



What is UP?

- A software engineering process (SEP)
 - defines the who, what, and how of developing software
 - Is the process in which we turn user requirements into software
- To model the “who” of SEP, UP uses the concept of the worker:
 - A role played by an individual or team within the project



UP Axioms

- UP has three basic axioms:
 - Use case and risk driven
 - Architecture centric
 - Iterative and incremental
- These are ways of capturing requirements
 - UP requirement driven??
 - UP is risk driven .. If you don't actively attack risks they will actively attack you!!



What is UP

- UP models the “when” as activities:
 - Tasks that will be performed by individuals or teams in the project
 - Individuals or teams will have specific roles when they perform certain activities
 - A sequence of activities is called a workflow
- The what are “artifacts”
 - Inputs and outputs to the project – ex
 - Source code, standards, executable programs, etc



Instantiating UP for your Project

- UP is a generic sw development process that must be instantiated for an organization and for each particular project → all sw projects tend to be different
- The instantiation process defining and incorporating:
 - In-house standards
 - Document templates
 - Tools – compilers, configuration management tools, etc
 - Databases – bug tracking, project tracking, etc
 - Lifecycle modifications- e.g. more sophisticated quality control measures



UP Axioms

- Develop and evolve a robust system architecture
- Architecture describes
 1. the strategic aspects of how the system is broken down into components
 2. How those components interact and are deployed



UP Process

- Iterative and incremental
- Each iteration contains all of the elements of a normal sw development project:
 - Planning
 - Analysis and design
 - Construction
 - Integration and test
 - Internal and external release
- Each iteration generates a baseline comprising a partially complete version of the final system, and documentation
 - Baselines build on each other over successive iterations
 - Increment - Difference between two consecutive baselines



Iteration Workflows

- Each iteration has five core workflows which specify what needs to be done and the skills needed to do it
- Five core workflows:
 - Requirements – what the system should do
 - Analysis – refining and structuring the requirements
 - Design – realizing the requirements in system architecture
 - Implementation – building the software
 - Test- verifying that the implementation works as desired
- Baseline:
 - Provides an agreed basis for further review and development
 - Changed only through formal procedures of configuration and change management

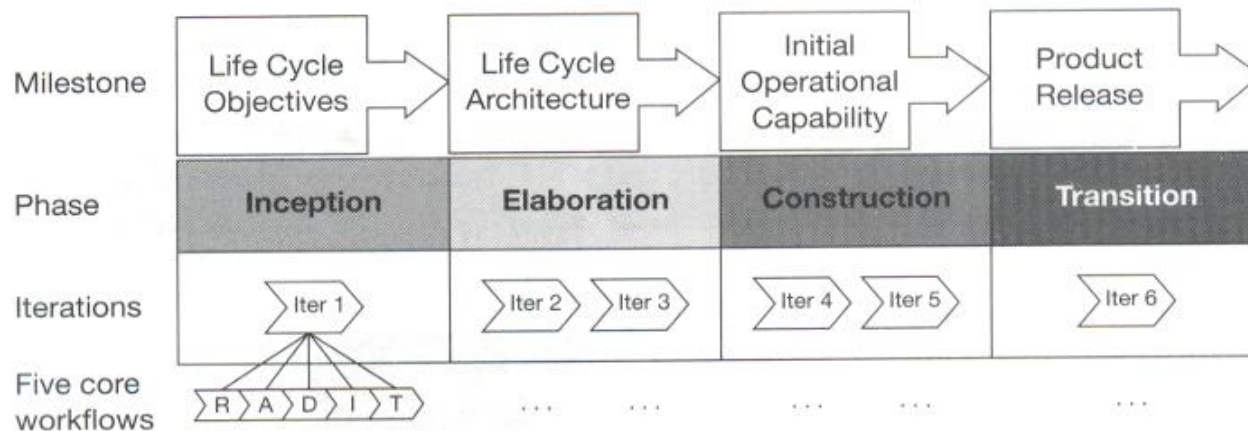


UP Structure

- Project lifecycle divided into four phases
 - **Inception** – Lifecycle objectives ..define scope of project (identify all actors and use cases)
 - **Elaboration** – Lifecycle Architecture ..Plan project, specify features
 - **Construction** – ..Build the product ..Initial Operational Capability
 - **Transition** – Product release ..Transition the product into end-user community
- Each ends with a major milestone

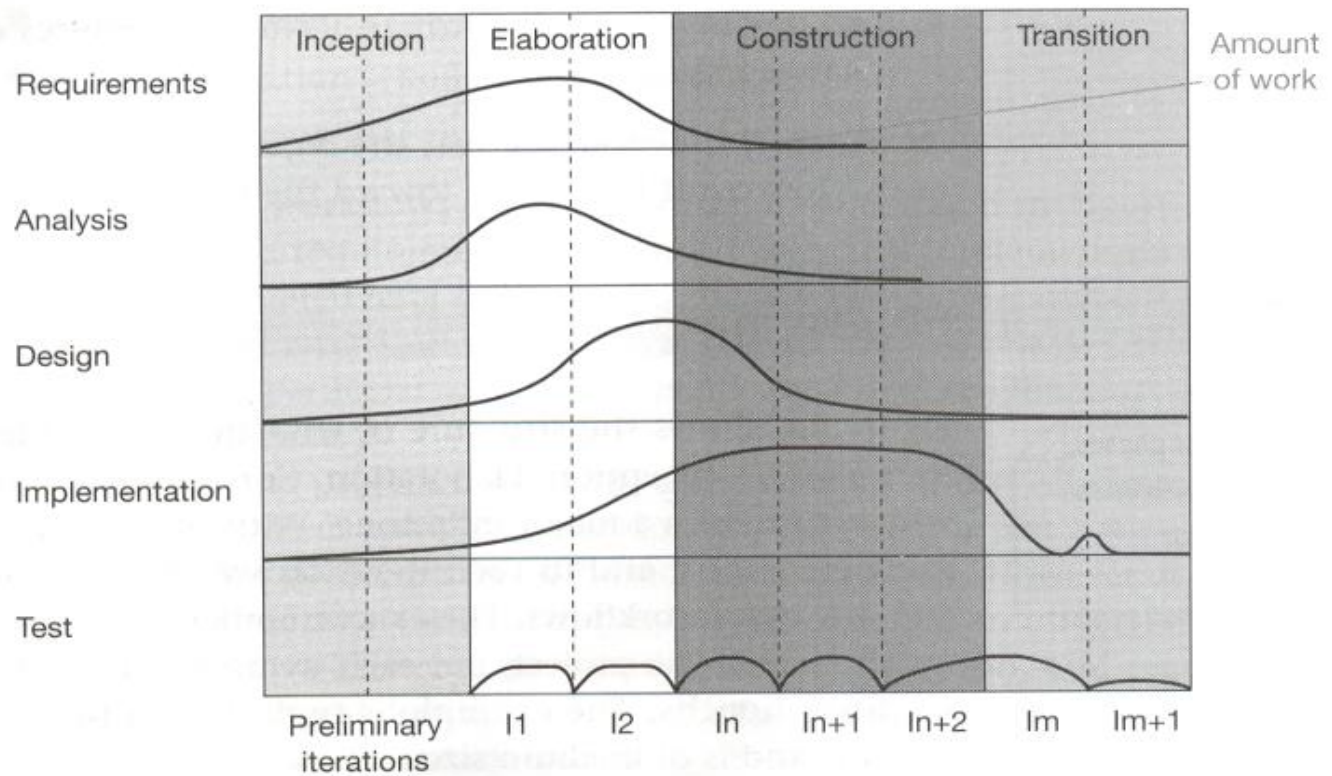
UP Structure

UML and UP



- As the project shifts through the phases the amount of work done in each workflow changes

Workflow





Workflow

- The curves indicate the relative amount of work done in each core workflows as the project progresses through the phases
- In Inception- most of the work is done in requirements and analysis
- In Elaboration the emphasis shifts to requirements, analysis and some design
- In Construction the emphasis is on design and implementation
- In Transition the emphasis is on implementation and test



UP Phases

- Every phase has
 - A goal
 - A focus of activity with one or more core workflow
 - A milestone



Inception goals

- Get the project off the ground
- Establish feasibility:
 - May involve prototyping to validate technological decisions or
 - proof of concept
- Create a business case to demo that the project will deliver quantifiable benefits
- Capture essential requirements to help scope the system
- Identify critical risks



Inception Focus

- Primary emphasis in inception is on requirements and analysis workflows
- Inception Deliverables:
 - Initial use case model
 - Initial Project Plan
 - Risk assessment document

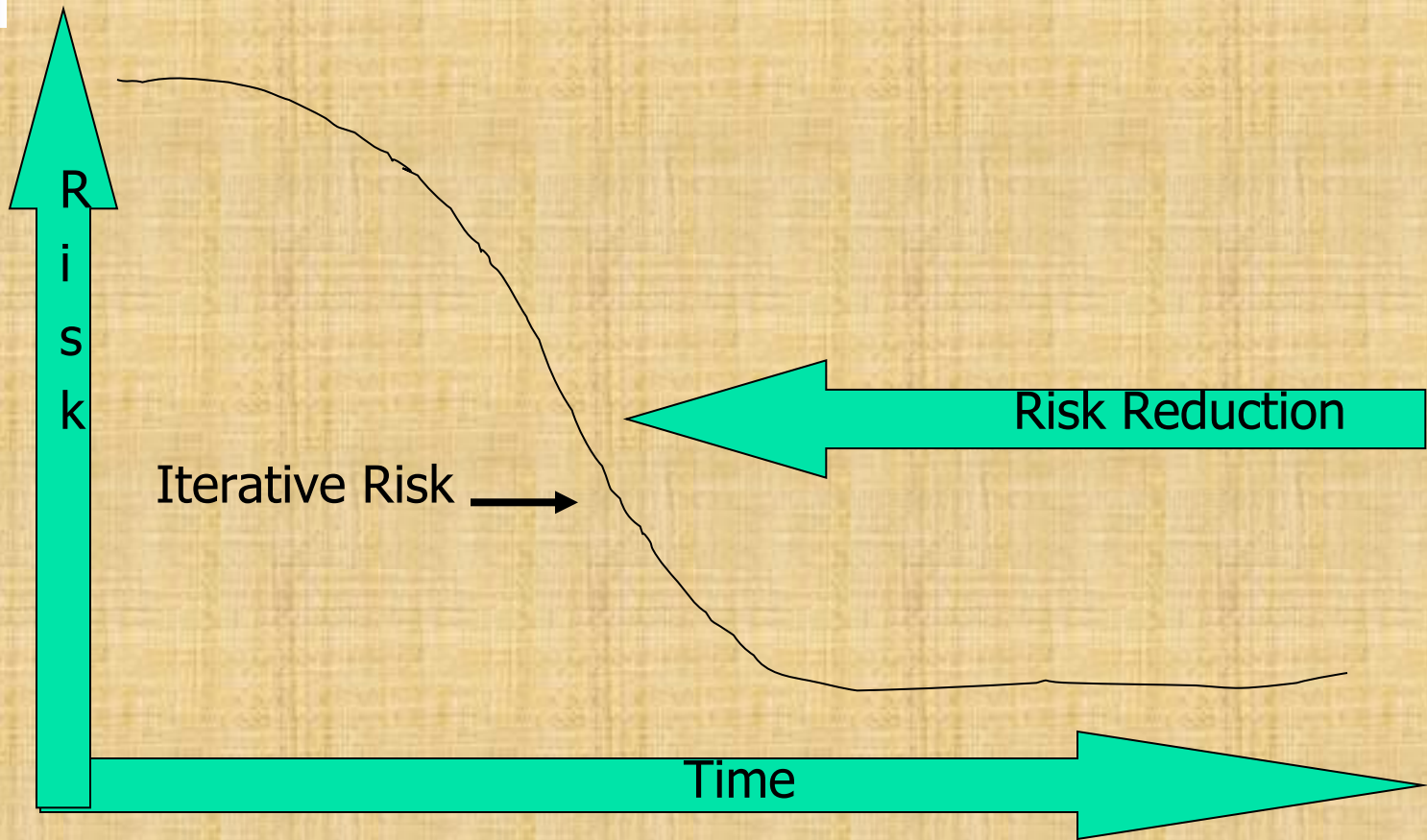


Iterations

- Earliest iterations address greatest risks
- Each iteration includes integration and test
- Iterations help:
 - Resolve risks before making large investments
 - Enable early feedback
 - Make testing and integration continuous
 - Focus project short-term objective milestones
 - Make possible deployment of partial implementation



Risk Profile





Elaboration Goals

- Create an executable architectural baseline
- Refine the Risk Assessment
- Define quality attributes (defect rates)
- Capture use cases to 80% of functional requirements
- Create a detailed plan for the construction phase
- Formulate a bid that includes resources, time, equipment, staff and cost



Elaboration Focus

- Requirements – refine system scope and requirements
- Analysis – establish what to build
- Design – create a stable architecture
- Implementation – build the architectural baseline
- Test – test the architectural baseline



Elaboration deliverables

- Architectural baseline
- Static Model
- Dynamic Model
- Use Case Model
- Updated Risk Assessment
- Updated Project Plan
- Sign-off document

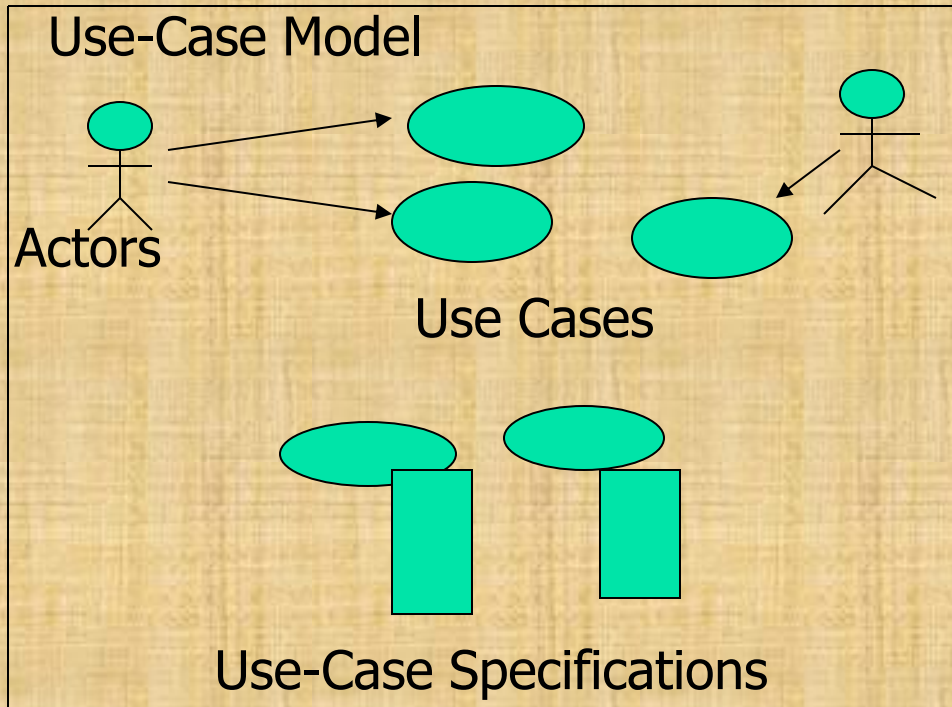


Requirements Discipline

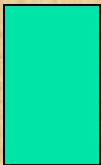
- The requirements discipline intends to:
 - Find agreement on what the system should do
 - Provide system developers with a better understanding of the system requirements
 - Define the boundaries of the system
 - Provide a basis for planning the technical contents of iterations
 - Provide a basis for estimating cost
 - Define a user-interface for the system, focusing on the needs and goals of the users

Requirements Artifacts

Relevant Requirements Artifacts



Glossary



Supplementary
Specification



Requirements Spec Phase

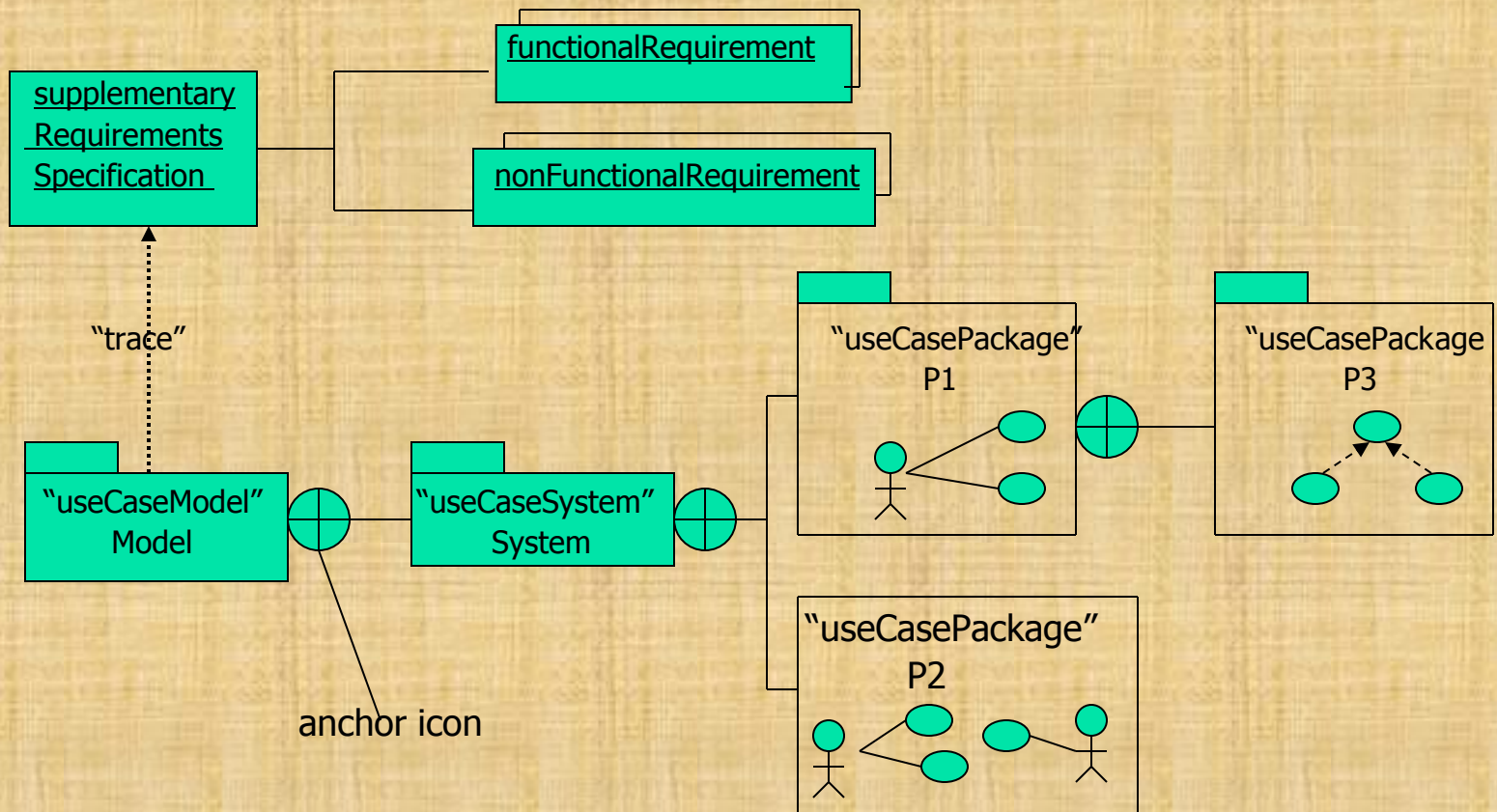
- Begins when the developers start modeling the requirements
- A CASE tool is used to enter, analyze and document the model
- The two most important specification techniques in OO analysis:
 - Class diagrams
 - Use case diagrams
- Spec doc also describe performance, 'look and feel', usability, maintainability, security, political and legal requirements



Specification Model

- Should be independent from the hardware/software platform on which the system is to be deployed
- Design Later!!!!

Requirements - Metamodel





Metamodel

- A lot of UML Syntax .. Discuss later
 - Folder icons are UML packages
 - UML grouping mechanism; contain groups of elements
 - Anchor icon indicates that the thing at the circle end contains the thing at the other end of the line
 - *supplementaryRequirementsSpecification* document (modeled as an object) contains many *functionalRequirements* and many *nonFunctionalRequirements*
 - Dashed arrow labeled <<trace >> indicate we can form the use case model by consideration of the *supplementaryRequirementsSpecification* and indicates an historical relationship between them
 - The use case model is composed of exactly one use case system
 - The system is made up of many use case packages (only three shown here)
- Requirements, use cases and actors to be covered later



UML

- A notation
- A way to document system specs
- Is not a methodology – it's a language
 - A notation simply tells you how to structure your system documentation
- Does not depend on a methodology
- Is composed of nine diagrams



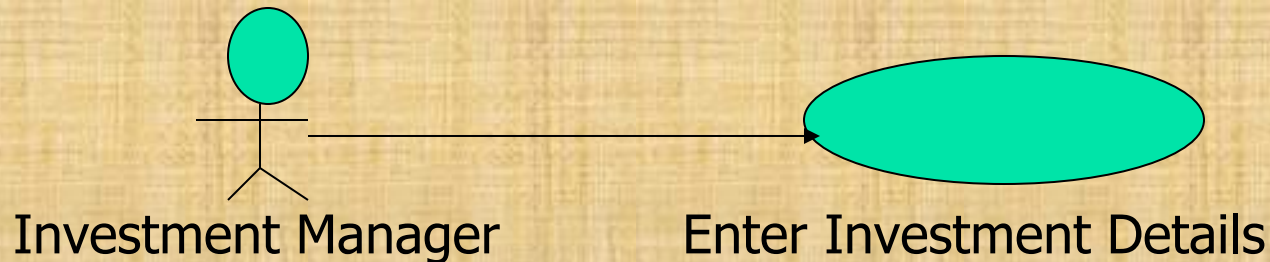
UML's nine Diagrams

- Use case diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram
- Class diagram
- Object diagram
- Component diagram
- Development diagram



Use Cases

- A tool that shows the **what** exclusively
- Often recommended as the primary tool for requirements gathering
- Are part of the UML language
 - UML – a notation for documenting system specification
- Use case example:





Use Cases

- Text descriptions of interaction between actors and computer system
- Use Case Diagrams
 - Graphical depictions of the relationships between actors and use cases and between use cases and another use case
- Use cases should drive not just requirements gathering, but the entire software development cycle



Exercise

- In your team, perform the following activities for the given problem statement:



Requirement Traceability

- Requirements must be traceable throughout the development lifecycle
- You should be able to ask any person in any role the questions in Table 1 which follows:



Traceability Defined by Role

Role	Traceability
Analyst/Designer	What reqmnts does this class on this class diagram relate to?
Developer	What reqmnts does the class you're programming relate to?
Tester	Exactly which requirements are you testing for when you execute this test case?
Maintenance programmer	What reqmnts have changed that require you to change the code that way?
Technical writer	What reqmnts relate to this section of the user manual?
Architect	What requirements define what this architectural component needs to do?
Data Modeler	What reqmnts drove the design of this entity or database table/index
Project Manager	What reqmnts will be automated in working code in this iteration



The questions

- Can rarely be answered in today's projects
- If answered, would:
 - Provide audit trail
 - Describe why the activities are being done
 - Help prevent developer goldplating:



Use Case Modeling

- A form of requirements engineering
- A different and complementary way of eliciting and documenting requirements
- Use case model
 - Describes what the system does
 - Serves as a contract between the customer, the users, and system developers
 - Allows
 - customers and users to validate that the system becomes what they expected
 - Developers to ensure that what they build is what is expected



Functional Requirements

- A statement of what the system should do –
- Ex. Collecting requirements for an ATM, some functional requirements might be:
 1. The ATM system shall check the validity of the inserted ATM card
 2. The ATM system shall validate the PIN entered by the customer
 3. The ATM system shall dispense no more than \$250 against any ATM card in any 24-hour period



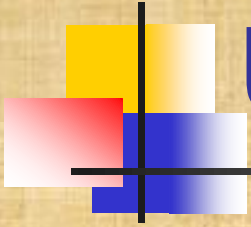
Non-functional Requirements

- A constraint placed on the system
- For the ATM system, might be:
 1. The ATM system shall be written in C
 2. The ATM system shall communicate with the bank using 256-bit encryption
 3. The ATM system shall validate an ATM card in three seconds or less
 4. The ATM system shall validate a PIN in three seconds or less



Use Case Model

- Consists of use cases and actors
- Each use-case in the model is described in detail, showing step by step how the system interacts with actors and what system does in the use case
- The Use-Case spec is the document where all of the use-case properties are documented



Use Case Modeling

- Steps:
 - Find the system boundary
 - Find the actors
 - Find the use cases:
 - Specify the use case
 - Create scenarios
- The output of these activities is the use case model



Use Case Modeling

- The Four components of this model:
 - Actors – roles played by people or things that use the system
 - Use cases – things that the actors can do with the system
 - Relationships – meaningful relationships between actors and use cases
 - System boundary – a box drawn around the use cases to denote the edge/boundary of the system being modeled
- Model provides a major source for objects and classes –



Rational Unified Process Description and Workflows

- **The RUP uses four elements to describe processes:**
 - **Workers** – describe a role, some people have many roles.
 - **Activities** – small, definable, reusable tasks that can be allocated to a single worker.



Description of Workflow (2)

- **Artifacts** – usually process deliverables, like: use cases, code, plans, test cases, test results.
- **Workflows** – coordinated sequences of activities.



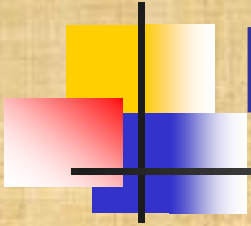
RUP Workflows

- **There are 9 workflows, 6 engineering workflows:**
 - Business modelling
 - Requirements
 - Development & Analysis
 - Implementation
 - Test
 - Deployment



RUP Workflow contd.

- **And 3 supporting workflows:**
 - Project management
 - Configuration and Change Management
 - Environment



Requirements Workflow

- Workflows are followed iteratively through each iteration of each phase of the RUP.
 - Effort expended in each workflow depends on the phase.
 - We'll look at the requirements workflow in more detail



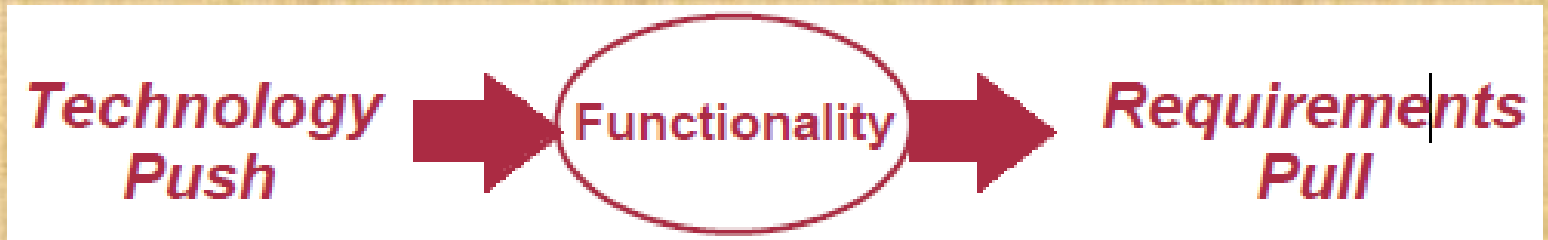
Capturing/discovering Requirements

- **More difficult and more important than writing code**
 - Users know what they have, not what they need
 - They will better understand what they need after they see it.
 - Nobody needed the WWW until they saw it.



Capturing/Discovering Requirements contd.

- User cannot envision the possibilities enabled by technology





Capturing/Discovering Requirements contd.

- Developer is rarely the user
- Diversity of users
- Support the user's mission, not only the user
- User needs and missions are constantly changing
- The new system will impact the user's needs, resulting in new system needs (cycle)
- Complex systems are never fully understood
 - understanding evolves as the system evolves
- Hard to understand the constraints of legacy systems and the system environment

Purpose of Requirements Workflow



- Aim development toward the right system
 - Other workflows focus on building the system right
- Describe what the system should and should not do
 - an agreement between customer (including user) and development organization
 - in the language of the customer/user

Tasks in Requirements Workflow

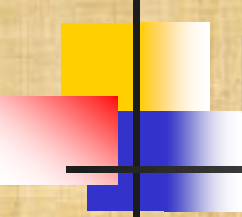


- List candidate requirements
- Understand system context
- Capture functional requirements
- Capture non-functional requirements
- Validate requirements (not well-developed in RUP)



List Candidate Requirements → Feature List

- **Candidate features that could become requirements**
 - Good ideas added to feature list
 - Features taken off list when they become formal requirements
- **Planning values**
 - Status
 - Cost
 - Priority
 - Risk



Understand system context Business or Domain Model

■ **Domain model**

- Identify and name important concepts and entities in the system context
- Identify and name relations between domain objects
- Glossary for now, possible classes in analysis and design workflows



Business or Domain Model?

- **Business model**
 - Domain (object) model plus
 - processes/behaviors
 - workers, their responsibilities and operations
- Decide whether to build a business model, a domain model, or simply a glossary of terms



Capture functional requirements

→ Use cases

- **Capture requirements as use cases**
 - Use case: a user's way of using the system
 - When an actor (user or external subsystem) uses the system, the system performs a use case
 - All use cases = all the things the system must do
- **Capture user interfaces that support the use cases**



Capture Non-functional Requirements Supplementary Req & Use Cases

- **System properties**

- Environmental or implementation constraints
 - **e.g. must have remote access or must run on Linux or WinNT**
- Qualities ("-ilities"): performance, reliability, security, maintainability, extensibility, usability, etc.



Capture Non-Func Req contd.

- **Tie to use cases or domain concepts, where possible**
 - **those that cannot be tied are listed as supplementary requirements**



Requirements in the Life Cycle Phases

- **Inception**

- identify most of the use cases to define scope
- detail critical use cases (10%)

- **Elaboration**

- detail the use cases (80% of the requirements)



Requirements contd.

- **Construction**

- identify and detail remaining use cases

- **Transition**

- track and capture requirements changes



Business Modeling

- **Business use case model**
 - processes (use cases) and users (actors) in roles
 - represents system from a usage perspective and outlines how it provides value to its users
- **Business object model**
 - how each use case is realized by a set of workers who are using business entities and work units



Requirements Workflow Description

- Artifacts created
- Workers participating
- Detailed workflow activity



Capturing Requirements as Use Cases

- **Use cases**

- Offer a systematic and intuitive way to capture functional requirements
- Focus on the value the system adds to each user or external system

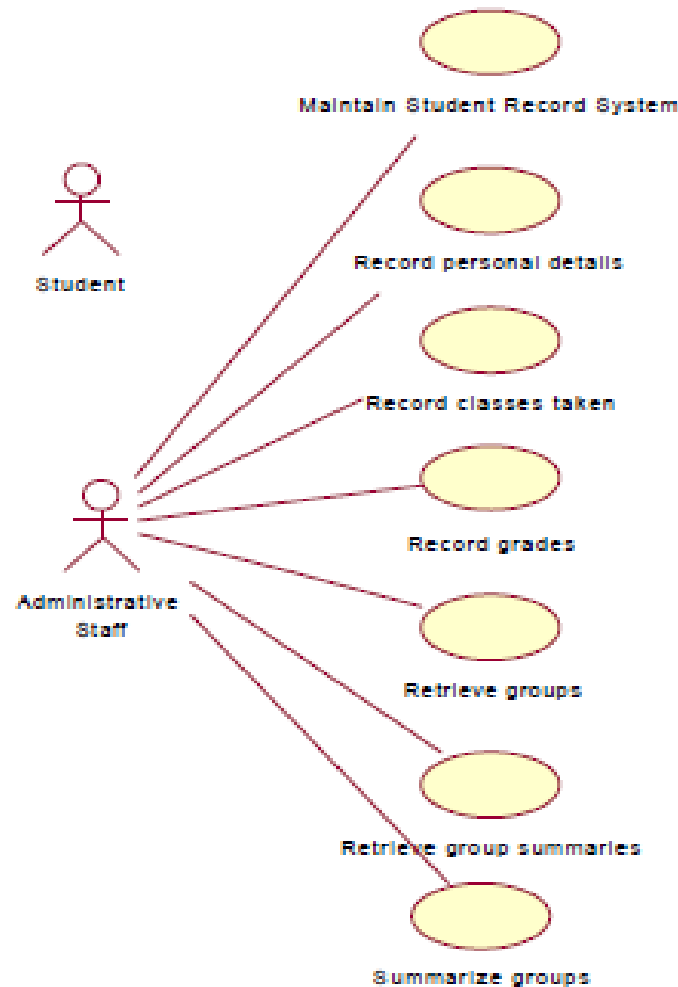


Capturing contd.

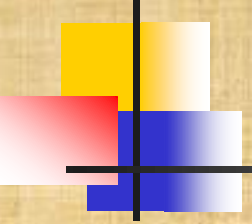
- Force analysts to think in terms of who the users are and their business or mission needs
- Drive the other development workflows

A University Support System Use Case Diagram

A use case diagram identifies
actors and names actors
actors and their use cases



Use-Case Model Artifact (Actors)



■ **Actors**

- Each type of user and each type of external system,
 - parties outside the system that interact with the system
 - Separate actor for each role of a user
- There will be (at least) one system use case for each actor role

Use-Case Model Artifact

(Use Case, Structure)

■ Use Case

- Scenarios giving the ways actors use the system
- Specifies main sequence and alternative sequences of actions/events that the system can perform



UC Model Artifacts contd.

- Includes special requirements specific to the use-case
- Can structure complex or large use case models
 - perspectives, packages, etc.



Other Requirements Workflow Artifacts

- **Architectural view of use-case model**
 - shows the architecturally significant use cases
 - important and critical functionality
 - important requirements to develop early
 - input to use case prioritization
 - corresponding use case realizations usually appear in architectural view of analysis and design models



Other Requirements contd.

- **Glossary**

- Important and common terms
- Consensus agreement on meaning
- Less formal view of business/domain model

- **User-interface prototype**

- help understanding of user/system interaction



Worker Responsibilities

- **System analyst**
 - Identify actors and use cases
 - Create complete and consistent set of use cases and requirements (but not for details of each individual use case)
 - Develop glossary to facilitate complete/consistent requirements set
- **Use-case specifier**
 - Detail one or more use cases
- **User-interface designer**
 - Define the “visual shape” of the user interface for one or more actors
 - layout, behavior, inter-screen flow
- **Architect**
 - Describe architectural view of use-case model



■ **Activity steps**

- Find actors
- Find use cases
- Describe each use case
- Describe use case model as a whole
(including glossary)



Finding Actors

- **Example actors or actor categories**
 - Business workers, business actors
 - The person/system asking the question, making the decision
 - External systems
 - System maintenance and operational support



Finding Actors (2)

- **Criteria**

- Must be at least one real user who can enact the candidate actor
- Minimum overlap between roles



Finding Actors (3)

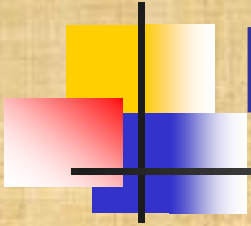
- **Capture**

- Actor name
- What the actor uses the system for; actor (role) needs and system responsibilities
- What the system uses the actor for; actor (role) responsibilities and system needs



Finding Use Cases

- **Examples: deliver an observable result of value to a particular actor**
 - Use case(s) for every role of every worker
 - Use case to support user's need to create, change, track, remove or study business objects
 - Use case to allow user to tell system of event or for system to tell user of event
 - Use cases for system startup, termination or maintenance



Finding Use Cases

- Use case name: verb phrase describing result of interaction
- Use case scope and boundaries are hard to find
 - Decouple them in time and data sharing
 - Iterate with architecture tasks



Describing the Use Cases

- **Description iterations add detail**

- 1.** Name
2. Few words
3. Few sentences to summarize actions
4. Detailed step-by-step action/response
5. Formal models



Use Case Model as a Whole

- **Model as a whole**

- Glossary of common terms
- Survey description: interaction of actors, interaction of use cases
 - Use case generalizations and extensions
 - Flows between use cases
 - Activity diagrams
 - Post-conditions of one use case establish pre-conditions of others
- Group by business use case, by actor, by coupling, etc.



Review

- **Review items**

- Use case list is complete
- Sequences of actions are correct, complete and understandable
- All use cases have value to actors

Prioritize and Detail Use Cases



- Prioritize based on clarity of system scope, architecture impact, and risk
 - Detail event flow, use case start/end, actor interaction
 - Pre-conditions; basic flow through states; post-conditions
 - Alternative flows to accommodate...
 - actor choice
 - influence of other actors
 - actor error
 - system error or failure

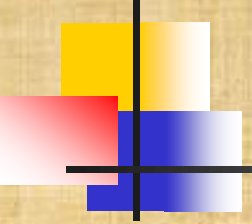


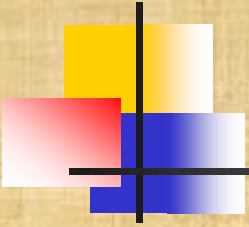
Detail Use Case Description

- Start states (preconditions)
- The first action to perform
- Action order
 - Basic/normal
 - Alternates
 - Constraints
- How the use case ends
- Possible end states (postconditions)
- System interaction with actor and what they exchange
 - Clarify which does what
 - Actor initiation, system response
 - System initiation, actor response
 - If "actor" is an external system, specify protocol
- Usage of system objects, values and resources
- Non-functional requirements

Review:

*understandable, correct,
complete, consistent*

- 
-
- Actor interacts by viewing and manipulating elements that represent attributes of use cases
 - Assure each use case is accessible to the actors through the user interface

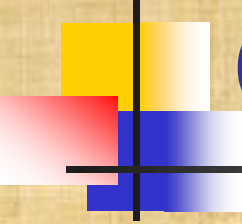


- Assure well-integrated, easy-to-use, consistent, navigable user interfaces
 - Analyze usability; don't be fooled by wording of use case
- Build physical prototype to validate UI and the use cases



Activity: Structure the Use-Case Model

- General and shared functionality: “uses”
 - Like inheritance: specific (real) uses general (abstract)
 - The generalization captures overlap between use cases
- Additional or optional functionality: “extends”



Activity: Structure the Use Case Model

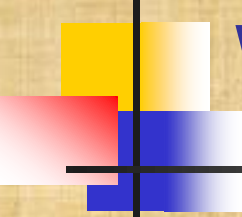
- **Be careful in structuring use-case model**
 - Reflect real use cases
 - Keep things understandable and manageable
 - Decompose functionality in the analysis model, not the use-case model
 - Object-oriented decomposition, not functional decomposition

Summary of Requirements Workflow



■ **Capture requirements as**

- Business model, domain model or glossary to capture system context
- Use-case model that captures functional requirements and use-case-specific nonfunctional requirements
 - Survey description of model as a whole
 - Set of diagrams
 - Detailed description of each use case



Summary of Requirements Workflow (2)

- Set of user-interface sketches/prototypes for each actor
- Supplementary requirements specification for requirements not specific to a use case

■ **Next steps**

- Use cases drive use-case realization in analysis and design and test cases in testing
- Analysis: reformulate use cases as interacting objects