Software Design & Architecture

Engr. Abdul-Rahman Mahmood

DPM, MCP, QMR(ISO9001:2000)

armahmood786@yahoo.com	alphasecure@gmail.com
alphapeeler.sf.net/pubkeys/pkey.htm	http://alphapeeler.sourceforge.net
pk.linkedin.com/in/armahmood	http://alphapeeler.tumblr.com
www.twitter.com/alphapeeler	armahmood786@jabber.org
www.facebook.com/alphapeeler	alphapeeler@aim.com
abdulmahmood-sss alphasecure	mahmood_cubix 48660186
armahmood786@hotmail.com	alphapeeler@icloud.com
http://alphapeeler.sf.net/me	http://alphapeeler.sf.net/acms/

Domain Model and User Stories

User Stories

- A User Story is one or more sentences in the everyday or business language of the end user or user of a system that captures what a user does or needs to do as part of his or her job function.
- It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper note card.

Domain Model

- Captures the most important types of objects in a system.
- Describing "things" in a system and how these things are related to each other.
- A "**thing**" can be an object, a class, an interface, a package or a subsystem, which is part of the system being developed.
- Very important process because it is employed throughout the entire system development life cycle.

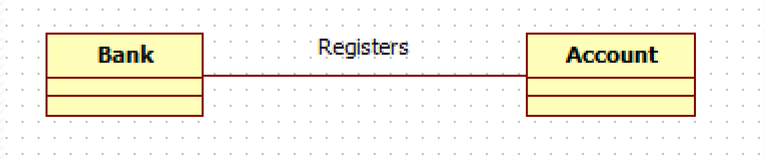
Characteristics of Domain Modeling

- Visual representation of conceptual classes.
- Associations and relationships between concepts (e.g Payment PAYS-FOR Sales).
- Attributes for information content (e.g. Sale records DATE and TIME).
- Does not include operations / functions.
- Does not describe software classes.
- Does not describe software responsibilities.

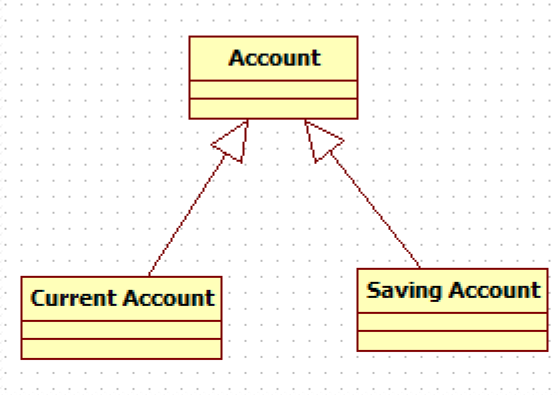
Steps To Create A Domain Model

- 1. Create User Stories
- 2. Identify candidate conceptual classes
- 3. Draw them in a UML domain model
- 4. Add associations necessary to record the relationships that must be retained
- 5. Add attributes necessary for information to be preserved
- 6. Use existing names for things, the vocabulary of the domain

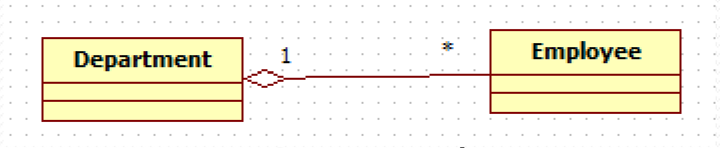
Relationships



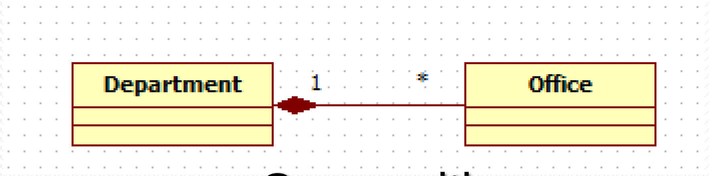
Association



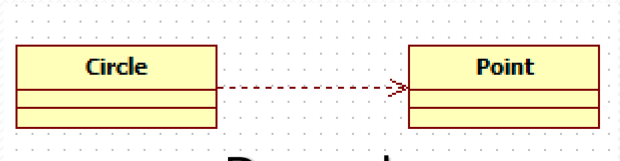
Generalization



Aggregation



Composition



Dependency

Example Of Domain Model

Online Video Sale & Rental Shop

High Level Requirements

The system must be able to keep track of which movie videos have been bought/rented and by whom.

classes & associations: customer *Buys* movie video;
customer *Rents* movie video

For videos bought, the system must record the quantity bought; for videos rented, the system must record which copy of the video has been rented and when it is due back.

classes & associations: customer Rents movie video;
–> movie video *Has* rental copy; customer *Rents* rental copy;

Attributes : Buys –> quantity;
Rentalcopy -> copyNumber, dateDue

The system must keep track of overdue rental videos and allow notices to be sent to customers who have videos overdue.

functional requirement: no new domain model requirements in this statement

The video shop will have a customer membership option for an annual fee, which will entitle the member to discounts (10%) on video sales and rentals.

generalization: Member *is a kind of* Customer

Member *Specializes* Customer

- Members should be able to make reservations for movie video rentals either in person at the store, by telephone or via the Web.
  - **classes & associations:** Member *Reserves* Rentalcopy

- A member can *reserve* at most five movie videos at any one time, but there is no limit on how many movie videos a member or nonmember can rent at any one time.
  - **constraint:** max-card(rental copy, *Reserves*) = 5
  - max-card(rental copy, *Rents*) = *

As an added feature, the video shop would like to allow customers (either members or nonmembers) to input, via the Web, mini-reviews (up to 100 words) and a rating (from 1, lowest, to 5, highest) of movies they have rented.

classes & associations: Customer *Provides* review *IsFor* Movie Video

–> Customer *Provides* Review;

MovieVideo *Has* Review

attributes: Review –> review text, rating

✓ These reviews should be anonymous if the customer so wishes (i.e., the customer can specify whether or not he wants his name to be made known when other customers browse the reviews).

Attributes: Review –> anonymous

✓ The video shop maintains the following information about all customers (members or nonmembers): name, address, phone number, fax number, age, sex, and email address

- **Attributes :** Customer–> name, address,
- phoneNumber, faxNumber, age, gender, email;

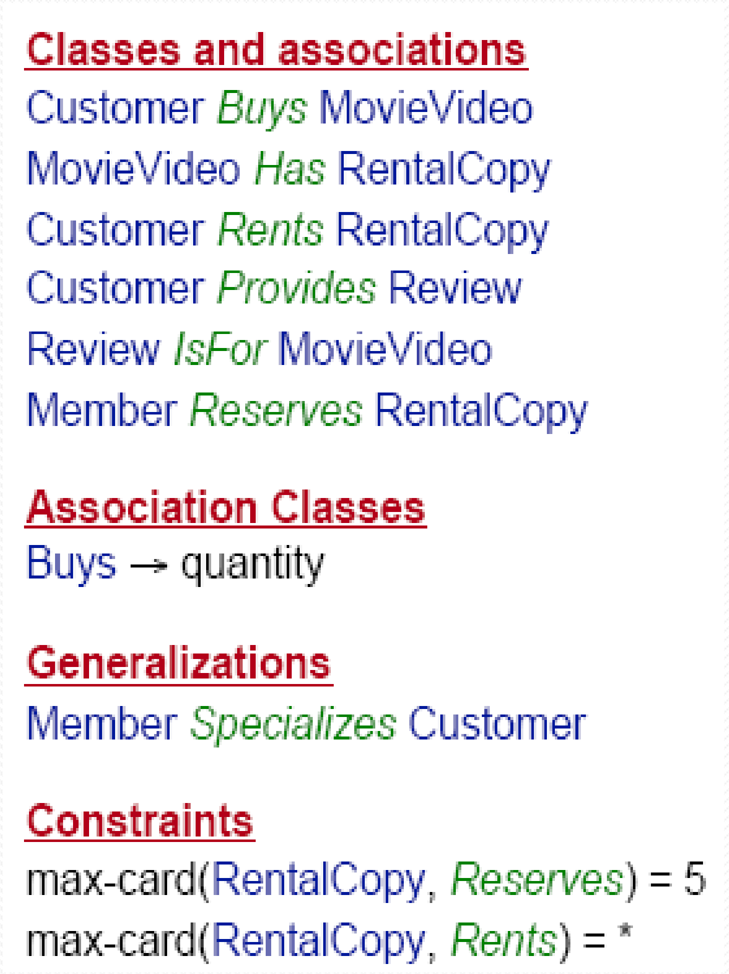
- ✓ In addition, members are assigned a membership number by the video shop when they become members and a password, which allows them to access the member's only area of the video shop's web site, including accessing and changing their personal information.

attributes: Member –>memberNumber, password

- An employee must be able to enter the basic information about a movie video (i.e., title, leading actor(s), director, producer, genre, synopsis, release year, running time, selling price, and rental price).

attributes: MovieVideo –> title, leadingActor[0..*], director, producer, genre, synopsis, releaseYear, runningTime, sellingPrice, rentalPrice

Summarization



Classes and associations

Customer *Buys* MovieVideo

MovieVideo *Has* RentalCopy

Customer *Rents* RentalCopy

Customer *Provides* Review

Review *IsFor* MovieVideo

Member *Reserves* RentalCopy

Association Classes

Buys → quantity

Generalizations

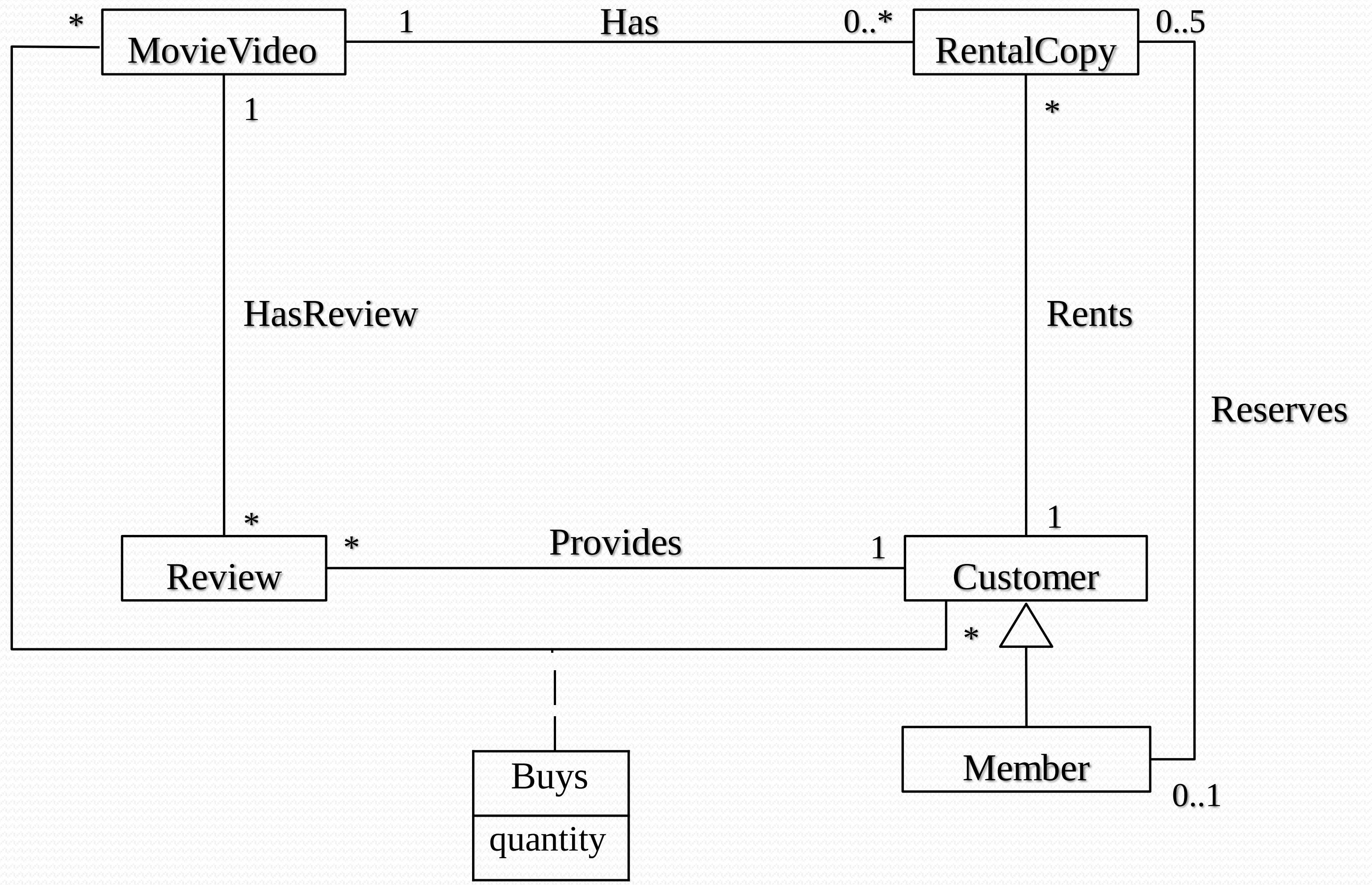
Member *Specializes* Customer

Constraints

max-card(RentalCopy, *Reserves*) = 5

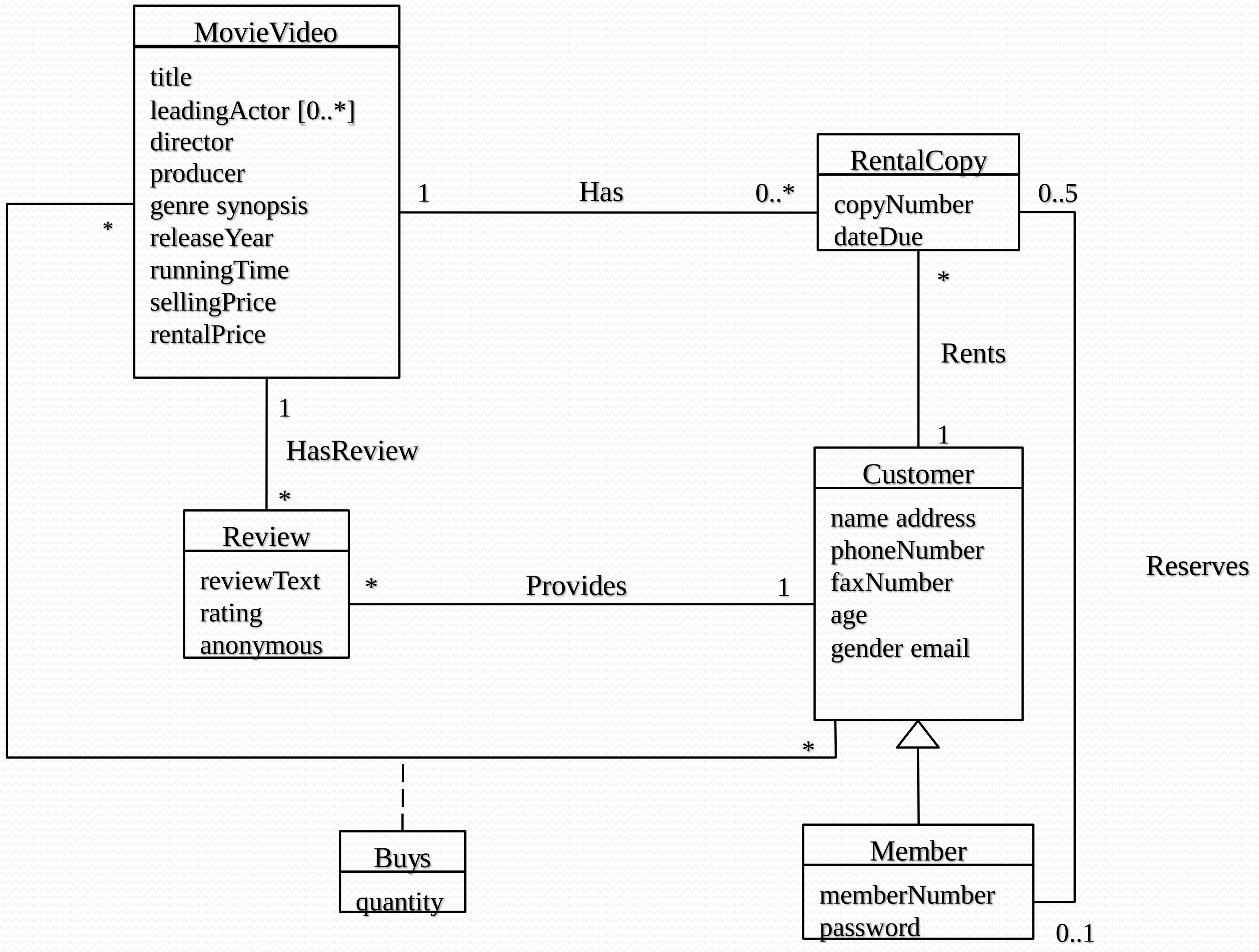
max-card(RentalCopy, *Rents*) = *

Solution

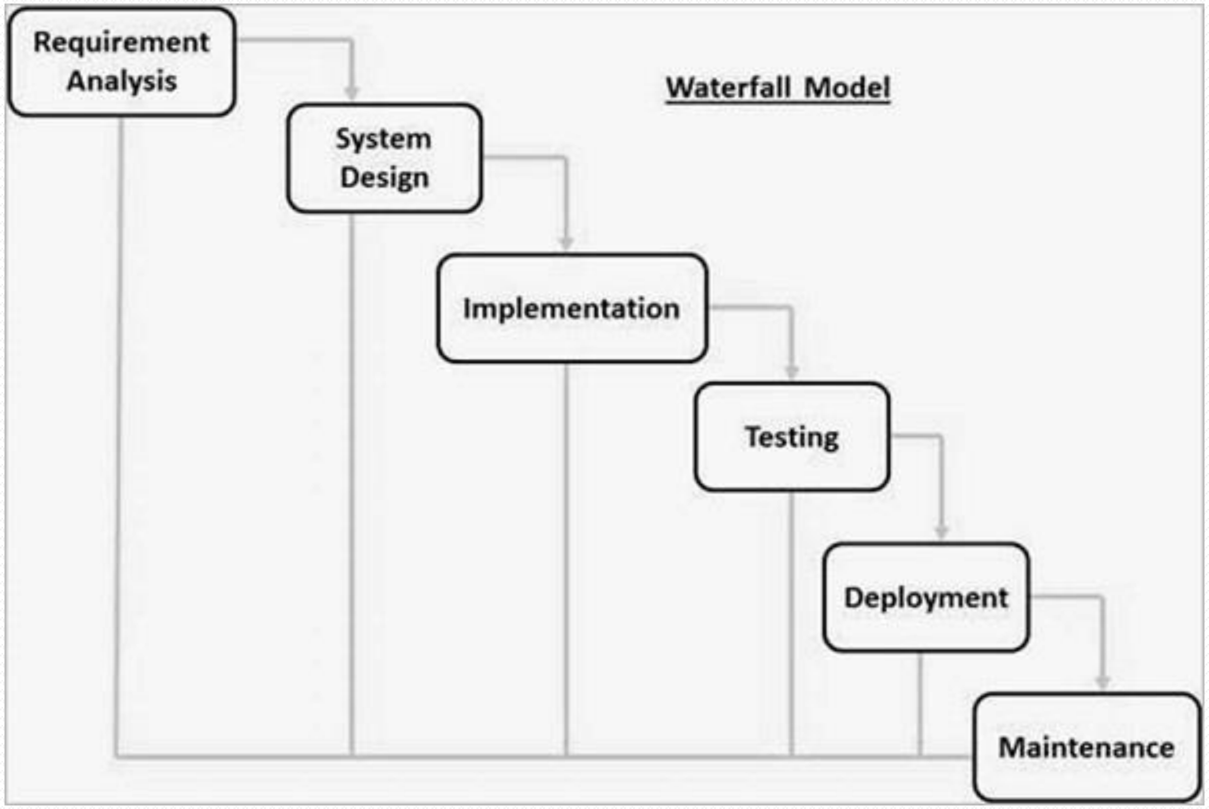


MovieVideo
RentalCopy
Review
Customer
Member
Buys
quantity
Has
HasReview
Rents
Reserves
Provides
1
0..*
0..5
*
1
*
*
1
*
1
*
0..1

Domain Model



Waterfall Model



Unified Process Introduction and History

What is the Unified Process

- UML driven Iterative process model (framework).
- OO methodology for large-scale software.
- Develops high-risk elements in early iterations
- Deliver value to customer
- Accommodate change early on in project
- Work as one team
- Adaptable methodology - can be modified for the specific software product to be developed
- 2D process : phases and workflows
- Utilizes Millers Law

Miller's Law

- Humans concentration on 7 *chunks* of information
- To handle larger chunks, use *stepwise refinement:*
  - <u>Concentrate</u> on most important aspects
  - <u>Postpone</u> aspects that are less critical
  - Results in product in small portions (incrimination)

- History of UP
  - Some roots in "*Spiral Model* " of Barry Boehm
  - Core initial development around 1995-1998
  - Canadian Air Traffic Control project as test bed
  - Philippe Kruchten : chief architect of UP/RUP
  - Rational Corporation had commercial product in mind (*RUP, now owned by IBM*) but also reached out to public domain (UP)

Unified Process Phases

Basic Characteristics of the Unified Process

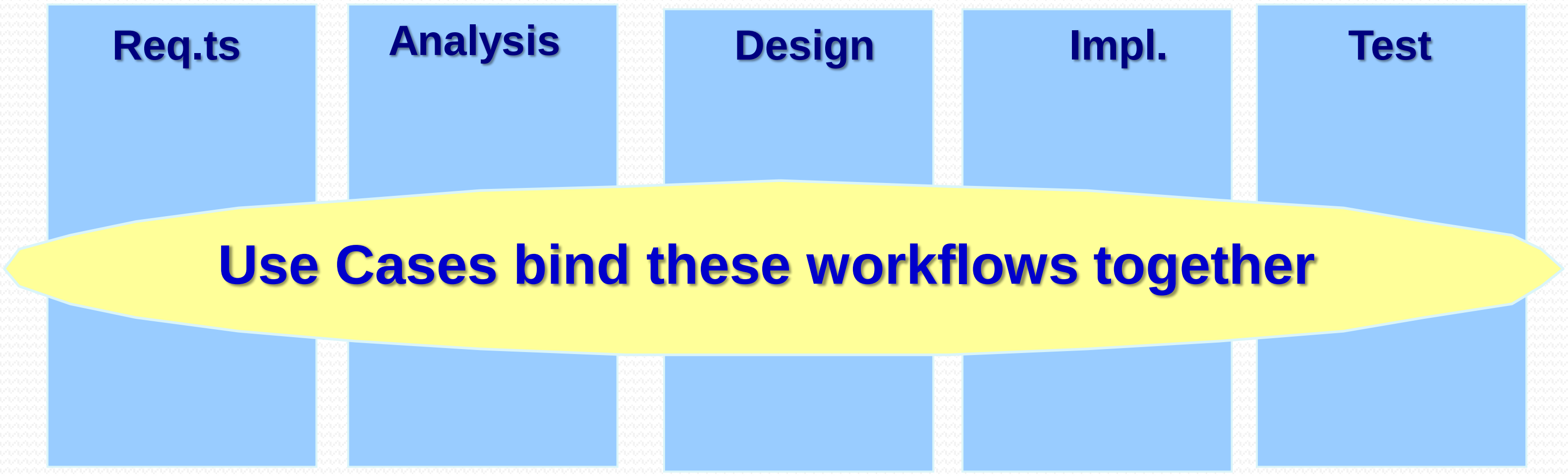
- Object-oriented
- Use-case driven
- Architecture centric
- Iteration and incrementation

Basic Characteristics of the Unified Process

- <u>Object-oriented</u>
  - Utilizes object oriented technologies.
  - Classes are extracted during OOA and designed during OOD.

<u>Use-case driven</u>

  - Utilizes use case model to describe complete functionality of the system



Basic Characteristics of the Unified Process

- <u>Architecture centric</u>
  - Focus core <u>architecture</u> in the <u>early iterations</u>
  - In earliest iterations, get high valued requirements
  - View of the whole design with the <u>important characteristics made more visible</u>
  - Expressed with <u>class diagram</u>

Basic Characteristics of the Unified Process

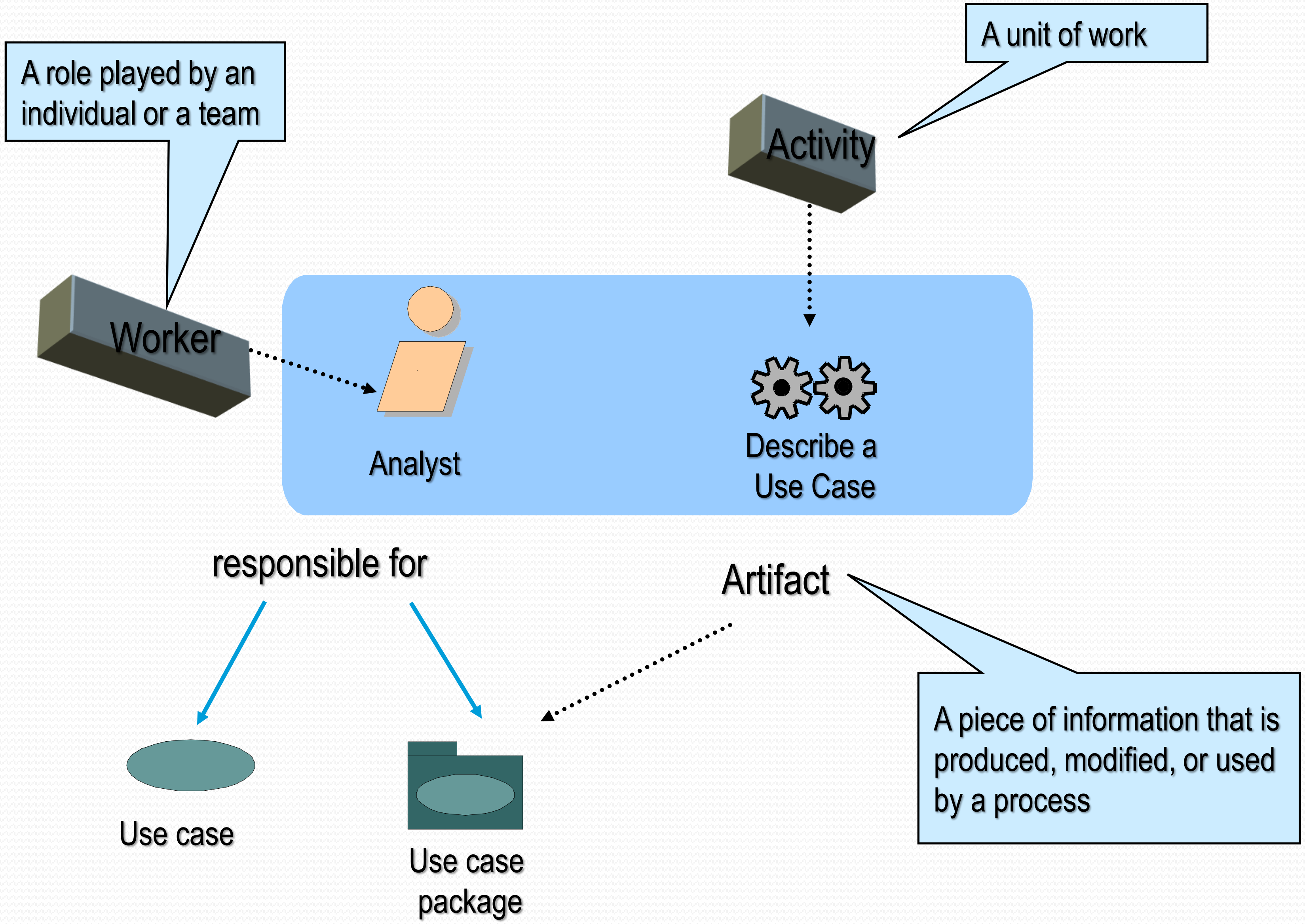
Iteration and incrementation

- Way to divide the work
- Iterations are steps in the process, and increments are growth of the product
- The basic software development process is iterative
  - Each successive version is intended to be closer to its target than its predecessor

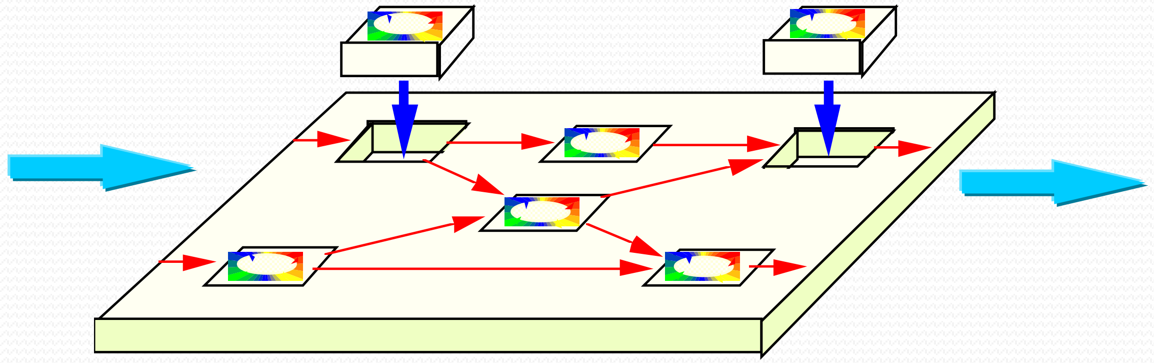
The Rational Unified Process

- RUP is a method of managing OO Software Development
- It can be viewed as a Software Development Framework which is extensible and features:
  - Iterative Development
  - Requirements Management
  - Component-Based Architectural Vision
  - Visual Modeling of Systems
  - Quality Management
  - Change Control Management

The Unified Process is Engineered


A role played by an individual or a team
A unit of work
Activity
Worker
Analyst
Describe a Use Case
responsible for
Artifact
A piece of information that is produced, modified, or used by a process
Use case
Use case package

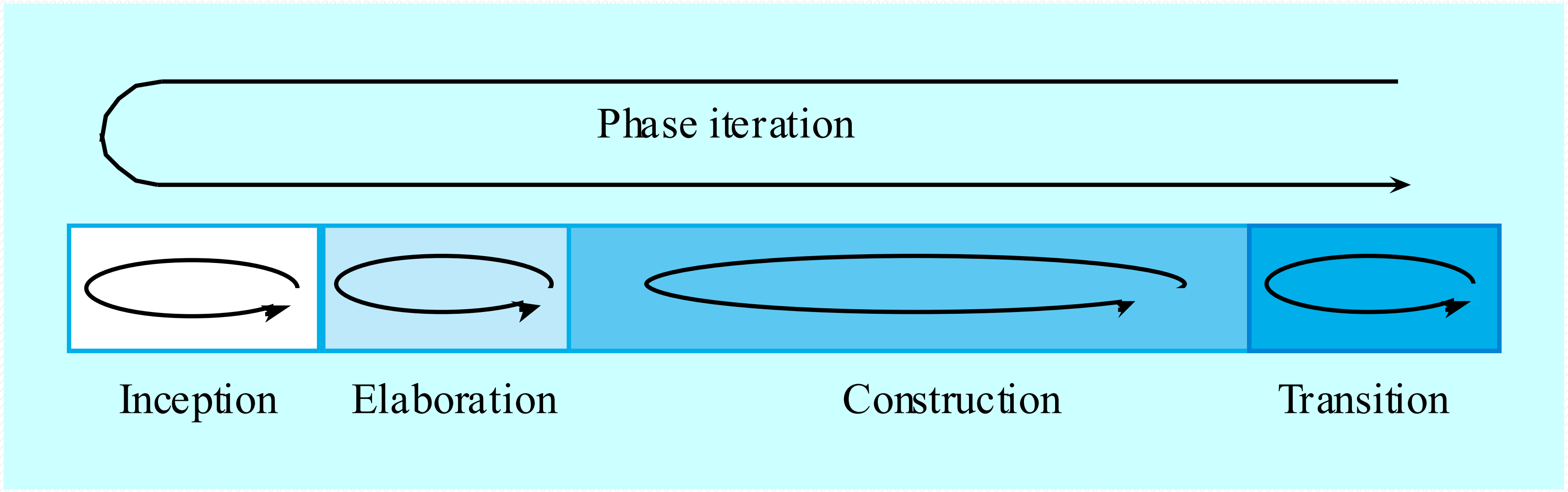
The Unified Process is a Process Framework



While the Unified Process is widely used, 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 concepts
  - IT IS A PROCESS FRAMEWORK for development

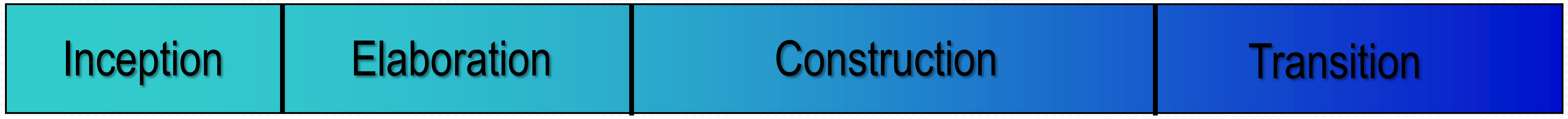
Unified Process Model



Goals and Features of Each Iteration

- Slowly chip away at the project risks:
  - performance risks
  - integration risks (different vendors, tools, etc.)
  - conceptual risks (hunt out analysis and design flaws)
- Perform a "miniwaterfall" project that ends with a delivery of something tangible in code.
- Each iteration is risk-driven.
- The result of a single iteration is an incremental improvement.

Unified Process Phases



- Inception
  - Define **business case**, **risks**, **10% requirements** identified, estimate next phase effort.
- Elaboration
  - **Understanding of problem / architecture**, **risk significant** units are coded/tested, **80%** requirements identified.
- Construction
  - System design, **programming and testing**.
- Transition
  - **Deploy** the system in its operating environment.

The Phases/Workflows of the Unified Process

- Phase is Business context of a step

Workflow is Technical context of a step

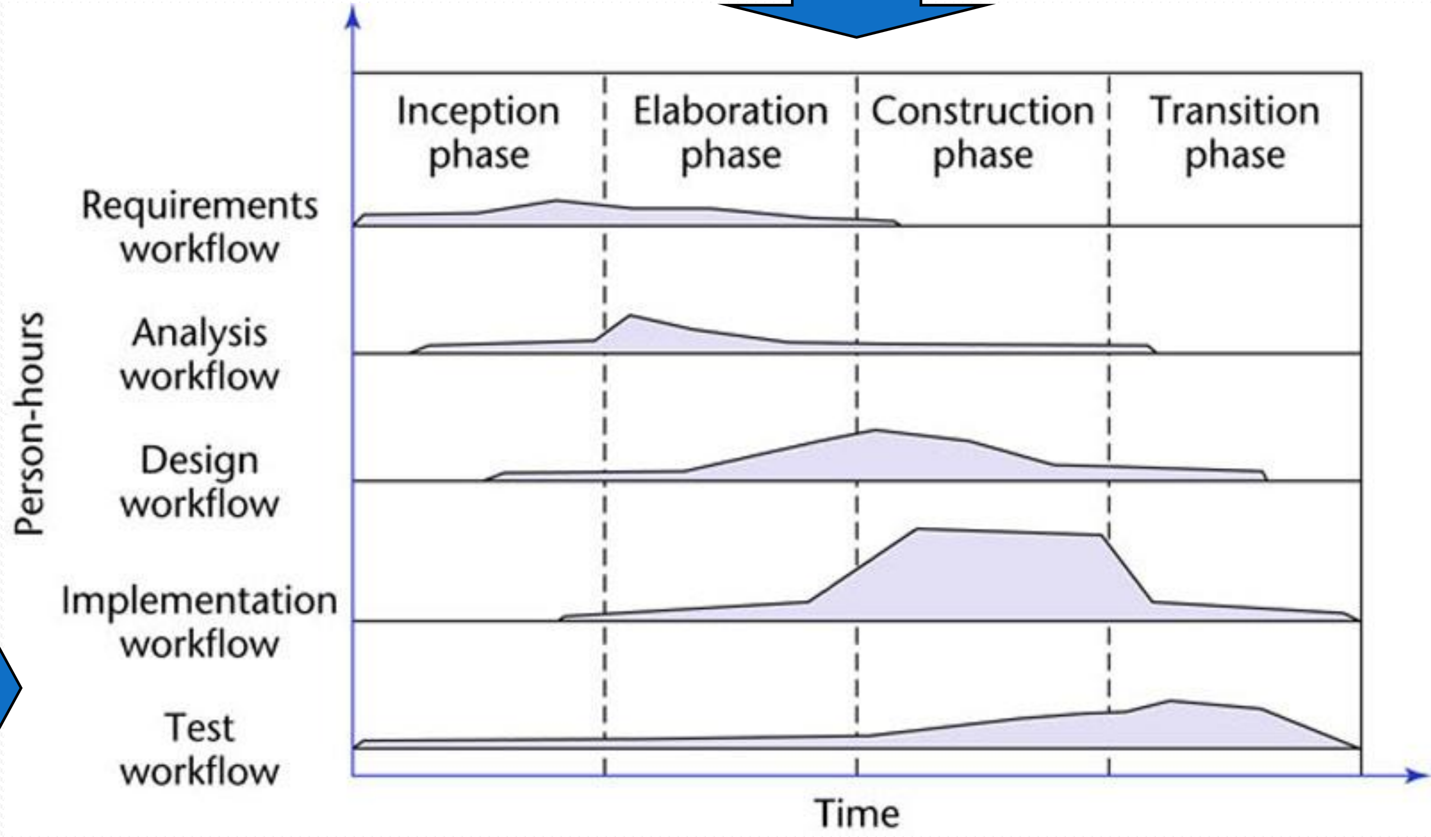


Figure 3.1

The Phases/Workflows of the Unified Process

- **NOTE: Most of the requirements work or workflow is done in the inception phase.**

- **However some is done later.**

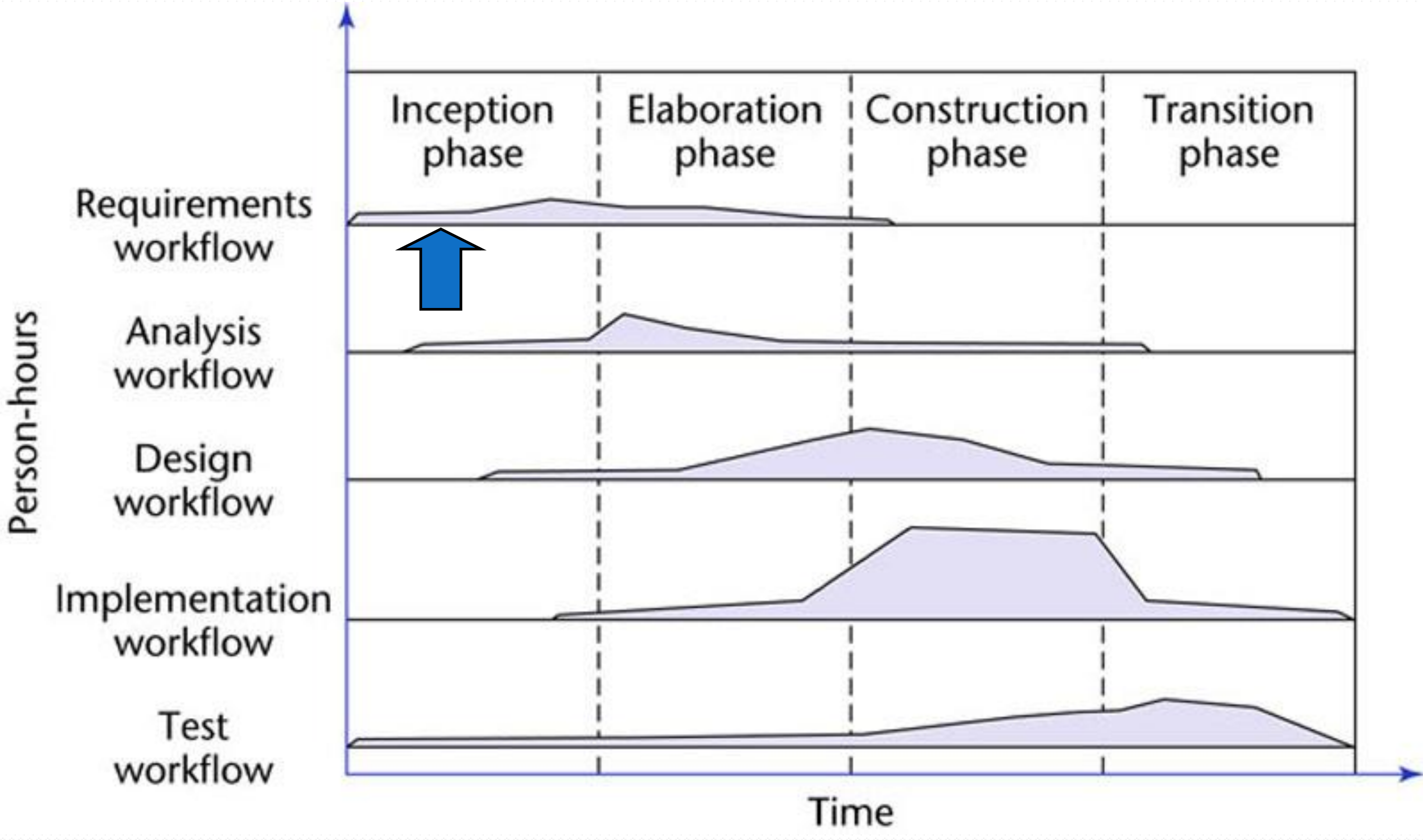


Figure 3.1

The Phases/Workflows of the Unified Process

- **NOTE: Most of the implementation work or workflow is done in construction**

- **However some is done earlier and some later.**

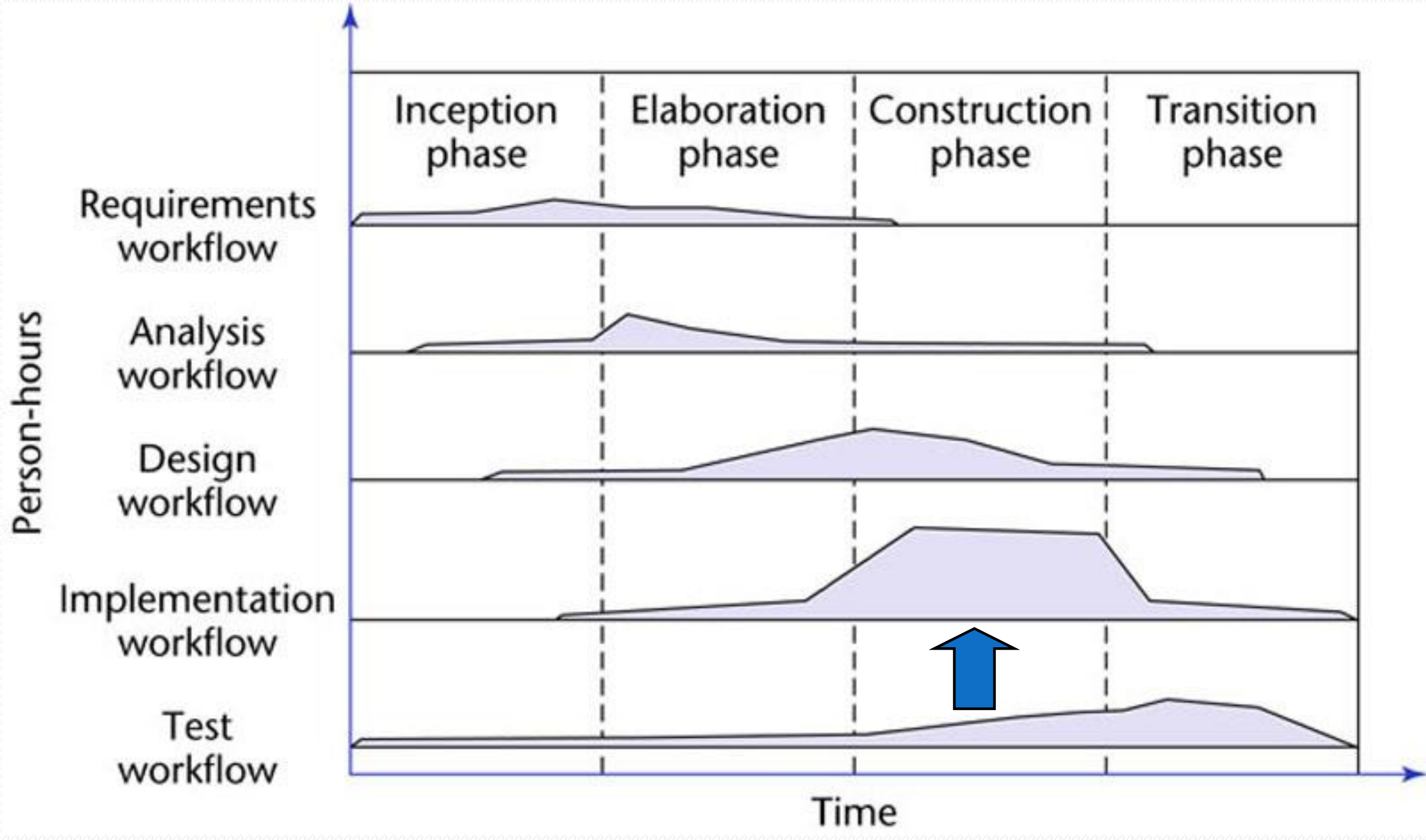


Figure 3.1

Phase Deliverables

Inception Phase	Elaboration Phase	Construction Phase	Transition Phase
• The initial version of the domain model • The initial version of the business model • The initial version of the requirements artifacts • A preliminary version of the analysis artifacts • A preliminary version of the architecture • The initial list of risks • The initial ordering of the use cases • The plan for the elaboration phase • The initial version of the business case	• The completed domain model • The completed business model • The completed requirements artifacts • The completed analysis artifacts • An updated version of the architecture • An updated list of risks • The project management plan (for the rest of the project) • The completed business case	• The initial user manual and other manuals, as appropriate • All the artifacts (beta release versions) • The completed architecture • The updated risk list • The project management plan (for the remainder of the project) • If necessary, the updated business case	• All the artifacts (final versions) • The completed manuals

UP Life cycle in four phases

- Inception
- Elaboration
- Construction
- Transition

The Enterprise Unified Process (EUP) adds two more phases to this:

- *Production:* keep system useful/productive after deployment to customer
- *Retirement:* archive, remove, or reuse etc.

Example roles in UP

- *Stake Holder:* customer, product manager, etc.
- *Software Architect:* established and maintains architectural vision
- *Process Engineer:* leads definition and refinement of Development Case
- *Graphic Artist:* assists in user interface design, etc.

Some UP guidelines

- *Attack risks early on* and continuously so, before they will attack you
- Stay focused on *developing executable software* in early iterations
- Prefer *component-oriented* architectures and *reuse existing components*
- Quality is a way of life, not an afterthought

Six best “must” UP practices

1. Time-boxed iterations: *avoid speculative powerpoint architectures”*

2. *Strive for cohesive architecture* and reuse existing components:

- e.g. core architecture developed by small, co-located team
- then early team members divide into sub-project leaders

Six best “must” UP practices

3. Continuously verify quality: *test early & often*, and realistically by integrating all software at each iteration
4. *Visual modeling*: prior to programming, do at least some visual modeling to explore creative design ideas

Six best “must” UP practices

5. *Manage requirements:* find, organize, and track requirements through skillful means

6. *Manage change:*
- disciplined configuration management protocol, version control,
- change request protocol
- baselined releases at iteration ends

Unified Process Workflows

The Unified Process

- The Unified Process IS A
- 2-dimensional systems development process described by a
  - set of phases and (dimension one)
  - Workflows (dimension two)

The Unified Process

- Phases
  - Describe the business steps needed to develop, buy, and pay for software development.
  - The business increments are identified as phases

- Workflows
  - Describe the tasks or activities that a developer performs to evolve an information system over time

Process Overview

	Phases (time)			
Workflow (tasks)	Inception	Elaboration	Construction	Transition
Requirements				
Analysis				
Design				
Implementation				
Test				

workflows

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system (see Chapter 29).
Project management	This supporting workflow manages the system development (see Chapter 5).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Primary Workflows

- The Unified Process
- PRIMARY WORKFLOWS
  - **Requirements workflow**
  - **Analysis workflow**
  - **Design workflow**
  - **Implementation workflow**
  - **Test workflow**
  - **Post delivery maintenance workflow**
- Supplemental Workflows
  - Planning Workflow

Planning Workflow

- Define scope of Project
- Define scope of next iteration
- Identify Stakeholders
- Capture Stakeholders expectation
- Build team
- Assess Risks
- Plan work for the iteration
- Plan work for Project
- Develop Criteria for iteration/project closure/success
- UML concepts used: initial Business Model, using class diagram

Requirements Workflow

- Primary focus
  - To determine the client’s needs by eliciting both functional and nonfunctional requirements
- Gain an understanding of the *application domain*
- Described in the language of the customer

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Requirements Workflow

- The aim is to determine the client's needs
- First, gain an understanding of the *domain*
  - How does the specific business environment work
- Second, build a business model
  - Use UML to describe the client's business processes
  - If at any time the client does not feel that the cost is justified, development terminates immediately
- It is vital to determine the client's constraints
  - Deadline -- Nowadays software products are often mission critical
  - Parallel running
  - Portability
  - Reliability
  - Rapid response time
  - Cost
- The aim of this *concept exploration* is to determine
  - What the client needs, and
  - *Not* what the client wants

Requirements Workflow

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

- List candidate requirements
  - textual feature list
- Understand system context
  - domain model describing important concepts of the context
  - business modeling specifying what processes have to be supported by the system using Activity Diagram
- Capture functional and nonfunctional requirements
  - Use Case Model
- Supplementary requirements
  - physical, interface, design constraints, implementation constraints

Analysis Workflow

- Primary focus
  - Analyzing and refining the requirements to achieve a detailed understanding of the requirements essential for developing a software product correctly
- To ensure that both the developer and user organizations understand the underlying problem and its domain
- Written in a more precise language

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Analysis Workflow

- The aim of the analysis workflow
  - To analyze and refine the requirements
- Two separate workflows are needed
  - The requirements artifacts must be expressed in the language of the client
  - The analysis artifacts must be precise, and complete enough for the designers
- Specification document ("specifications")
  - Constitutes a contract
  - It must not have imprecise phrases like "optimal," or "98 percent complete"
- Having complete and correct specifications is essential for
  - Testing, and
  - Maintenance
- The specification document must not have
  - Contradictions
  - Omissions
  - Incompleteness

Analysis Workflow

- Structure the Use Cases
- Start reasoning about the internal of the system
- Develop Analysis Model: Class Diagram and State Diagram
- Focus on what is the problem not how to solve it
- Understand the main concepts of the problem
- Three main types of classes stereotypes may be used:
  - Boundary Classes: used to model interaction between system and actors
  - Entity Classes: used to model information and associated behavior deirectly derived from real-world concept
  - Control Class: used to model business logic, computations transactions or coordination.
- The specification document must not have
  - Contradictions
  - Omissions
  - Incompleteness

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Design Workflow

- The aim of the design workflow is to refine the analysis workflow until the material is in a form that can be implemented by the programmers
  - Determines the internal structure of the software product

Design Workflow

The goal is to refine the analysis workflow until the material is in a form that can be implemented by the programmers

- Many nonfunctional requirements need to be finalized at this time, including: Choice of programming language, Reuse issues, Portability issues.

Classical Design

- Architectural design
  - Decompose the product into modules
- Detailed design
  - Design each module using data structures and algorithms

Object Oriented Design

- Classes are extracted during the object-oriented analysis workflow, and
  - Designed during the design workflow

Design Workflow

General Design

- Refine the Class Diagram
- Structure system with Subsystems, Interfaces, Classes
- Define subsystems dependencies
- Capture major interfaces between subsystems
- Assign responsibilities to new design classes
- Describe realization of Use Cases
- Assign visibility to class attributes
- Design Databases and needed Data Structures
- Define Methods signature
- Develop state diagram for relevant design classes
- Use Interaction Diagram to distribute behavior among classes
- Use Design Patterns for parts of the system

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Design Workflow

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

- Architectureal Design
- Identify Design Mechanisms
  - Refine Analysis based on implementation environment
  - Characterize needs for specific mechanisms (inter-process communication, real-time computation, access to legacy system, persistence, …)
  - Assess existing implementation mechanisms
- Identify Design Classes and Subsystems
  - A Subsystem is a special kind of Package which has behavioral semantics (realizes one or more interfaces)
  - Refine analysis classes
  - Group classes into Packages
  - Identify Subsystems when analysis classes are complex
    - Look for strong interactions between classes
    - Try to organize the UI classes into a subsystem
    - Separate functionality used by different actors in different subsystems
    - Separate subsystems based on the distribution needs
  - Identify Interfaces of the subsystems

Implementation Workflow

- The aim of the implementation workflow is to implement the target software product in the selected implementation language

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Implementation Workflow

- Distribute the system by mapping executable components onto nodes in the deployment model
- Implement Design Classes and subsystems through packaging mechanism:
  - package in Java, Project in VB, files directory in C++
- Acquire external components realizing needed interfaces
- Unit test the components
- Integrate via builds

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Test Workflow

- Carried out in parallel with other workflows
- Primary purpose
  - To increase the quality of the evolving system
- The test workflow is the responsibility of
  - Every developer and maintainer
  - Quality assurance group

Workflow (tasks)
Requirements
Analysis
Design
Implementation
Testing

Test Workflow

- Develop set of test cases that specify what t
in the system
  - many for each Use Case
  - each test case will verify one scenario of the use case
  - based on Sequence Diagram
- Develop test procedures specifying how to perform test cases
- Develop test component that automates test procedures

Deployment Workflow

- Activities include
  - Software packaging
  - Distribution
  - Installation
  - Beta testing

Deployment Workflow

- Producing the Software
  - Output of implementation is tested executables.
  - Must be associated with other artifacts to constitute a complete product:
    - Installation scripts
    - User documentation
    - Configuration data
    - Additional programs for migration: data conversion.
  - In some cases:
    - different executables needed for different user configurations
    - different sets of artifacts needed for different classes of users:
      - new users versus existing users,
      - variants by country or language

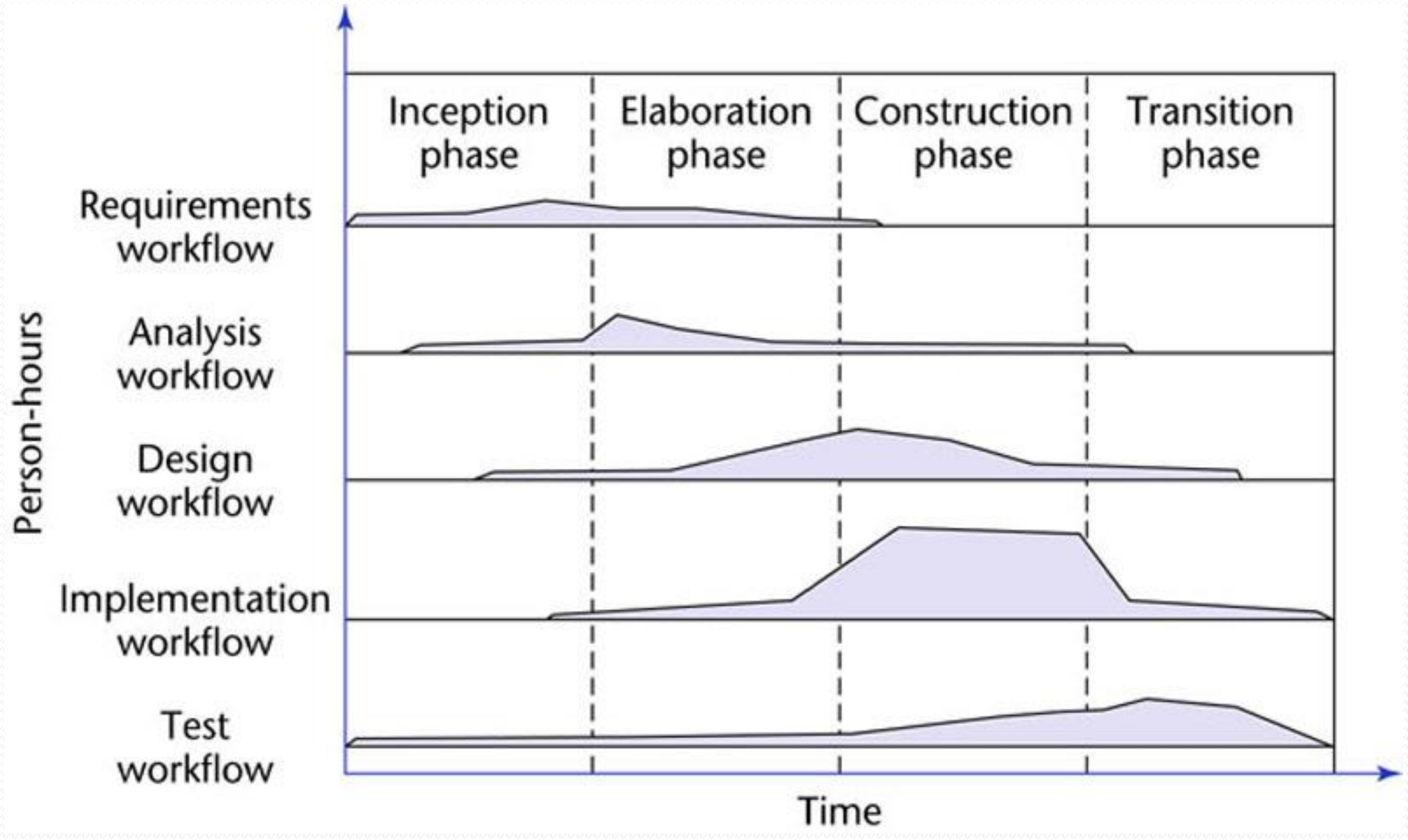
Deployment Workflow

- Producing the Software (continued)
  - For distributed software, different sets may have to be produced for different computing nodes in the network Packaging the Software
  - Distributing the Software
  - Installing the Software
  - Migration
  - Providing Help and Assistance to Users
  - Acceptance

Iterations and Workflow

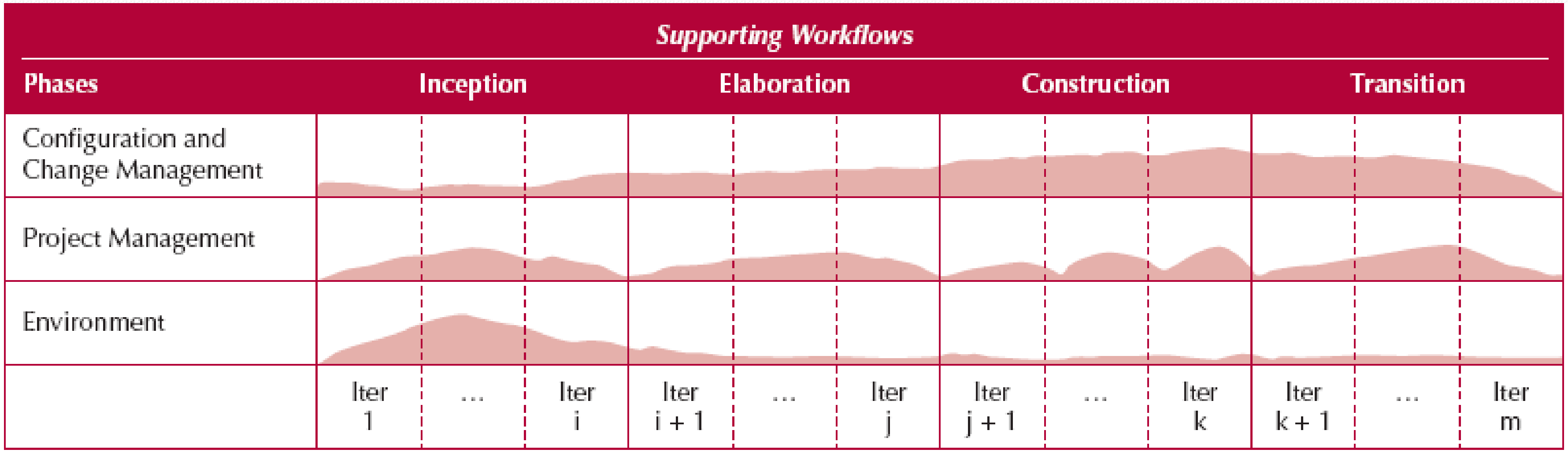
Phases

Core Workflows



Iterations

Supporting Workflows of The Unified Process



Software Project Management Plan

- Once the client has signed off the specifications, detailed planning and estimating begins

- We draw up the software project management plan, including
  - Cost estimate
  - Duration estimate
  - Deliverables
  - Milestones
  - Budget

- This is the earliest possible time for the SPMP

Post delivery Maintenance

- Post delivery maintenance is an essential component of software development
  - More money is spent on post delivery maintenance than on all other activities combined
- Problems can be caused by
  - Lack of documentation of all kinds
- Two types of testing are needed
  - Testing the changes made during post delivery maintenance
  - Regression testing
- All previous test cases (and their expected outcomes) need to be retained

Retirement

- Software is can be made unmaintainable because
  - A drastic change in design has occurred
  - The product must be implemented on a totally new hardware/operating system
  - Documentation is missing or inaccurate
  - Hardware is to be changed—it may be cheaper to rewrite the software from scratch than to modify it
- These are instances of maintenance (rewriting of existing software)
- True retirement is a rare event
- It occurs when the client organization no longer needs the functionality provided by the product