Requirements-based Unified Modeling Language™ (UML™)

A Borland White Paper

By Joseph D. Schulz, Product Line Solutions Manager

September, 2003
Contents

Introduction ......................................................................................................................... 3
  The dreaded “M” word .................................................................................................... 3
  Unified Modeling Language™ ....................................................................................... 4
  Use case models ............................................................................................................ 4
  Requirements based UML™ ......................................................................................... 5

UML™ Overview .............................................................................................................. 6
  What is UML™? ............................................................................................................ 6
  A typical UML™ process ............................................................................................. 7

UML™ requirements ........................................................................................................ 7
  UML™ requirements example ..................................................................................... 8
  Benefits of UML™ requirements ............................................................................... 9
  Drawbacks to UML™ requirements .......................................................................... 10

Requirements-based UML™ (RBU) overview ................................................................. 11
  What is RBU? ............................................................................................................. 11
  Typical RBU process ................................................................................................. 12
  RBU Example ........................................................................................................... 17
  Benefits of the RBU approach .................................................................................. 21

References ..................................................................................................................... 22
Introduction

The dreaded “M” word

Every project needs one and every developer follows one, whether formal or informal, prescribed or ad-hoc. Unfortunately, for most IS professionals the word “methodology” invokes dreadful images of the worst kind. It often implies reams of unnecessary work, impossibly rigid standards, and lots of wasted time. When the average developer hears the dreaded word, they usually assume that the related project is doomed.

Of course, just the opposite is true. A development project that doesn’t actively use some sort of methodology has relatively little chance of success. If such a project does succeed, it is merely through coincidence or sheer dumb luck. These are the kinds of projects that ramble about generating lots of paperwork but relatively few measurable results. They miss every major deadline because they change directions so frequently, and usually require large quantities of rework to “fix” previous mistakes. In a project without a methodology, there is usually no such thing as a frozen deliverable, so consequently there are no deliverables.

So, then what exactly is a methodology? In its simplest form, a methodology is a set of steps to accomplish a task. That’s it. No fancy buzzwords or expensive terminology, just a set of steps. It is a plan that describes each task and its sequence relative to the others. After all, any job worth doing is worth planning for. As the old military adage goes, “If you fail to plan, you are planning to fail.”

Of course, a robust methodology can also include many other components. Strictly speaking, a complete methodology includes not only task descriptions but also supporting components like task standards, technique guidelines, deliverable outlines, and quality metrics. These items, though, are merely present to supplement the basic purpose of the methodology, which is to identify and prioritize the work to be done. That is, to describe the set of steps needed to accomplish the goal.
Unified Modeling Language™

The latest emerging industry-standard in the object-oriented methodology arena is the Unified Modeling Language™, commonly referred to as UML™. UML is a software modeling standard managed by the Object Management Group™ (OMG™), an industry consortium of software companies. Prior to the OMG, each company or consultant authored proprietary competing methodologies and realized the significant benefits of a truly global standard for Object Oriented Analysis and Design (OOAD). By combining much of their previous work, the UML standard was born.

Of course, UML is not actually a methodology. Rather, it is a notational standard that can be used to implement the tasks within a methodology. By having a common notation, methodology and tool vendors can easily develop complementary solutions without requiring retraining of the workforce. While the disparate UML-based methodologies may define differing sets of tasks, the techniques all employ the same graphical constructs.

Use case models

The UML specification includes graphical notations for many different diagram types, with most being optional steps based on the complexity of the application being developed. Relatively speaking, the “first” deliverable described in the UML notation is the Use Case diagram. A Use Case diagram graphically depicts the interaction between system users (i.e., “actors”) and system functions (i.e., “use cases”). Subsequent UML diagrams build on this basic information to identify the system components, methods, and packages necessary.

Unfortunately, this approach that begins with use cases creates a glaring deficiency in most UML-based methodologies; that is, they don’t address the business-oriented application requirements. Instead of first defining the purpose and objectives for the development effort, UML methods begin by jumping directly to the software functions. This presupposes that the use case participants already know why they need a system and what the optimal solution should look like.
In reality, the most important part of any systems development effort is to first establish a clear understanding of the problem so that potential solutions can be effectively weighed. To do this, the key business requirements must be defined, including the return-on-investment justification for each. Once this objective baseline is established, proposed alternatives can then be measured to determine which best solves the stated problem. Without this requirements analysis, a UML-based approach may only help to deliver the wrong application faster and cheaper.

**Requirements based UML™**

One possible solution to this problem is the use of “Requirements-Based UML” (RBU). RBU is a structured approach for incorporating business-oriented requirements analysis into a UML-centric development method. It balances the need for non-technical business analysis against the need for the structured technical approach defined in UML. Furthermore, it identifies business requirements analysis as a precursor to software-centric use case modeling efforts.

RBU also relies on a more natural, textual format for requirements deliverables. Non-technical staff members are generally more comfortable with words than diagrams, so RBU business requirements are defined in sentences and paragraphs. These textual descriptions are then related to the graphical objects defined in the UML deliverables.

After the first level of UML diagrams is completed (use case models, collaboration diagrams, etc.), the requirements are refined into more detailed textual technical specifications. In turn, these specifications are then related to the next round of UML diagram objects. This process of textual requirements leading UML modeling can continue to whatever level of detail is appropriate for the specific project.

By using this alternating approach with requirements and diagram objects, a more complete analysis and design model is produced. This provides a clearer picture of the application environment, including not only answering the “How?” questions for the application but also clarifying the “Why?” and “What?” as well. All too often development teams are eager to rush into coding and the latter two questions remained unvisited. It is these types of projects...
that are most often cancelled or rejected by the customers because they provide little business value.

**UML™ Overview**

**What is UML™?**

The Unified Modeling Process (UML) is a common notation for structured modeling within an OOAD framework. It was originally developed by several of the leading OOAD methodologists as a means to help standardize the types and format of deliverables produced by the competing OOAD methods. While not strictly a methodology itself, UML describes the notation that methodology outputs employ.

The current UML notational standard addresses the system analysis, design, and deployment steps in a development lifecycle. A new draft standard of UML, v2.0, is currently in RFI review and will extend the current standard to include a range of other activities. The most notable addition expected in v2.0 is a common notation for business process redesign.

*Figure 1: Traditional UML process flow*
A typical UML™ process

A UML-based development methodology usually involves a series of graphical models which are used to define the functional and technical aspects of an application system. Each model depicts a diagrammatic representation of one aspect of the application and is integrated with the other model objects. These models are then used as the component specifications for the construction phase of the project.

As shown in the graphic, the first and primary model developed in most UML-based methods is the Use Case diagram. Use Case diagrams are used to identify the external system boundary for an application by depicting the system functions (“use cases”) that external entities (“actors”) are able to interact with. Use Case diagrams are generally developed in very close collaboration with the application’s ultimate customers or sponsors.

These Use Case diagrams are often then more fully described by the creation of either Object Sequence diagrams and/or Object Collaboration diagrams. Both of these diagram types serve to more fully describe the Use Case by including the nature and order of each of the major work steps within the Use Case. Together, some combination of these three diagrams provide the functional application requirements for a system.

UML™ requirements

In the context of a UML-based method as outlined above, the term “requirements” generally refers to a set of technical specifications that describe the software features in an application. These requirements are imperative statements of functionality that must exist in the developed code and are written as “Plain Language” textual sentences or paragraphs. This deliverable is often named the “System Requirement Specification” (SRS).

Most often, these “UML requirements” included in the SRS are developed as extensions of a Use Case diagram. For each Use Case defined, the complete set of mandatory characteristics is identified and documented in clear, concise language. Modelers will then use these
requirement definitions to help complete and validate the systems design models, ensuring coverage of all required functionality.

The SRS generally includes both functional and non-functional requirements. Functional requirements state a capability that invokes or performs an actor-oriented transaction. Non-functional requirements, on the other hand, state a characteristic of the application which limits or bound a designer’s ability to develop a solution. Non-functional requirements usually include information about traits like performance and capacity limits, security rules and responsibilities, and/or technological considerations.

**UML™ requirements example**

In the example pictured below, a Use Case has been named “Enter Product Order.” This Use Case would exist on one or more Use Case diagrams and would be detailed with the inclusion of a Use Case narrative (e.g., pre-conditions, post-conditions, etc.). In the diagram, the appropriate actor(s) would also be associated to the Use Case.

As part of the transition from system analysis to system design, UML requirements would then be defined for this Use Case. These requirements would itemize the specific features necessary in the software in order to fully accomplish the “Enter Product Order” Use Case. As mentioned above, this might include both functional requirements and non-functional requirements.

One such UML requirement for the “Enter Product Order” Use Case might be the statement that “The system shall include a menu option to add new customer orders for saleable products…” As a result of this requirement, when the user interface class is developed in the Class diagram, the system designer will know to include a method to invoke this process. This may also be reflected in the appropriate State Transition diagram(s) as an event which triggers a change in state.
Benefits of UML™ requirements

The advantage of developing a structured SRS which includes both models and textual requirements is twofold. First, the textual statements are often more communicative than UML notation to non-technical customers as they provide a written description of the functionality that anyone can read. The graphical notations can often be daunting for an uneducated customer, and so the textual descriptions are more comfortable for those without any previous UML training.

Secondly, the textual requirements provide a place to document software features that may not be readily apparent or do not exist in the graphical models. For example, non-functional characteristics like hardware constraints are difficult to include in UML models because they are typically global issues that cannot be incorporated into the description of just one model object.

So, UML requirements become the document-centric “bridge” between the graphical system analysis deliverables (Use Case diagrams, etc.) and the graphical system design deliverables (Class diagrams, etc.). Most often these requirements are managed with a word processing
Requirements based Unified Modeling Language™ (UML™)

application and reviewed and approved in document format. This comfortable paradigm mimics traditional document-oriented analysis techniques.

**Drawbacks to UML™ requirements**

However, there are also disadvantages to limiting the requirements process to technical feature descriptions. First and foremost, by beginning the analysis process with Use Case definitions, the focus is immediately on the design of the software. Since Use Cases describe systemic solutions to problems, the derived UML requirements will address only the systemic characteristics as well.

With this approach, the only solution that can be developed will be one that can be automated with an application. This virtually ignores the relevant business issues that may be all or part of the problem as well. Often a minor business process redesign (like job function reorganization) can facilitate a more efficient application or even eliminate the need for an application at all.

Also, UML requirements tend to include a lot of technical language since they are describing technical features. This is typically because they are written for the development organization to use as an input to the system design process. However, another goal of a structured requirements analysis is to validate the system analysis deliverables and the customers needed to do this are often non-technical. So, the personnel with the appropriate business knowledge may not be able to adequately understand the requirements definitions.

Finally, another problem with UML requirements is that they tend to focus on one business transaction at a time. Since they are most often derived from Use Cases, the requirements are documented with an eye toward that one transaction and often ignore the business workflow surrounding it. Without a highly structured reuse analysis, it is often possible to end up with highly efficient transactions that contain a lot of business redundancy between them.
**Requirements-based UML™ (RBU) overview**

**What is RBU?**

Requirement-based UML (RBU) is a structured approach for integrating formal requirements analysis into a UML-based analysis and design effort. It balances the need for non-technical business analysis against the need for the system-oriented approach defined in UML by including a multi-level requirements definition. Instead of just the technical feature descriptions captured in traditional UML requirements (see previous chapter), RBU defines multiple requirements deliverables with a specific focus for each. Simply put, requirements management becomes a lifecycle task that runs in parallel with the OOAD tasks.

As with UML requirements, RBU requirements deliverables are defined in a natural, textual format. This allows non-technical customers to more comfortably review and understand the requirements information. These textual descriptions are then related to the relevant graphical objects defined in the UML-based deliverables.

Often, development teams prematurely rush into the development of the application solution and ignore the larger business issues. This can easily lead to inappropriate or expensive
technological solutions. By using alternating “rounds” of modeling and textual specifications, the RBU approach helps to temper that tendency and delivers a richer, more complete picture of the business problem. In this manner, it helps to ensure a higher quality, more cost-effective solution.

In addition, the RBU approach addresses Quality Assurance as a lifecycle task as well. Requirements are thoroughly tested before any development is performed, detecting conflicts and omissions that would stall later development. At each stage, the QA/testing plan is refined and more detail is added until specific test cases have been identified. By developing the test cases from the requirements rather than the code, a more complete test harness can be established.

**Typical RBU process**

The RBU technique begins with a textual specification of all of the requirements for any solution to the business problem. These requirement statements define both the functionality required in the solution as well as the boundaries the solution must operate within. All of these requirement statements should be specified in a non-technical, “plain language” format. Ideally, the customers will define these textual requirements themselves without restatement by the development staff.

These requirements, usually referred to as “customer requirements,” should be defined without regard to how the application will look. Specifically, they should not reference menu options or screen formats. The intent is to capture a definition of the business process needed to completely solve the business problem. In essence, they answer the question of “What?” not “How?”.

Once the Customer Requirements are defined, they can then be used as the basis for developing the Use Case diagrams. Specifically, each functional requirement identified in the Customer Requirements will initially correspond to one Use Case if it can be automated. If it can’t be automated, then a manual transition plan will need to be developed.
While it may appear redundant to develop a one-to-one correspondence between Customer Requirements and Use Cases, it isn’t since further refinement will be performed on the Use Case diagram during system analysis. The mapping is only one-to-one at the beginning of this stage. Once the Use Case diagram is refined with “Extends” and “Uses” relationships, the mapping becomes a many-to-many relationship.

It is important, though, that this reuse analysis be performed on the Use Cases and not on the Customer Requirements. This is to avoid corrupting the real business requirements with artificial technological constraints. All too often users are forced to redesign their business process in order to accommodate technology rather than the reverse. As the saying goes, “Just because you know how to use a hammer, not every problem is a nail.” That is, define the customer requirements for a solution before a specific technology is applied.

As part of the transition to system design, both the Customer Requirements and the system analysis models are used to develop Functional Requirements. As with the Customer Requirements, the Functional Requirements are stated in a textual format. However, unlike the Customer Requirements, the Functional Requirements define the technical features of the
Requirements based Unified Modeling Language™ (UML™)

application rather than the business needs. They are used to define the “How?” for the application.

At this point, relationships should be established between the Functional Requirements and the previous deliverables. For example, each Functional Requirement should be an implementation of at least one Customer Requirement and at least one Use Case. These relationships are then analyzed to look for inconsistencies in the model. For example, if any Customer Requirement does not have at least one “child” Functional Requirement, there is a gap somewhere in the Use Case model. On the other hand, if there are Functional Requirements without at least one “parent” Customer Requirement, then the scope of the original project has been increased, either intentionally or unintentionally.

Figure 5: RBU functional requirements

Once the Functional Requirements have been fully defined and quality-checked, they are used in conjunction with the Use Case model to develop the Class model. As with the Functional Requirements, relationships should be established to the “parent” objects in the previous deliverables and used to look for inconsistencies and omissions in the Class model.
Finally, the system design and implementation models are then used to develop the application code itself. By using this “matrix” approach to building the UML deliverables, the resulting application is more complete and of higher quality.

If a formal business process model is desired, the RBU process can be extended to support this work as well. Although the UML specification does not currently include notation for business process models, there are many popular methodologies which do. For example, Activity diagrams and Context Diagrams are often used by many modeling tools to illustrate this type of work.

In these cases, additional diagrams are developed before the UML deliverables listed above are created. Both of these diagram types show the flow of work through an organization without regard for job titles and application boundaries. Their purpose is to identify and optimize the organizational process before an application system is designed.

Using the RBU approach, these business diagrams would be preceded by a set of requirement definitions. These requirements, called “Business Requirements” define the goals of the organization, including the objective metrics used to measure success. Business Requirements then become the guiding principles used to govern which possible business process is the most desirable.
As with previous requirements, Business Requirements are related to other requirements and the modeling objects. Specifically, Business Requirements should be related to the Customer Requirements necessary to accomplish the goals and to the processes in the business models that implement them. Again, these relationships can be inspected for inconsistencies before moving forward in the development lifecycle.

Finally, as mentioned earlier, the RBU method includes a third parallel activity for Quality Assurance and Testing. This set of work is performed by the QA organization and is used to detect flaws in the requirements and models deliverables and to develop the test harness used for verification of the application code. By deriving the test cases from the requirements in a progressive manner, the resulting test plan will be more complete and should validate both functional and operational performance.

At first glance, the RBU approach may seem to introduce additional work on the project team because there are more steps than in a traditional UML-based method. However, in the opinion of this author, the work that RBU dictates is not additional work, but rather it is a matter of formalizing work that is already being done with informal methods. By purposefully addressing these steps, the quality of the work will increase and productivity may actually improve. At the very least, the quality of the software product itself will be measurably higher.
RBU Example

To demonstrate the RBU technique, consider the example of the “Enter Product Order” Use Case shown in the previous chapter. How did the user and/or development staff conclude that this Use Case was necessary? How did they know which system functions would be appropriate to solve the business problem?

Most often, the answer to these questions is that the Information Systems staff asked the customer what the solution should be. This assumes, however, that the customer has the information and expertise necessary to make this decision. Very often that is not the case and dangerous assumptions are introduced into the development effort before a single line of code is written.

In an RBU-based project, the Use Case model would be preceded by a study of the concrete business drivers for the project. Specifically, an ROI analysis should be done to identify the rationale for the project and the expected gains to be achieved from a successful deployment. Such a study can be documented in the form of Business Requirements and organized by the primary business areas affected. In our example, there are four possible business areas that are part of a strategic initiative to reduce the cost of goods sold by 10%.

Figure 8: Sample RBU business requirements
Figure 9: Sample RBU customer requirements

For each of these four business areas, an event context diagram can then be used to identify the process requirements that are part of the business solution to the problem. The primary actors are illustrated and the context-level events and responses are defined. This context diagram can then be used as the high-level vision and scope document for our project.

The next step is to “decompose” the context diagram into the set of Customer Requirements which document the business process needed to solve the problem. In our example, the Customer Requirements would include information about not only the two automated tasks (“Record Order” and “Generate Invoice”) but also the two manual tasks that surround them (“Produce Quote” and “Ship Product”). Together, these four tasks identified in the Customer Requirements describe the complete business solution necessary to solve the stated problem. Now the development team has sufficient information to make an informed decision about which tasks can be automated and how best to implement them.
After reviewing these Customer Requirements, only two Use Cases are defined initially. These would be named “Enter Product Order” and “Generate Invoice” and would be related to the two Customer Requirements that are being automated. During the course of the system analysis, however, additional Use Cases might be identified in order to encapsulate reusable logic (e.g., “Calculate Shipping Charges”) or to extend the model for alternate courses (e.g., “Non-Profit Invoices”). These additional Use Cases would not be directly related to Customer Requirements but would instead derive their relationships through other Use Cases.

Figure 11: Sample RBU functional requirements
Once the Use Case model was completed, the Functional Requirements would then identify the functional and non-functional software features needed to automate the Use Case definitions. In our example, the “Calculate Shipping Charges” Use Case would be broken down into the discrete set of requirements necessary to completely specify the application functionality needed. This includes not only functional requirements but also the non-functional characteristics that need to be adhered to (e.g., security requirements, performance requirements, scalability requirements, etc.).

\[\text{Figure 12: Sample RBU class diagram}\]

Finally, these Functional Requirements would, in turn, be used to help define the objects in the more traditional system design models (e.g., Class diagram, State Transition diagram, etc.). Traditionally the work the of a “cowboy coder,” an RBU system design is now the culmination of a well-structured and highly organized application definition. Functional Requirements are typically packaged into a Systems Requirements Specification (SRS) document, and it is this document which is delivered to the application programmers for them to use as the guide when building the source code components.
Benefits of the RBU approach

There are many benefits to using the RBU approach instead of traditional UML-based methods that treat requirement as “features.” The most important is that a structured requirements-based approach to development will dramatically improve the level of communication between end-users and the development staff. By providing a non-threatening textual format for deliverables, customers without training in UML notation are able to participate in the application specification. The requirements documents produced will be easier to read and more likely to be reviewed by the appropriate customers. All of this means more feedback which will lead to higher quality deliverables.

Another major benefit is that RBU provides a facility to document the entire business solution, not just the automated subset of it. Where UML modeling techniques make the assumption that a systemic solution is available, RBU requirements do not. This, in turn, provides a much richer picture of the solution and allows the project team to make more informed decisions about what automated functionality can and should be included in the application.

In fact, business improvements are often suggested as a result of the RBU requirements analysis that have nothing to do with application development. These solutions would typically not even be discussed during a UML-based project. Just as often, applications cannot efficiently solve the root cause of the business problem being solved because it is not an automated problem. It is important to understand this before an expensive development project is launched and then later cancelled due to lack of substantive results.

Finally, a third major benefit of RBU is that the scope of the application can be managed based on customer needs rather than on software features. By using the requirements as an integral part of the change control process, change requests can be evaluated on the basis of the business improvement in the Business Requirements and/or Customer Requirements. Then, using the relationships established between the various RBU deliverables, the complete impact of a change can be determined before the change is approved.
References


