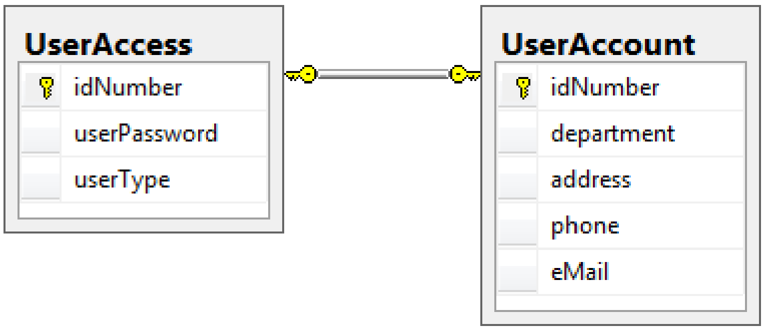


Figure 8.69 Database Tables for Iteration E2

Implementing Elaboration Iteration E2 related components takes several days that is detailed in the Iteration Plan of the Elaboration Iteration E2. These implementation activities lead team members to achieve objectives with expected results for the e-Reserve system within Elaboration Iteration E2.

In the next step, the implemented features are tested. First of all, unit tests are added in this iteration to exercise the system's user operations capability that determines whether the individual units of source code fit for use. To perform these unit tests, new test suite is created and implemented for the user operations. It is important to verify that all the unit tests are executed successfully. After this verification the log is prepared in which the results of unit tests for the user operations are presented. Now it is time to prepare test cases to determine whether e-Reserve system meets the required specifications by using the set of conditions and variables for the Elaboration Iteration E2 features. So the test suite is updated to store these test cases. After execution of test cases, the result of each test case is reported in a Test Log (see Appendix C.II.28) as shown in Table 8.12 that depicts a part of log for the test case insert user. This Test Log is a raw data that will subsequently be analyzed to help determine the results of some aspect of the test effort within the Test Results (see Appendix C.II.29) and more detailed in Test Evaluation Summary (see Appendix C.II.32). The defects that are captured during these test activities are detailed in Defect Report (see Appendix C.II.30 and C.II.31) documents.

Finally new version of build for the e-Reserve system is ready that is developed within the Elaboration Iteration E2. Now we have to prepare an Iteration Plan (see Appendix C.II.34) for the next iteration which will be the first iteration of the Construction Phase.

Again we are in a critical point that we reached to the end of Elaboration Phase. So we have to make a decision to continue on Construction Phase or go back to Elaboration Phase. This decision is made by checking the objectives of Lifecycle Architecture Milestone that whether it is satisfied or not.

Table 8.12 Test Case Result of Insert User

Date	Test Module	Test Case	Input	Expected Output	Actual Output	Recommendation
01/21/09	Insert User	Successful Insertion	Valid Page Info Valid User Type Info Valid Form Status Valid User ID Valid Password Valid User Type Valid Account Info	Message is shown. User account is inserted successfully.	Display the expected output correctly.	
		Successful Home Redirection	Valid Page Info Valid User Type Info	Selected user form is displayed.	Display the expected output correctly.	
		Invalid Home Redirection	Invalid Page Info Invalid User Type Info	Error message is displayed. Librarian page cannot be shown.	Display the expected output correctly.	
		No Menu	Valid Page Info Invalid User Type Info	Error message is displayed. Menu cannot be shown.	Display the expected output correctly.	
		Successful Inner Redirection	Valid Page Info Valid User Type Info	Selected form is displayed.	Display the expected output correctly.	
		Invalid Inner Redirection	Invalid Page Info Valid User Type Info	Error message is displayed. Insert New Account page cannot be shown.	Display the expected output correctly.	
		Invalid Form	Valid Page Info Valid User Type Info Invalid Form Status	Error message is displayed. Form for inserting a new account is not available.	Error message cannot be displayed. Form does not reloaded.	Check database connections after insertion operation.
		Incorrect Validation (No User ID)	Valid Page Info Valid User Type Info Valid Form Status No User ID Valid Password Valid User Type	Error message is displayed. Wait on insert new account page to enter User ID.	Display the expected output correctly.	

8.3 Lifecycle Architecture Milestone

Lifecycle Architecture Milestone marks the end of the Elaboration Phase as shown in Figure 8.1. It is the second important major milestone of the project that we reached at the end of second iteration of Elaboration Phase. Now we are standing at a point that we have to you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks.

Evaluation criteria for the Elaboration Phase can be listed as follows:

- The Vision and requirements of e-Reserve system are stable.
- The architecture is stable.
- The key approaches that are used in test and evaluation are proven.
- Test and evaluation of executable prototypes from the two elaboration iterations have demonstrated that the major risk elements have been addressed and have been credibly resolved.
- The Iteration Plan for the construction phase has sufficient detail and fidelity to allow the work to proceed.
- Stakeholders agree that the current vision can be met if the current plan is executed in the context of the presented architecture to develop the entire system.
- The rate of actual resource expenditure to planned expenditure is acceptable.

The project may be aborted or considerably rethought if it fails to reach this milestone same as in the previous milestone. The decision to proceed to the Construction Phase is made based on mitigating the technical risks that are identified.

Elaboration Phase Iteration E2 work products are tabulated in Table 8.13. All the work products are given in the Appendix C of the thesis as C.I.1-39 and C.II.1-35, also on CD to be reached by:

- ~/Appendices/AppendixC/ElaborationIterationE1
- ~/Appendices/AppendixC/ElaborationIterationE2

Table 8.13 Elaboration Iteration E2 Work Products (APPENDIX C.II)

eReserve_SoftwareDevelopmentPlan_3.0
eReserve_WorkOrder_3.0
eReserve_RiskList_3.0
eReserve_IterationAssessment_3.0
eReserve_StatusAssessment_3.0
eReserve_Glossary_3.0
eReserve_ChangeRequestCR_02_1.0
eReserve_UseCaseModel_3.0
eReserve_SoftwareRequirementsSpecifications_3.0
eReserve_SupplementarySpecification_3.0
eReserve_AnalysisModel_2.0
eReserve_UseCaseRealizationSpecification_2.0
eReserve_DesignModel_2.0
eReserve_DesignModel_2.1
eReserve_SoftwareArchitectureDocument_3.0
eReserve_ImplementationModel_2.0
eReserve_SoftwareArchitectureDocument_3.1
eReserve_UseCaseRealizationSpecification_2.1
eReserve_NavigationMap_2.0
eReserve_UserInterfacePrototype_2.0
eReserve_DataModel_2.0
eReserve_IntegrationBuildPlan_2.0
eReserve_TestSuite_2.0
eReserve_DeveloperTest_2.0
eReserve_TestLog_2.0
eReserve_TestCase_2.0
eReserve_TestSuite_2.1
eReserve_TestLog_2.1
eReserve_TestResults_2.0
eReserve_DefectReportDF_02_1.0
eReserve_DefectReportDF_03_1.0
eReserve_TestEvaluationSummary_2.0
eReserve_ProjectPhasePlan_3.0
eReserve_IterationPlanC1_1.0
eReserve_Build_2.0

CHAPTER 9

CONSTRUCTION PHASE

We already completed the Inception and Elaboration phases of our project successfully and ready for the next phase. The next point in delivery processes is the Construction phase that focuses on completing the analysis then, design and the implementation of the system as mentioned in Chapter 3. Construction Phase of our project consists of three iterations in which each of them includes several activities and concluded by a milestone as shown in the Figure 9.1.

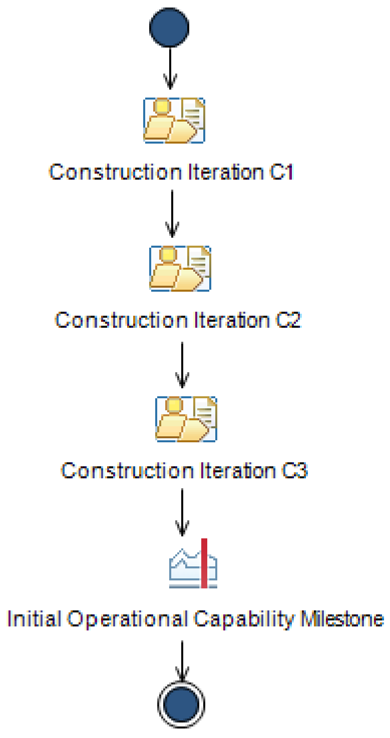
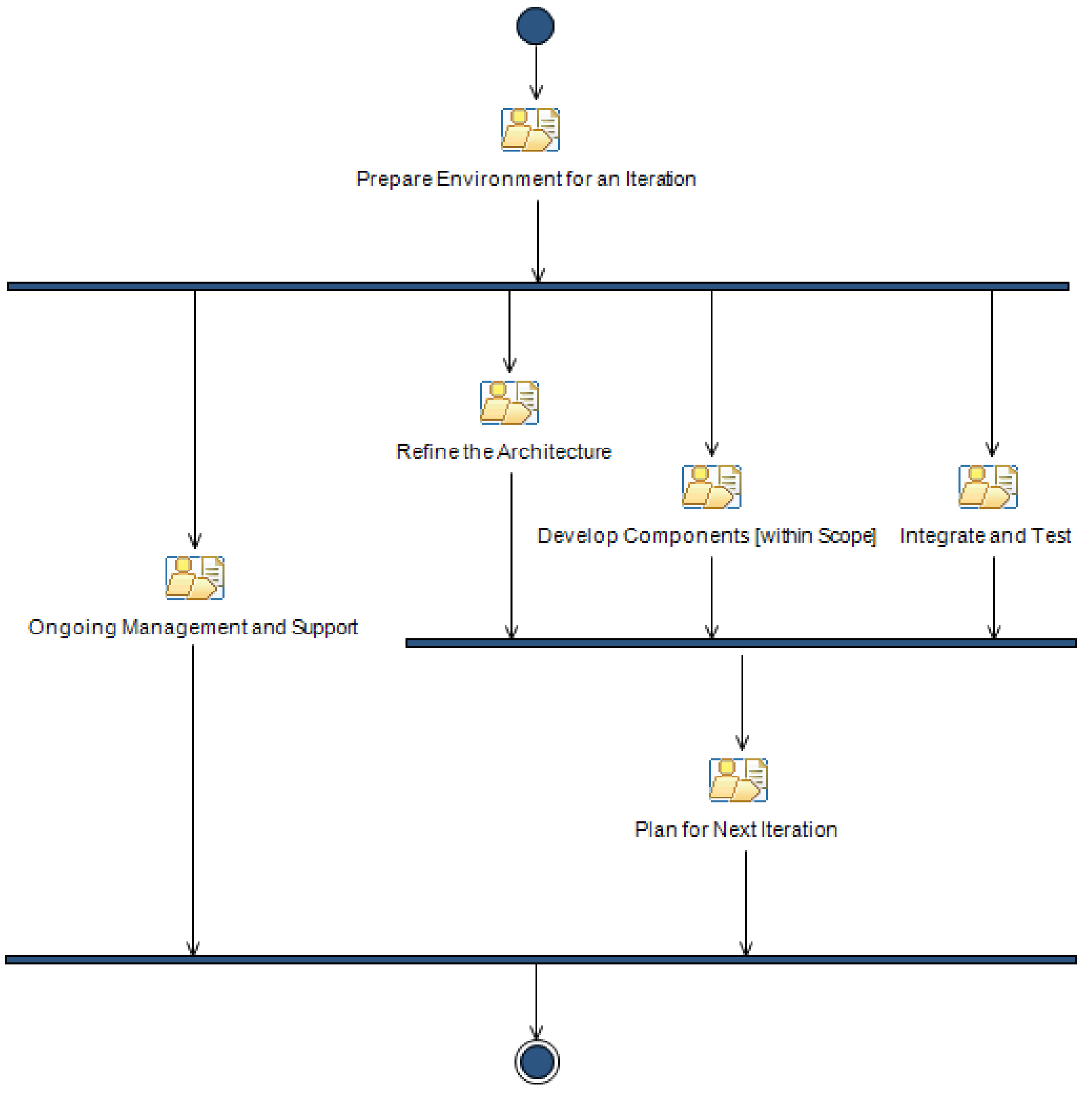

Construction Iteration C1
Construction Iteration C2
Construction Iteration C3
Initial Operational Capability Milestone

Figure 9.1 Construction Phase

9.1 Construction Iterations

The Construction phase has three iterations that are decided for e-Reserve project. The activities performed in Construction Iterations of the construction phase are shown in the Figure 9.2. The activity diagram in the Figure 9.2 and all of its activities and tasks can be obtained from RMC and modified to adapt on projects.



Prepare Environment for an Iteration
Refine the Architecture
Develop Components [within Scope]
Integrate and Test
Ongoing Management and Support
Plan for Next Iteration

Figure 9.2 Construction Phase Activity Diagram

All of the three iterations performed in Construction phases, consist of same activities and tasks with the same work breakdown structure. As we mentioned in Chapter 3, this repetition is the major functionality of iterative development methodology.

The work breakdown structure that is depicted in Figure 9.2 contains the same activities with Elaboration Phase iterations but not all of them. Within elaboration phase, we mainly focus on to baseline the architecture of the system. Also design and implementation of the system is done with a little effort. Because of this, elaboration phase includes activities that are used for implementation issues. We know that construction phase iterations focuses on design and implementation of the current system. So iterations for the construction phase have to include activities that are related with design and implementation. As a result we should use some of the activities that were already used in elaboration phase iterations which are highly related with design and implementation issues. By using RMC, we can easily see the differences and the similarities between iterations that are performed in elaboration and construction phases that are depicted in Figure 9.3.

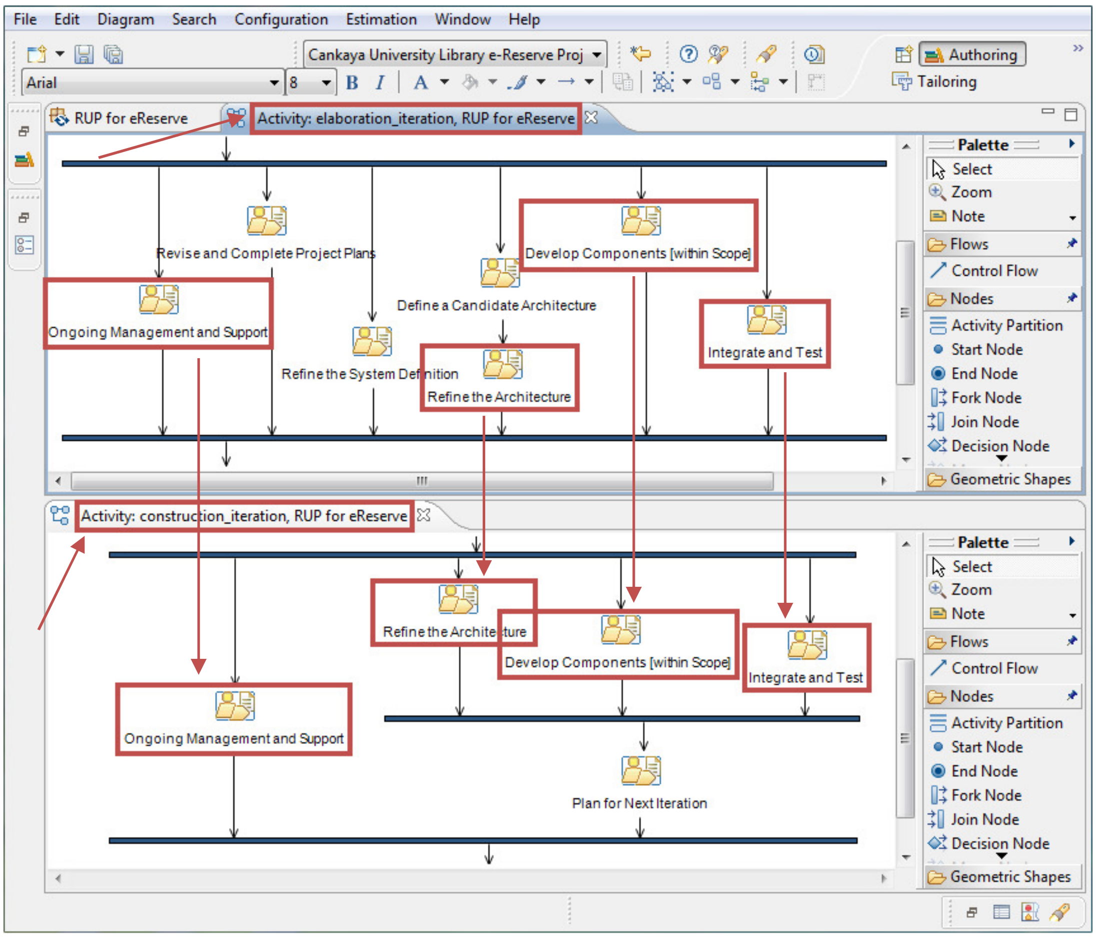

File Edit Diagram Search Configuration Estimation Window Help
Cankaya University Library e-Reserve Proj
Authoring
Tailoring
Arial
RUP for eReserve
Activity: elaboration_iteration, RUP for eReserve
Revise and Complete Project Plans
Develop Components [within Scope]
Define a Candidate Architecture
Ongoing Management and Support
Refine the System Definition
Refine the Architecture
Integrate and Test
Activity: construction_iteration, RUP for eReserve
Refine the Architecture
Develop Components [within Scope]
Integrate and Test
Ongoing Management and Support
Plan for Next Iteration
Palette
Select
Zoom
Note
Flows
Control Flow
Nodes
Activity Partition
Start Node
End Node
Fork Node
Join Node
Decision Node
Geometric Shapes

Figure 9.3 Elaboration and Construction Phase Activity Diagrams

The top window shows a part of activity diagram that represents the activities used in an iteration of elaboration phase. The bottom window shows a part of activity diagram that represents the activities used in an iteration of construction phase. As shown in the Figure 9.3 some of the activities within elaboration phase iterations are also used in the construction phase iterations. As we mentioned before, these activities are highly related with design and implementation of the system, so they are performed again in construction phase iterations with more effort.

All tasks and activities performed in each iteration are depicted briefly in iteration plans. The Iteration Plan (see Appendix C.II.34) for the Construction Iteration C1 is created at the end of Elaboration Iteration E2. The Iteration Plan (see Appendix D.I.31) for the Construction Iteration C2 is created at the end of Construction Iteration C1. Finally, the Iteration Plan (see Appendix D.II.31) for the Construction Iteration C3 is created at the end of Construction Iteration C2. All of these plans can be found in related work products.

In the previous iterations of Elaboration Phase, we implemented the system access capability and user operations of the e-Reserve System. These are the core capabilities for the system that will form a baseline for our Construction Phase. The use cases that are implemented within each iteration of Construction Phase are listed as follows.

We give system the capability of course operations that includes the use cases which are implemented within Construction Iteration C1 as follows:

- Activate Course
- Update Course
- Deactivate Course
- Search Course

We give system the capability of material operations that includes the use cases which are implemented within Construction Iteration C2 as follows:

- Insert Material
- Update Material
- Delete Material

We update the system capability of material operations by giving additional properties that includes the use cases which are implemented within Construction Iteration C3 as follows:

- View Material
- Search Material
- Download Material

Construction phase is the main development phase during which the first operational release of the product is realized. The analysis and design activities in the Elaboration phase have shown what to do and how to do it. The critical parts have also been implemented and interfaces stabilized to make sure that the design works.

The construction phase involved a continuation of design and implementation of components in the project, finalization of more predictive components such as the user interface, and the identification of limitations within the system. The iterations of construction phase are focused on implementation of features. During these iterations all high priority and major features were implemented and tested. In the last iteration of the construction phase development was completed for the overall e-Reserve system.

As we mentioned before, the primary goal of Construction Phase is the design and implementation of components for e-Reserve System. Within each iteration of Construction Phase, firstly all iteration specific components are designed in details. As an example, in Construction Iteration C1, while we are studying on Activate Course use case the design element named as CourseData is produced as shown in Figure 9.4. The design element CourseData can also be found in Design Model (see Appendix D.I.11) in more details.

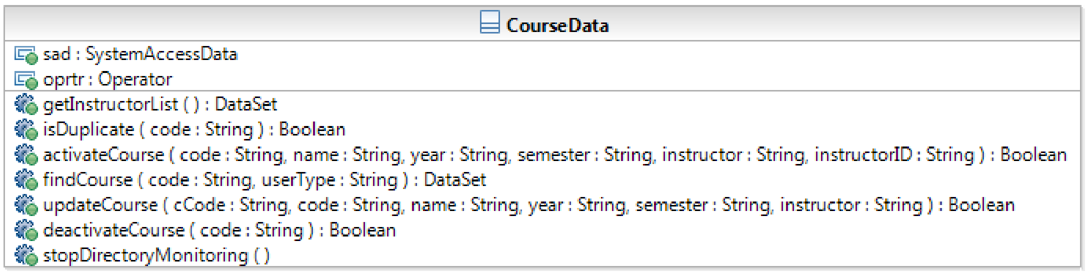
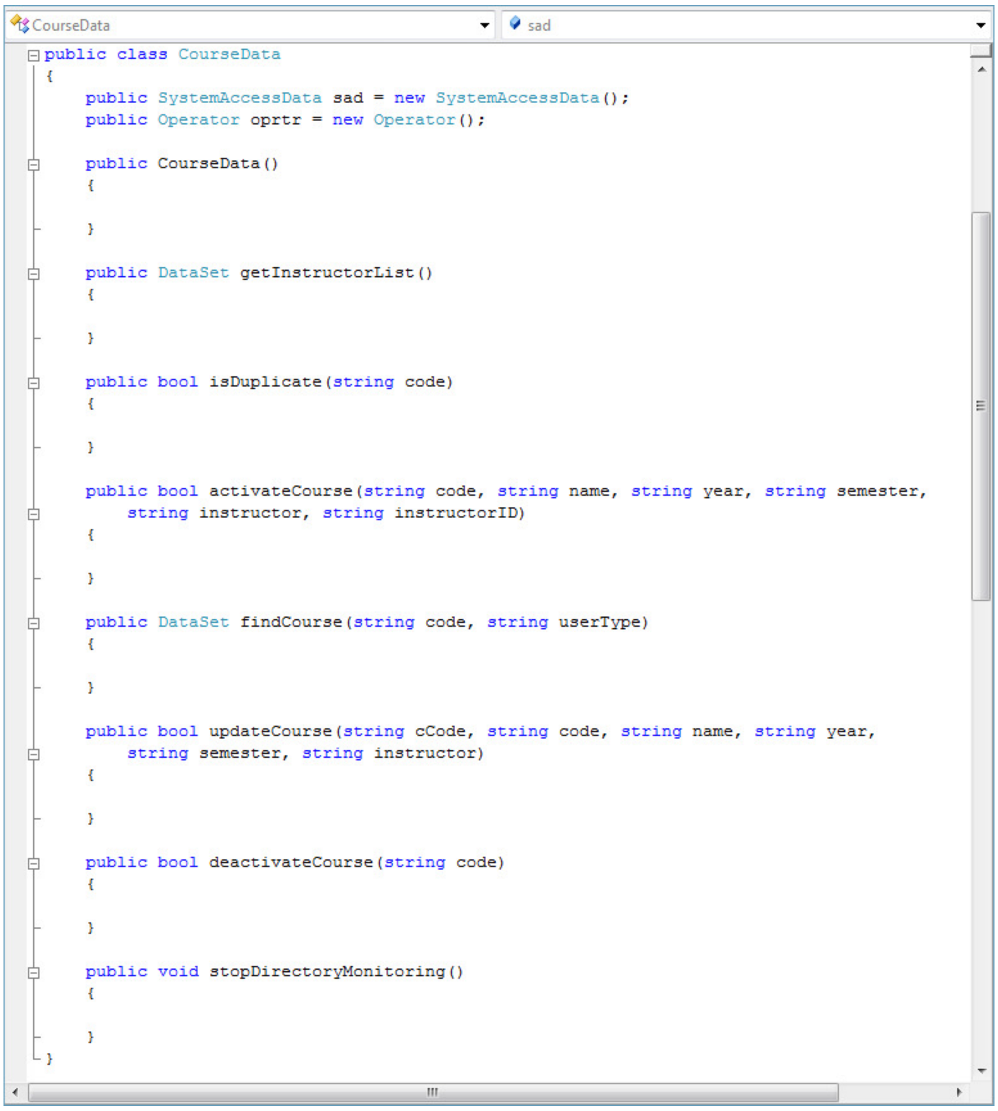

```
CourseData
sad : SystemAccessData
oprtr : Operator
getInstructorList ( ) : DataSet
isDuplicate ( code : String ) : Boolean
activateCourse ( code : String, name : String, year : String, semester : String, instructor : String, instructorID : String ) : Boolean
findCourse ( code : String, userType : String ) : DataSet
updateCourse ( cCode : String, code : String, name : String, year : String, semester : String, instructor : String ) : Boolean
deactivateCourse ( code : String ) : Boolean
stopDirectoryMonitoring ( )
```

Figure 9.4 Design Class of CourseData

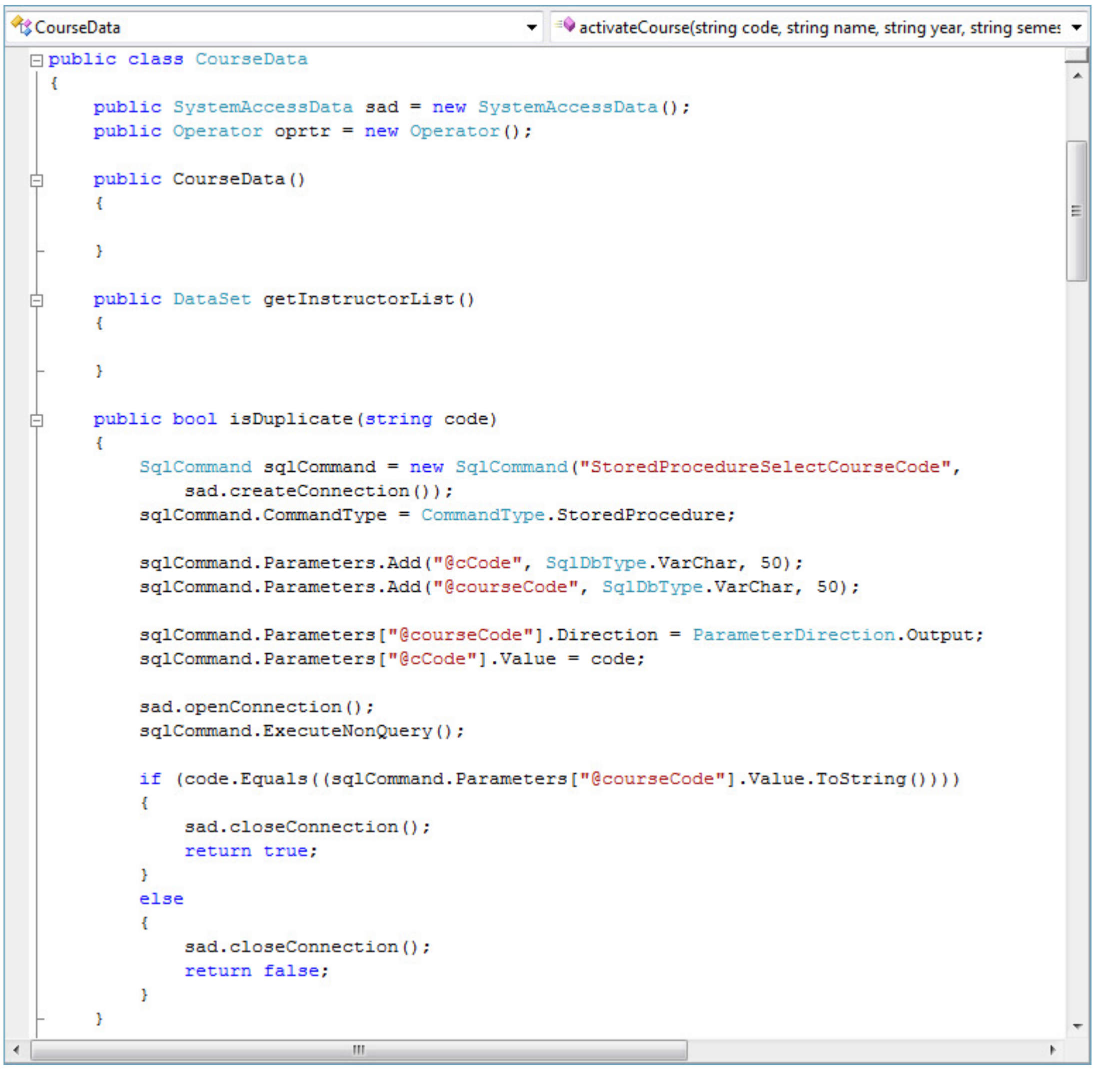
The most important activity of Construction Phase iterations is implementation of the components that are designed within these iterations. Implementation has a great effort within this phase. Implementation of iteration specific components is performed in a thoughtful manner by satisfying all requirements that are captured during the design activities. In the previous example we examined the design of CourseData element during Construction Iteration C1. Now, the CourseData element is implemented as shown in Figure 9.5 by adhering to restrictions of the design that are previously defined as depicted in Figure 9.4. All attributes and methods for the CourseData class are clearly implemented as shown in Figure 9.5. Now the system is ready to be programmed for the desired activities that are previously defined in Activate Course use case.



```
public class CourseData
{
    public SystemAccessData sad = new SystemAccessData();
    public Operator oprtr = new Operator();

    public CourseData()
    {

    }

    public DataSet getInstructorList()
    {

    }

    public bool isDuplicate(string code)
    {

    }

    public bool activateCourse(string code, string name, string year, string semester,
        string instructor, string instructorID)
    {

    }

    public DataSet findCourse(string code, string userType)
    {

    }

    public bool updateCourse(string cCode, string code, string name, string year,
        string semester, string instructor)
    {

    }

    public bool deactivateCourse(string code)
    {

    }

    public void stopDirectoryMonitoring()
    {

    }
}
```

Figure 9.5 CourseData Class Implementation

As we mentioned before, the system is designed using Design Classes and in the next step all of these design issues are satisfied during the implementation of classes. All attributes and methods of our CourseData class are shown in Figure 9.5. All of these methods have empty bodies that are ready to provide the required functionality of the system. So all method bodies are completed based on the required functionality that is necessary to satisfy the system integrity. As an example the first method in CourseData class is the isDuplicate method. This method provides one of the control mechanisms of the system that checks for the duplicate insertions on accounts. This method is firstly created by the help of its related Design Class with an empty body. Finally, the method implementation is completed by providing its behavior to perform successfully.


CourseData
activateCourse(string code, string name, string year, string seme:

```
public class CourseData
{
    public SystemAccessData sad = new SystemAccessData();
    public Operator oprtr = new Operator();

    public CourseData()
    {

    }

    public DataSet getInstructorList()
    {

    }

    public bool isDuplicate(string code)
    {
        SqlCommand sqlCommand = new SqlCommand("StoredProcedureSelectCourseCode",
            sad.createConnection());
        sqlCommand.CommandType = CommandType.StoredProcedure;

        sqlCommand.Parameters.Add("@cCode", SqlDbType.VarChar, 50);
        sqlCommand.Parameters.Add("@courseCode", SqlDbType.VarChar, 50);

        sqlCommand.Parameters["@courseCode"].Direction = ParameterDirection.Output;
        sqlCommand.Parameters["@cCode"].Value = code;

        sad.openConnection();
        sqlCommand.ExecuteNonQuery();

        if (code.Equals((sqlCommand.Parameters["@courseCode"].Value.ToString())))
        {
            sad.closeConnection();
            return true;
        }
        else
        {
            sad.closeConnection();
            return false;
        }
    }
```

Figure 9.6 CourseData Method Implementation

Again some of the artifacts from previous iterations are updated with changes implemented during iterations. Remaining work products are created newly to suit the specific objectives of each iteration of construction phases that are listed in the iteration plans. Complete set of work products that are created and updated within the Construction Iteration C1 can be found in Appendix D.I, work products for Construction Iteration C2 can be found in Appendix D.II, and work products for Construction Iteration C3 can be found in Appendix D.III.

Again we are in a critical point that we reached to the end of Construction Phase. The finalization of the construction phase marked the completion of all major components and modules of the e-Reserve system. At the conclusion of the Construction phase, we expect the product is suitable for beta testing or end user testing. We have to make a decision to continue on Transition Phase or postpone it. This decision is made by checking the objectives of Initial Operational Capability Milestone that whether it is satisfied or not.

9.2 Initial Operational Capability Milestone

As it is shown in Figure 9.1, Initial Operational Capability Milestone marks the end of the Construction Phase. It is the third important major milestone of the project that we reached at the end of third iteration of Construction Phase. Now we are standing on a point that e-Reserve system is ready to be handed over to the Transition Phase. All functionality has been developed and all alpha testing has been completed.

Evaluation criteria for the Construction Phase involves the answers of the questions listed as follows:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all the stakeholders ready for the transition into the user community?
- Are actual resource expenditures versus planned still acceptable?

If the project fails to reach this milestone for any reason then, move to the Transition Phase may have to be postponed by one release.

Construction Phase Iteration C1, C2 and C3 work products are tabulated in Table 9.1, Table 9.2 and Table 9.3 respectively. All the work products are given in the Appendix D of the thesis as D.I.1-32, D.II.1-32 and D.III.1-32, also on CD to be reached by:

- ~/Appendices/ AppendixD/ConstructionIterationC1
- ~/Appendices/ AppendixD/ConstructionIterationC2
- ~/Appendices/ AppendixD/ConstructionIterationC3

Table 9.1 Construction Iteration C1 Work Products (APPENDIX D.I)

eReserve_WorkOrder_4.0
eReserve_RiskList_4.0
eReserve_IterationAssessment_4.0
eReserve_StatusAssessment_4.0
eReserve_Glossary_4.0
eReserve_ChangeRequestCR_03_1.0
eReserve_UseCaseModel_4.0
eReserve_SoftwareRequirementsSpecifications_4.0
eReserve_AnalysisModel_3.0
eReserve_UseCaseRealizationSpecification_3.0
eReserve_DesignModel_3.0
eReserve_DesignModel_3.1
eReserve_SoftwareArchitectureDocument_4.0
eReserve_ImplementationModel_3.0
eReserve_SoftwareArchitectureDocument_4.1
eReserve_UseCaseRealizationSpecification_3.1
eReserve_NavigationMap_3.0
eReserve_UserInterfacePrototype_3.0
eReserve_DataModel_3.0
eReserve_IntegrationBuildPlan_3.0
eReserve_TestSuite_3.0
eReserve_DeveloperTest_3.0
eReserve_TestLog_3.0
eReserve_TestCase_3.0
eReserve_TestSuite_3.1
eReserve_TestLog_3.1
eReserve_TestResults_3.0
eReserve_DefectReportDF_04_1.0
eReserve_TestEvaluationSummary_3.0
eReserve_ProjectPhasePlan_4.0
eReserve_IterationPlanC2_1.0
eReserve_Build_3.0

Table 9.2 Construction Iteration C2 Work Products (APPENDIX D.II)

eReserve_WorkOrder_5.0
eReserve_RiskList_5.0
eReserve_IterationAssessment_5.0
eReserve_StatusAssessment_5.0
eReserve_Glossary_5.0
eReserve_ChangeRequestCR_04_1.0
eReserve_UseCaseModel_5.0
eReserve_SoftwareRequirementsSpecifications_5.0
eReserve_AnalysisModel_4.0
eReserve_UseCaseRealizationSpecification_4.0
eReserve_DesignModel_4.0
eReserve_DesignModel_4.1
eReserve_SoftwareArchitectureDocument_5.0
eReserve_ImplementationModel_4.0
eReserve_SoftwareArchitectureDocument_5.1
eReserve_UseCaseRealizationSpecification_4.1
eReserve_NavigationMap_4.0
eReserve_UserInterfacePrototype_4.0
eReserve_DataModel_4.0
eReserve_IntegrationBuildPlan_4.0
eReserve_TestSuite_4.0
eReserve_DeveloperTest_4.0
eReserve_TestLog_4.0
eReserve_TestCase_4.0
eReserve_TestSuite_4.1
eReserve_TestLog_4.1
eReserve_TestResults_4.0
eReserve_DefectReportDF_05_1.0
eReserve_TestEvaluationSummary_4.0
eReserve_ProjectPhasePlan_5.0
eReserve_IterationPlanC3_1.0
eReserve_Build_4.0

Table 9.3 Construction Iteration C3 Work Products (APPENDIX D.III)

eReserve_WorkOrder_6.0
eReserve_RiskList_6.0
eReserve_IterationAssessment_6.0
eReserve_StatusAssessment_6.0
eReserve_Glossary_6.0
eReserve_ChangeRequestCR_05_1.0
eReserve_UseCaseModel_6.0
eReserve_SoftwareRequirementsSpecifications_6.0
eReserve_AnalysisModel_5.0
eReserve_UseCaseRealizationSpecification_5.0
eReserve_DesignModel_5.0
eReserve_DesignModel_5.1
eReserve_SoftwareArchitectureDocument_6.0
eReserve_ImplementationModel_5.0
eReserve_SoftwareArchitectureDocument_6.1
eReserve_UseCaseRealizationSpecification_5.1
eReserve_NavigationMap_5.0
eReserve_UserInterfacePrototype_5.0
eReserve_DataModel_5.0
eReserve_IntegrationBuildPlan_5.0
eReserve_TestSuite_5.0
eReserve_DeveloperTest_5.0
eReserve_TestLog_5.0
eReserve_TestCase_5.0
eReserve_TestSuite_5.1
eReserve_TestLog_5.1
eReserve_TestResults_5.0
eReserve_DefectReportDF_06_1.0
eReserve_TestEvaluationSummary_5.0
eReserve_ProjectPhasePlan_6.0
eReserve_IterationPlanT1_1.0
eReserve_Build_5.0

CHAPTER 10

TRANSITION PHASE

We completed the Construction phase of our project successfully and ready for the next and the last phase of our project. The next point in delivery processes is the Transition phase that focuses on beta testing and deployment as mentioned in Chapter 3. Transition Phase of our project will consist of two iterations in which each of them includes several activities and concluded by a milestone as shown in the Figure 10.1.

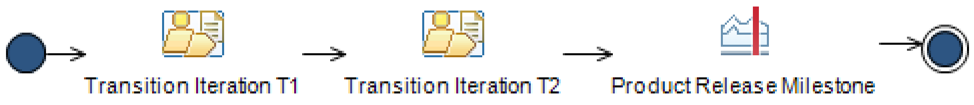


Figure 10.1 Transition Phase

10.1 Transition Iterations

The Transition phase has two major iterations that are decided for e-Reserve project. The activities performed in Transition Iterations of the transition phase are shown in the Figure 10.2. The activity diagram in the Figure 10.2 and all of its activities and tasks can be obtained from RMC and modified to adapt on any type of project.

All of the two iterations performed in Transition phases, consist of same activities and tasks with the same work breakdown structure as already done in the previous phases which are Elaboration and Construction.

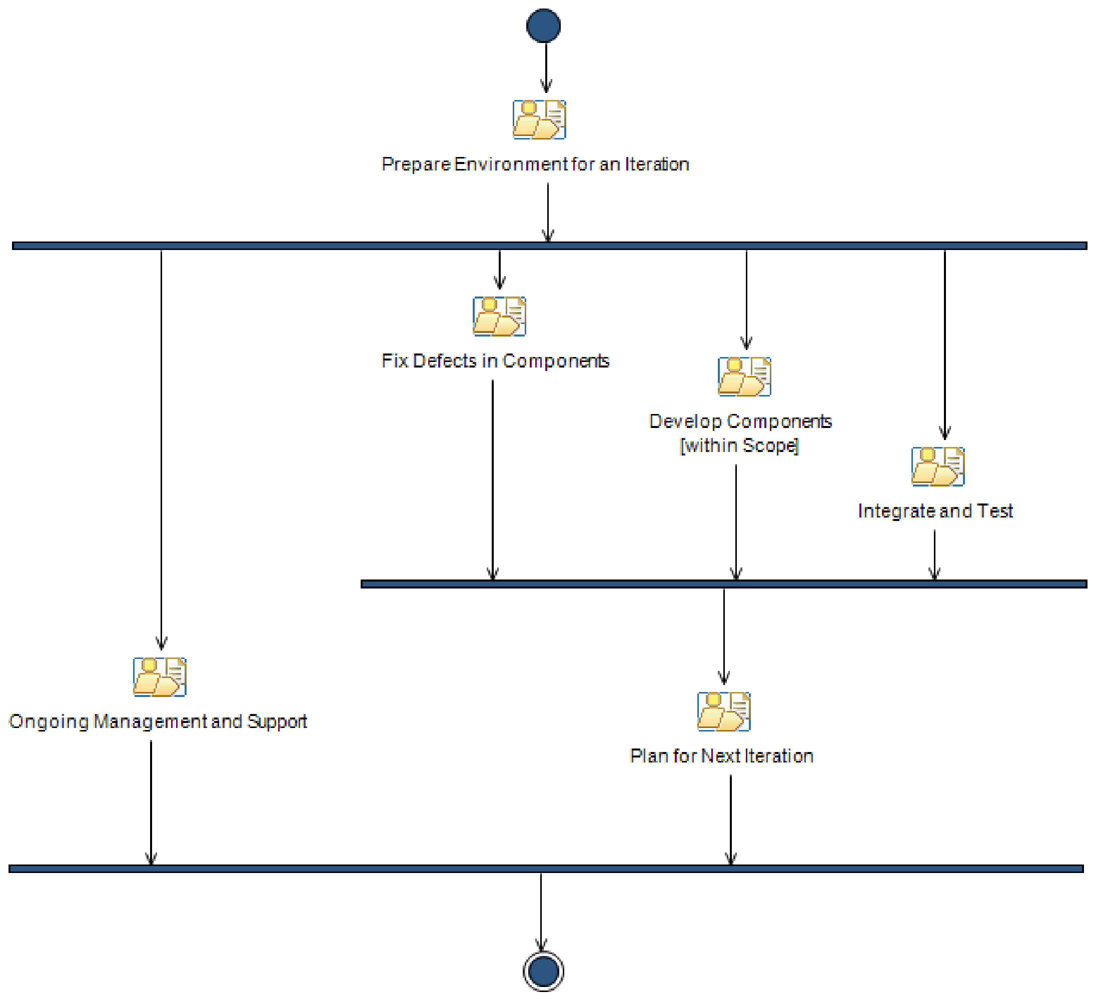

Prepare Environment for an Iteration
Fix Defects in Components
Develop Components [within Scope]
Integrate and Test
Ongoing Management and Support
Plan for Next Iteration

Figure 10.2 Transition Phase Activity Diagram

The work breakdown structure that is depicted in Figure 10.2 contains the same activities with Construction Phase iterations but not all of them. Within construction phase, we mainly focus on to design and implementation of the e-Reserve system by refining it. Also testing of the system is performed as done in elaboration phase. Testing activities has a great importance both in elaboration and construction phases. These testing activities provide a feedback to team members about the status, stability and reliability of the system that is developed up to that point.

We know that transition phase iterations focuses on testing and deployment of the current system. While performing testing activities within transition phase it is possible to capture any new defects that could not be handled before. So there must be some activities that are related to design and implementation again in iterations of transition phase to fix these captured defects. However these design and implementation activities will not be as detailed as in construction phase in which there will be only small changing, if needed. So iterations for the transition phase have to include activities that are related with design, implementation and test. As a result we should use some of the activities that were already used in construction phase iterations which are highly related with design, implementation and test issues. By using RMC, we can easily see the differences and the similarities between iterations that are performed in construction and transition phases that are depicted in Figure 10.3.

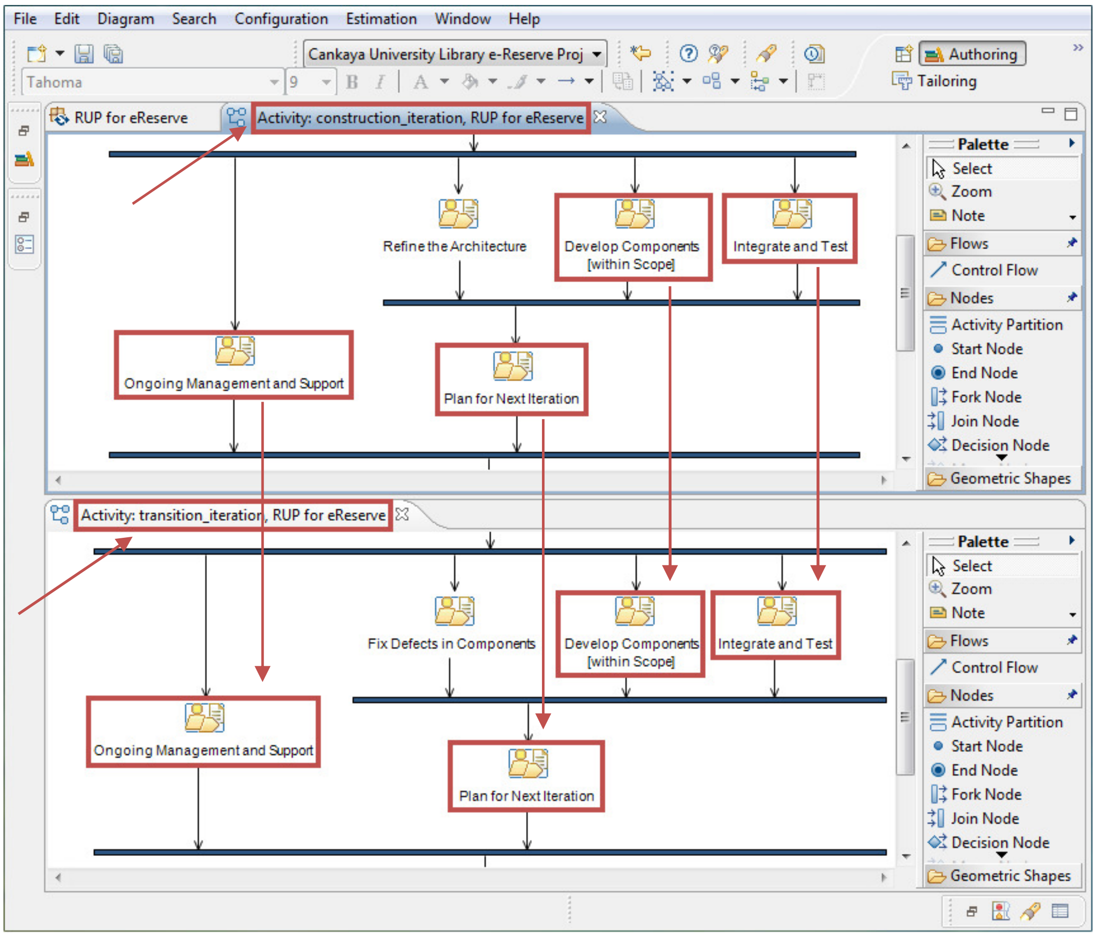


Figure 10.3 Construction and Transition Phase Activity Diagrams

The top window shows a part of activity diagram that represents the activities used in an iteration of construction phase. The bottom window shows a part of activity diagram that represents the activities used in an iteration of transition phase. As shown in the Figure 10.3 some of the activities within construction phase iterations are also used in the transition phase iterations. As we mentioned before, these activities are highly related with design, implementation and test of the system, so they are accommodated again in transition phase iterations and ready for use if needed after performing the activities that are related with fixing defects that will be captured.

As shown in Figure 10.3 the main difference between the construction phase iterations and the transition phase iterations is the activity called "Fix Defects in Components" which is placed in the iterations of transition phase. This is the key activity that forms the iterations of transition phase. Fix Defects in Components activity completes a part of the implementation so that it can be delivered for integration, if any defect is captured.

All tasks and activities performed in each iteration are depicted briefly in iteration plans. The Iteration Plan (see Appendix D.III.31) for the Transition Iteration T1 is created at the end of Construction Iteration C3. Finally, the Iteration Plan (see Appendix E.I.7) for the Transition Iteration T2 is created at the end of Transition Iteration T1. All of these plans can be found in related work products.

In the previous iterations of Elaboration and Construction phases, we implemented the system access capability, user operations, course operations, material operations, and browse operations of the e-Reserve System. These capabilities form the overall structure of the e-Reserve system by implementing the following use cases:

- Login
- Logout
- Insert User
- Update User
- Delete User
- Activate Course
- Update Course
- Deactivate Course
- Search Course
- Insert Material
- Update Material
- Delete Material
- View Material
- Search Material
- Download Material

Each iteration of the transition phase tries to improve the functionality of these use cases by resolving defects. Final release of the system includes all of these use cases that are implemented successfully without any defects at the end of the transition phase.

As we mentioned in this section, the Transition phase focuses on delivering the system into production. In other words, the principal objective of the transition phase is to integrate the product in the user's environment and correct the operational version until customers provide positive acceptance tests. The transition phase involved final user feedback and acceptance, rollout of beta testing and bug elimination. There will be testing by both system testers and end-users, and corresponding rework and fine tuning. Our e-Reserve system is a Web application for the Internet. So the beta version is tested by a group of acceptance testers before going online.

As we did in the previous phases, some of the artifacts from previous iterations are updated with changes occurred while iterations are performed. Remaining work products are created newly to suit the specific objectives of each iteration of transition phases that are listed in the iteration plans. Complete set of work products that are created and updated within the Transition Iteration T1 can be found in Appendix E.I, and work products for Transition Iteration T2 can be found in Appendix E.II.

Again we are in a critical point that we reached to the end of Transition Phase. The finalization of the transition phase marked the completion of components and modules, tests of the e-Reserve system with corrected defects. At the conclusion of the Transition phase, we expect the product ready for release. We have to make a decision to release the product or to postpone it. This decision is made by checking the objectives of Product Release Milestone that whether it is satisfied or not.

10.2 Product Release Milestone

Product Release Milestone marks the end of the Transition Phase in Figure 10.1 also the end of the project. It is the fourth important major milestone of the project that we reached at the end of second iteration of Transition Phase. Now we are standing on a point that e-Reserve system is ready to be released. All functionality has been developed and all beta testing has been completed. At this point, we have to decide if the objectives were met, and if we should start another development cycle. In some cases this milestone may coincide with the end of the inception phase for the next cycle. The Product Release Milestone is the result of the customer reviewing and accepting the project deliverables.

Evaluation criteria for the Transition Phase involves the answers of the questions listed as follows:

- Is the user satisfied?
- Are actual resources expenditures versus planned expenditures acceptable?

If the project fails to reach this milestone for any reason then, release of the product may have to be postponed.

Transition Phase Iteration T1 and T2 work products are tabulated in Table 10.1 and Table 10.2 respectively. All the work products are given in the Appendix E of the thesis as E.I.1-7 and E.II.1-5, also on CD to be reached by:

- ~/Appendices/ AppendixE/TransitionIterationT1
- ~/Appendices/ AppendixE/TransitionIterationT2

Table 10.1 Transition Iteration T1 Work Products (APPENDIX E.I)

eReserve_WorkOrder_7.0
eReserve_RiskList_7.0
eReserve_IterationAssessment_7.0
eReserve_StatusAssessment_7.0
eReserve_TestEvaluationSummary_6.0
eReserve_ProjectPhasePlan_7.0
eReserve_IterationPlanT2_1.0

Table 10.2 Transition Iteration T2 Work Products (APPENDIX E.II)

eReserve_WorkOrder_8.0
eReserve_RiskList_8.0
eReserve_IterationAssessment_8.0
eReserve_StatusAssessment_8.0
eReserve_TestEvaluationSummary_7.0

CHAPTER 11

SUMMARY AND CONCLUSIONS

11.1 Summary

This thesis has described the adoption of RUP on a software development project as a case study. Without any reliable document, developing a required system or understanding of an existing software system can be a very wearisome and expensive task. Consequently, some standard methodology should be applied on software development projects to manage them properly.

After some introductory remarks in Chapter1, in Chapter 2, we started with a literature review concerning with UML. UML plays a central role to capture requirements, to analyze and design the system. It provides the communication between the project team members and stakeholders that reduces the misunderstood issues. Today UML uses version 2.0 which is the strongest version and has more advantages than its older versions. So we concentrated on version 2.0 with its thirteen diagrams in our study.

Chapter 3 also continues with literature review concerning with RUP. There are totally nine disciplines defined in RUP, six of which are core disciplines and three core supporting disciplines. These disciplines provide us to construct and manage our project safely with high quality. All nine disciplines should be applied carefully within a project lifecycle that uses RUP to obtain a successfully developed product by the help of best practices that is suitable for a wide range of projects and organizations.

In order to achieve determined thesis goals we have used two different IBM tools which are IBM Rational Method Composer (RMC) and IBM Rational Software Modeler (RSM) that are briefly explained in Chapter 4. RUP has to be applied correctly to get a successful result from the project. At this point RMC provides everything you need to conduct your software development project based on RUP. It helps you to define, configure and tailor processes with its guidance in details. During the development of the project we produce some work products which requires some modeling work. The models that are needed in development were produced using the RSM which is a visual modeling and design tool of IBM based on UML 2.0.

We treated a real world problem for our software development project in Chapter 5 as a case study. Features and functioning of the existing system that is still working in the organization is summarized to form a better understanding for the structure of the system. Existing problems, in other words the complaints of users, of this system is determined in details. Possible solutions are identified to reduce or eliminate these problems according to the requests of users.

After determining the problem, project is initiated as discussed in Chapter 6. In the first step, RMC environment is prepared by creating method plug-ins where all content about the project is organized in this package. RMC also provides its method plug-ins to reference other method plug-ins that will have content contributed to extended or replaced. Then the method content is created to provide step-by-step explanations, describing how specific development goals are achieved independent of the placement of these steps within a development lifecycle. By creating method configuration we are able to specify working sets of content and processes for a specific context. In the next step, we begin to construct our capability patterns that are used as building blocks to assemble delivery processes. These capability patterns form four sequential phases of our project. The construction of capability patterns concludes with defining them by identifying activities and tasks for each of these activities. Finally in the last step of RMC preparation, delivery processes are created that describes a complete and integrated approach for performing a specific type of project. By the help of delivery processes we can easily figure out what is produced, how it is produced and the required staffing for the entire project lifecycle. The construction of delivery processes concludes with defining them by using the capability patterns that we have constructed in the previous step. As a result, RMC preparation is terminated when we obtained delivery processes including inception, elaboration, construction, and transition phases. For UML modeling of the system RSM environment is prepared by creating model project that holds our UML 2.0 model artifacts which will be created while the development of the project continues.

During the development of our project we perform four sequential phases as defined in delivery processes. They are Inception Phase, Elaboration Phase, Construction Phase, and Transition Phase which are discussed in details respectively in Chapter 7, Chapter 8, Chapter 9, and Chapter 10, respectively. In Inception Phase, we define the scope of the system and establish the feasibility of the system. In Elaboration Phase, we capture the functional requirements of the system. In Construction Phase, we focus on completing the analysis of the system, performing the majority of the design and the implementation of the system. Finally in the Transition Phase, we move the system into the user's environment. These four phases have a great importance on managing iterative-incremental software development projects based on RUP methodology. Each phase concludes with a milestone that checks whether the requirements for the current phases are satisfied or not. This control mechanism provides project managers to take decisions about the continuation of the project. These four sequential phases are split into several iterations. Number of iterations may vary from project to project depending on features and requirements of the software development project. In our project Inception Phase has one iteration, Elaboration Phase has two iterations, Construction Phase has three iterations, and Transition Phase has two iterations. All iterations consist of several activities. Some of these activities contain same features but some of them differentiate from each other depending on the phases they belong. Processing activities in parallel, which are not sequential, prevents loss of time and provides resolving possible risks quickly. Each activity consists of one or more tasks. Tasks are the major elements where the actual part of the job is done. Each task is concluded by a work product that points the critical parts and summarizes the task. Work products are important to complete the project successfully and achieve its objectives. So they are prepared correctly by following unique method that will be easily understood by team members and an appropriate format with universal practices. All these structures are properly established and managed by the help of the RMC tool. Here the most important thing is to produce work products properly and in time, because some of these work products are used as input to initiate another task. So management of the process is critical. Some work products require using UML diagrams in order to perform desired analysis and design of the system. This problem was overcome by using the RSM tool based on UML 2.0. Both RMC and RSM are IBM developed and supported tools.

The work products resulted using RMC for the software development are placed as APPENDIX B, C, D and E. Their lists are given at the end of thesis. The documentation of appendices B-E is placed on a CD that is attached to the Thesis at the end.

11.2 Conclusions

In this thesis we have presented how RUP is applied on a software development project. We studied a real world problem as a case study to reflect most possible problems encountered by project teams. Eventually, we tried to find suitable solutions to such problems.

A successful software development means not only a reliable and secure product; it should also be ready on the date previously agreed. So accurate scheduling is an important issue in software development projects while managing the development of the project. Projects that do not end on delivery dates causes budget to be exceeded and especially credibility of the developer organization gets a loss by customers. It is possible to prevent these major problems by applying RUP on our software development projects. For this purpose, we use RMC tool to apply RUP properly and efficiently. During our project, we have experienced that, RMC provides guidelines for project team members to develop projects safely and rapidly. Each task within an activity tells clearly which job to do at that moment. Such an approach eliminates the confusion within the project team. Everybody knows their liabilities and concentrates on it. Every work product that is produced at the end of each task provides a safe development activity and saves time through previously determined risks on the project. All components used in the processes ranging from tasks to phases are identified in detailed plans based on RUP methodology. Thus, the desired product has been approached in a safe manner at the end of each successful iteration. All actions that are taken within our software development project are clearly explained step by step by the help of RMC tool.

As noted earlier, we studied a small sized software development project. An overall attitude of software groups about small sized projects is that, RUP for such projects is not agile enough and is too rigid. At this point we have benefited from RMC tool to show that this claim is not true. RMC includes several best practices and templates for many kind of projects including small sized ones. We can easily add, remove or change processes based on these templates and best practices of RMC to adapt them into our small sized project. This feature allows us to get a smaller and lighter process framework and validate the decisions with real work efforts. So the desired agility was protected in terms of overall functioning of RUP methodology. On the other hand, successfully created and updated work products, that are a consequence of applying RUP, provide a clear and safe path in the process during the development of our project. Our study shows that creation of work products does not cause loss of agility. Contrarily well defined work products form a better understanding about the system that is to be developed. Also it provides a better communication between team members and especially stakeholders which eliminates major problems, such as scheduling and budget problems, encountered in software development projects. As a result, RMC supports many kind of projects regardless of their sizes. Thus, desired agility is reached also in small projects that use RUP.

A criticism that is often cited in the software engineering circles is the large volume of documentation that RUP produces. At the end of the project the bulk of the documentation is placed in the APPENDIX. That material is summarized in the following table:

Table 11.1 Volume of Work Products

APPENDIX	A	B	C		D			E	
			I	II	I	II	III	I	II
No. of Pages	9	181	256	311	392	460	535	42	34
Total	**2220**								

As it is seen in the above table even for such a small project hundreds even a few thousand pages of documentation is tremendous. In order to see the real picture we put no limitation on the number and size of the work products of the project. We, therefore, may say that volume would not differ much for medium or even large projects. The crucial point here is that RMC provides us the relevant templates and even examples so that filling them up are not that hard. Moreover, some of the information in the templates is repeated. One can, therefore, reduce the size easily for practical limits. In fact, one can state that the original number of pages of work products is about 684. Yet, for critical and long life application projects, the author strongly believes that the documentation, say long, will be very vital and useful for efficient maintenance and enhancement purposes.

REFERENCES

Ambler, S. W. (2005a), *A Manager's Introduction to The Rational Unified Process (RUP)*, Prentice Hall.

Ambler, S. W. (2005b), *The Elements of UML 2.0 Style*, Cambridge University Press.

Ambler, S. W., J. Nalbone and M. Vizdos (2005), *Enterprise Unified Process: Extending the Rational Unified Process*, Prentice Hall.

Aniszczyk, C. and Gallardo, D. (2007), *Get started with the Eclipse Platform,* IBM Corporation Software Group, New York.

Bell, D. (2003), UML *basics: An introduction to the Unified Modeling Language*, The Rational Edge.

Boggs, W. and M. Boggs (2002), *Mastering UML with Rational Rose*, SYBEX.

Booch, G., J. Rumbaugh and I. Jacobson (2005), *The Unified Modeling Language User Guide*, Addison Wesley.

Brown, A.W. (2008), *MDA Redux: Practical Realization of Model Driven Architecture,* ACM/IEEE International Conference on Composition Based Software Systems (ICCBSS) 2008, Washington, DC.

Cernosek, G. (2004), *Next-generation model-driven development,* IBM Corporation Software Group, New York.

Cernosek, G. and E. Naiburg (2004), *A technical discussion of software modeling: The Value of Modeling*, IBM Corporation Software Group, New York.

Erickson, M. and McIntyre, A. (2001), *What is Eclipse, and how do I use it?*, IBM Corporation Software Group.

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition, Addison Wesley.

Gornik, D. (2001), *IBM Rational Unified Process: Best Practices for Software Development Teams*, IBM Corporation Software Group, New York.

Hamilton, K. and R. Miles (2006), *Learning UML 2.0*, O'Reilly.

Haumer, P. (2005*), IBM Rational Method Composer: Part 1: Key concepts*, The Rational Edge.

Hunt, J. (2003), *Guide to the Unified Process featuring UML, Java and Design Patterns,* Second Edition, Springer.

IBM (June 2006a), *On Demand Business: IBM Rational Method Composer*, IBM Corporation Software Group, New York.

IBM (December 2006b), *On Demand Business: IBM Rational Method Composer*, IBM Corporation Software Group, New York.

IBM (2007a), *On Demand Business: Supporting Agile Development*, IBM Corporation Software Group, New York.

IBM (2007b), *Rational Unified Process*, RAH11027usen-00.

IBM Rational University (2004), *Essentials of Visual Modeling with UML 2.0*, IBM Corporation, California.

IBM Redbooks (2007), *The IBM Rational Unified Process for System z*, Vervante.

Kroll, P. and B. MacIsaac (2006), *Agility and Discipline Made Easy: Practices from OpenUP and RUP*, Addison Wesley.

Kroll, P. and W. Royce (2005), *Key principles for business-driven development*, The Rational Edge.

Kruchten, P. (2003), *The Rational Unified Process: An Introduction*, Third Edition, Addison Wesley.

Mittal, K. (2005), *Introducing Rational Software Modeler: Gain true application visualization,* IBM Developer Works.

Pilone, D. and N. Pitman (2005), *UML 2.0 in a Nutshell*, O'Reilly.

Selic, B. (2005), *Unified Modeling Language version 2.0*, IBM Developer Works.

Shuja, A. K. and J. Krebs (2008), *IBM Rational Unified Process Reference and Certification Guide*, IBM Press.

Smith, B. (2006), *Model Structure Guidelines For Rational Software Modeler, Rational Systems Developer, and Rational Software Architect ("Traditional RUP" Orientation),* IBM Corporation Software Group.

Smith, W. T. (2008), *IBM Rational Architecture Management software model structure guidelines: Part 1. Fundamentals*, IBM Corporation Software Group.

Wessberg, M. (2005), *Introducing the IBM Rational Unified Process essentials by analogy*, IBM Developer Works.

West, D. (2002), *Planning a Project with the Rational Unified Process*, IBM Corporation Software Group.

http://library.cankaya.edu.tr/faaliyetraporu.htm

APPENDIX A

RUP DISCIPLINE WORKFLOWS

The capability patterns can be in the form of discipline workflows. These discipline workflows are captured from the RUP poster [e.g. IBM, 2007b] and more details can be found in RMC.

1. PROJECT MANAGEMENT

1.1 Conceive New Project
- 1.1.1 Identify and Assess Risks
- 1.1.2 Develop Business Case
- 1.1.3 Initiate Project
- 1.1.4 Project Approval Review

1.2 Evaluate Project Scope and Risk
- 1.2.1 Identify and Assess Risks
- 1.2.2 Develop Business Case

1.3 Plan the Project
- 1.3.1 Develop Measurement Plan
- 1.3.2 Develop Risk Management Plan
- 1.3.3 Develop Product Acceptance Plan
- 1.3.4 Develop Problem Resolution Plan
- 1.3.5 Develop Quality Assurance Plan
- 1.3.6 Define Project Organization and Staffing
- 1.3.7 Define Monitor and Control Processes
- 1.3.8 Plan Phases and Iterations
- 1.3.9 Compile Software Development Plan
- 1.3.10 Project Planning Review

1.4 Plan Reminder of Initial Iteration
- 1.4.1 Develop Iteration Plan
- 1.4.2 Develop Business Case
- 1.4.3 Iteration Plan Review

1.5 Monitor & Control Project
- 1.5.1 Schedule and Assign Work
- 1.5.2 Monitor Project Status
- 1.5.3 Report Status
- 1.5.4 Handle Exceptions and Problems
- 1.5.5 Project Review Authority (PRA) Project Review
- 1.5.6 Organize Review
- 1.5.7 Conduct Review

1.6 Manage Iteration
- 1.6.1 Acquire Staff
- 1.6.2 Initiate Iteration
- 1.6.3 Iteration Evaluation Criteria Review
- 1.6.4 Identify and Assess Risks
- 1.6.5 Assess Iteration
- 1.6.6 Iteration Acceptance Review

1.7 Reevaluate Project Scope and Risk
- 1.7.1 Identify and Assess Risks
- 1.7.2 Develop Business Case

1.8 Plan for Next Iteration
- 1.8.1 Develop Iteration Plan
- 1.8.2 Develop Business Case
- 1.8.3 Iteration Plan Review

1.9 Refine the Development Plan
- 1.9.1 Develop Measurement Plan
- 1.9.2 Develop Risk Management Plan
- 1.9.3 Develop Product Acceptance Plan
- 1.9.4 Develop Problem Resolution Plan
- 1.9.5 Develop Quality Assurance Plan
- 1.9.6 Define Project Organization and Staffing
- 1.9.7 Define Monitor and Control Processes
- 1.9.8 Plan Phases and Iterations
- 1.9.9 Compile Software Development Plan
- 1.9.10 Project Planning Review

1.10 Close Out Phase
- 1.10.1 Prepare for Phase Close-Out
- 1.10.2 Lifecycle Milestone Review

1.11 Close Out Project
- 1.11.1 Prepare for Project Close-Cut
- 1.11.2 Project Acceptance Review

2. BUSINESS MODELING

2.1 Assess Business Status
- 2.1.1 Assess Target Organization
- 2.1.2 Set and Adjust Objectives
- 2.1.3 Identify Business Goals and KPIs
- 2.1.4 Business Architectural Analysis
- 2.1.5 Capture a Common Business Vocabulary
- 2.1.6 Maintain Business Rules

2.2 Describe Current Business
- 2.2.1 Assess Target Organization
- 2.2.2 Set and Adjust Objectives
- 2.2.3 Identify Business Goals and KPIs
- 2.2.4 Find Business Actors and Use Cases
- 2.2.5 Business Architectural Analysis
- 2.2.6 Capture a Common Business Vocabulary
- 2.2.7 Maintain Business Rules
- 2.2.8 Functional Area Analysis

2.3 Define Business
- 2.3.1 Identify Business Processes
- 2.3.2 Refine Business Process Definitions
- 2.3.3 Design Business Process Realizations
- 2.3.4 Define Business Operations
- 2.3.5 Refine Roles and Responsibilities

2.4 Explore Process Automation
- 2.4.1 Set and Adjust Objectives
- 2.4.2 Define Automation Requirements
- 2.4.3 Construct Business Architectural
- 2.4.4 Proof-of-Concept

2.5 Develop Domain Model
- 2.5.1 Capture a Common Business Vocabulary
- 2.5.2 Maintain Business Rules
- 2.5.3 Business Architecture Analysis
- 2.5.4 Detail a Business Entity
- 2.5.5 Review the Business Analysis Model

3. REQUIREMENTS

3.1 Analyze the Problem
- 3.1.1 Capture a Common Vocabulary
- 3.1.2 Find Actors and Use Cases
- 3.1.3 Develop Vision
- 3.1.4 Develop Requirements Management Plan

5.3 Service Realization
- 5.3.1 Realize Decisions

5.4 Implement Components
- 5.4.1 Plan Subsystem Integration
- 5.4.2 Implement Design Elements
- 5.4.3 Analyze Runtime Behavior
- 5.4.4 Implement Testability Elements
- 5.4.5 Implement Developer Test
- 5.4.6 Execute Developer Tests
- 5.4.7 Review Code

5.5 Integrate Each Subsystem
- 5.5.1 Implement Developer Test
- 5.5.2 Execute Developer Test
- 5.5.3 Integrate Subsystem

5.6 Integrate the System
- 5.6.1 Integrate System

6. TEST

6.1 Define Evaluation Mission
- 6.1.1 Identify Test Motivators
- 6.1.2 Agree on the Mission
- 6.1.3 Identify Targets of Test
- 6.1.4 Define Assessment and Traceability Needs
- 6.1.5 Identify Test Ideas
- 6.1.6 Define Test Approach

6.2 Verify Test Approach
- 6.2.1 Define Test Environment Configurations
- 6.2.2 Identify Testability Mechanisms
- 6.2.3 Define Testability Elements
- 6.2.4 Define Test Details
- 6.2.5 Implement Test
- 6.2.6 Implement Test Suite
- 6.2.7 Obtain Testability Commitment

6.3 Validate Build Stability
- 6.3.1 Define Test Details
- 6.3.2 Implement Test
- 6.3.3 Execute Test Suite
- 6.3.4 Analyze Test Failure
- 6.3.5 Determine Test Results
- 6.3.6 Assess and Advocate Quality

6.4 Test and Evaluate
- 6.4.1 Define Test Details
- 6.4.2 Implement Test
- 6.4.3 Implement Test Suite
- 6.4.4 Execute Test Suite
- 6.4.5 Analyze Test Failure
- 6.4.6 Structure the Test Implementation
- 6.4.7 Identify Test Ideas
- 6.4.8 Determine Test Results

6.5 Achieve Acceptable Mission
- 6.5.1 Assess and Improve Test Effort
- 6.5.2 Assess and Advocate Quality
- 6.5.3 Determine Test Results

6.6 Improve Test Assets
- 6.6.1 Prepare Guidelines for the Project
- 6.6.2 Define Test Approach
- 6.6.3 Define Testability Elements
- 6.6.4 Structure the Test Implementation
- 6.6.5 Identify Test Ideas
- 6.6.6 Define Test Details
- 6.6.7 Define Assessment and Traceability Needs
- 6.6.8 Implement Test
- 6.6.9 Implement Test Suite

7. DEPLOYMENT

7.1 Plan Deployment
- 7.1.1 Develop Deployment Plan
- 7.1.2 Define Bill of Materials

7.2 Develop Supporting Material
- 7.2.1 Develop Training Material
- 7.2.2 Develop Support Material
- 7.2.3 Create Product Artwork
- 7.2.4 Develop Installation Work Products

7.3 Manage Acceptance Test
- 7.3.1 Manage Acceptance Test
- 7.3.2 Support Development
- 7.3.3 Execute Test Suite
- 7.3.4 Determine Test Results

7.4 Produce Deployment Unit
- 7.4.1 Write Release Notes
- 7.4.2 Create Deployment Unit

7.5 Beta Test Product
- 7.5.1 Manage Beta Test

7.6 Manage Acceptance Test for Custom Install
- 7.6.1 Manage Acceptance Test
- 7.6.2 Support Development
- 7.6.3 Execute Test Suite
- 7.6.4 Determine Test Results

7.7 Package Product
- 7.7.1 Release to Manufacturing
- 7.7.2 Verify Manufactured Product

7.8 Provide Access to Download Site
- 7.8.1 Provide Access to Download Site

8. CONFIGURATION & CHANGE MANAGEMENT

8.1 Manage Change Requests
- 8.1.1 Submit Change Request
- 8.1.2 Update Change Request
- 8.1.3 Review Change Request
- 8.1.4 Confirm Duplicate or Reject CR
- 8.1.5 Schedule and Assign Work
- 8.1.6 Verify Changes in Build

8.2 Plan Project Configuration & Change Control
- 8.2.1 Establish Configuration Management (CM) Policies
- 8.2.2 Write Configuration Management (CM) Plan
- 8.2.3 Establish Change Control Process

8.3 Create Project Configuration Management (CM) Environments
- 8.3.1 Set Up Configuration Management (CM) Environment
- 8.3.2 Create Integration Workspaces

8.4 Monitor & Report Configuration Status
- 8.4.1 Report on Configuration Status
- 8.4.2 Perform Configuration Audit

8.5 Change & Deliver Configuration Items
- 8.5.1 Create Development Workspace
- 8.5.2 Make Changes
- 8.5.3 Deliver Changes
- 8.5.4 Update Workspace
- 8.5.5 Create Baseline
- 8.5.6 Promote Baselines

8.6 Manage Baselines & Releases
- 8.6.1 Create Deployment Unit
- 8.6.2 Create Baselines
- 8.6.3 Promote Baselines

9. ENVIRONMENT

9.1 Prepare Environment for Project
- 9.1.1 Tailor the Process for the Project
- 9.1.2 Develop Development Case
- 9.1.3 Prepare Guidelines for the Project
- 9.1.4 Prepare Templates for the Project
- 9.1.5 Select and Acquire Tools

9.2 Prepare Environment for an Iteration
- 9.2.1 Develop Development Case
- 9.2.2 Prepare Guidelines for the Project
- 9.2.3 Develop Manual Styleguide
- 9.2.4 Prepare Templates for the Project
- 9.2.5 Launch Development Process
- 9.2.6 Set Up Tools
- 9.2.7 Verify Tool Configuration & Installation

9.3 Support Environment for an Iteration
- 9.3.1 Support Development

APPENDIX B

INCEPTION PHASE WORK PRODUCTS

I. Inception Iteration I1 Work Products (B.I)

B.I.1	eReserve_RiskList_1.0
B.I.2	eReserve_BusinessCase_1.0
B.I.3	eReserve_SoftwareDevelopmentPlan_1.0
B.I.4	eReserve_ReviewRecord_11_11_08_1.0
B.I.5	eReserve_DevelopmentCase_1.0
B.I.6	eReserve_ConfigurationManagementPlan_1.0
B.I.7	eReserve_SoftwareDevelopmentPlan_1.1
B.I.8	eReserve_ProjectPhasePlan_1.0
B.I.9	eReserve_IterationPlanI1_1.0
B.I.10	eReserve_WorkOrder_1.0
B.I.11	eReserve_StatusAssessment_1.0
B.I.12	eReserve_Vision_1.0
B.I.13	eReserve_SoftwareArchitectureDocument_1.0
B.I.14	eReserve_SoftwareRequirementsSpecifications_1.0
B.I.15	eReserve_Vision_1.1
B.I.16	eReserve_Glossary_1.0
B.I.17	eReserve_UseCaseModel_1.0
B.I.18	eReserve_SupplementarySpecification_1.0
B.I.19	eReserve_DeploymentModel_1.0
B.I.20	eReserve_SoftwareArchitectureDocument_1.1
B.I.21	eReserve_ReferenceArchitecture_1.0
B.I.22	eReserve_TestStrategy_1.0
B.I.23	eReserve_WorkOrder_1.1
B.I.24	eReserve_IterationAssessment_1.0
B.I.25	eReserve_ProjectPhasePlan_1.1
B.I.26	eReserve_IterationPlanE1_1.0

All of the Inception Iteration I1 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixB/InceptionIterationI1

Samples and detailed explanations for work products can be found in RMC.

APPENDIX C

ELABORATION PHASE WORK PRODUCTS

I. Elaboration Iteration E1 Work Products (C.I)

C.I.1	eReserve_SoftwareDevelopmentPlan_2.0
C.I.2	eReserve_IntegrationBuildPlan_1.0
C.I.3	eReserve_WorkOrder_2.0
C.I.4	eReserve_RiskList_2.0
C.I.5	eReserve_IterationAssessment_2.0
C.I.6	eReserve_StatusAssessment_2.0
C.I.7	eReserve_UseCaseModel_2.0
C.I.8	eReserve_Glossary_2.0
C.I.9	eReserve_ChangeRequestCR_01_1.0
C.I.10	eReserve_DevelopmentInfrastructure_1.0
C.I.11	eReserve_UseCaseModel_2.1
C.I.12	eReserve_SoftwareRequirementsSpecifications_2.0
C.I.13	eReserve_SupplementarySpecification_2.0
C.I.14	eReserve_SoftwareArchitectureDocument_2.0
C.I.15	eReserve_AnalysisModel_1.0
C.I.16	eReserve_UseCaseRealizationSpecification_1.0
C.I.17	eReserve_DesignModel_1.0
C.I.18	eReserve_DesignModel_1.1
C.I.19	eReserve_DesignModel_1.2
C.I.20	eReserve_SoftwareArchitectureDocument_2.1
C.I.21	eReserve_ImplementationModel_1.0
C.I.22	eReserve_SoftwareArchitectureDocument_2.2
C.I.23	eReserve_UseCaseRealizationSpecification_1.1
C.I.24	eReserve_NavigationMap_1.0
C.I.25	eReserve_UserInterfacePrototype_1.0
C.I.26	eReserve_DataModel_1.0
C.I.27	eReserve_IntegrationBuildPlan_1.1
C.I.28	eReserve_TestSuite_1.0
C.I.29	eReserve_DeveloperTest_1.0
C.I.30	eReserve_TestLog_1.0
C.I.31	eReserve_TestCase_1.0

C.I.32 eReserve_TestSuite_1.1
C.I.33 eReserve_TestLog_1.1
C.I.34 eReserve_TestResults_1.0
C.I.35 eReserve_DefectReportDF_01_1.0
C.I.36 eReserve_TestEvaluationSummary_1.0
C.I.37 eReserve_ProjectPhasePlan_2.0
C.I.38 eReserve_IterationPlanE2_1.0
C.I.39 eReserve_Build_1.0

All of the Elaboration Iteration E1 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixC/ElaborationIterationE1

Samples and detailed explanations for work products can be found in RMC.

II. Elaboration Iteration E2 Work Products (C.II)

C.II.1 eReserve_SoftwareDevelopmentPlan_3.0
C.II.2 eReserve_WorkOrder_3.0
C.II.3 eReserve_RiskList_3.0
C.II.4 eReserve_IterationAssessment_3.0
C.II.5 eReserve_StatusAssessment_3.0
C.II.6 eReserve_Glossary_3.0
C.II.7 eReserve_ChangeRequestCR_02_1.0
C.II.8 eReserve_UseCaseModel_3.0
C.II.9 eReserve_SoftwareRequirementsSpecifications_3.0
C.II.10 eReserve_SupplementarySpecification_3.0
C.II.11 eReserve_AnalysisModel_2.0
C.II.12 eReserve_UseCaseRealizationSpecification_2.0
C.II.13 eReserve_DesignModel_2.0
C.II.14 eReserve_DesignModel_2.1
C.II.15 eReserve_SoftwareArchitectureDocument_3.0
C.II.16 eReserve_ImplementationModel_2.0
C.II.17 eReserve_SoftwareArchitectureDocument_3.1
C.II.18 eReserve_UseCaseRealizationSpecification_2.1
C.II.19 eReserve_NavigationMap_2.0
C.II.20 eReserve_UserInterfacePrototype_2.0
C.II.21 eReserve_DataModel_2.0
C.II.22 eReserve_IntegrationBuildPlan_2.0
C.II.23 eReserve_TestSuite_2.0
C.II.24 eReserve_DeveloperTest_2.0
C.II.25 eReserve_TestLog_2.0
C.II.26 eReserve_TestCase_2.0
C.II.27 eReserve_TestSuite_2.1
C.II.28 eReserve_TestLog_2.1
C.II.29 eReserve_TestResults_2.0
C.II.30 eReserve_DefectReportDF_02_1.0
C.II.31 eReserve_DefectReportDF_03_1.0

C.II.32 eReserve_TestEvaluationSummary_2.0
C.II.33 eReserve_ProjectPhasePlan_3.0
C.II.34 eReserve_IterationPlanC1_1.0
C.II.35 eReserve_Build_2.0

All of the Elaboration Iteration E2 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixC/ElaborationIterationE2

Samples and detailed explanations for work products can be found in RMC.

APPENDIX D

CONSTRUCTION PHASE WORK PRODUCTS

I. Construction Iteration C1 Work Products (D.I)

D.I.1	eReserve_WorkOrder_4.0
D.I.2	eReserve_RiskList_4.0
D.I.3	eReserve_IterationAssessment_4.0
D.I.4	eReserve_StatusAssessment_4.0
D.I.5	eReserve_Glossary_4.0
D.I.6	eReserve_ChangeRequestCR_03_1.0
D.I.7	eReserve_UseCaseModel_4.0
D.I.8	eReserve_SoftwareRequirementsSpecifications_4.0
D.I.9	eReserve_AnalysisModel_3.0
D.I.10	eReserve_UseCaseRealizationSpecification_3.0
D.I.11	eReserve_DesignModel_3.0
D.I.12	eReserve_DesignModel_3.1
D.I.13	eReserve_SoftwareArchitectureDocument_4.0
D.I.14	eReserve_ImplementationModel_3.0
D.I.15	eReserve_SoftwareArchitectureDocument_4.1
D.I.16	eReserve_UseCaseRealizationSpecification_3.1
D.I.17	eReserve_NavigationMap_3.0
D.I.18	eReserve_UserInterfacePrototype_3.0
D.I.19	eReserve_DataModel_3.0
D.I.20	eReserve_IntegrationBuildPlan_3.0
D.I.21	eReserve_TestSuite_3.0
D.I.22	eReserve_DeveloperTest_3.0
D.I.23	eReserve_TestLog_3.0
D.I.24	eReserve_TestCase_3.0
D.I.25	eReserve_TestSuite_3.1
D.I.26	eReserve_TestLog_3.1
D.I.27	eReserve_TestResults_3.0
D.I.28	eReserve_DefectReportDF_04_1.0
D.I.29	eReserve_TestEvaluationSummary_3.0
D.I.30	eReserve_ProjectPhasePlan_4.0
D.I.31	eReserve_IterationPlanC2_1.0

D.I.32 eReserve_Build_3.0

All of the Construction Iteration C1 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixD/ConstructionIterationC1

Samples and detailed explanations for work products can be found in RMC.

II. Construction Iteration C2 Work Products (D.II)

D.II.1 eReserve_WorkOrder_5.0
D.II.2 eReserve_RiskList_5.0
D.II.3 eReserve_IterationAssessment_5.0
D.II.4 eReserve_StatusAssessment_5.0
D.II.5 eReserve_Glossary_5.0
D.II.6 eReserve_ChangeRequestCR_04_1.0
D.II.7 eReserve_UseCaseModel_5.0
D.II.8 eReserve_SoftwareRequirementsSpecifications_5.0
D.II.9 eReserve_AnalysisModel_4.0
D.II.10 eReserve_UseCaseRealizationSpecification_4.0
D.II.11 eReserve_DesignModel_4.0
D.II.12 eReserve_DesignModel_4.1
D.II.13 eReserve_SoftwareArchitectureDocument_5.0
D.II.14 eReserve_ImplementationModel_4.0
D.II.15 eReserve_SoftwareArchitectureDocument_5.1
D.II.16 eReserve_UseCaseRealizationSpecification_4.1
D.II.17 eReserve_NavigationMap_4.0
D.II.18 eReserve_UserInterfacePrototype_4.0
D.II.19 eReserve_DataModel_4.0
D.II.20 eReserve_IntegrationBuildPlan_4.0
D.II.21 eReserve_TestSuite_4.0
D.II.22 eReserve_DeveloperTest_4.0
D.II.23 eReserve_TestLog_4.0
D.II.24 eReserve_TestCase_4.0
D.II.25 eReserve_TestSuite_4.1
D.II.26 eReserve_TestLog_4.1
D.II.27 eReserve_TestResults_4.0
D.II.28 eReserve_DefectReportDF_05_1.0
D.II.29 eReserve_TestEvaluationSummary_4.0
D.II.30 eReserve_ProjectPhasePlan_5.0
D.II.31 eReserve_IterationPlanC3_1.0
D.II.32 eReserve_Build_4.0

All of the Construction Iteration C2 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixD/ConstructionIterationC2

Samples and detailed explanations for work products can be found in RMC.

III. Construction Iteration C3 Work Products (D.III)

D.III.1 eReserve_WorkOrder_6.0
D.III.2 eReserve_RiskList_6.0
D.III.3 eReserve_IterationAssessment_6.0
D.III.4 eReserve_StatusAssessment_6.0
D.III.5 eReserve_Glossary_6.0
D.III.6 eReserve_ChangeRequestCR_05_1.0
D.III.7 eReserve_UseCaseModel_6.0
D.III.8 eReserve_SoftwareRequirementsSpecifications_6.0
D.III.9 eReserve_AnalysisModel_5.0
D.III.10 eReserve_UseCaseRealizationSpecification_5.0
D.III.11 eReserve_DesignModel_5.0
D.III.12 eReserve_DesignModel_5.1
D.III.13 eReserve_SoftwareArchitectureDocument_6.0
D.III.14 eReserve_ImplementationModel_5.0
D.III.15 eReserve_SoftwareArchitectureDocument_6.1
D.III.16 eReserve_UseCaseRealizationSpecification_5.1
D.III.17 eReserve_NavigationMap_5.0
D.III.18 eReserve_UserInterfacePrototype_5.0
D.III.19 eReserve_DataModel_5.0
D.III.20 eReserve_IntegrationBuildPlan_5.0
D.III.21 eReserve_TestSuite_5.0
D.III.22 eReserve_DeveloperTest_5.0
D.III.23 eReserve_TestLog_5.0
D.III.24 eReserve_TestCase_5.0
D.III.25 eReserve_TestSuite_5.1
D.III.26 eReserve_TestLog_5.1
D.III.27 eReserve_TestResults_5.0
D.III.28 eReserve_DefectReportDF_06_1.0
D.III.29 eReserve_TestEvaluationSummary_5.0
D.III.30 eReserve_ProjectPhasePlan_6.0
D.III.31 eReserve_IterationPlanT1_1.0
D.III.32 eReserve_Build_5.0

All of the Construction Iteration C3 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixD/ConstructionIterationC3

Samples and detailed explanations for work products can be found in RMC.

APPENDIX E

TRANSITION PHASE WORK PRODUCTS

I. Transition Iteration T1 Work Products (E.I)

E.I.1	eReserve_WorkOrder_7.0
E.I.2	eReserve_RiskList_7.0
E.I.3	eReserve_IterationAssessment_7.0
E.I.4	eReserve_StatusAssessment_7.0
E.I.5	eReserve_TestEvaluationSummary_6.0
E.I.6	eReserve_ProjectPhasePlan_7.0
E.I.7	eReserve_IterationPlanT2_1.0

All of the Transition Iteration T1 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixE/TransitionIterationT1

Samples and detailed explanations for work products can be found in RMC.

II. Transition Iteration T2 Work Products (E.II)

E.II.1	eReserve_WorkOrder_8.0
E.II.2	eReserve_RiskList_8.0
E.II.3	eReserve_IterationAssessment_8.0
E.II.4	eReserve_StatusAssessment_8.0
E.II.5	eReserve_TestEvaluationSummary_7.0

All of the Transition Iteration T2 work products can be found in the CD with detailed documentation and can be accessed separately as follows:

- ~/Appendices/AppendixE/TransitionIterationT2

Samples and detailed explanations for work products can be found in RMC.

APPENDIX F

IBM RATIONAL TOOL PLUG-INS

All RMC and RSM materials that are used in our software development project can be found in the CD as in the form of plug-ins with details and can be accessed separately as follows:

- ~/Appendices/AppendixF/RMC
- ~/Appendices/AppendixF/RSM

These plug-ins can be easily integrated into RMC and RSM tools to reach the content.