Analysis and Design with UML
Agenda

- Benefits of Visual Modeling
- History of the UML
- Visual Modeling with UML
- The Rational Iterative Development Process
What is Visual Modeling?

“Modeling captures essential parts of the system.”

Dr. James Rumbaugh

Visual Modeling is modeling using standard graphical notations

Computer System
Visual Modeling Captures Business Process

*Use Case Analysis is a technique to capture business process from user’s perspective*
Visual Modeling is a Communication Tool

Use visual modeling to capture business objects and logic

Use visual modeling to analyze and design your application
Visual Modeling
Manages Complexity

The human mind can only handle 7 plus or minus 2 things at once
Visual Modeling Defines Software Architecture

User Interface (Visual Basic, Java)

Business Logic (C++, Java)

Database Server (C++ & SQL)

Model your system independent of implementation language
Visual Modeling Promotes Reuse

Reusable Components

Multiple Systems
What the Iterative Life Cycle Is Not

- It is not hacking
- It is not a playpen for developers
- It is not unpredictable
- It is not redesigning the same thing over and over until it is perfect
- It is not an excuse for not planning and managing a project
- It is not something that affects only the developers on a project
What the Iterative Life Cycle Is

- It is planned and managed
- It is predictable
- It accommodates changes to requirements with less disruption
- It is based on evolving executable prototypes, not documentation
- It involves the user/customer throughout the process
- It is risk driven
Three Important Features of the Iterative Approach

- **Continuous integration**
  - Not done in one lump near the delivery date

- **Frequent, executable releases**
  - Some internal; some delivered

- **Attack risks through demonstrable progress**
  - Progress measured in products, not documentation or engineering estimates
Resulting Benefits

- Releases are a forcing function that drives the development team to closure at regular intervals
  - Cannot have the “90% done with 90% remaining” phenomenon

- Can incorporate problems/issues/changes into future iterations rather than disrupting ongoing production

- The project’s supporting elements (testers, writers, toolsmiths, CM, QA, etc.) can better schedule their work
Risk Profile of an Iterative Development

Risk

Time

Preliminary Iteration
Architect. Iteration
Architect. Iteration
Devel. Iteration
Devel. Iteration
Devel. Iteration
Transition Iteration
Transition Iteration
Post-deployment

Inception
Elaboration
Construction
Transition

Waterfall
Risk Management Phase-by-Phase

- **Inception**
  - Bracket the project’s risks by building a proof of concept

- **Elaboration**
  - Develop a common understanding of the system’s scope and desired behavior by exploring scenarios with end users and domain experts
  - Establish the system’s architecture
  - Design common mechanisms to address system-wide issues
Risk Management Phase-by-Phase (cont.)

■ Construction
  - Refine the architecture
  - Risk-driven iterations
  - Continuous integration

■ Transition
  - Facilitate user acceptance
  - Measure user satisfaction

■ Post-deployment cycles
  - Continue evolutionary approach
  - Preserve architectural integrity
Risk Reduction Drives Iterations

Initial Project Risks
Initial Project Scope

Define scenarios to address highest risks

Plan Iteration N
- Cost
- Schedule

Develop Iteration N
- Collect cost and quality metrics

Assess Iteration N

Revise Overall Project Plan
- Cost
- Schedule
- Scope/Content

Revised Project Risks
- Reprioritize

Risks Eliminated

Iteration N
Use Cases Drive the Iteration Process

Inception $\rightarrow$ Elaboration $\rightarrow$ Construction $\rightarrow$ Transition

Iteration 1 $\rightarrow$ Iteration 2 $\rightarrow$ Iteration 3

“Mini-Waterfall” Process

- Iteration Planning
- Rqmts Capture
- Analysis & Design
- Implementation
- Test
- Prepare Release

Copyright © 1997 by Rational Software Corporation
The Iteration Life Cycle: A Mini-Waterfall

- Results of previous iterations
- Up-to-date risk assessment
- Controlled libraries of models, code, and tests

Selected scenarios

Iteration Planning
Requirements Capture
Analysis & Design
Implementation
Test
Prepare Release

Release description
Updated risk assessment
Controlled libraries
Detailed Iteration Life Cycle Activities

- **Iteration planning**
  - Before the iteration begins, the general objectives of the iteration should be established based on
    - Results of previous iterations (if any)
    - Up-to-date risk assessment for the project
  - Determine the evaluation criteria for this iteration
  - Prepare detailed iteration plan for inclusion in the development plan
    - Include intermediate milestones to monitor progress
    - Include walkthroughs and reviews
Detailed Iteration Life Cycle Activities (cont.)

- Requirements Capture
  - Select/define the use cases to be implemented in this iteration
  - Update the object model to reflect additional domain classes and associations discovered
  - Develop a test plan for the iteration
Detailed Iteration Life Cycle Activities (cont.)

- **Analysis & Design**
  - Determine the classes to be developed or updated in this iteration
  - Update the object model to reflect additional design classes and associations discovered
  - Update the architecture document if needed
  - Begin development of test procedures

- **Implementation**
  - Automatically generate code from the design model
  - Manually generate code for operations
  - Complete test procedures
  - Conduct unit and integration tests
Test
- Integrate and test the developed code with the rest of the system (previous releases)
- Capture and review test results
- Evaluate test results relative to the evaluation criteria
- Conduct an iteration assessment

Prepare the release description
- Synchronize code and design models
- Place products of the iteration in controlled libraries
Work Allocation Within an Iteration

- Work to be accomplished within an iteration is determined by
  - The (new) use cases to be implemented
  - The rework to be done

- Packages make convenient work packages for developers
  - High-level packages can be assigned to teams
  - Lower-level packages can be assigned to individual developers

- Use Cases make convenient work packages for test and assessment teams

- Packages are also useful in determining the granularity at which configuration management will be applied
  - For example, check-in and check-out of individual packages

Copyright © 1997 by Rational Software Corporation
Iteration Assessment

- Assess iteration results relative to the evaluation criteria established during iteration planning:
  - Functionality
  - Performance
  - Capacity
  - Quality measures

- Consider external changes that have occurred during this iteration
  - For example, changes to requirements, user needs, competitor’s plans

- Determine what rework, if any, is required and assign it to the remaining iterations
Selecting Iterations

- **How many iterations do I need?**
  - On projects taking 18 months or less, 3 to 6 iterations are typical

- **Are all iterations on a project the same length?**
  - Usually
  - Iteration length may vary by phase. For example, elaboration iterations may be shorter than construction iterations
The First Iteration

- The first iteration is usually the hardest
  - Requires the entire development environment and most of the development team to be in place
  - Many tool integration issues, team-building issues, staffing issues, etc. must be resolved

- Teams new to an iterative approach are usually overly-optimistic

- Be modest regarding the amount of functionality that can be achieved in the first iteration
  - Otherwise, completion of the first iteration will be delayed,
  - The total number of iterations reduced, and
  - The benefits of an iterative approach reduced
There Is No Silver Bullet

- Remember the main reason for using the iterative life cycle:
  - You do not have all the information you need up front
  - Things will change during the development period

- You must expect that
  - Some risks will not be eliminated as planned
  - You will discover new risks along the way
  - Some rework will be required; some lines of code developed for an iteration will be thrown away
  - Requirements will change along the way