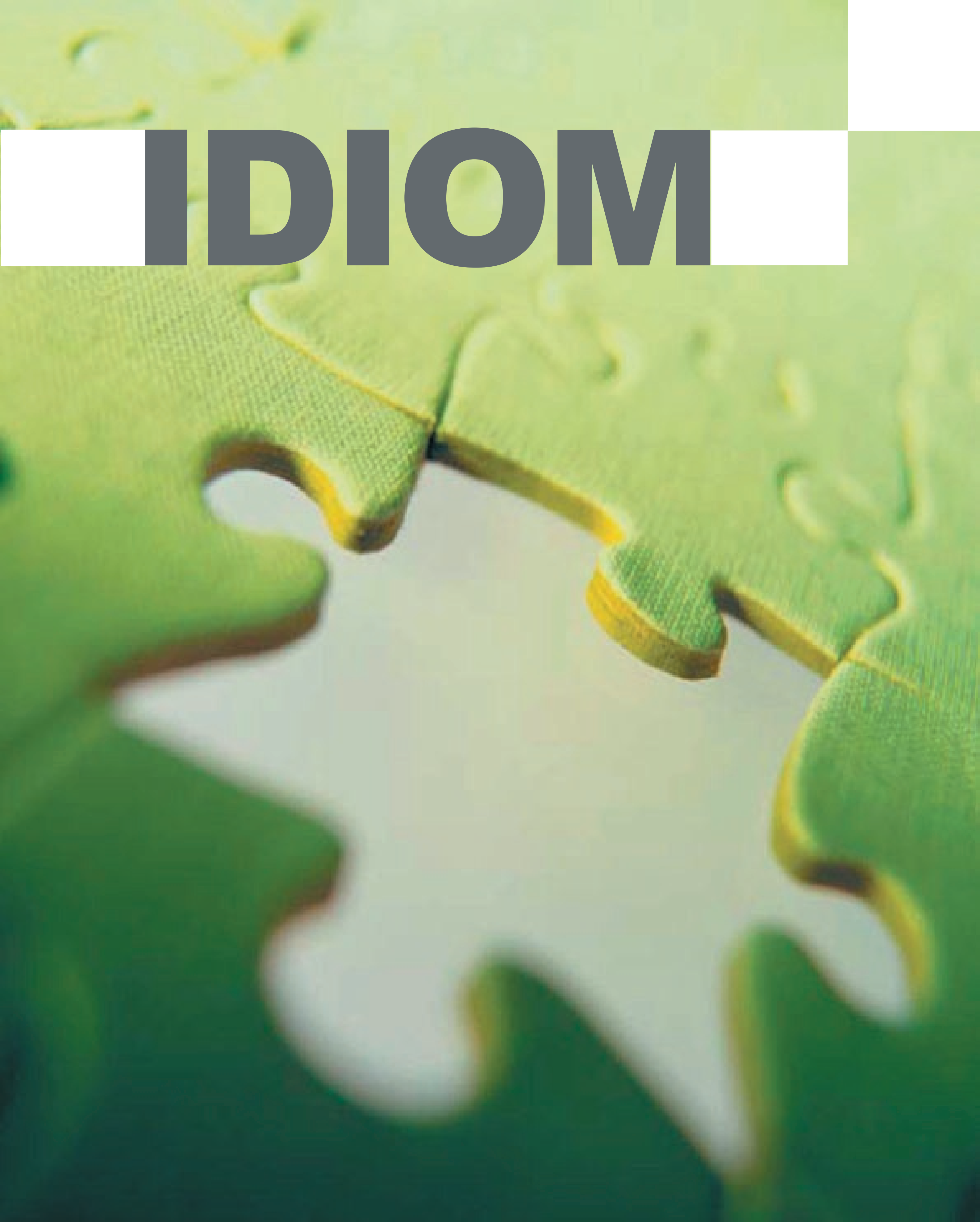
IDIOM



IDIOM and Rational Unified Process

Introduction

The purpose of this paper is to describe how business rules are managed within the Rational Unified Process (RUP) and to outline how IDIOM can be integrated with RUP.

A brief outline of RUP is provided, followed by a description of the capture, use and management of business rules within a RUP software development project. Finally, the integration of IDIOM into a software development project using RUP is described. This description includes the benefits provided by the use of IDIOM.

Rational Unified Process

The Rational Unified Process[1] is a software development process developed by the Rational Corporation that is intended to help ensure the success of large software development projects.
Key features of RUP are:

- Architecture centric
- Use Case driven – the core mechanism for capturing requirements in RUP is the use-case
- Iterative and incremental – RUP based projects are developed in a series of controlled iterations
- Uses UML as a representation tool

The architecture of a system is the blueprint for the application. It defines the overall structure of an application and determines how the functional requirements of the system will be satisfied.

Use case modeling is a method of capturing the functional requirements of the system. A use case captures requirements as a sequence of actions that the system will perform to yield an observable result to a particular actor.

Software development projects that use RUP are executed as a series of iterations or mini-projects that incrementally build up the capabilities of the system under development. Within each iteration, the process defines four phases and allocates the major workflows amongst these phases. **Figure 1** provides a summary of these phases and the allocation of the workflow amongst the phases. As can be seen, the two workflows that are of principle interest in capturing business rules – Business Modeling and Requirements capture – have the bulk of their activity performed in the Inception and Elaboration phases.
The business rules are converted into executable code during the Construction phase, primarily via the Analysis & Design and Implementation & Test workflows.

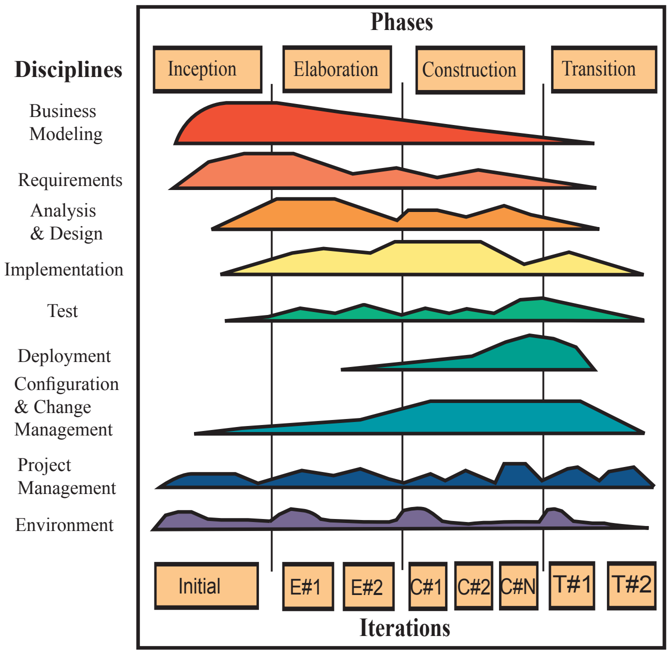


Figure 1 Rational Unified Process Phases

Unified Modeling Language

The Unified Modeling Language (UML) is a graphical object-oriented modeling language, originally developed by Rational and since adopted as a standard notation by the Object Management Group (OMG)[2]. UML defines a number of standard diagram types. These are summarized in **Table 1**. UML is an integral component of RUP. It is used to express the results of many of the RUP activities, including:

- Business Process Modeling
- Business Object Modeling
- Requirements capture via use case modeling
- Architecture
- Software Design

Business Rules within RUP

This section provides a brief description of the place of business rules within RUP. It concentrates on two major areas within the Rational Unified Process in which business rules are captured. These are the Business Modeling and Requirements workflows. In addition, the impact of business rules on the Design, Implementation and Test workflows is also briefly described.

[1] The Unified Software Development Process - Ivar Jacobson, Grady Booch, James Rumbaugh; Addison Wesley, 1999

[2] http://www.omg.org

Table 1 UML Diagram Types

Diagram	Purpose
Use Case	Model the relationship between actors and the use-cases of the system
State	Model the lifecycle of an object, including its possible states and the transitions between states
Class	Model the major entities in the system and their static relationships
Object	Model snapshots of the object instances of a system at a particular point in time
Sequence	Model the dynamic interactions between objects of the system
Collaboration	Same as sequnce diagram but with a different view
Activity	Model the activities that occur during the realization of a use case scenario
Component	Model the physical implementation of a system in terms of components and their dependencies
Deployment	Model the allocation of software components to hardware resources

Business Modeling

The purpose of the business modeling activity in RUP is to:

- ❑ Capture the structure and dynamics of the business for which the system is being developed
- ❑ Derive the functional requirements for the system
- ❑ Understand problems and identify potential improvements in the business process

There are several artifacts produced during business modeling. Of particular interest from a business rules perspective, are:

- ❑ Business Glossary – Defines terms that are important to the business
- ❑ Business Rules – Declarations of policy and constraints
- ❑ Business Object Model – Identifies the core entities of interest to the business and the relationships between them.

The RUP Business Rules artifact is captured as a single document containing the definition of the business rules. RUP allows business rules to be specified in the business rules document using a number of options. These include:

- ❑ Object Constraint Language (OCL)
- ❑ Near English
- ❑ UML models

The RUP *Business Rules Guidelines* provides guidelines on how to express business rules graphically in the various UML models.
A summary of how to express the various types of business rules using different UML diagrams is provided in **Table 2**.

In practice, using UML class diagrams to express business rules is not very effective. This is a problem of object-oriented (OO) design as much as a problem with UML. Amongst the major design principles of OO design are the principles of abstraction and encapsulation. A class is an abstract representation of some concrete entity (e.g. Customer) and encapsulates the data related to the abstraction in question, along with the methods or operations that manipulate that data. There are many instances of business rules that do not fit cleanly within a single class. For example, a business rule calculation may require data from two or more classes. Following good OO design principles, the definition of such business rules will end up being spread amongst multiple classes. This makes it harder to verify the correctness of the business rule.

Another problem with using UML to capture business rules is that UML diagrams in themselves are not adequate to express all business rules. To help alleviate this, the OMG has adopted the Object Constraint Language (OCL).

Table 2 Business Rules expression in UML

Rule Type	Model	How to express
Stimulus and response	Activity	As decisions and conditional paths
Action constraint	Activity	As conditional paths
Structure constraint	Class	Constraints on associations between classes
Inference	Activity	As alternative paths and activities
Computation	Class	Methods and relationships between classes in the object mode

The OCL is a declarative language that allows constraints to be added to UML models to help remove ambiguity from the diagrams. OCL expressions use a formal syntax to specify constraints.

Requirements Capture

The second major RUP workflow that requires the capture of business rules is the Requirements workflow. The purpose of the requirements workflow includes:

- Establishing what the system is to do
- Establishing the boundaries of what the system is to do.
- Communicating the requirements of the system to the developers

There are two major categories of requirements for any system. These are:

- Functional requirements – the business functions and features the system must provide
- Non-functional requirements – attributes of the system that are not functional but affect how the system is to be developed. These include quality attributes, such as performance and scalability, and other technical, regulatory or legal constraints.

The non-functional requirements are captured in a supplementary requirements specification. RUP mandates that functional requirements are captured using use case models. Use case models capture requirements as a series of use cases and use case diagrams, which depict the relationships between the various use cases and the actors who interact with them. As mentioned earlier, a use case captures requirements as a sequence of actions that the system will perform to yield an observable result to a particular actor. In addition to the primary scenario, a use case may also include action sequences for alternative scenarios. Use cases are documented as textual documents, following a standard template. **Table 3** describes the standard set of properties used in a use case description.

As seen, the RUP use case documentation does not make any explicit provision for documenting or associating business rules with the use case. One possibility is to document the business rules in the special requirements area of the use case. There are shortcomings in this approach, however. One of the biggest problems is that in many cases a business rule applies to more than one use case. These shared business rules do not belong to any single use case. Either they are stored in a single repository, such as the business rules document, and referenced from multiple use cases, or the rules are defined in multiple use cases.

Analysis and Design

The Analysis and Design workflow is concerned with the analysis of the functional and non-functional requirements of the system and the subsequent design of an architecture and software components that will satisfy the requirements. RUP does not provide an explicit activity for mapping business rules to the design. However, it is standard practice for the design of components to include any relevant business rules.

Implementation

Subsequent to the Analysis and Design workflow is the Implementation workflow. The primary activities of this workflow are the construction and integration of the software components as identified in the Analysis and Design workflow. In theory, this workflow does not require any explicit knowledge of the business rules other than how to implement them, as specified in the component design. In practice, there is generally a significant amount of refinement of business rules performed during this workflow.

Table 3 Properties of a use case

Property	Decription
Name	Name of the use case
Description	Short summary description of the purpose of the use case
Pre-conditions	Constraints that must hold for the use case to be invoked
Post-conditions	Constraints that must hold upon the completion of the use case
Extension points	List of points within the main and alternative flows where behavior can be extended by extension use cases
Relationships	Relationships between the use case and other use cases and actors
Special requirements	Requirements of the use case that are not considered part of the use case model but impact the design or implementation
Main flow	Textual description of the flow of events for the main scenario of the use case
Alternative flows	If there are alternative flows through the use case, they are documented as alternative sequences
Activity diagrams	UML activity diagrams can be used to provide a graphical representation of the primary and alternative flows
Supplementary diagrams	Other diagrams that may help the understanding of the use case. This can include screen prototypes.

Test

The major activities of the test workflow are the design, implementation and execution of tests to ensure the constructed software meets its requirements. The various requirements documents and models, including the business rules document, are used as inputs to the design of the tests. The functional tests must verify that the business rules are correctly implemented.

Using IDIOM with RUP

This section provides a description on how to use IDIOM to manage business rules within a RUP software development process. Use of IDIOM during the major workflows of RUP is described, along with the subsequent benefits for the project.

Business Rules Capture

As described in the previous section, business rules are considered an important part of RUP. However, neither of the two modeling technologies used within RUP - use case and UML based OO modeling – are entirely suitable for capturing business rules.
To summarize the problems with business rules capture in standard RUP:

- Lack of single, unified method of expressing business rules
- Shared business rules that apply to more than one requirement
- Definition of business rules using UML class diagrams results in fragmented business rule definitions, which are difficult to verify

IDIOM can be used to manage the Decisions within a RUP process during both the business process modeling and the requirements modeling workflows. The major change to RUP to allow the use of IDIOM is to use the IDIOM Decision Manager exclusively to capture the Decisions. The facts and terms of the business process model can continue to be captured using a business object model.

During the business process modeling workflow, instead of capturing business rules in a business rules document, Decisions should be entered directly into the IDIOM Decision Manager. A pre-condition of being able to do this is that a Business Object model has been defined and made available to the Decision Suite as an XML schema. This increases the importance of the business object modeling activity.

Capturing Decisions in the IDIOM Decision Manager provides a single unified repository of the business Decisions. Furthermore, there is a single consistent notation that is used for all specifiable Decisions. In contrast, the standard RUP technique can result in a document (or several documents) with business rules expressed using a conglomeration of different UML model diagrams.

During the requirements workflow of RUP, the Decisions captured in the IDIOM Decision Manager can be elaborated and refined as new knowledge is gained. Furthermore, new Decisions can be added to the repository as they are discovered. Rather than duplicating the Decisions in the use case model of the system, each use case should provide a cross-reference to applicable Decisions in the IDIOM repository.

Finally, during the software construction workflow, the Decisions captured in IDIOM can be automatically converted into executable code and used directly in the application via the IDIOM Decision Engine. This eliminates the risk of the implemented Decisions diverging from their specification.

Business Object Model

The IDIOM Decision Suite relies on the presence of a business object model defined using an XML schema. This corresponds very closely to the business object model defined during the business modeling workflow of RUP. UML class diagrams can be used to model the business objects. There are several possibilities for automatically transforming a business object model, represented as a UML class diagram, into an XML schema suitable for use with the IDIOM Decision Suite.

The first method was originally designed as a method of using UML to design XML schemas. It is described in a paper available from the Rational web site[3]. This method relies on modeling the class diagram in a UML tool that supports the export of model information in XMI format. XMI is an XML based model information interchange format defined by the OMG[4]. The basic technique is to create a class diagram using a special set of UML stereotypes to delineate how the classes are to be transformed into the schema definition. The next step is to export the model as XMI. The XMI representation of the model must then be transformed into the XML schema definition. There are several ways to carry out this step, but the usual approach is to use an XSLT[5] processor to perform the transformation using XSLT stylesheets.

The drawback to the approach discussed above is that its design intention is to allow XML schema definitions to be modeled and designed using UML, rather than allowing general UML business object models to be transformed into an XML schema. This is highlighted by the requirement to annotate the model with XML schema specific stereotypes and tagged values. A more general approach would allow any UML business object model to be transformed into an XML schema, without requiring the model to provide any special information to control the production of the schema. Fortunately, version 2.0 of the XMI specification includes production rules that detail how to generate XML schemas from UML models.

Requirements Management

For any reasonably complex system there will be a large number of interrelated requirements. A necessary management task is identifying dependencies of the requirements and determining the impact of a change in a requirement. There is a workflow in RUP for managing changes in requirements. This workflow contains an activity for dependency management. In addition, commercial requirements management tools exist that help with these activities. One such tool is Rational RequisitePro. It is used to build a database of all features and requirements and the dependencies between them. It produces a traceability matrix that can be used to determine and assess the impact of a change to a particular requirement.

It is desirable to be able to determine the impact of Decision changes on the requirements management activity and tools. At this stage, the IDIOM Decision Repository does not directly integrate with any of the commercial requirements management tools. However, the requirements management tools can perform their function as long as they have a list of the Decisions that are maintained in the IDIOM Repository. This can be done by copying the high level Decision names from the Decision Manager directly into the requirements management tool. A necessary part of the rules capture process will be to update the requirements tool database whenever the IDIOM Decision repository is altered. It will then be possible to determine the impact on the system of any change to a Decision.

Analysis and Design

The use of the IDIOM Decision Suite in the Analysis and Design workflow is quite straightforward, but has a significant impact on the overall project. The integration of the IDIOM Decision

[3] UML for XML Schema Mapping Specification - Grady Booch, Magnus Christerson, Matthew Fuchs, Jari Koistinen; Rational Software Corporation, 1999.
http://www.rational/com/media/uml/resources/media/uml_xmlschema33.pdf

[4] XML Metadata Interchange (XMI) Specification, Version 2.0 - The Object Management Group, May 2003 07-21
http://www.omg.org/cgi-bin/apps/doc?formal/03-05-02

[5] XSL Transformations (XSLT) Version 1.0 World Wide Web Consortium, 16 November, 1999
http://www.w3.org/TR/xslt

Engine into the software architecture must be established. The IDIOM Architecture Briefing provides a summary of how this can be done and details the subsequent impact of this activity on the overall quality of the resulting system.

The design of the software components is greatly simplified when using the IDIOM Decision Suite. Rather than having to design the implementation of the business rules, the software designer(s) only have to make provision in the component design for the invocation of the Decision Engine at the appropriate points of execution. It is not even necessary to know what Decisions will exist at this stage.

Implementation

The major activity of the implementation workflow that is concerned with the IDIOM Decision Suite is the integration of the Decision Engine into the application. The implementation of the business object model in a manner that allows business objects to be transferred between the application and the Decision Engine is also performed in this workflow. The bulk of the software can be implemented without any knowledge of the business Decisions. The Decision Engine completely insulates the application from any knowledge of the Decision implementation.

The IDIOM Decision Generator automatically generates executable code from the Decisions defined in the repository. The generated code can be integrated with the main application at deployment/build time. It is not needed while the application code is being developed. This allows a high degree of parallelism to be achieved in the capture of business Decisions and the design and implementation of the application code, leading to a reduction in risk and development time.

Test

IDIOM assists in the test workflow. First, by separating the business Decisions from the remainder of the application, they can be tested separately. Secondly, the IDIOM Test Executive allows the Decisions to be tested independently from the application. Application tests can be designed without having to consider validating the correctness of the business Decisions. This simplifies the testing process and shortens the testing cycle.

Summary

The capture of business rules or Decisions is an integral part of the Rational Unified Process (RUP). The two main workflows where business rules are captured are the business process modeling and requirements modeling workflows. The IDIOM Decision Manager can easily be integrated into RUP to ease the capture of business rules. The benefits of using IDIOM for business rules management within a project using RUP include:

- ❑ Single repository of all business Decisions.
- ❑ Enforces best-practice of business object modeling.
- ❑ Provides single consistent notation for all Decisions.
- ❑ Eliminates the problem of business Decisions being scattered amongst multiple use cases.
- ❑ IDIOM business Decision notation does not obfuscate the Decisions. They are easily verifiable by business users.
- ❑ Decisions can be directly converted into executable code, eliminating the risk of errors being introduced in the construction phase.
- ❑ Functional testing of the software is simplified by being able to separate the testing of business Decisions and the application logic.