CBOE Object-Oriented Development Process

Mark Woyna, Brad Samuels, Dave Wegener

07/10/98 4:12 PM

Table of Contents

Tables

Introduction

This document describes the development process of a project that is based partially on the Rational Objectory process. The first section is based on the definition for capturing business requirements by definition of the actors and the procedures that they use to fulfill the function of the business. The other sections are the definition of the Objectory Development Case for CBOE Object Oriented projects.

To have a greater understanding of this document it is strongly suggested that the reader take a cursory look at the Objectory documentation. This would include the overviews of the processes used in Requirements Capture, Analysis and Design, Implementation and Testing.

This document does not go into in depth description of every artifact or report that is listed. Reference these items In the Objectory document.

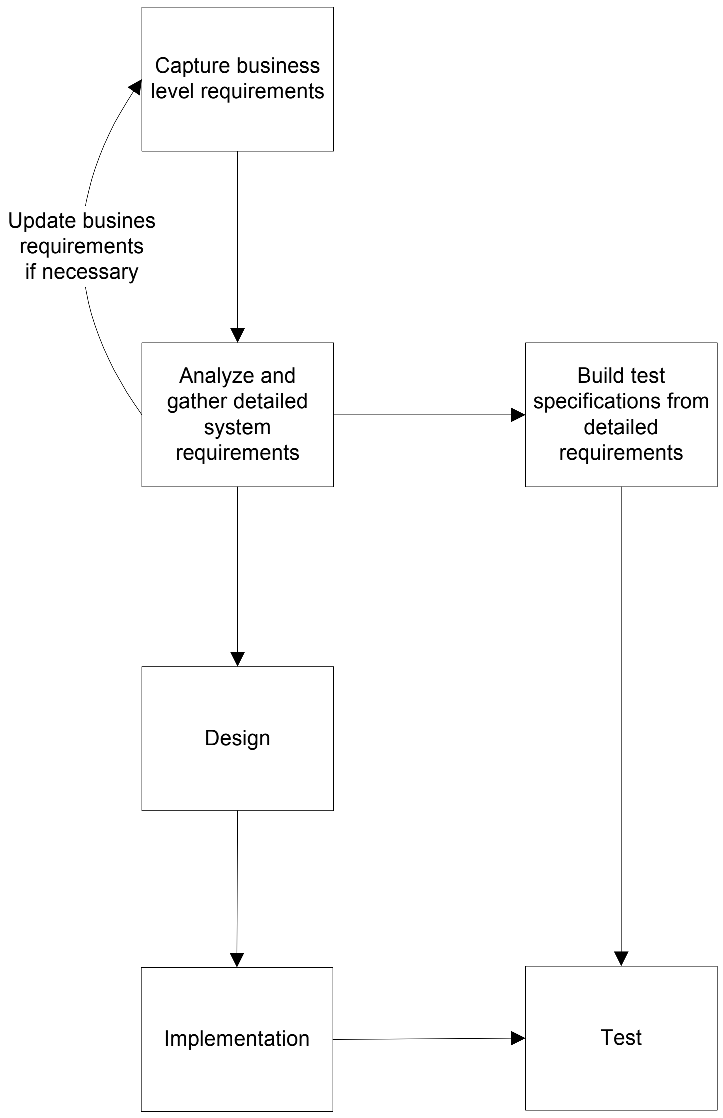
Philosophy

CBOE Systems Development has determined that the use of a development process will greatly enhance the quality and robustness of software projects that are developed for the company. There is a thought that when using a development process that it should be streamlined enough to permit iterative development rather than a waterfall approach. The amount of paper documentation should be kept to a minimum by using modeling techniques that can capture as much of the requirements and design as possible and will allow maintainability when modifications are made to the requirements which will show the effects of the changes throughout the model.

When using a development process it will also support a “Top Down” and “Bottom Up” approach to reach the goal of implementing a system or modifications to existing systems. This is crucial to have development efforts to be accomplished in a timely manner. On the same note there are some pros and cons to having both of these approaches being used in parallel. The “Top Down” and “Bottom Up” portions need to be synchronized so that the system to be developed is not missing the mark of the requirements and that the business requirements are complete.

Concurrency

This diagram illustrates some of the concurrency in the initial analysis that should occur.


Capture business level requirements
Update busines requirements if necessary
Analyze and gather detailed system requirements
Build test specifications from detailed requirements
Design
Implementation
Test

The Tools

CBOE has standardized on the following tools to implement this development process for software systems under an Object Oriented approach.

Rational Objectory

Objectory process captures many of the best practices in modern software development in a form that can be tailored for a wide range of projects and organizations.

Objectory is an *iterative* process. Given today's sophisticated software systems, it is not possible to sequentially first define the entire problem, design the entire solution, build the software, and finally test the product. An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations. This approach gives better flexibility in accommodating new requirements or tactical changes in business objectives, and allows the project to *identify and resolve risks* earlier.

Objectory is a *controlled* process. This iterative approach is only possible however through very careful *requirements management,* and *change control* to ensure at every point in time a common understanding of the expected functionality, the expected level of quality, and to allow a better control of the associated costs and schedules.

Objectory activities create and maintain *models*. Rather than focusing on producing large amounts of paper documents, Objectory emphasizes the development and maintenance of *models* - semantically rich representations of the software system under development.

Objectory focuses on early development and base lining of a robust software *architecture,* which facilitates parallel development, minimizes rework, increases reusability and maintainability. This architecture is used to plan and manage the development around the use of software *components*.

Objectory development activities are driven by *use cases*. The notion of use cases, and scenarios drive the process flow from requirements capture through testing, and provides coherent and traceable threads through both the development and the delivered system.

Objectory supports *object-oriented techniques*. Several of the models are object-oriented models, based on the concepts of objects, classes, and associations between them. These models, like many other technical artifacts, use the Unified Modeling Language (UML) as the common notation.

Objectory supports *component-based software development*. Components are nontrivial modules, subsystems that fulfill a clear function, and that can be assembled in a well-defined architecture, either ad hoc, or some component infrastructure such as the Internet, CORBA, COM/DCOM, for which an industry of reusable components is emerging.

Objectory is a *configurable* process. No single process is suitable for all software development. Objectory fits small development teams, as well as large development organizations. Objectory is founded on a simple and clear process architecture that provides commonality across a family of processes, and yet can be varied to accommodate different situations. It contains guidance on how to configure the process to suit the needs of a given organization.

Objectory encourages objective on-going *quality* control. Quality assessment is built into the process, in all activities, involving all participants, using objective measurements and criteria, and not treated as an afterthought, or a separate activity performed by a separate group.

Objectory is supported by *tools* that automate large parts of the process. They are used to create and maintain the various artifacts - models in particular - of the software engineering process: visual modeling, programming, testing, and so on. They are invaluable in supporting all the bookkeeping associated with the change management, as well as the configuration management that accompanies each iteration.

Rational RequistePro

RequisitePro is a software tool that helps find, document, organize, and track changes to user requirements, software specifications, and test cases. RequisuitePro integrates Microsoft Word with a secure, multi-user requirements database.

Rational Rose

Rational Rose supports Objectory by providing a means to create and maintain the models developed in some of the process components:

· The use-case model and the domain model (optional) in Requirements Capture.

· The design model in Analysis & Design.

· The implementation model in Implementation.

Rational SoDA

SoDA can automatically create documents from Rational Rose models and RequisitePro documents. This is accomplished through specialized Word templates.

Microsoft Word95

This will be the standard word processor that for all documents and templates.

Business Requirements

The purpose of the Business Requirements is to capture the fundamental actions and objects that the business uses for the day to day operations. The way the requirements are captured is at a level of what the actors in the business are responsible for, any “procedures” or “procedural systems” that the actors use to fulfill their responsibilities and artifacts or “objects” that are used to complete the responsibility.

When describing “procedures” or “procedural systems” that are currently implemented as software systems, the goal is to capture the purpose and the abstract interface that supports the business for that system and not a detailed system description of the actual interface. The abstract interface is defined through the actors use of the process. Rules are captured as part of the description of a process.

The requirements will be captured in a Business Object Model. This model will be used to drive the analysis and systems requirements that will be generated for the development of the systems to support the business. This model will also be used to verify existing requirements and design that have been previously developed.

Also, this section defines some business level technology, integration and architecture requirements that are used to guide the analysis in the next section.

Capture Requirements from Model

Sequence diagrams are shown with a use case. Use cases are determined from actor responsibilities.

Table 1 Business Requirements Artifacts

Artifact	*How to Use*
Business Object Model	Formal-External
Actor	Formal-External
Actor Package	Informal
Subject	Formal-External
Subject Package	Informal
Use Case	Formal-External
Use Case Package	Informal
Traceability Database	Formal-Internal
Technology Architecture Document	Formal-Internal
Integration Architecture Document	Formal-Internal
Systems Architecture Document	Formal-Internal

Table 2 Business Requirement Reports

Report	*How to Use*
Actor Report	Formal-External
Subject Report	Formal-External
Use Case Report	Formal-External
Business Traceability Report	Formal-Internal

Business Object Model

The **business object model** is a model that describes the business’ requirements in terms of use cases, actors and subjects.

The model will be implemented in the Rose Use Case View. The configuration management of this artifact will be in the main Rose model file.

Actor

This is the representation of a workers, roles or group entities that interact with the business. Roles are the common responsibilities of workers or group entities that can be defined abstractly.

The configuration management of this artifact will be by the Rational Rose model file.

Actor Package

An **actor package** is a collection of actors; it is used to structure the actors by dividing it into smaller parts.

The configuration management of this artifact will be by the Rational Rose model file.

Subject

This is a representation of an object that is used in the business or a procedure that is done to handle an object for the business.

The configuration management of this artifact will be by the Rational Rose model file.

Subject Package

A **subject package** is a collection of subjects; it is used to structure the business model subjects by dividing it into smaller parts.

The use case package will be an organization by functionality. The configuration management of this artifact will be by the Rational Rose model file.

Use Case

This is the representation of an action which the business must support. It is captured by describing the action in text, the involvement of actors and subjects in a use case diagram, and how the action is accomplished through sequence diagrams. The configuration management of this artifact will be by the Rational Rose model file.

Use Case Package

A **use-case package** is a collection of use cases, relationships, diagrams, and other packages; it is used to structure the use-case model by dividing it into smaller parts.

The use case package will be an organization by functionality. The configuration management of this artifact will be by the Rational Rose model file.

Traceability Database

This database will capture relationships of use cases, actors, subjects, objects and classes that represent the business requirements and how they are supported through the systems to be developed. In the context of the business requirements the use cases, actors and subjects are traced for tracking completeness of the business model. Requirements are determined from subjects and actors.

- Business Requirements
  - Business Use Cases
    - Business Subjects
    - Business Actors
    - Business Roles
      - Business Actors

The traceability is determined from the business model.

This will be implemented in a Requisite Pro database. The configuration management of this will be through the Requisite Pro software package.

Technology Architecture Document

This document describes the computing platforms, fundamental infrastructure components and development environment for CBOE's future architecture that enables distributed object systems.

The configuration management will be through the default configuration management tool.

Integration Architecture Document

This document describes an integration architecture that focuses on establishing the overall vision for integrating application systems by providing a constructive software environment and strategies for external system integration. The configuration management will be through the default configuration management tool.

Systems Architecture Document

This document describes the high level systems layering architecture that will be guide the construction of the business systems. The configuration management will be through the default configuration management tool.

Actor Report

This will be a SoDA document that will describe the actors in the actor package and their relationships to use cases. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Subject Report

This will be a SoDA document that will describe the subjects in the subject package and their relationships to use cases. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Use Case Report

This will be a SoDA document that will describe the use cases in the use case package and their relationships to actors and subjects. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Business Traceability Report

This report is generated from the Traceability Database which will identify requirements and how they are supported by use cases and the underlying support for the use case by subjects, actors, and roles. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Analysis and Requirements Capture

The purposes of Analysis Requirements Capture are:

- To provide the translation from the Business Model Use Case Requirements to a system view that can fulfill the Business Requirements.
- Show traceability for realized objects to use cases.
- To come to an agreement with the users on what the system should do.
- To give system developers a better understanding of the requirements on the system.
- To delimit the system.
- To provide a basis for planning the technical contents of iterations.
- To achieve these goals, a *use-case model* is developed that describes *what* the system will do - an effort that views users as important sources of information (in addition to system requirements). The use-case model can then serve as a contract between the users and the system developers, which allows:
    1. Users to validate that the system will become what they expected.
    2. System developers to build what is expected.

The analysis of the business requirements to be implemented as a system are captured in this stage. The business requirements should have subjects that can be used as the foundation for the system to be built. This will allow for the realization of objects or services that can be re-used throughout the systems that support the business. The subjects are refined at this stage to become interfaces for the analysis system.

The capturing of requirements from the users pertaining to the use of the system should be realized in use cases that describe the functionality they will need and also to reference these requirements to the business use case that this requirement will support.

There is analysis that must be done to relate the requirements from the business model to the analysis model and vice versa (since we are using both “Top Down” and “Bottom Up” approaches). This will be done and both models verified for correctness and any adjustments should be made to reflect the results of the analysis. There is no set rules for how this is done. The suggestion is to use the packages as the guidelines for linking between the models and use the traceability matrix to set the links for the requirements.

When the links in the traceability matrices have been completed, a review of the use cases and objects can be done to see if a use case has been satisfied by objects and that an object is not a loner by not having an associated use case. This may seem limited off the bat, but someone with domain knowledge will still need to review these matrices to determine that the model is complete and the requirements are being satisfied. To accomplish this there is some level of interpretation that is required to understand the model.

Table 3 Analysis and Requirements Artifacts

Artifact	*How to Use*
Analysis Model	Formal-External
Use Case Package	Informal
Use Case	Formal-External
Use Case Realization	Informal
Object Realization/Interface Specification	Formal-Internal
Traceability Database	Formal-Internal
Glossary	Formal-External
Supplemental Specifications	Formal-External
Application(s) Architecture Document	Formal-Internal

Table 4 Analysis and Requirement Reports

Report	*How to Use*
Actor Report	Formal-External
Use Case Report	Formal-External
Use Case Model Survey	Formal-External
Use Case Realization Report	Casual
Object/Interface Specification Report	Formal-Internal
Analysis Traceabilty Report	Formal-Internal
Requirement Specification	Formal-External

Analysis Model

The **analysis model** is a model that describes a system's functional requirements in terms of use cases.

The analysis model will be implemented in the Rose Use Case View. The configuration management of this artifact will be in the main Rose model file that is under the default configuration management tool.

Use Case Package

A **use-case package** is a collection of use cases, actors, relationships, diagrams, and other packages; it is used to structure the use-case model by dividing it into smaller parts.

The use case package will be an organization by functionality. A separate Actor Package should be created to contain all actors for the project. This may be dynamic over time due to the scope of the individual use cases. The configuration management of this artifact will be the analysis model file.

Use Case

A **use-case instance** is a sequence of actions a system performs that yields an observable result of value to a particular actor.

A **use-case type** defines a set of use-case instances.

There are several key words in this definition:

· *Use-case instance.* The sequence referred to in the definition is really a specific flow of events through the system, or an instance. Many flow of events are possible, and many may be very similar. To make a use-case model understandable, you should group similar flows of events into a use-case type. Identifying and describing a use case really means identifying and describing the type.

· *System performs.* This means that the system provides the use case. An actor communicates with a use-case instance of the system.

· *An observable result of value.* You can put a value on a successfully performed use case. A use case should make sure that an actor can perform a task that has an identifiable value. This is very important in determining the correct level or granularity for a use case. Correct level refers to achieving use cases that are not too small. In certain circumstances, you can use a use case as a planning unit in the organization that includes the actor.

· *Actions.* An action is a computational or algorithmic procedure; it is invoked either when the actor provides a signal to the system or when the system gets a time event. An action may imply signal transmissions to either the invoking actor or other actors. An action is atomic, which means it is performed either entirely or not at all.

· *A particular actor.* The actor is key to finding the correct use case, especially because the actor helps you avoid use cases that are too large. It is important to begin with individual (human) actors, or instances of actors. It is a good idea when determining suitable actors to name at least two or, if possible, three people who would be able to perform as an individual actor. Suppose you are developing an object-oriented modeling tool. There are really two actors in this example: a developer, someone who develops systems using the tool as support, and a system administrator, someone who manages the tool. Each of these actors has his own demands on the system, and will therefore require his own set of use cases.

The use case will be documented with a SoDA Template and a Rose Use Case Diagram. The SoDA Use Case Document that is created will be linked into the use case definition in the Rose Use Case Specification. The Rose use case will be managed as part of the use case package. The configuration management of the SoDA Use Case Document will be the default configuration management tool.

Use Case Realization

A use-case realization describes how a particular use case is realized within the anlaysis model, in terms of collaborating objects.

This has also been termed as a scenario diagram. This is documented in the Use Case as a sequence diagram. The configuration management will be supplied by the use case package.

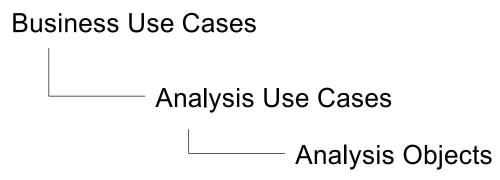
Object Realization/Interface Specification

An object realization is a class that is defined to support the a use case when it is realized within the analysis model. This may be interpreted as an interface for a subsystem or an object that is used by the subsystem. The subject from the business model is refined.

This is documented in the Logical section as a class. The configuration management will be supplied by the analysis model file.

Traceability Database

The traces for analysis use cases are added to the corresponding business use cases. New analysis objects are traced to analysis use cases.



Glossary

There is one *Glossary* for the system. The *Glossary* will be used identify the common terms and acronyms that are used in the system. The configuration management will be through the default configuration management tool.

Supplemental Specifications

The Supplementary Specifications capture the system requirements that are not readily capturable in the use cases of the use-case model. Such requirements include; legal and regulatory requirements, and application standards; quality attributes of the system to be built, including usability, reliability, performance and supportability requirements; other requirements such as operating systems and environments, compatibility requirements, and design constraints. The configuration management will be through the default configuration management tool.

Application(s) Architecture Document

The Application(s) Architecture Document describes the software and system architecture by means of different architectural views. This can be done for a set of relating applications or for a single application that is being developed. This document should contain any information pertaining to layering, third party software specifications and infrastructure architecture. The representation of this information is free form and has no set rules for modeling. The architecture or other system specific information that pertains to the use case view architecture will added to the document. The configuration management will be through the default configuration management tool.

Actor Report

This will be a SoDA document that will describe the actors in the actor package and their relationships to use cases. The configuration management will be through the default configuration management tool.

Use Case Report

This is a generated document from a customized SoDA Word template for each Use Case. The content will be a Use Case Diagram from Rose with the Use Case text from the Use Case Document. There will be one Use Case Report per Use Case. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Use Case Model Survey

This is a generated document form a customized SoDA Word template. This is a document of the Rose Use Case View Diagram. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Requirement Specification

This is a generated document from a customized SoDA Word template. This is a document that has all the use case documents and use case diagrams with a cover page and a boilerplate introduction. . This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Use Case Realization Report

This is a generated document form a customized SoDA Word template. This will contain the realization sequence diagrams, the use case diagram and the use case document. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Object/Interface Specification Report

This is a generated document form a customized SoDA Word template. This will contain the realization of objects or interfaces that have been identified within the analysis model. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Analysis Traceability Report

This report is generated from the Traceability Database which will identify the use cases, object realizations and the business requirements that these items are supporting. This will show the analysis use case hierarchy backwards to the business requirements. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Workflow

No changes from the suggested Objectory Work Flow.

Design

The purposes of Design are:

· To describe how the system should realize all the requirements on it.

· To build a system that is resilient to changes in the system requirements, and in the implementation environment.

· To develop a stable software architecture in early iterations to mitigate risks associated with architecture.

· To develop specifications that can be used as input to implementation and testing activities.

To achieve these goals, a *design model* is developed on the basis of the results from requirements capture (the use-case model), and of guidelines on how to use the implementation environment. The design model serves as an abstraction of the source code; in other words, the design model acts as a "blueprint" of how the source code is written and organized into manageable units.

Table 5 Design Artifacts

Artifact	*How to Use*
Design Model	Formal-Internal
Design Package	Formal-Internal
Class	Formal-Internal
Traceability Database	Formal-Internal
Application(s) Architecture Document	Formal-External

Table 6 Design Reports

Report	*How to Use*
Class Report	Casual
Design Model Survey	Casual
Design Specification	Casual
Design Traceability Report	Casual

Design Model

The design model is adapted to model the real implementation environment, and serves as an abstraction of the source code. It is a "blueprint" of how the source code is structured and written.

The design model is a hierarchy of packages (*design subsystems* and *design-service packages*), with "leaves" that are *classes* or *use-case realizations*.

The design model will be implemented in Rose. The configuration management of this artifact will be in the main Rose model file.

Design Package

A design package is a collection of classes, relationships, use-case realizations, diagrams, and other packages; it is used to structure the design model by dividing it into smaller parts.

The layering will be assigned by View, Controller or Model. A design package will be assigned an owner for maintenance. The configuration management of this artifact will be by the Rational Rose category file.

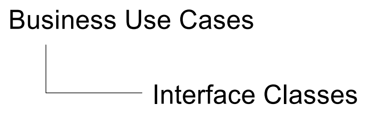
Class

A **class** is a description of a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics.

The class will be documented within Rose in a specified design package. The configuration management will be supplied by the design package.

Traceability Database

This will trace design interface classes to analysis objects or applicable analysis use cases. Any supporting design classes will be traced to the respective interface class. This may be in a separate database.



Application(s) Architecture Document

The architecture or other system specific information that pertains to the design, will added to the existing applications architecture document started in the analysis phase. The configuration management will be through the default configuration management tool.

Class Report

This is a generated document form a customized SoDA Word template. This contains the description of the class, i.e. methods and instance variables, and any documentation that was entered into the Rose Class Specification. This will not be in configuration management since it is only generated from managed artifacts.

Design Model Survey

This is a generated document form a customized SoDA Word template. This documents the packages and classes in the model with relationships. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Design Specification

This is a generated document form a customized SoDA Word template. This documents the packages and classes in the model with the class diagrams and sequence diagrams. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Design Traceability Report

This report is generated from the Traceability Database which will identify the use cases, object realizations and the business requirements that the classes are supporting. This will show the interface classes and the traced analysis use case or analysis object. This will not be in configuration management since it is only generated from managed artifacts. This document will be time stamped.

Workflow

No changes from the suggested Objectory Work Flow.

Implementation

The Rational Objectory process describes “Implementation” as:

· Defining the organization of the code, in terms of implementation subsystems organized in layers.

· Implementing classes and objects in terms of components (source files, binaries, executables, and others).

· Testing the developed components as units.

· Integrating the results produced by individual implementers (or teams), into an executable system.

The Implementation process component limits its scope to how individual classes are to be unit tested. System tests and integration tests are described in the Test process component.

Table 7 Implementation Artifacts

Artifact	*How to Use*
Implementation Model	Informal
Implementation Subsystem	Formal-Internal
Component	Formal-Internal
Integration Build Plan	Casual
Software Architecture Document	Formal-External

Table 8 Implementation Activities

Activity	*How to Use*
Code Reviews	Formal-Internal

Implementation Model

The implementation model will be implemented in Rational Rose and capture the main component view of the system. The configuration management of this artifact will be in the main Rational Rose model file.

Implementation Subsystem

Each implementation subsystem will be a component that contains the logical view packages that the subsystem realizes. The configuration management of this artifact will be by the Rational Rose category file.

Component

Each component represents a piece of software code, a file containing information, or an aggregate of other components, for example, an application consisting of several executables. It will be documented as a Rational Rose Component Package. The configuration management of this artifact will be by the Rational Rose category file.

Integration Build Plan

The Integration Build Plan provides a detailed plan for integration within an iteration. It will be documented as a Word document. There will be no configuration management of this artifact. This document will be time stamped.

Software Architecture Document

The architecture or other system specific information that pertains to the implementation architecture will be added to this document. The configuration management will be through the default configuration management tool.

Workflow

No changes from the suggested Objectory Work Flow.

Testing

The Rational Objectory process describes "Testing" as:

· Verifying the interaction between objects.

· Verifying the proper integration of all components of the software.

· Verifying that all requirements have been correctly implemented.

· Identifying and ensuring defects are addressed prior to the deployment of the software.

In many organizations, software testing accounts for 30 to 50 percent of software development costs. Yet most people believe that software is not well tested before it is delivered. This contradiction is rooted in two clear facts. First, testing software is enormously difficult. The different ways a given program can behave are unquantifiable. Second, testing is typically done without a clear methodology and without the required automation or tool support. While the complexity of software makes complete testing an impossible goal, a well-conceived methodology and use of state-of-the-art tools, can greatly improve the productivity and effectiveness of the software testing.

For "safety-critical" systems where a failure can harm people (such as air-traffic control, missile guidance, or medical delivery systems), high-quality software is essential for the success of the system produced. For a typical MIS system, this situation is not as painfully obvious, but the impact of a defect can be very expensive.

Well-performed tests, initiated early in the software lifecycle, will significantly lower the cost of completing and maintaining the software. It will also greatly reduce the risks or liabilities associated with deploying poor quality software, such as poor user productivity, data entry and calculation errors, and unacceptable functional behavior. Nowadays, many MIS system are "mission-critical", that is, companies cannot fulfill their functions and experience massive losses when failures occur. For example: banks, or transportation companies. Mission-critical systems must be tested using the same rigorous approaches used for safety-critical systems.

Table 9 Testing Artifacts

Artifact	***How to Use***
Test Case	Formal-Internal
Test Procedure	Formal-Internal
Test Packages	Formal-Internal
Test Classes	Formal-Internal
Test Components	Informal
Test Subsystems	Informal
Defects	Formal-Internal
Test Plan	Formal-External

Table 10 Testing Reports

Activity	***How to Use***
Test Survey	Formal-External
Test Evaluation Report	Formal-Internal

Test Case

The test case will be implemented in Rational Requisite Pro and document inputs, execution conditions, and expected results for a given path through a Use Case. The configuration management of this artifact will be through Requisite Pro.

Test Procedure

A test procedure is a set of detailed instructions for the set-up, execution, and evaluation of results for a given test case (or set of test cases). Test procedures will be captured in Requisite Pro. The configuration management of this artifact will be through Requisite Pro.

Test Packages

A test packages is used to contain test classes. It is documented in the Design Model. The configuration management of this artifact will be by the Rational Rose category file.

Test Classes

Each test class provides a realization of one or more test procedures. Test classes will be documented in the Design Model and built during Implementation. The configuration management of this artifact will be by the Rational Rose category file.

Test Components

The test components represent the realized test procedures in Implementation. These can be source code files, test scripts, or a combination. They will be documented in the Component View of Rational Rose. The configuration management of this artifact will be by the Rational Rose category file.

Test Subsystems

The test subsystems are used to group the test components into more manageable pieces. They will be documented in the Component View of Rational Rose. The configuration management of this artifact will be by the Rational Rose category file.

Defects

A defect is a product anomaly. A defect can be any kind of issue you want tracked and resolved. Defects will be documented and maintained in a defect-tracking tool.

Test Plan

The test plan contains information about the purpose and goals of testing within the project. It is maintained in a Word document. This document will be time stamped.

Test Survey

The test survey includes the test plan and a report of all test cases and test procedures. It is maintained in a Word document using a SODA template. There will be no configuration management of this artifact. This document will be time stamped.

Test Evaluation Report

The purpose of a test evaluation report is to provide information on the results of running a series of test cases. It is maintained as a Word document and may be produced by testing tools. There will be no configuration management of this artifact. This document will be time stamped.

Workflow

No changes from the suggested Objectory Work Flow.

Reference

Objectory 4.1

This is the version of Rational Objectory that is the basis for this development case.

Glossary

To be skipped:	This artifact is not required in this specific process.
Casual:	This artifact is generated and used as work information, it is not reviewed and approved by anyone. It is often a temporary artifact that is discarded after project.
Informal:	The artifact is used in the process but informal; it is not delivered, and maybe never defined on paper, but someone has approved it.
Formal-Internal:	The artifact is formally produced and used internally at some milestone.
Formal-External:	The artifact is part of a delivery at some milestone, and requires some form of approval by the buyer, the customer, or some other external stakeholder.