IBM Software Group | Rational Software France

Object-Oriented Analysis and Design with UML2 and Rational Software Modeler

PART II – Object-Oriented Analysis

Rational. software

e business on demand software

Table of Contents

IBM Software Group | Rational Software France

Object-Oriented Analysis and Design with UML2 and Rational Software Modeler

05. Introduction to RUP

e business on demand software

Success Rates of Software Development Projects

"Standish Group" CHAOS Chronicles	Year	Success Rate
First "Chaos" Report	1994	16 %
"Extreme Chaos"	2000	28 %
Last "Chaos" Report	2003	34 %

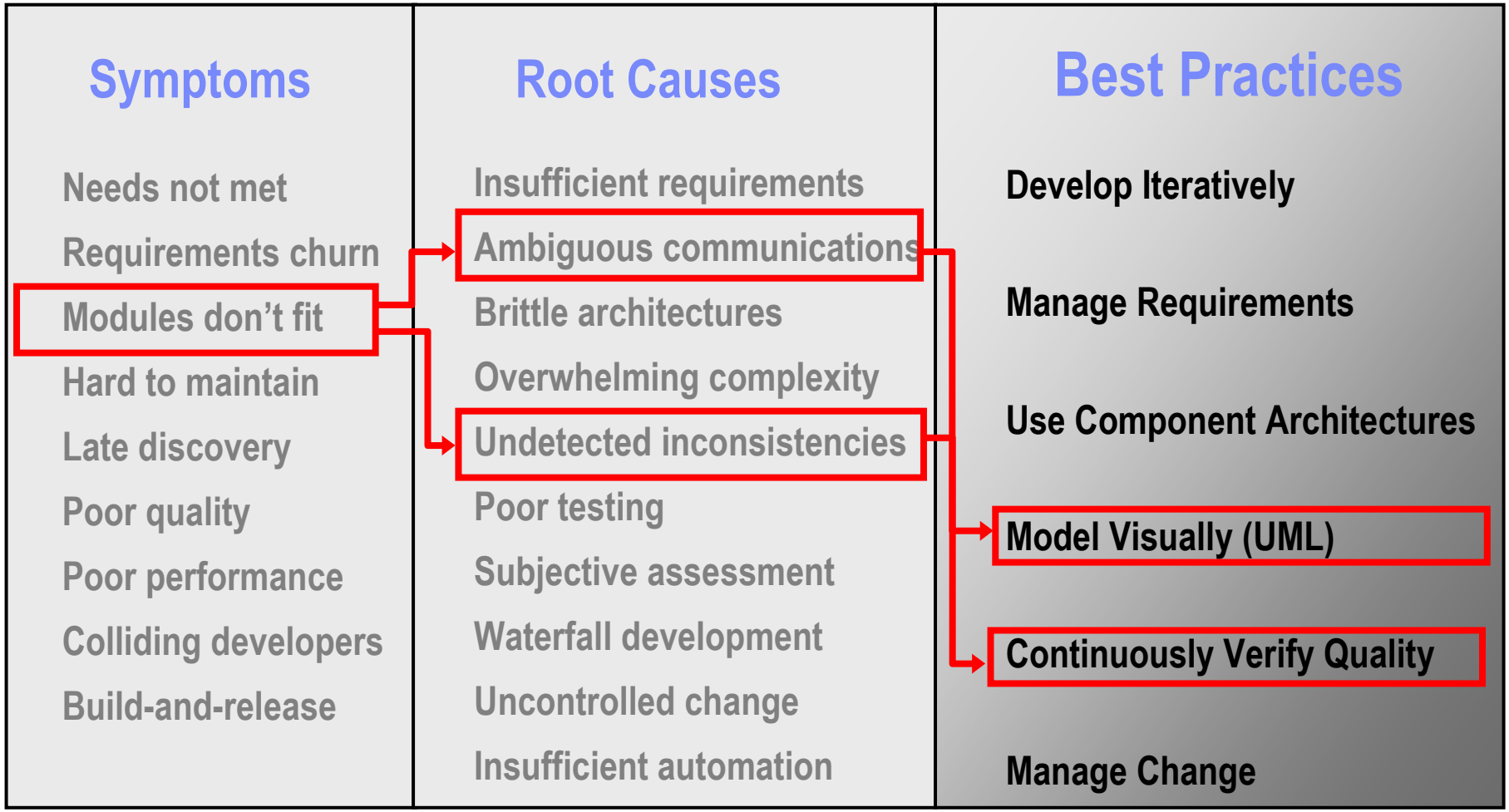
- Success = project delivered on time, within budget and meeting the needs of the users

"We know why projects fail, we know how to prevent their failure -- so why do they still fail?" - Martin Cobb

Symptoms of Software Development Problems

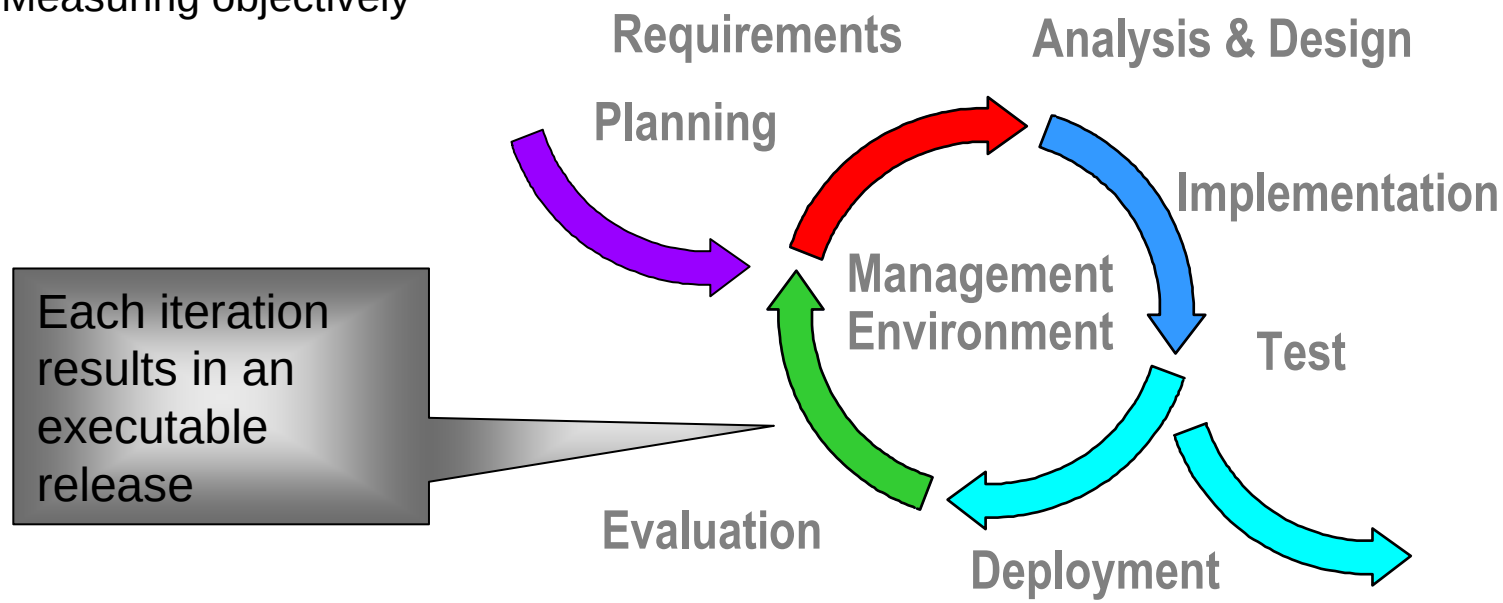
- ✓ User or business needs not met
- ✓ Requirements not addressed
- ✓ Modules not integrating
- ✓ Difficulties with maintenance
- ✓ Late discovery of flaws
- ✓ Poor quality of end-user experience
- ✓ Poor performance under load
- ✓ No coordinated team effort
- ✓ Build-and-release issues

Trace Symptoms to Root Causes



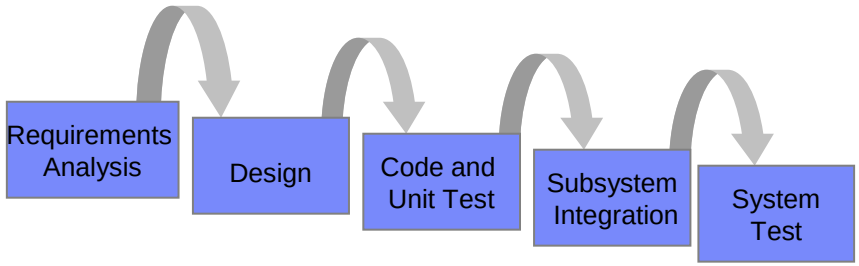
Definition of Iterative Development

- Iterative development = steering a project by using periodic objective assessments, and re-planning based on those assessments
- Good iterative development means:
  - Addressing risks early
  - Using an architecture-driven approach
  - Measuring objectively

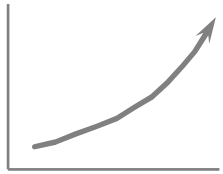


Contrasting Traditional and Iterative Processes

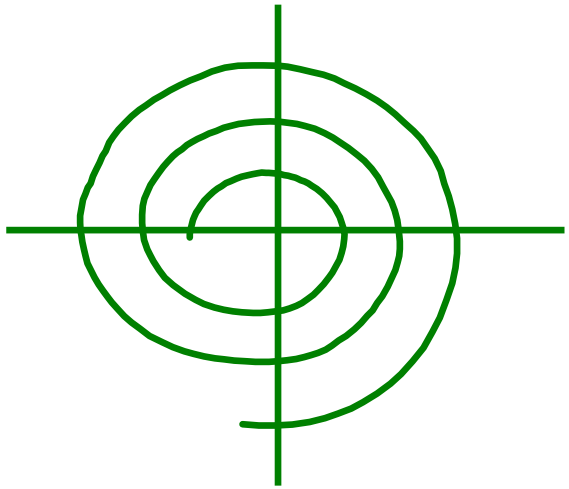
Waterfall Process



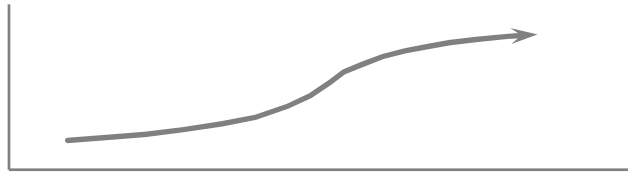
- Requirements-driven and mostly custom development
- Late risk resolution
- Diseconomy of scale



Iterative Process



- Architecture-driven and component-based
- Early risk resolution
- Economy of scale



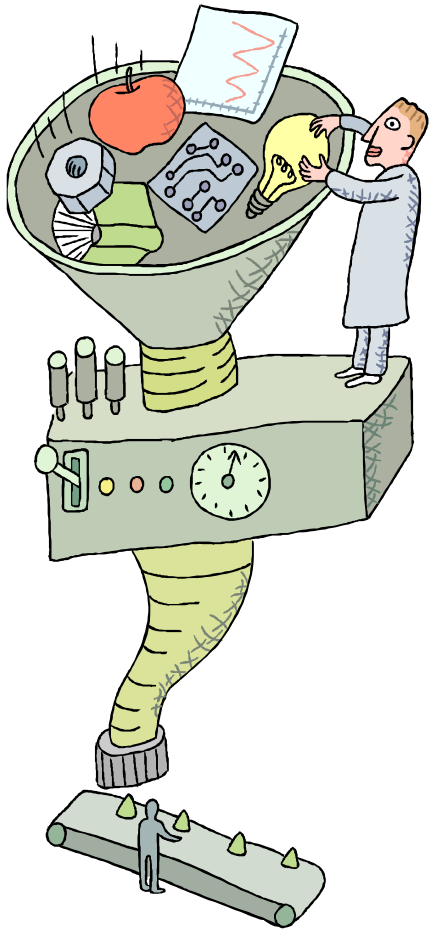
Iterations and Phases

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architecture Iteration	Architecture Iteration	Developme nt Iteration	Developme nt Iteration	Development Iteration	Transition Iteration	Transition Iteration

- **Inception**: To achieve concurrence among all stakeholders on the lifecycle objectives for the project
- **Elaboration**: To baseline architecture providing a stable basis for the design and implementation efforts in Construction
- **Construction**: To complete the development of the product
- **Transition**: To ensure the product is available for its end users

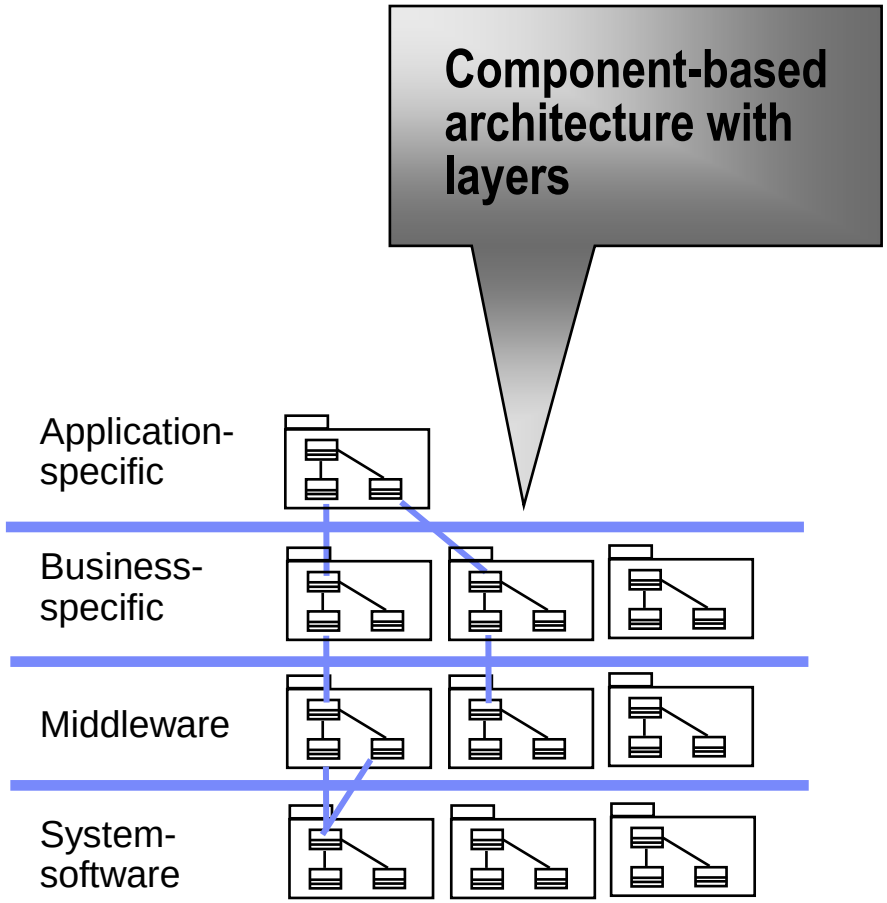
Managing Requirements

- Ensures that you:
  - Solve the right problem
  - Build the right system
- By taking a systematic approach to
  - Eliciting
  - Organizing
  - Documenting
  - Managing
- The changing requirements of a software application



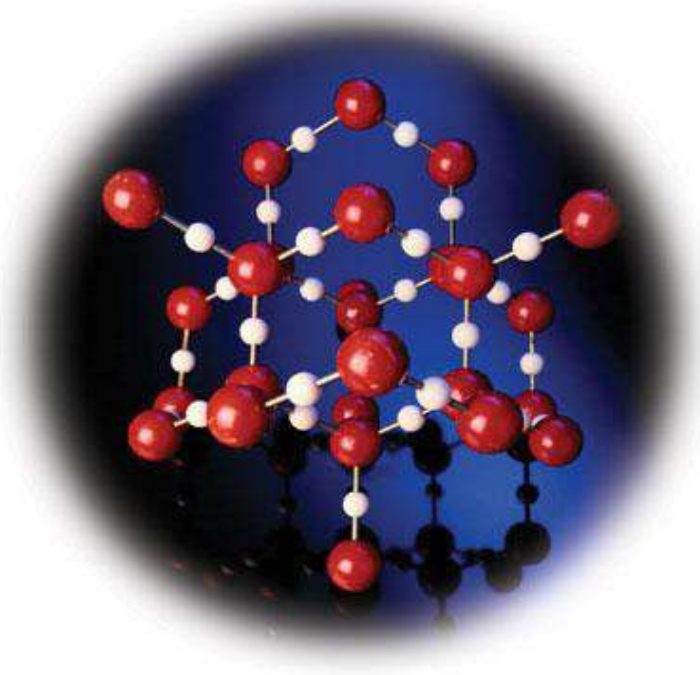
Use Component-Based Architectures

- Basis for reuse
  - Component reuse
  - Architecture reuse
- Basis for project management
  - Planning
  - Staffing
  - Delivery
- Intellectual control
  - Manage complexity
  - Maintain integrity



Model Visually (UML)

- Captures structure and behavior
- Shows how system elements fit together
- Keeps design and implementation consistent
- Hides or exposes details as appropriate
- Promotes unambiguous communication
  - The UML provides one language for all practitioners



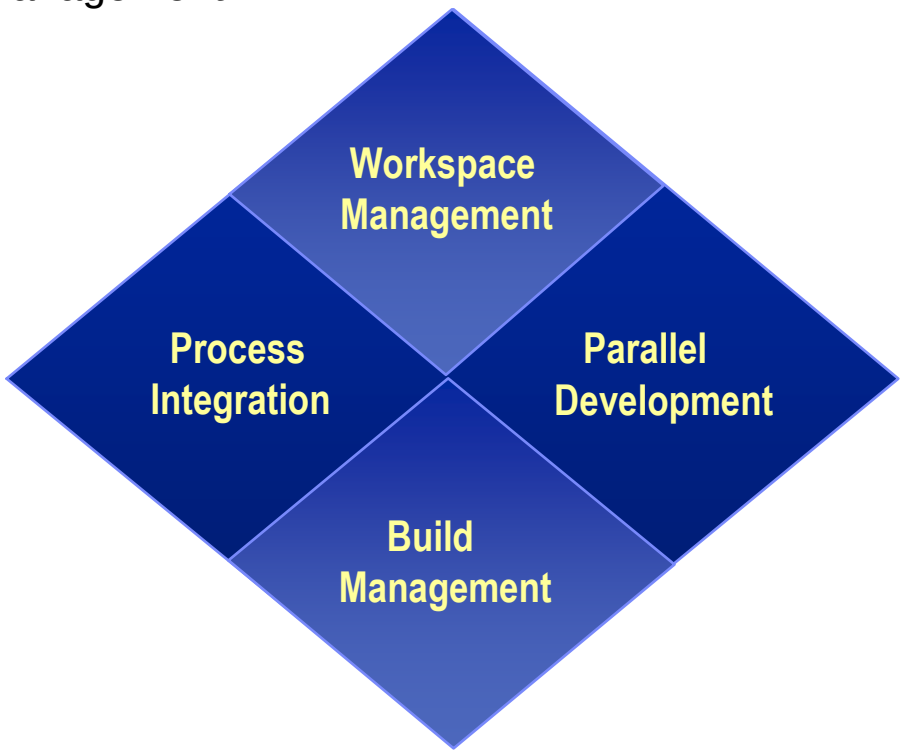
Continuously Verify Quality

Software problems are
100 to 1000 times more costly
to find and repair after deployment

Manage Change

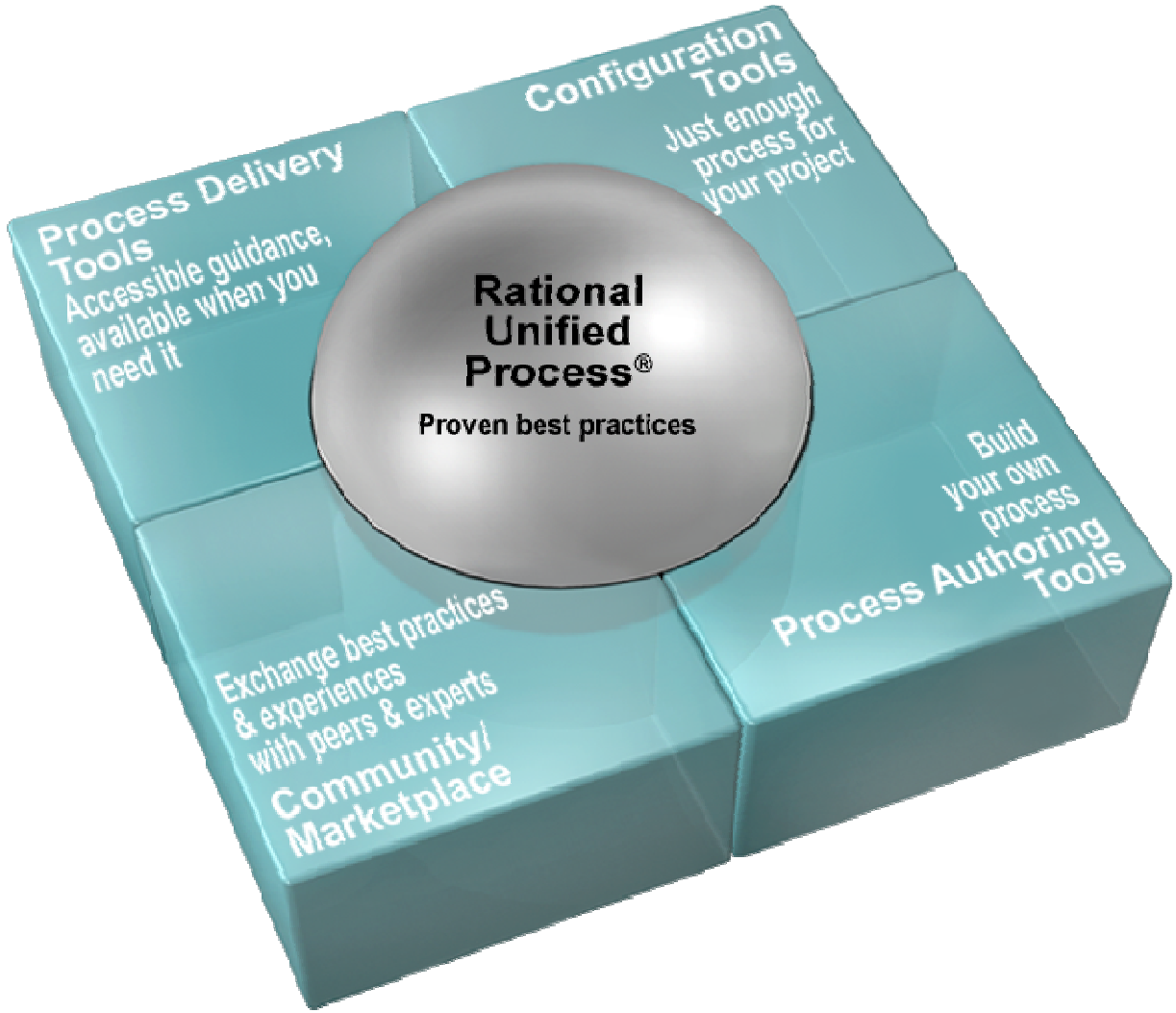
- To avoid confusion, have:
  - Secure workspaces for each developer
  - Automated integration/build management
  - Parallel development

Configuration Management is more than just check-in and check-out

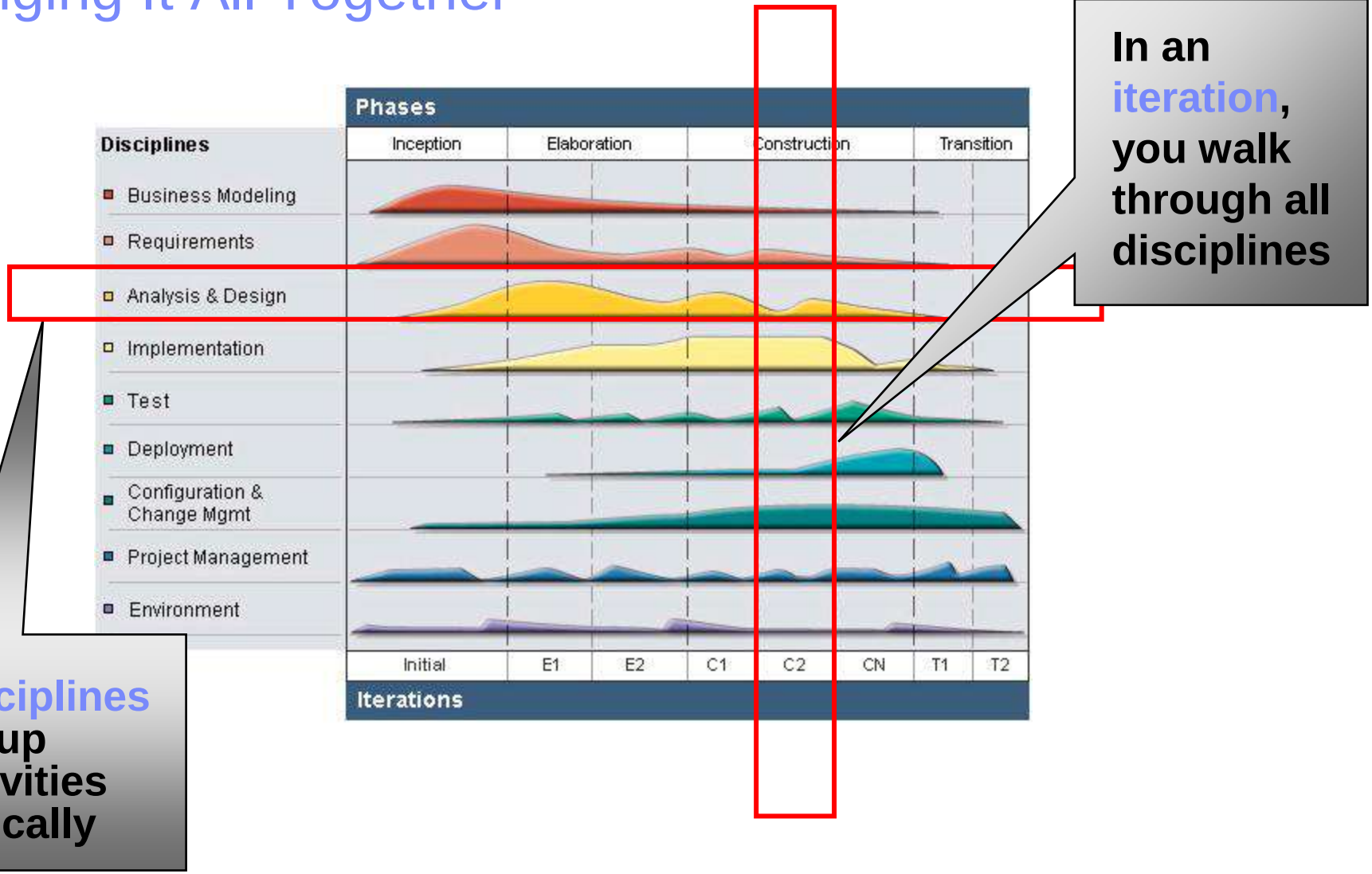


Rational Unified Process Implements Best Practices

- Iterative approach
- Guidance for activities and artifacts
- Process focus on architecture
- Use cases that drive design and implementation
- Models that abstract the system



Bringing It All Together



IBM Software Group | Rational Software France

Object-Oriented Analysis and Design with UML2 and Rational Software Modeler

06. Requirements Overview

Rational. software

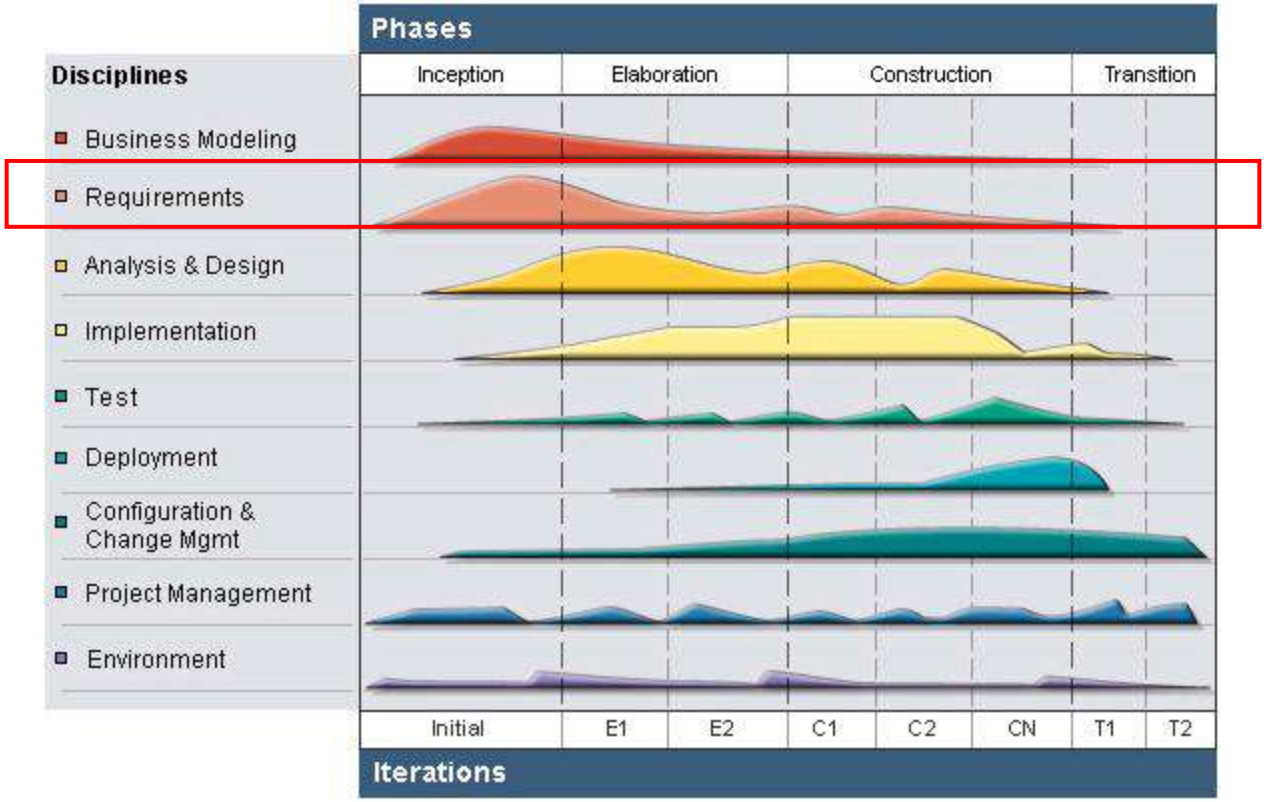
e business on demand software

Where Are We?

- Introduction to Use-Case Modeling
  - Find Actors and Use Cases
  - Other Requirements Management Artifacts

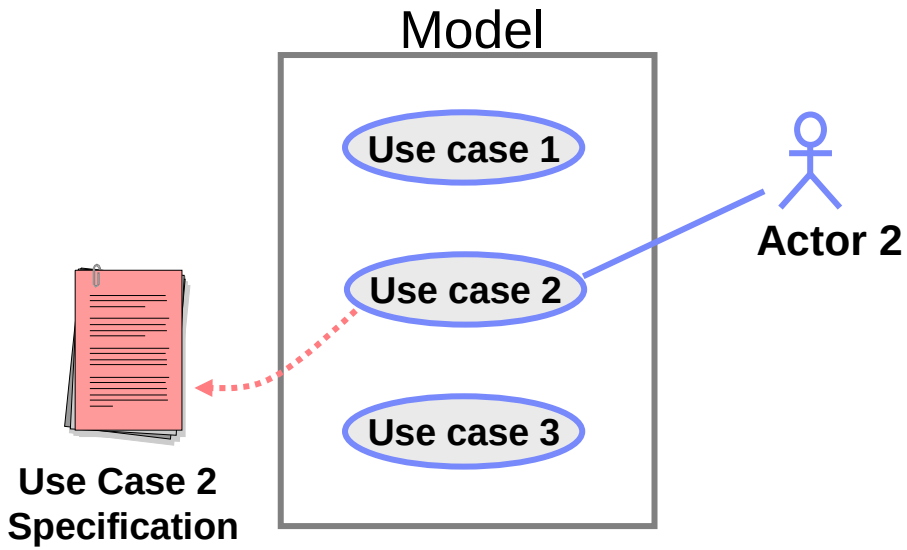
Requirements in Context

- The purpose of Requirements is to:
  - Elicit stakeholder requests and transform them into a set of requirements work products that scope the system to be built and provide detailed requirements for what the system must do
- RUP recommends a use-case driven approach

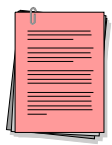


What Is Use-Case Modeling?

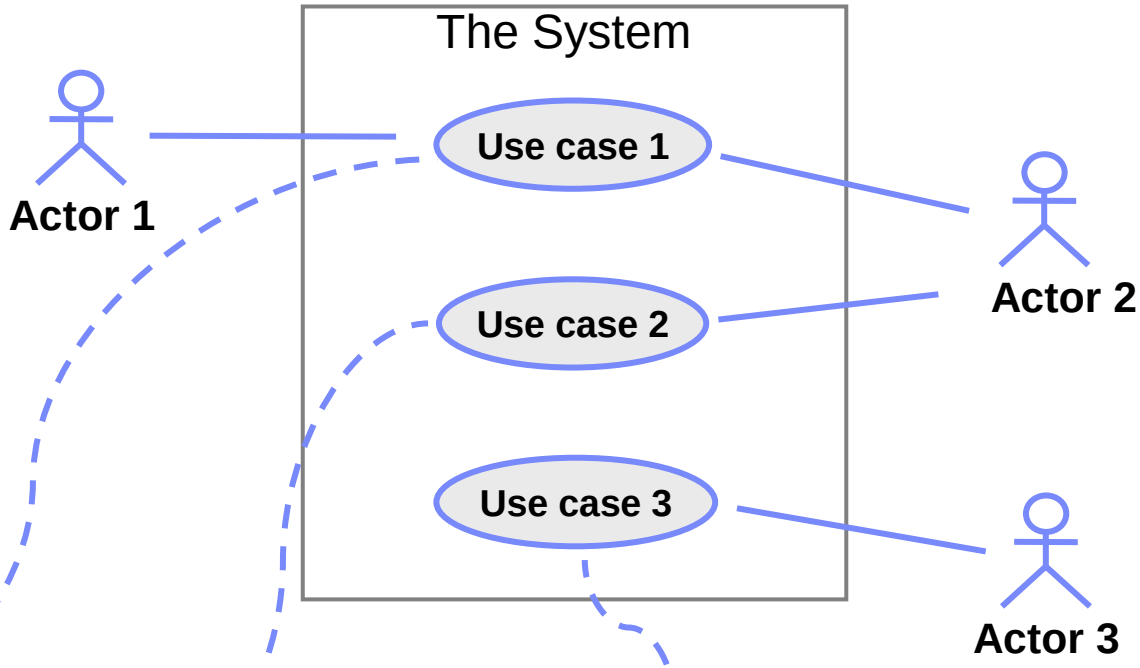
- Links stakeholder needs to software requirements
- Defines clear boundaries of a system
- Captures and communicates the desired behavior of the system
- Identifies who or what interacts with the system
- Validates/verifies requirements
- Is a planning instrument

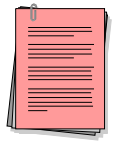


A Use-Case Model is Mostly Text



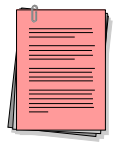
Use-Case-Model Survey
- survey description
- list of all actors
- list of all use cases





Use-Case 1 Spec
- brief description
- flow of events

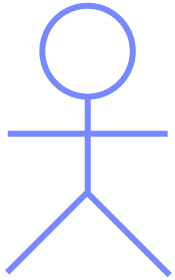
Use-Case 2 Spec
- brief description
- flow of events



Use-Case 3 Spec
- brief description
- flow of events

Major Concepts in Use-Case Modeling

- An actor represents anything that interacts with the system

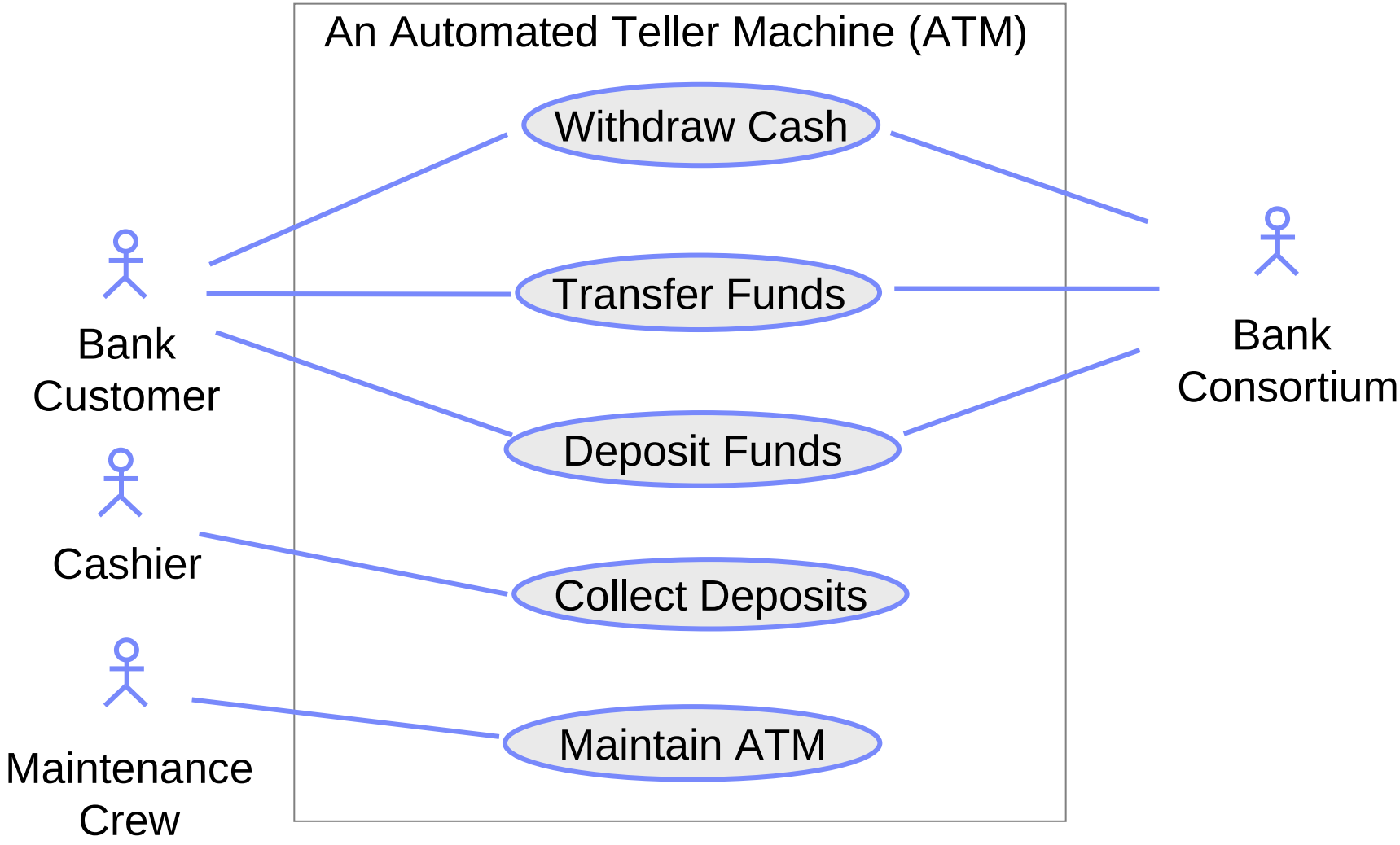


Actor

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

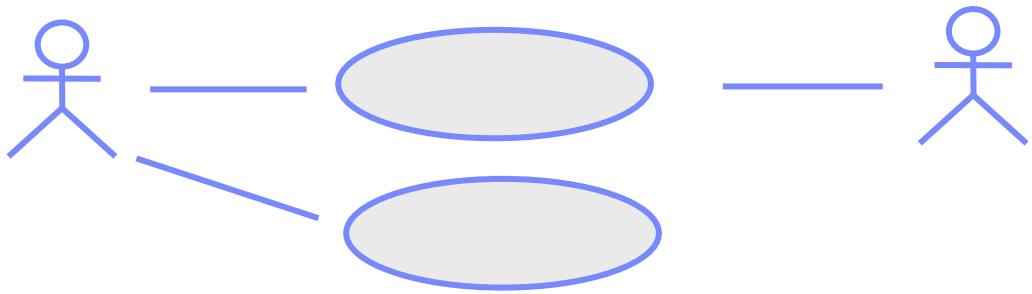
Use Case

Use-Case Diagram



Use Cases Contain Software Requirements

- Each use case:
  - Describes actions the system takes to deliver something of value to an actor
  - Shows the system functionality an actor uses
  - Models a dialog between the system and actors
  - Is a complete and meaningful flow of events from the perspective of a particular actor



Benefits of Use Cases

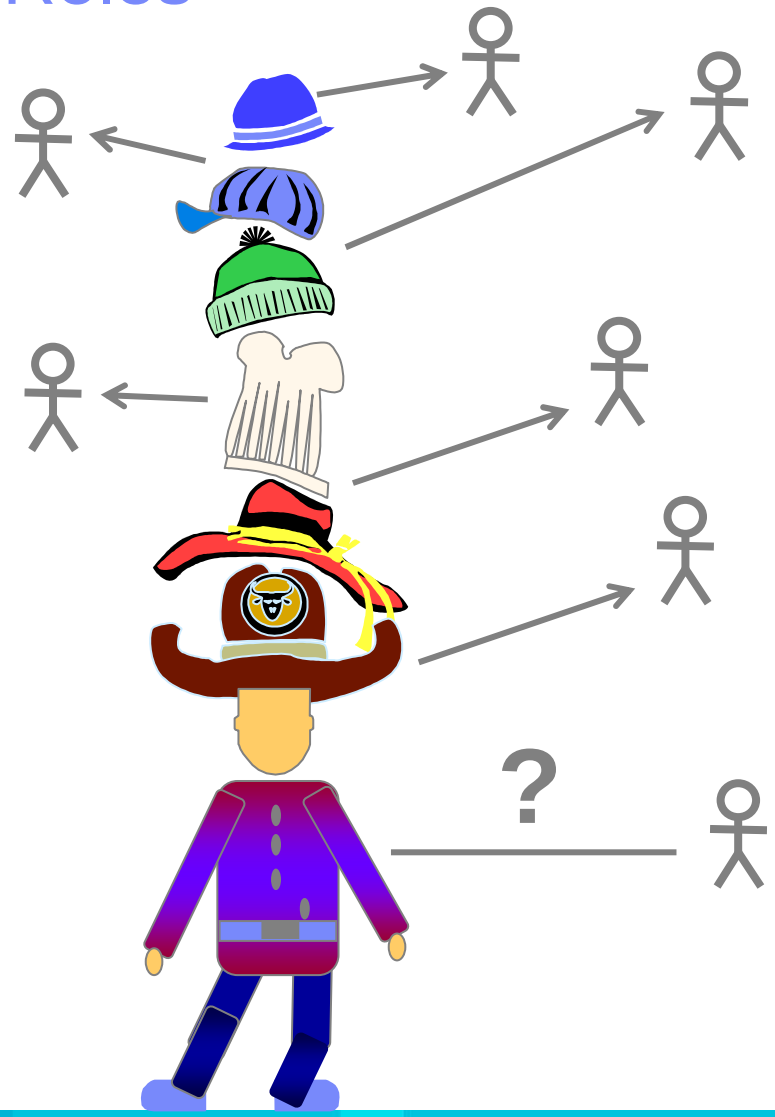
- Give context for requirements
  - Put system requirements in logical sequences
  - Illustrate why the system is needed
  - Help verify that all requirements are captured
- Are easy to understand
  - Use terminology that customers and users understand
  - Tell concrete stories of system use
  - Verify stakeholder understanding
- Facilitate agreement with customers
- Facilitate reuse: test, documentation, and design

Where Are We?

- Introduction to Use-Case Modeling
- ⇨ Find Actors and Use Cases
- Other Requirements Management Artifacts

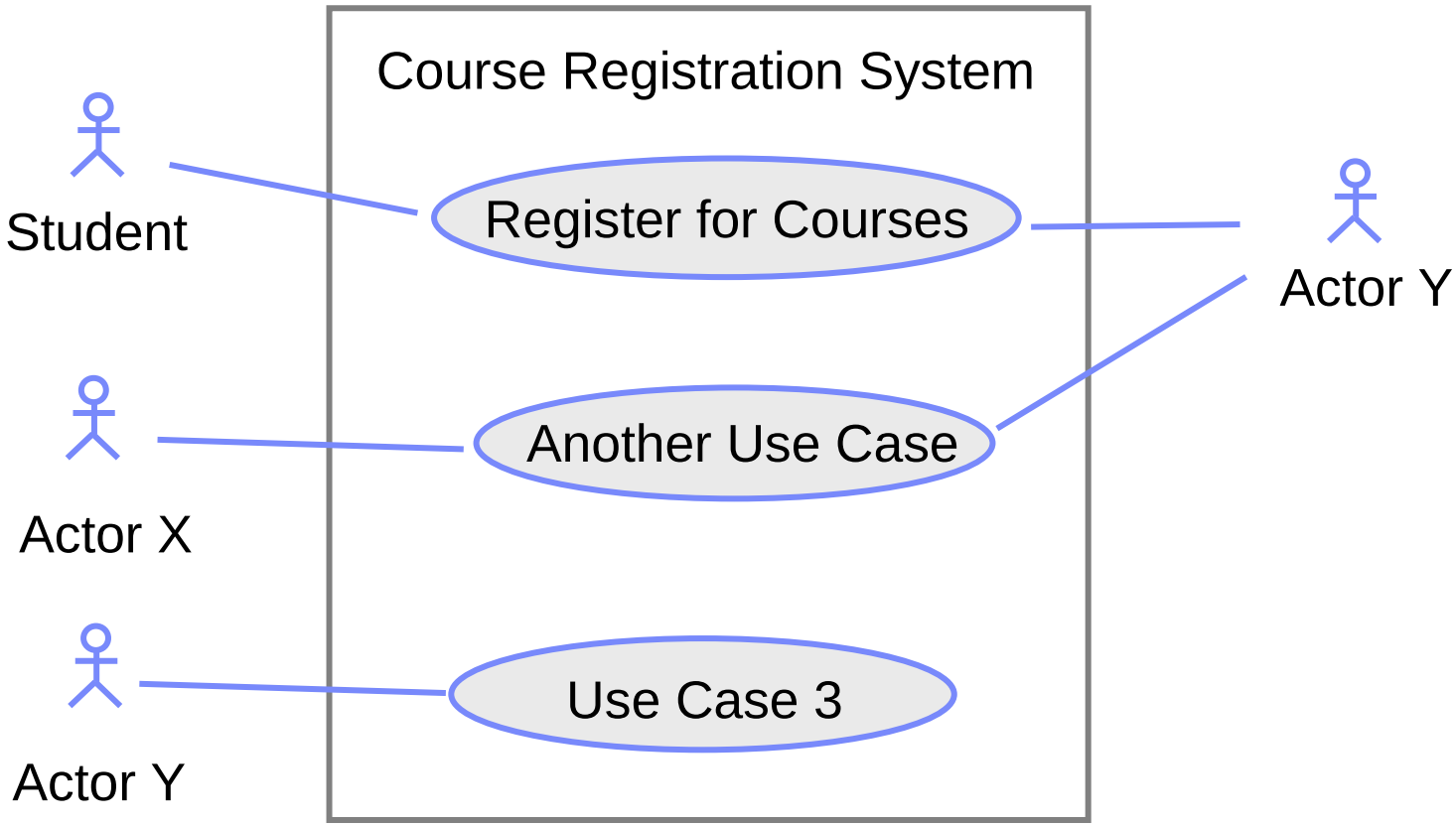
Define Actors: Focus on the Roles

- An actor represents a role that a human, hardware device, or another system can play in relation to the system
- Actor names should clearly denote the actor's role



Case Study: Course Registration System

- Review the problem statement provided in the Course Registration Requirements Document



How Should I Name a Use Case?

- Indicate the value or goal of the actor
- Use the active form; begin with a verb
- Imagine a to-do list
- Examples of variations
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration
  - Use Registration System

Which variations show the value to the actor? Which do not?
Which would you choose as the use-case name? Why?

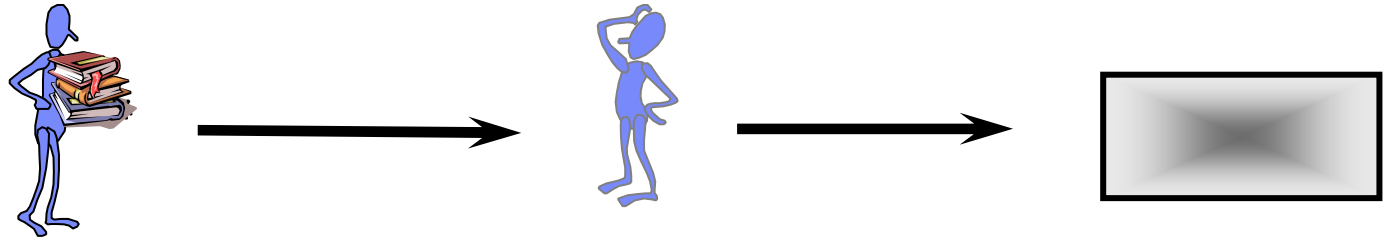
Steps for Creating a Use-Case Model

1. Find actors and use cases
   - Identify and briefly describe actors
   - Identify and briefly describe use cases
2. Write the use cases
   - Outline all use cases
   - Prioritize the use-case flows
   - Detail the flows in order of priority

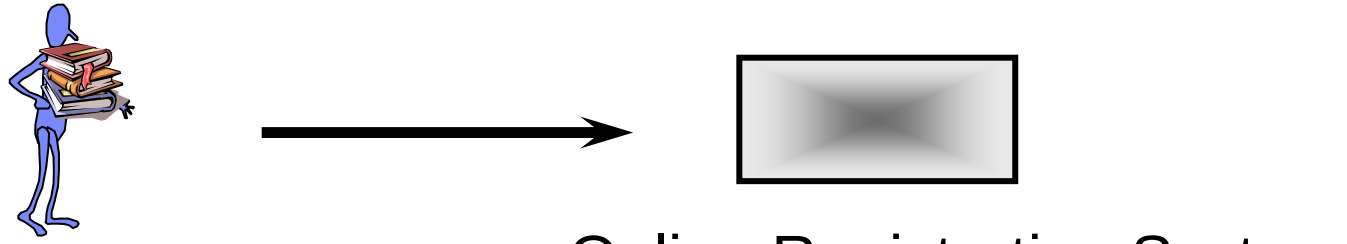
} Outside the scope of this course

Find Actors

- Who is pressing the keys (interacting with the system)?



The student never touches the system; the registrar operates it.
Or, are you building an Internet application?

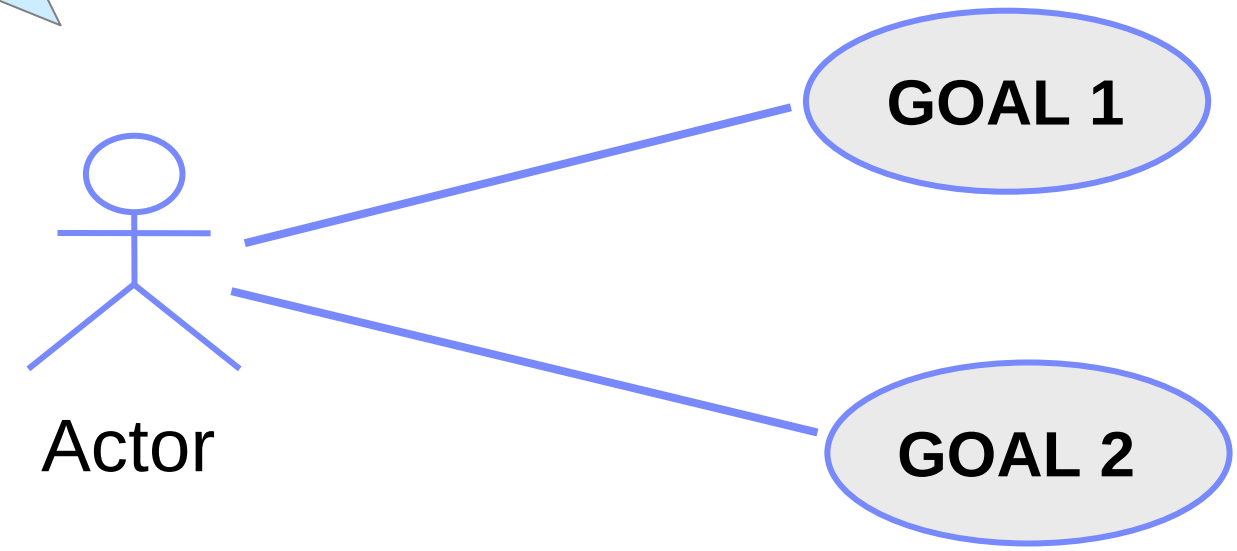


Identify Actors

- Who/what uses the system?
- Who/what gets information from this system?
- Who/what provides information to the system?
- Where in the company is the system used?
- Who/what supports and maintains the system?
- What other systems use this system?

Find Use Cases

What goal am I trying to achieve by using the system?



Identify Use Cases

- What are the goals of each actor?
  - Why does the actor want to use the system?
  - Will the actor create, store, change, remove, or read data in the system? If so, why?
  - Will the actor need to inform the system about external events or changes?
  - Will the actor need to be informed about certain occurrences in the system?
- Does the system supply the business with all of the correct behavior?

Group Exercise

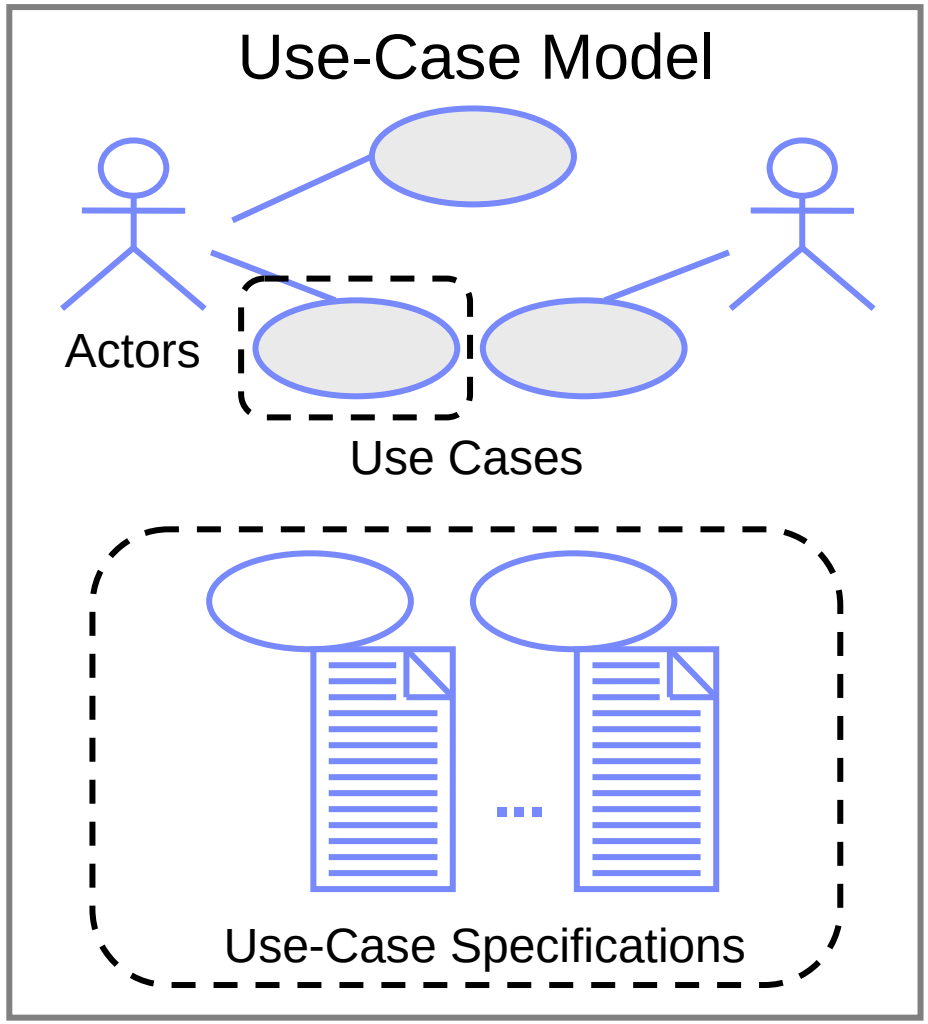
- Identify the actors who interact with the Course Registration System
- Identify use cases for the system
- Sketch a use-case diagram

Where Are We?

- Introduction to Use-Case Modeling
- Find Actors and Use Cases
- ⇨ Other Requirements Management Artifacts

Use-Case Specifications

- Name
- Brief description
- Flow of Events
- Relationships
- Activity diagrams
- Use-Case diagrams
- Special requirements
- Pre-conditions
- Post-conditions
- Other diagrams



Use-Case Flow of Events

- Has one normal, *basic flow*
- Several *alternative flows*
  - Regular variants
  - Odd cases
  - Exceptional flows for handling error situations

A Scenario Is a Use-Case Instance

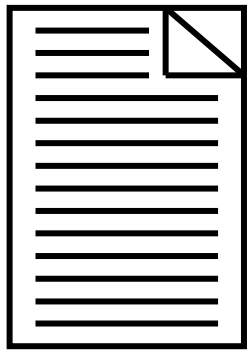
Scenario 1

Log on to system.
Approve log on.
Enter subject in search.
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

Scenario 2

Log on to system.
Approve log on.
Enter subject in search.
Invalid subject.
Re-enter subject.
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

Glossary



Glossary

Course Registration System Glossary

1. Introduction

This document is used to define terminology specific to the problem domain, explaining terms, which may be unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal *data dictionary*, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

2. Definitions

The glossary contains the working definitions for the key concepts in the Course Registration System.

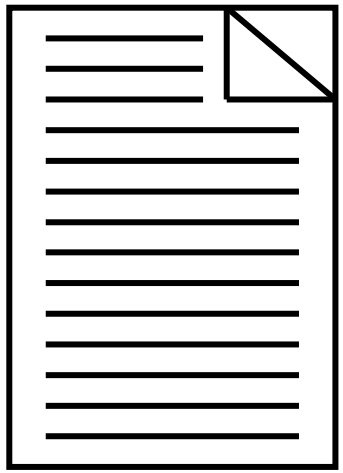
2.1 Course: A class offered by the university.

2.2 Course Offering: A specific delivery of the course for a specific semester – you could run the same course in parallel sessions in the semester. Includes the days of the week and times it is offered.

2.3 Course Catalog: The unabridged catalog of all courses offered by the university.

Supplementary Specification

- Functionality
- Usability
- Reliability
- Performance
- Supportability
- Design constraints



Supplementary Specification