Metodologias de Desarrollo de Software I

Unified Process

Ivar Jacobson
Grady Booch
James Rumbaugh

Agenda

- Unified Process Overview
- Phases
  - Inception phase
  - Elaboration phase
  - Construction phase
  - Transition phase
- Process Components
  - Requirements Capture
  - Analysis & Design
  - Implementation
  - Test
- Estimating Work with Use Cases

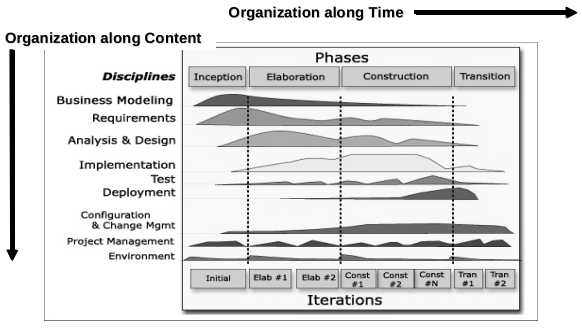
What is Unified Process (UP)?

- Unified Process is a software engineering process
- Unified Process is an iterative and incremental process
- Unified Process development activities
  - create and maintain UML-based models
  - are driven by use cases
- Unified Process is supported by tools
- Unified Process is designed for flexibility and extensibility

Process Overview - Two Dimensions

- Along time, the life-cycle aspects of the process as it will unroll itself. Represents the dynamic aspects of the process, and is expressed in terms of *development cycles*, *phases* and *iterations*
- Along process components, which groups activities logically by nature. Represents the static aspects of the process: how it is described in terms of process components (*activities*, *workflows*, *artifacts*, and *workers*)

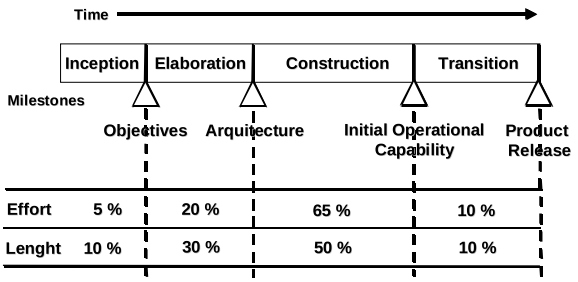
Process Overview



Cycles and Phases

- Dynamic organization of the process along time
- The software lifecycle is broken into cycles of development, and each cycle is divided in four consecutive phases:
  - Inception phase
  - Elaboration phase
  - Construction phase
  - Transition phase

Phases, Milestones and Effort



Effort	5 %	20 %	65 %	10 %
Lenght	10 %	30 %	50 %	10 %

Inception Phase

- Establishes the business case for the system and delimit the project scope
- Identify all external entities with which the system will interact (actors) and defines the nature of this interaction at a high-level (use cases)
- The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan
- At the end of the Inception phase, you examine the lifecycle objectives of the project and decide whether or not to proceed with the development

Inception Phase: Major Deliverables

- Project description
- Initial Market and Risk assesment
- Business model (optional)
  - Business use case, Business collaboration, Organizational Units
- Domain model and Glossary
- Initial Use case model (10 % - 20 % Complete)
  - Use case diagram
  - Textual description of actors and use cases (only the most relevant)
- Project proposal

Inception Phase: Market Factors

- Identified by asking
  - Who or what the competition is?
    - What are competing companies doing right or doing wrong?
  - What technologies you are depending on, such as: web, object databases,…
  - Market trends that influence your project
  - Future trends you are depending on, such as: more home offices, more small companies
  - It is possible to be:
    - Not fast enough to market
    - Too fast to market

Inception Phase: Risk Factors

- People
  - Team not experienced
  - Team not familiar with the technologies to be used
  - Unable to hire people with the right background
- System
  - Number of transactions per time frame
  - Number of expected users
  - Expected duration of some functionality
  - Legacy systems you have to interface with
  - Failure
  - Security

Inception Phase: Risk Factors

- Resource
  - Too many users
  - Too short a schedule
  - Supplier can't deliver product we depend on
- Technology
  - Dependence on a technology that changes
- Corporate
  - Lack of user acceptance
  - Too fast company growth

Elaboration Phase

- Develop a common understanding of the system's scope and desired behavior by exploring scenarios with end users and domain experts
  - Describes most of the use cases and takes into account some of the constrains: non-functional requirements
  - Establish the system's architecture
  - Risk evaluation (requirements, technological, political)
- At the end of the Elaboration phase, you examine the detailed system objectives and scope, the choice of an architecture, and the resolution of major risks

Elaboration Phase: Major Deliverables

- Use case model (at least 80 % complete)
  - Use case Specifications
    - Detailed basic paths
    - Alternative paths
  - Activity diagrams
- User interface diagrammed (optional)
- Architecture Description
- Executable architectural prototype
- Revised risk list
- Project plan
  - Estimating Work with Uses Cases

Construction Phase

- Iteratively and incrementally develop a complete product
- This implies describing the remaining use case, fleshing out the design, and completing the implementation and test of the software
- At the end of the Construction phase, you decide if the software, the sites, the users are all ready to go operational

Construction Phase: Deliverables

- Iteration plans
- Code
- Test plans
- Test results
- Review of iterations
- The software product integrated on the adequate platform
- User guides
- Training manuals
- Demos
- Sales and marketing materials

Transition Phase

- Facilitate user acceptance
  - Beta Testing to validate the new system against user expectations. Measure user satisfaction
- Achieve user self-supportability training them
- Parallel operation with a legacy system that it is replacing
  - Conversion of operational databases
- At the end of the Transition phase you decide whether the lifecycle objectives have been met, and possibly if you should start another development cycle

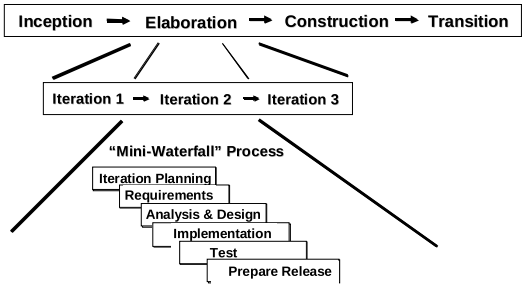
Iterations

- Each phase in the Unified Process can be further broken down into iterations
- An *iteration* is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows incrementally from iteration to iteration to become the final system
- Each iteration goes through all aspects of software development, i.e., all process components

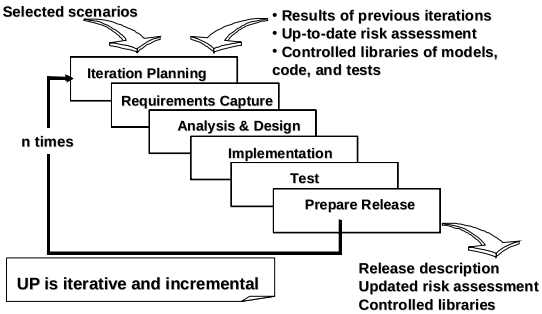
Iteration Life Cycle - Iteration Planning

- 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

The Iteration Life-cycle



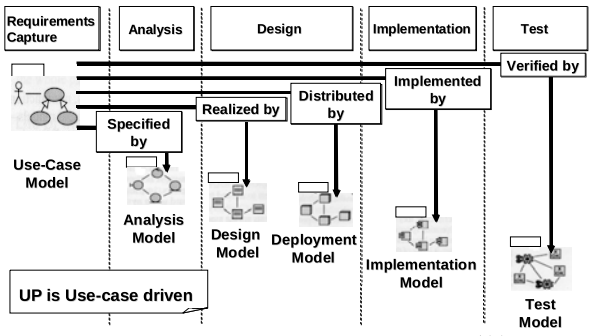
The Iteration Life-cycle: A Mini-Waterfall



Process Components

- UP is composed of 8 process components, which are described by activities, workflows, workers, and artifacts.
- Engineering Process Components
  - Requirements Capture
  - Analysis & Design
  - Implementation
  - Test
- Supporting Components
  - Management, Environment and Deployment

Process Components and Models



Elements of Process Components

- Who? *Worker* represents a position that can be assigned to a person or a team, and it specifies required responsibilities and abilities
- What? *Artifact* is a general term for any kind of description or information created, produced, changed, or used by workers (a model, a model element, or a document)
- How? *Activity* is a piece of work that a worker performs in a process component
- When? *Workflows* illustrate workers collaborations
  - Activity diagram

Iteration Life Cycle - Requirements Capture

- The goal is to describe *what* the system should do and allows the developers and the customer to agree on that description
- Requirements capture results in a use-case model and some supplementary requirements
- Use cases represent the behavior of the system

Iteration Life Cycle - Requirements Capture

- Nonfunctional requirements common for many or all use cases are kept in a separate document and are known as the *supplementary requirements*
- The same use-case model is used during requirements capture, analysis, design, and test

Requirements Capture – Non-functional requirements

- Usability
  - What is known about the users of this system?
  - Are they not used to computers, power users, or somewhere in between?
  - How easy must this system be to use?
- System
  - What kind of system will this software run on?
  - Do we have to port to multiple platforms?
  - Must the software support multiple simultaneous users?

Requirements Capture – Non-functional requirements

- Security
  - What are the needs for secure login or secure transmission of data?
- Persistence
  - Do we have any persistent data?
  - Are there any requirements on the database to use?
- Integration with Other Systems
  - Does this software have to integrate with other software or hardware?

Requirements Capture – Non-functional requirements

- Redundancy
  - Are there any needs for redundant data, subsystems, processes, or hardware?
- Performance
  - Are there any restrictions on how slow or fast the system or any part of it will run?
- Size
  - Are there any restrictions on the size of the system or any part of it?

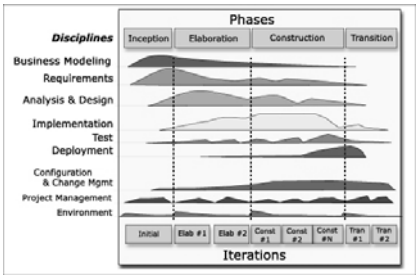
Requirements Capture – Non-functional requirements

- Internationalization
  - Does the software have to support any character set worldwide or only some of them?
  - What information must be translated?

Iteration Life Cycle - Requirements Capture

- During the Inception phase, analysts identified most use cases in order to delimit the system and scope the project and to detail the most critical ones (less than 10%)
- During the Elaboration phase, analysts capture most of the remaining requirements (about the 80%)
- The remaining requirements are captured (and implemented) during the Construction phase
- There is almost no requirements capture during the Transition phase

Iteration Life Cycle - Requirements Capture



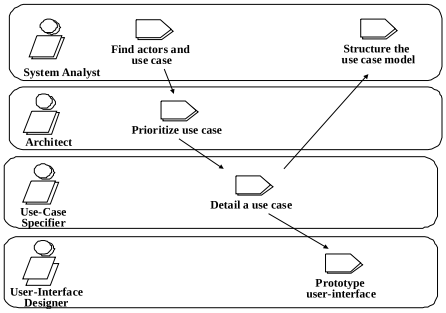
Requirements Capture - Artifacts

- *Use-Case model*, allows software developers and the customer to agree on the requirements (Special Requirements)
- *Actors Description*, parties (human, software, hardware) outside the system that collaborate with the system
- *Architecture description* contains an architectural view of the use-case model depicting the architecturally significant use cases

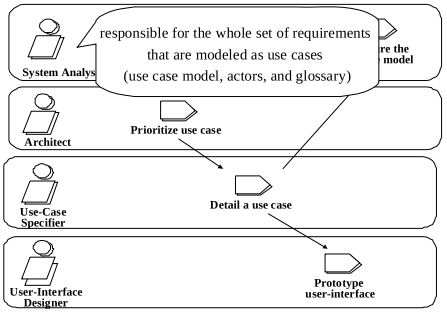
Requirements Capture - Artifacts

- *Glossary*, defines important and common terms used by analysts when they describe the system
- *User-Interface prototype*, help during requirements capture

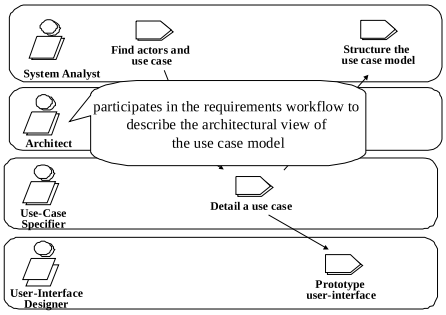
Requirements Capture Workflow



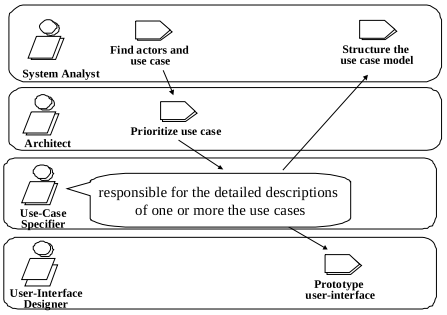
Requirements Capture Workflow



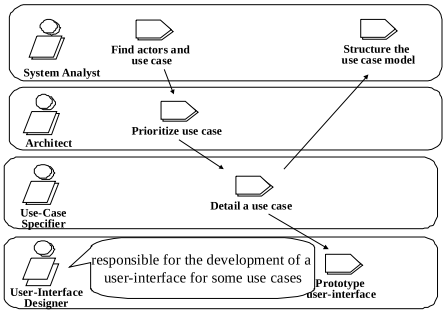
Requirements Capture Workflow



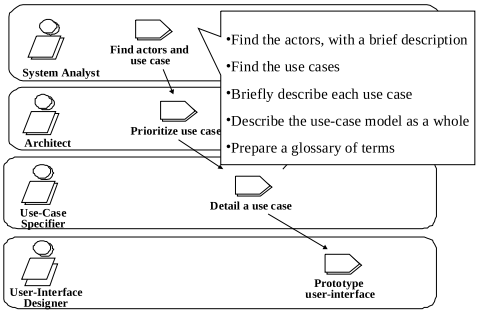
Requirements Capture Workflow



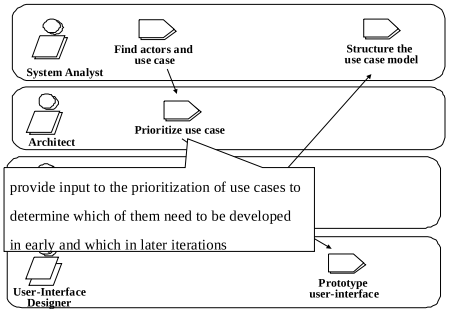
Requirements Capture Workflow



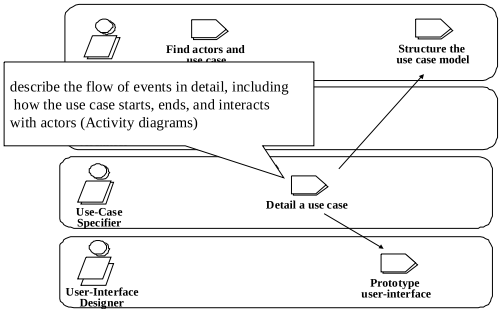
Requirements Capture Workflow



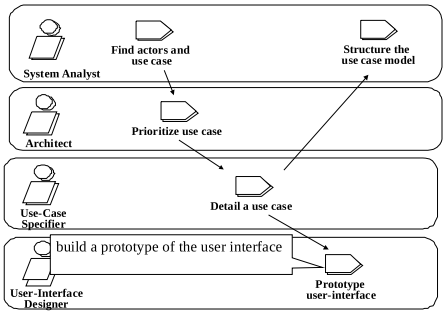
Requirements Capture Workflow



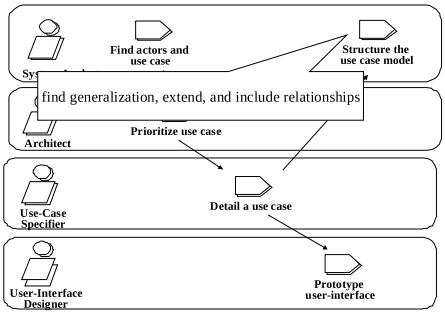
Requirements Capture Workflow



Requirements Capture Workflow



Requirements Capture Workflow



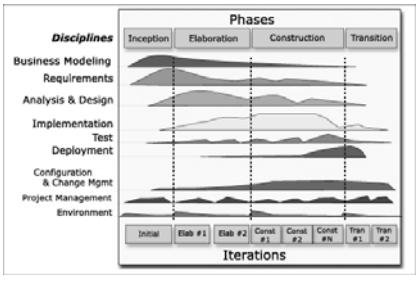
Iteration Life Cycle - Analysis

- **In Analysis the requirements are analyzed as described in the requirements capture by refining and structuring them**
- **The purpose is to achieve a more precise understanding of the requirements and to achieve a description of the requirements**
- **The analysis model**
  - **is described using the language of the developers**
  - **can be viewed as a first cut at a design model**

Iteration Life Cycle - Analysis

- **Analysis is the focus during the initial elaboration iterations**
- **It contributes to a sound and stable architecture and facilities an in-depth understanding of the requirements**
- **Later, during the end of elaboration and in construction when the architecture is stable and requirements are understood, the focus is on design and implementation**

Iteration Life Cycle - Analysis



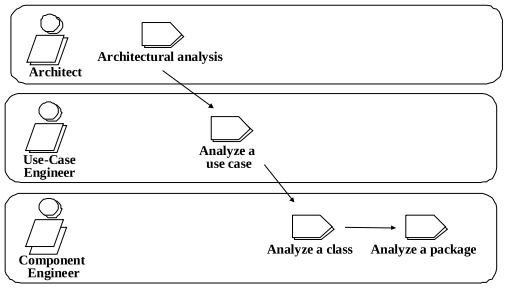
Analysis - Artifacts

- ***Analysis class* represents an abstraction of one or several classes and/or subsystems in the system's design**
  - **Boundary class**
  - **Entity class**
  - **Control class (coordination, sequencing, transactions, and control)**

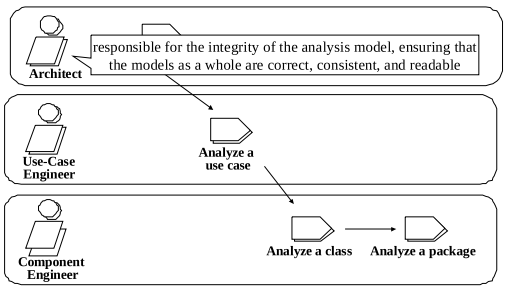
Analysis - Artifacts

- *Use-Case realization - Analysis*
  - **Class diagrams (boundary, control, entity classes)**
  - **Interaction diagrams (collaboration diagrams)**
  - **Flow of events (additional text explaining collaboration diagrams)**
  - **Special requirements**
- ***Analysis package* provides a means of organizing the artifacts of the analysis model**
- ***Architecture description* contains an architectural view of the analysis model depicting its architectural significant artifacts**

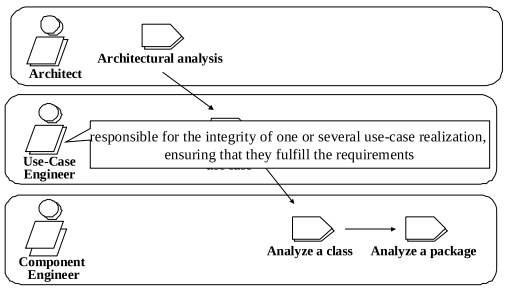
Analysis - Workflow



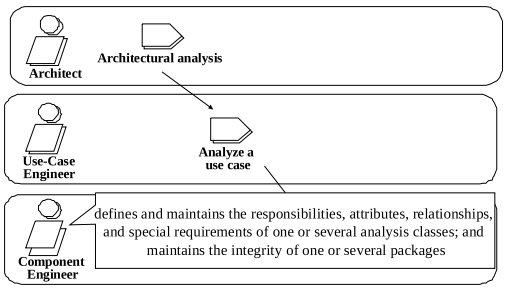
Analysis - Workflow



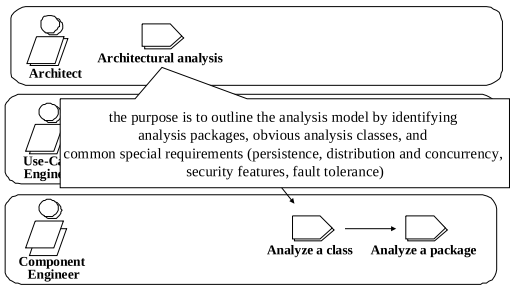
Analysis - Workflow



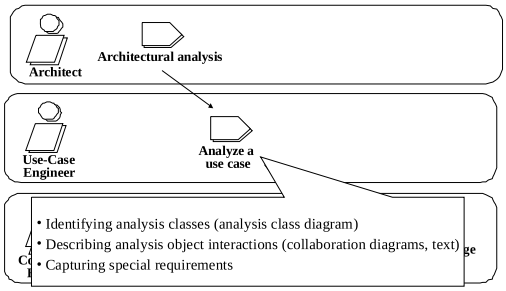
Analysis - Workflow



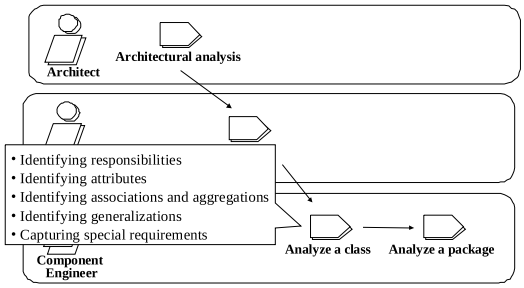
Analysis - Workflow



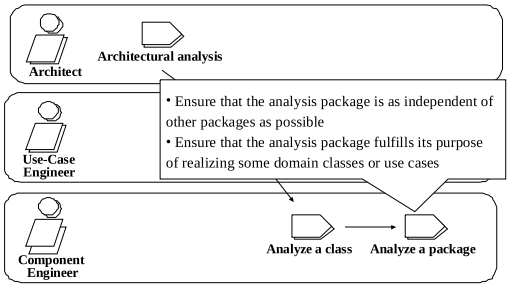
Analysis - Workflow



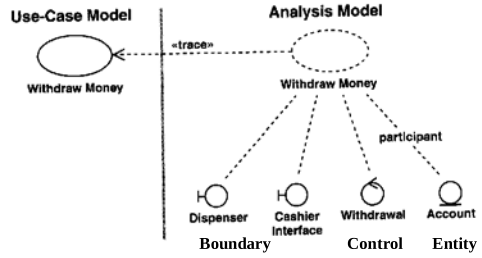
Analysis - Workflow



Analysis - Workflow

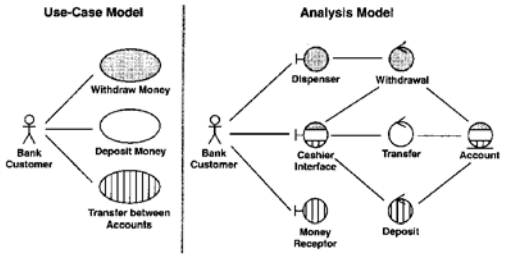


Analysis Realization

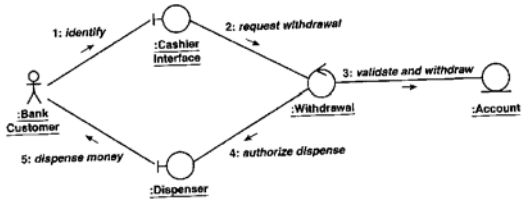


Analysis classes

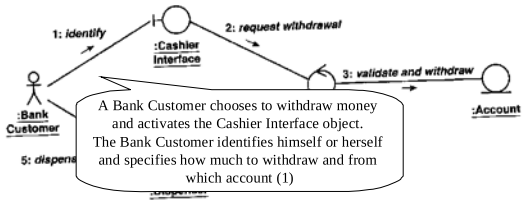
Analysis Realization



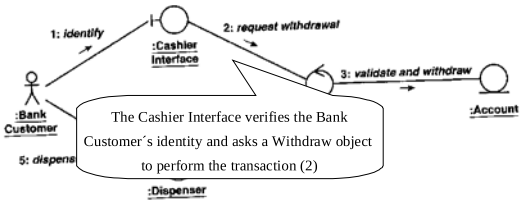
Analysis Realization



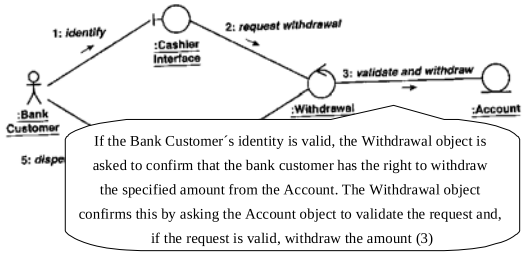
Analysis Realization



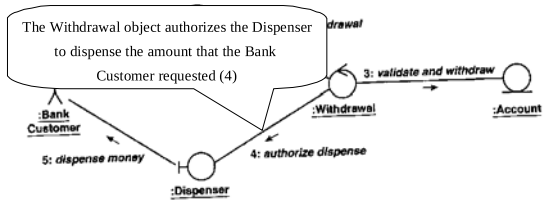
Analysis Realization



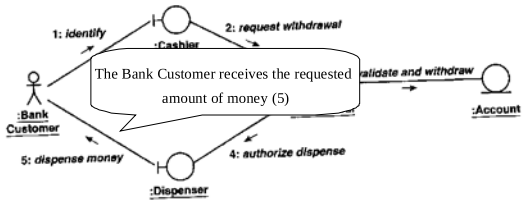
Analysis Realization



Analysis Realization



Analysis Realization



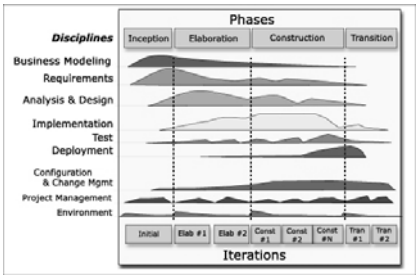
Iteration Life Cycle - Design

- **In design we shape the system and find its form that lives up all requirements made on it**
- **The essential input to design is the result of analysis, the analysis model**
- **The purpose of design is acquire an understanding of issues regarding nonfunctional requirements and constraints related to programming languages, component reuse, operating systems, distribution, and concurrency, database, user-interface technologies**

Iteration Life Cycle - Design

- Design is focus during the end of elaboration and beginning of construction iterations
- It contributes to a sound and stable architecture and creates a blueprint for the implementation model
- During the construction phase, when the architecture is stable and requirements are well understood, the focus shifts to implementation

Iteration Life Cycle - Design



Design - Artifacts

- *Design model*, object model that describes the physical realization of use cases by focusing on how functional and nonfunctional requirements impact that system under consideration
- *Design class*, abstraction of a class or similar construct in the system's implementation

Design - Artifacts

- *Use-Case realization - Design*, is a collaboration within the design model that describes how a specific use case is realized and performed, in terms of design classes and their objects
  - Class diagram
  - Interaction diagram (sequence diagram)
  - Flow of event, textual description that explains and complements the diagrams
  - Implementation requirements, textual description that collects requirements, such as nonfunctional requirements

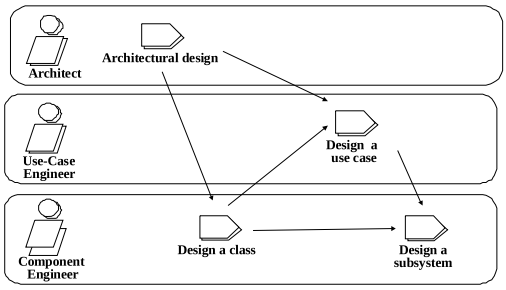
Design - Artifacts

- *Design subsystem*, provides a means of organizing the artifacts of the design model in more manageable pieces. A subsystem can consist of design classes, use-case realizations, interfaces, and other subsystems
- *Interfaces*, are used to specify the operations provided by design classes and subsystems
- *Architecture description* (View of the Design Model), contains an architectural view of the design model depicting its architecturally significant artifacts

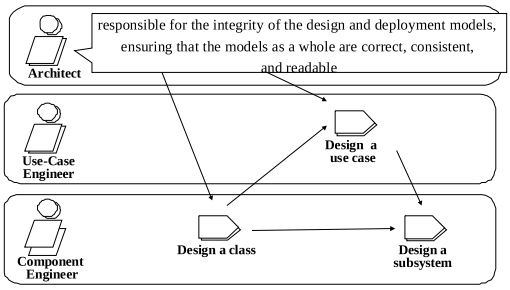
Design - Artifacts

- *Deployment model*, is an object model that describes the physical distribution of the system in terms of how functionality is distributed among computational nodes
- *Architecture description* (View of the Deployment Model), contains an architectural view of the deployment model depicting its architecturally significant artifacts

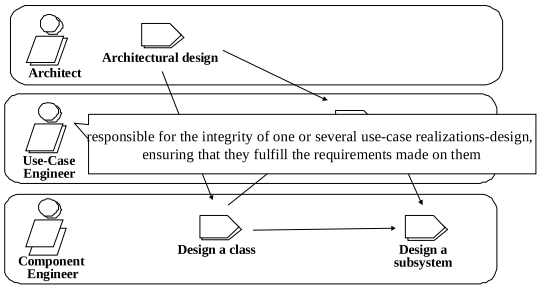
Design Workflow



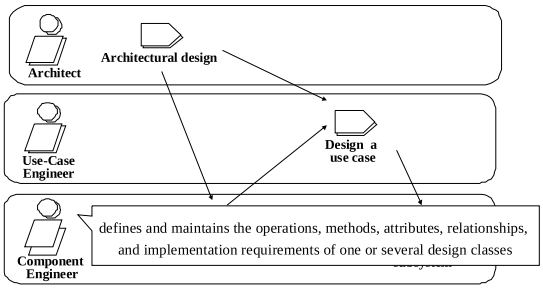
Design Workflow



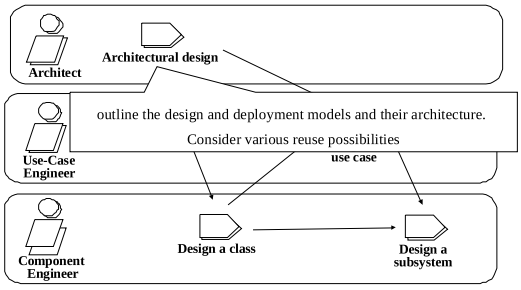
Design Workflow



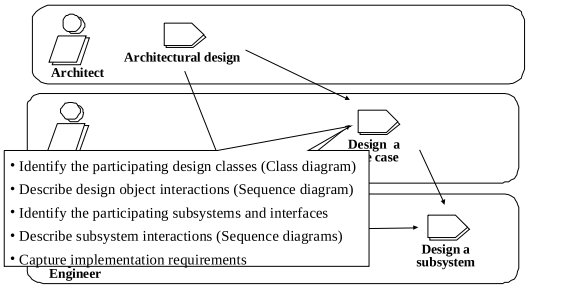
Design Workflow



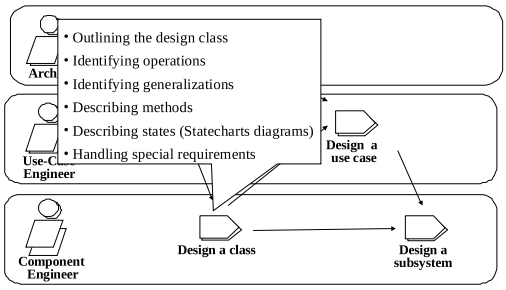
Design Workflow



Design Workflow



Design Workflow



Design Realization

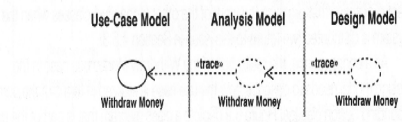


FIGURE 3.7 Use-case realizations in different models.

Design Realization

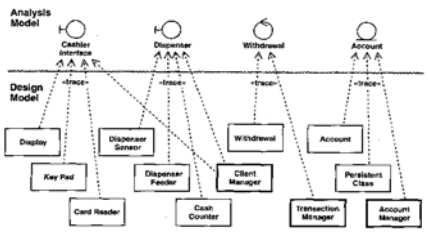


FIGURE 3.8 Design classes in the design model tracing to analysis classes in the analysis model.

Design Realization

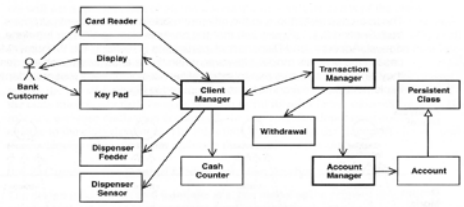


FIGURE 3.9 A class diagram that is part of a realization of the Withdraw Money use case in the design model. Each design class participates and plays roles in the use-case realization.

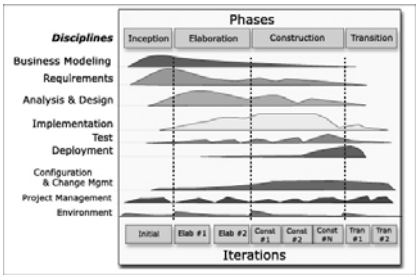
Iteration Life Cycle - Implementation

- The system is realized through implementation producing the *sources* (source-code files, header files, makefiles, and so on) that will result in an executable system
- The sources are described in an *implementation model* that consists of modules structured into implementation packages
- Implementation includes testing the separate classes and/or packages, but not testing that the packages/classes work together

Iteration Life Cycle - Implementation

- Implementation is the focus during the construction iterations
- Implementation is also done during elaboration to create the executable architectural baseline and during transition to handle late defects such as those found when beta releasing the system

Iteration Life Cycle - Implementation



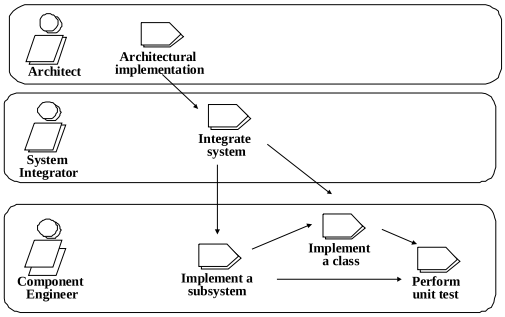
Implementation - Artifacts

- ***Implementation model*, describes how elements in the design model, such as design classes, are implemented in terms of components such as code files, executables, and so on**
- ***Component*, physical packaging of model elements, such as design classes in the design model**
- ***Implementation subsystem*, provides a means of organizing the artifacts of the implementation model into more manageable pieces (package in Java, project in Visual Basic, directory in C++)**

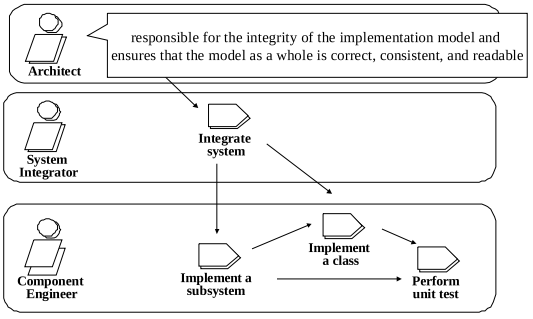
Implementation - Artifacts

- ***Interface*, specifies the operations implemented by components and implementation subsystems**
- ***Architecture description* (View of the Implementation Model), contains an architectural view of the implementation model depicting its architecturally significant artifacts**
- ***Integration build plan*, the software is built incrementally in manageable steps so that each step yields small integration or test problems**

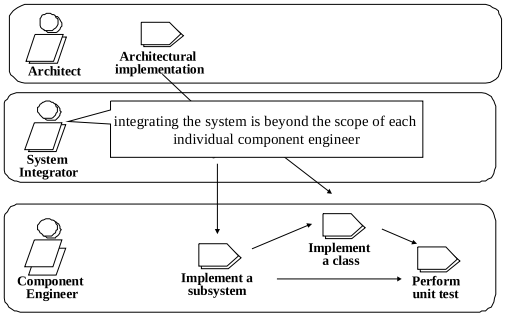
Implementation Workflow



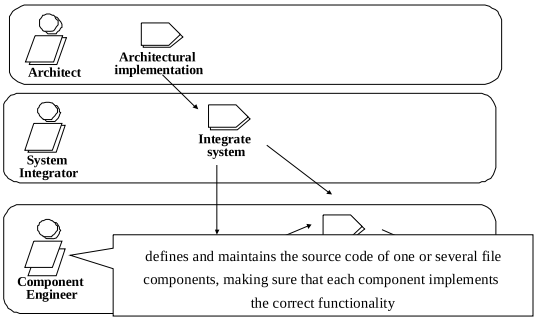
Implementation Workflow



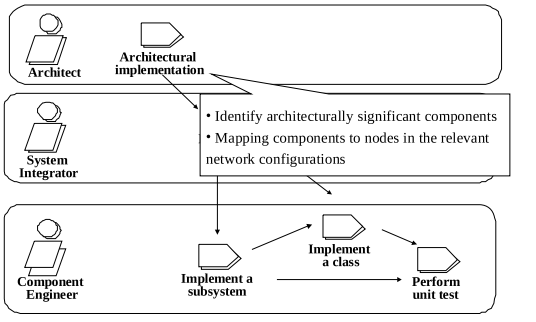
Implementation Workflow



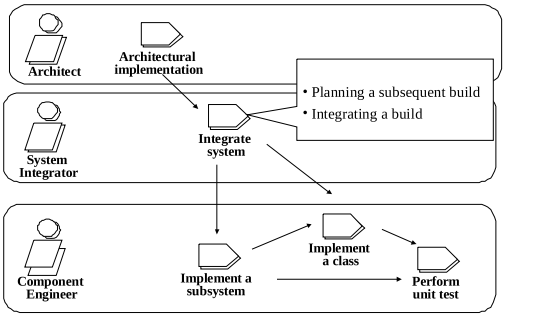
Implementation Workflow



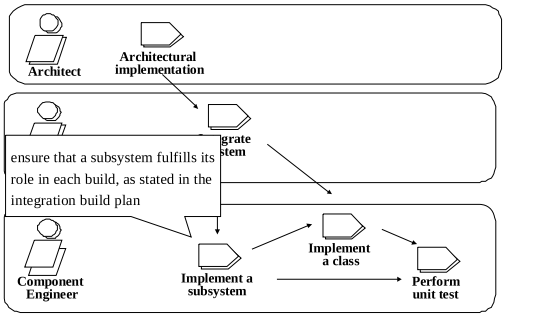
Implementation Workflow



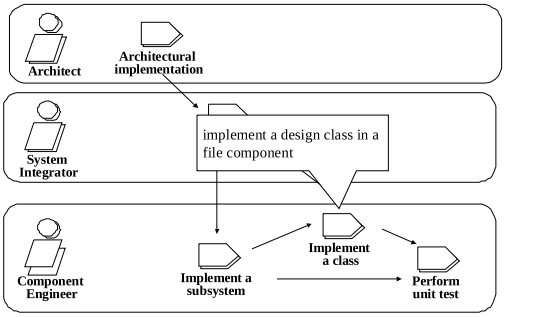
Implementation Workflow



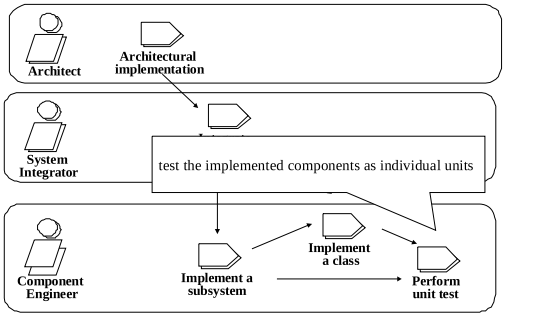
Implementation Workflow



Implementation Workflow



Implementation Workflow



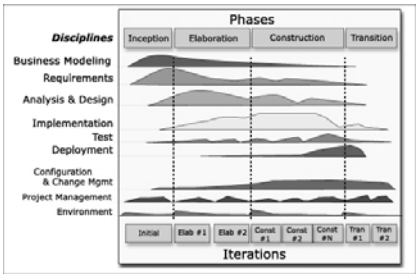
Iteration Life Cycle - Test

- 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
- First test each use case separately to verify that its participating classes work together correctly
- Then you test (certain aspects of) the system as a whole with use-case descriptions as input to this test

Iteration Life Cycle - Test

- Some initial test planning may occur during inception when the system is scoped
- Testing is primarily employed when each build (as an implementation result) is integration and system tested
- During transition phase, the focus shifts toward fixing defects detected during early usage and toward regression testing

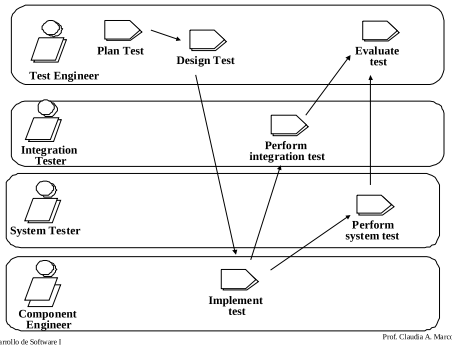
Iteration Life Cycle - Test



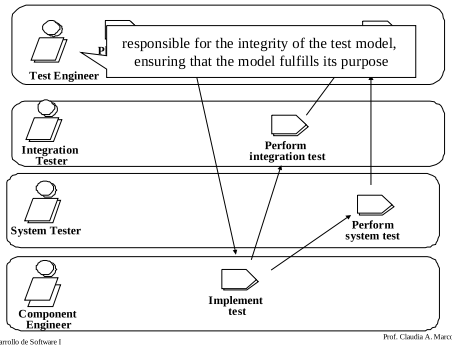
Test - Artifacts

- *Test model*, describes how executable components in the implementation model are tested by integration and system tests
- *Test cases*, specify what to test which input or result, and under which conditions to test
- *Test procedures*, specify how to perform the test cases
- *Test components*, automate the test procedures
- Testing also results in a test plan, evaluation of the performed tests, and defects that can be feedback to other core workflows, such as design and implementation

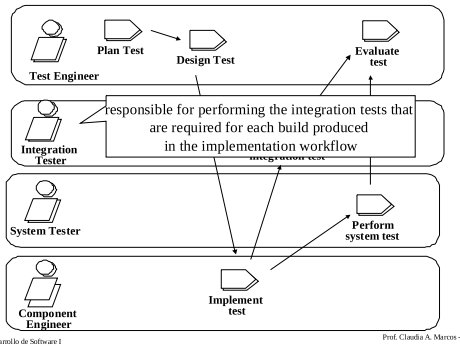
Test Workflow



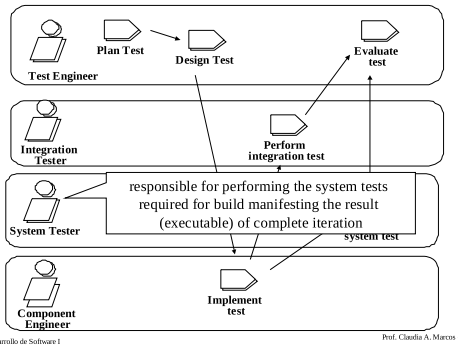
Test Workflow



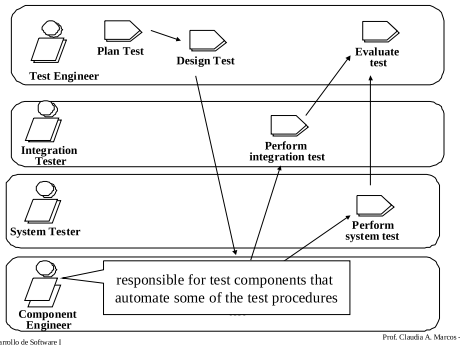
Test Workflow



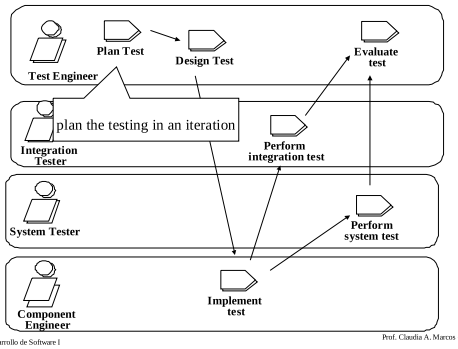
Test Workflow



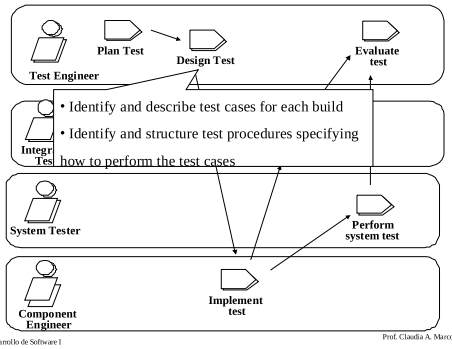
Test Workflow



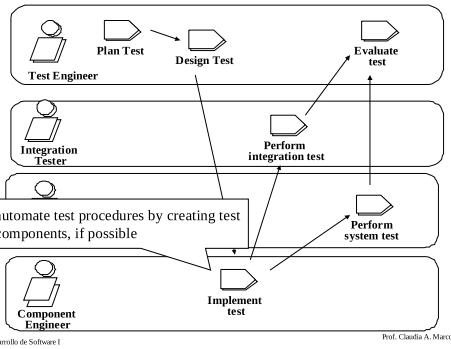
Test Workflow



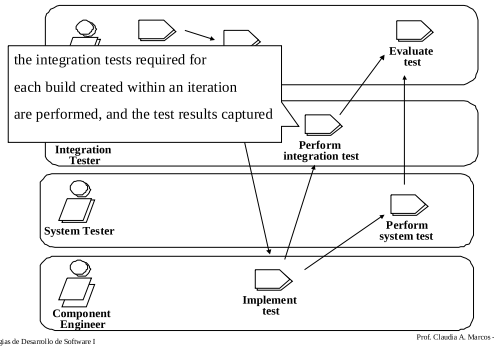
Test Workflow



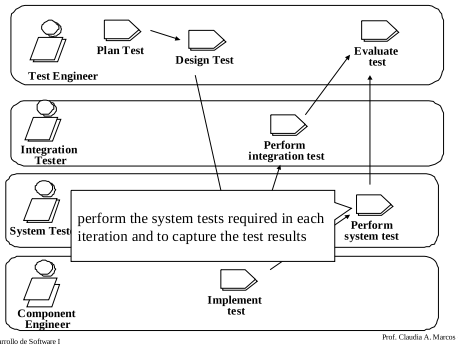
Test Workflow



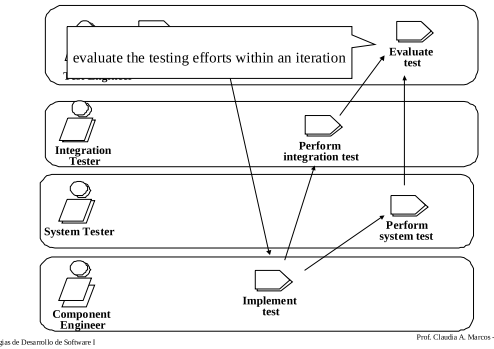
Test Workflow



Test Workflow



Test Workflow



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**

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

- **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

- 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

Detailed Iteration Life Cycle Activities

- Implementation
  - Automatically generate code from the design model
  - Manually generate code for operations
  - Complete test procedures
  - Conduct unit and integration tests

Detailed Iteration Life Cycle Activities

- 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

Work Allocation Within an Iteration

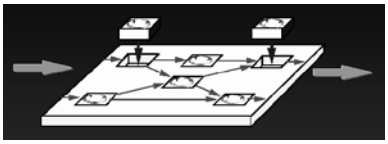
- Use Cases make convenient work packages for test and assessment teams
- Packages are also useful in determining the granularity at which configuration management will be applied
  - For example, check-in and check-out of individual packages

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

There Is No Universal Process

- The Unified Process is designed for flexibility and extensibility
  - Allows a variety of lifecycle strategies
  - Selects what artifacts to produce
  - Defines activities and workers
  - Models process concepts



References

- **The Unified Software Development Process**

Authors: I. Jacobson, G. Booch and J. Rumbaugh
Editorial: Addison Wesley; Date: January 1999
ISBN: 0-201-57169-2

- **UML Distilled: Applying the Standard Object Modeling Language**

Authors: M. Fowler and K. Scott
Editorial: Addison Wesley; Date: January 1998
ISBN: 0-201-32563-0

References

- **Applying Use Cases: A practical guide**

Authors: G. Schneider and J. P. Winters
Editorial: Addison Wesley; Date: February 2001
ISBN: 0-201-70853-1

- **Web Sites**
  - **Rational Software Corporation**

http://www.rational.com

  - **Herramientas basadas en UML**

http://www.objectsbydesign/tools/umltools_byPrice.html