Modern Methods in Software Engineering

ID2207

Introduction

www.imit.kth.se/courses/2G1522

Course info

www.imit.kth.se/courses/2G1522

Coordinator and lecturer

Mihhail Matskin
misha@imit.kth.se
tel. 08-790 41 28

Assignments and project responsible

Abdul Haseeb
ahaseeb@kth.se

Leif Lagerback
leifl@kth.se

Lectures:

Mondays (the first week we have a lecture on Friday also)

Thursdays

Exercises:

Fridays (next week on Monday)

Written examination (3 p.)

October 26 at 15-19

Registration at least 14 days before exam period

Homework and project assignments (2 p.)

Homeworks

Start Date	Due Date	Description
2007-09-10	2007-09-17	Homework 1
2007-09-14	2007-09-20	Homework 2
2007-09-21	2007-09-27	Homework 3
2007-09-28	2007-10-04	Homework 4

It is assumed that the Homeworks are done by groups of 2 student - there will not be bonus for doing them alone

Project

Aims of the project

- To get practice in Extreme Programming approach

Your task

Apply some elements of Extreme programming approach to solving the problem of implementing Academic payroll system

Provide an analysis and comparison of your experience to develop a code with analysis-design-implementation cycle and with XP approach

The project must be done by groups of 2 students

Bonus

1. Delivering all Homeworks 1,2,3,4 in due time gives 5 bonus points to written exam (this assumes that all Homeworks are approved)
2. There will be some bonus questions in Homeworks which give bonus points to exam
3. In case of Late Submission of any Homework, No bonus points will be awarded for the "in-time submission of homeworks". But the bonus points for the Bonus questions will still be granted.
4. Delivering project work before October 26 gives 5 bonus points to written exam (this assumes that the project work is approved)

ALL Bonus points are only valid for the first exam on October 26

Course literature

- **Object-Oriented Software Engineering: Using UML, Patterns and Java: International Edition, 2/E Bernd Bruegge, Allen H. Dutoit, ISBN: 0131911791, Publisher: Prentice Hall, Copyright: 2004, Format: Paper; 800 pp Published: 16 Oct 2003**
  (available in the Kista Electrum book store)
- **Lecture notes**
- **Additional articles in the curriculum will be added during the course**

Very Tentative Lecture Plan

	Date	Lecture
1	03.09.2007	Introduction and Software Lifecycles
2	06.09.2007	UML Basics
3	07.09.2007	Requirements Elicitation
4	13.09.2007	Requirements Analysis
5	17.09.2007	System Design
6	20.09.2007	Object Design
7	24.09.2007	Object Design
8	27.09.2007	Move to code, Testing
9	01.10.2007	Extreme Programming
10	04.10.2007 05.10.2007	Guest lectures from Microsoft "Overview of the Microsoft Software development process“ & “Software Factories

Objectives of the Course

- Learn about Software Engineering methods:
  - how to build complex software systems when the context frequently changes
- Learn methods for dealing with complexity and changes in software construction
- Be able to evaluate and choose methods for producing a high quality software system within time
- Get technical knowledge and some managerial knowledge for software construction

Introduction Content

- Introduction
- Software nature
- Definitions of Software Engineering
- Dealing with complexity
  - Abstraction
  - Decomposition
  - Hierarchy
- Software development concepts and activities
- Software development lifecycle

Literature used

- Text book “Object-Oriented Software Engineering: Using UML, Patterns and Java” International Edition, 2/E Bernd Bruegge, Allen H. Dutoit

  Chapters 1, 15, 16

What is Software?

- Software has dual role:
  - it is a product
  - it is a vehicle for developing products
- Software is
  - computer programs
  - associated documentation
  - associated data that is needed to make the programs operatable

Software's Nature

- Software is
  - intangible – it is difficult to understand development efforts
  - easily reproducible – cost is not in manufacturing but in development
  - labor-intensive – hard to automate
  - easy to modify
  - a logical rather than physical product – it doesn't wear out
  - untrained people can hack something together

Adopted from Lethbridge and Laganière.

Software's nature

- Demand for software is high and rising
- In many cases software has poor design and it getting worse
- "software crisis" in a permanent state
- We have to learn to engineer software

Example: Space Shuttle Software

- Cost: $10 Billion, millions of dollars more than planned
- Time: 3 years late
- Quality: First launch of Columbia was cancelled because of a synchronization problem with the Shuttle's 5 onboard computers.
  - Error was traced back to a change made 2 years earlier when a programmer changed a delay factor in an interrupt handler from 50 to 80 milliseconds.
  - The likelihood of the error was small enough, that the error caused no harm during thousands of hours of testing.
- Substantial errors still exist.
  - Astronauts are supplied with a book of known software problems "Program Notes and Waivers".

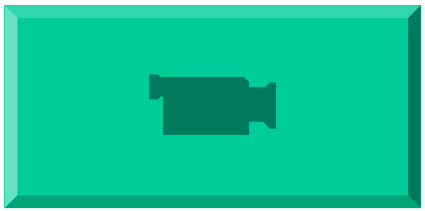
More software failures

- **Item:** In the summer of 1991, telephone outages occurred in local telephone systems in California and along the Eastern seaboard. These breakdowns were all the fault of an error in signaling software. Right before the outages, DSC Communications (Plano, TX) introduced a bug when it changed three lines of code in the several-million-line signaling program. After this tiny change, nobody thought it necessary to retest the program.
- **Item:** In 1986, two cancer patients at the East Texas Cancer Center in Tyler received fatal radiation overdoses from the Therac-25, a computer-controlled radiation-therapy machine. There were several errors, among them the failure of the programmer to detect a race condition (i.e., miscoordination between concurrent tasks).
- **Item:** A New Jersey inmate escaped from computer-monitored house arrest in the spring of 1992. He simply removed the rivets holding his electronic anklet together and went off to commit a murder. A computer detected the tampering. However, when it called a second computer to report the incident, the first computer received a busy signal and never called back.

From http://www.byte.com/art/9512/sec6/art1.htm

Do you know this product?

- This is an actual public demo of the Windows 98 USB plug in functionality, shortly before its initial release



Software failures – some reasons

- Seldom occurring situations are not taken into account
- Users can actively misusing systems
- Management failures
- Unnecessary complexity
- . . .

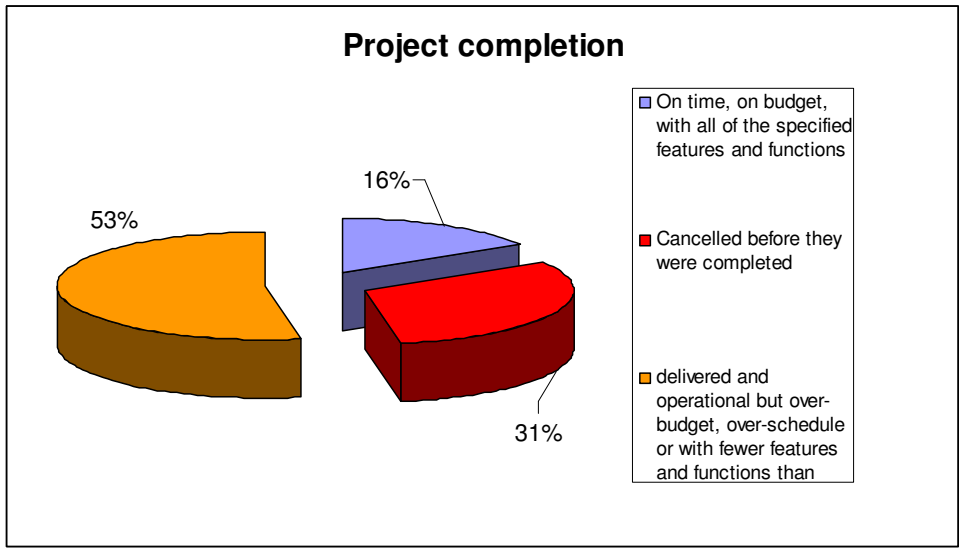
Software project failures

- Terminated for non-performance of contract.
- Completed but the system is not deployed as users cannot or will not use it.
- Completed but the system does not meet the originally promised
  - cost
  - schedule.
  - quality.
  - capability.
- Completed but the system could not be evolved in a cost-effective manner

Adopted from Charette 1989

Software projects

- The Standish Group delivered "Chaos Report"
  - 365 IT executives in US companies in diverse industry segments.
  - 8,380 projects
- In 1994 only 16.2% of software projects were completed on-time and on-budget (for large and complex systems it is 9%)
  - average time overrun = 222%.
  - average cost overrun = 189%
  - 61% of originally specified features included
- In 2003 , it is 34% of projects completed on-time and on-budget



What is Software Engineering?

- *Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines* [from Fritz Bauer in Pressman 1997, pA-1]
- **sound engineering principles** implies:
  - a specification followed by an implementation
  - a demonstration that the solution does what it is supposed to do
  - the adoption of sound project management practices
  - the availability of a range of tools and techniques
- **obtain economically** implies:
  - productivity estimation measures
  - cost estimation and measures
- **reliability and efficiency** imply:
  - performance measures
  - standards
  - quality

What is Software Engineering?

The IEEE describes Software Engineering as *"the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software"*

- **quantifiable** implies measurement
- the discipline is not only concerned with **development** but also **maintenance** and **operation**

What is Software Engineering?

- Brygge&Dutoit:
  - Software Engineering is a collection of techniques, methodologies and tools that help with the production of
    - a high quality software system
    - with a given budget
    - before a given deadline
    - while change occurs.

Software Engineering

- The amateur software engineer is always in search of magic, some sensational method or tool whose application promises to render software development trivial. It is the mark of the professional software engineer to know that no such panacea exists.
  - Grady Booch, in Object-Oriented Analysis and Design

What is Software Engineering?

- Software engineering is a modeling activity.
- Software engineering is a problem-solving activity.
- Software engineering is a knowledge acquisition activity.
- Software engineering is a rationale-driven activity.

Scientist, Engineer and Software Engineer

- (Computer) scientist
  - concerns theories and methods that underlie computer and software systems
  - has no time constraints
- Engineer
  - works in application specific domain
  - has time constrains
- Software engineer
  - woks in multiple application domain
  - has time constraints
  - changes can occur in requirements and technology

Factors of Software Design

- Complexity
  - The problem domain is difficult because of often we are not experts in it
  - The development process is very difficult to manage
  - Software is extremely flexible – easy to modify
- Changes
  - each change makes next change more expensive
  - the cost of implementing changes grows in time

How to deal with complexity?

- Abstraction

  -- Ignore non-essential details - modeling
- Decomposition

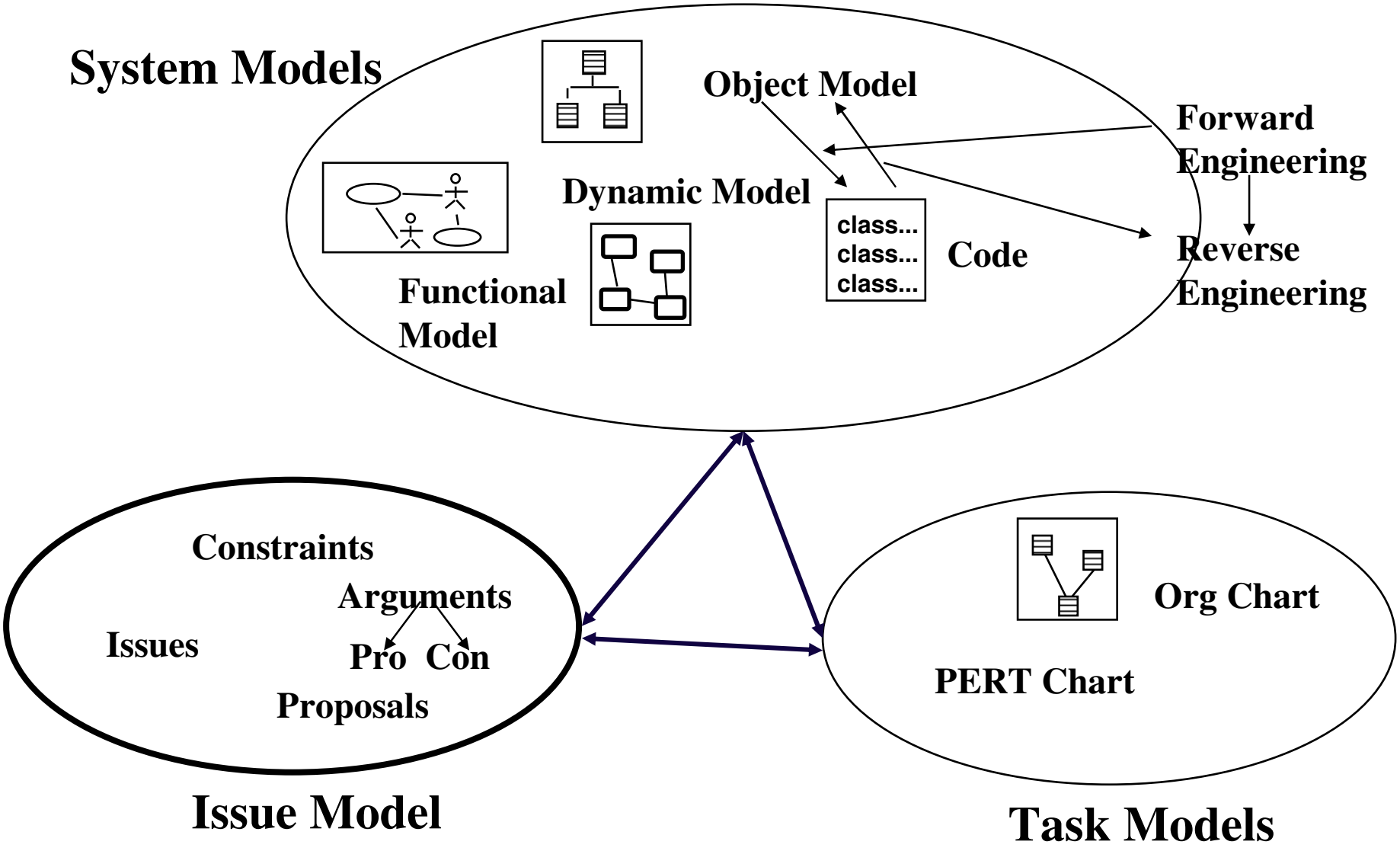
  -- Break problem into sub-problems
- Hierarchy

  -- Simple relationship between chunks

Abstraction/Modeling

- System Model:
  - Object Model: What is the structure of the system? What are the objects and how are they related?
  - Functional model: What are the functions of the system? How is data flowing through the system?
  - Dynamic model: How does the system react to external events? How is the event flow in the system ?
- Task Model:
  - What are the dependencies between the tasks?
  - How can this be done within the time limit?
  - What are the roles in the project or organization?
- Issues Model:
  - What are the open and closed issues? What constraints were posed by the client? What resolutions were made?

Abstraction/Models



An Example of System Models

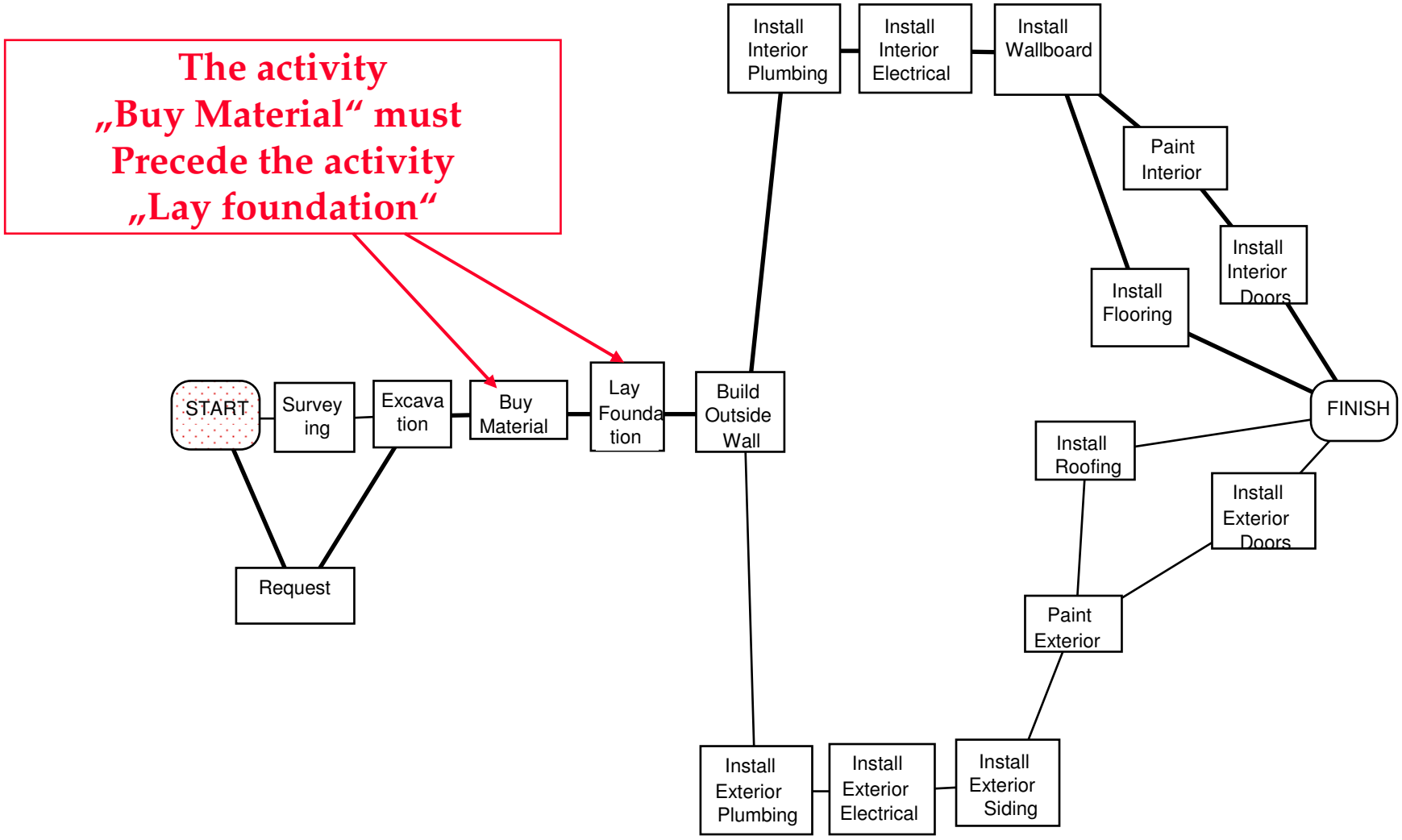
Problem description: A shop lists its products. Each product has a code. A product lists shops.

Object model (UML Class Diagram):

Code

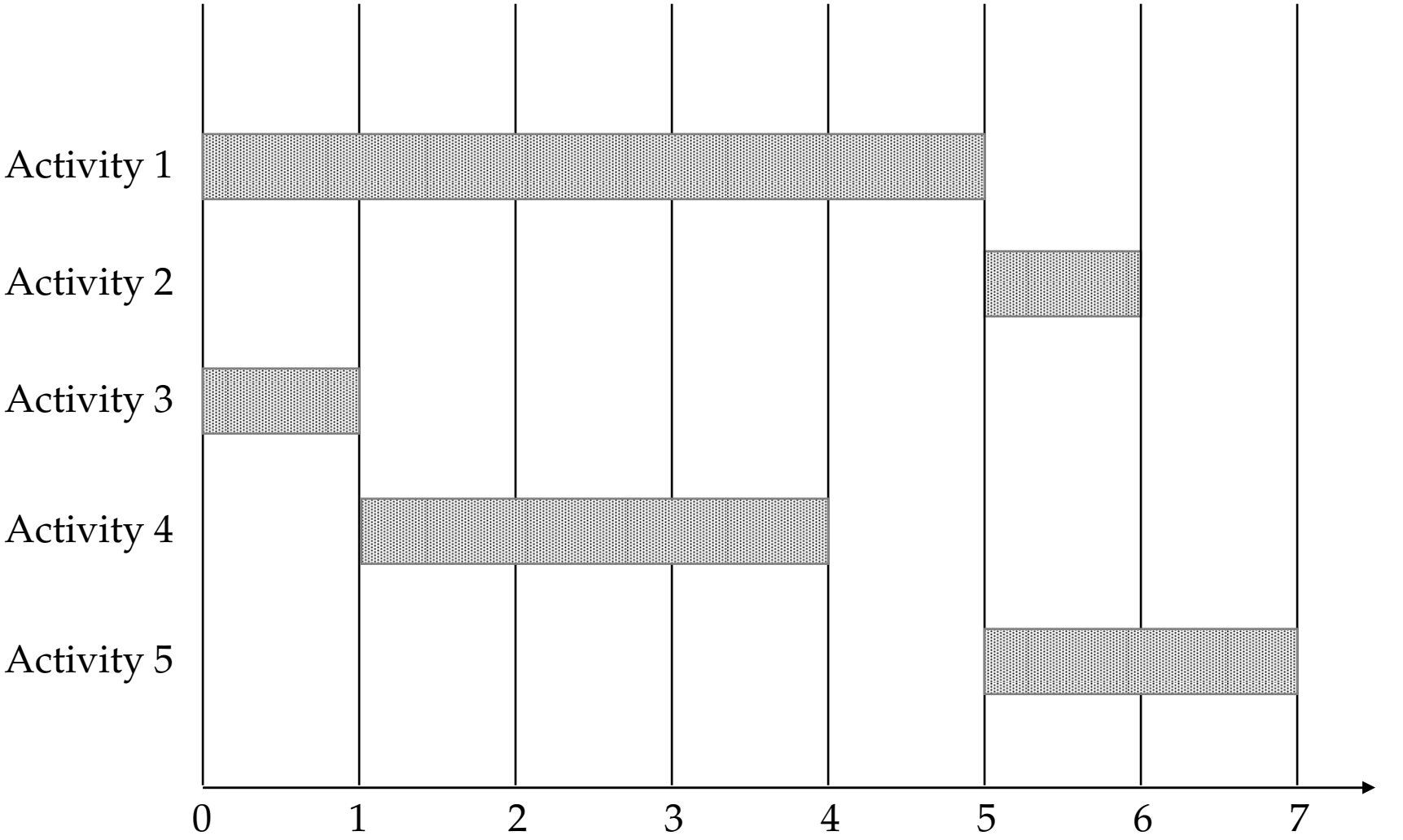
```
public class Shop
{
      public Vector m_Product = new Vector();
};

public class Product
{
      public int m_productCode
      public Vector m_Shop = new Vector();
};
```

Building a House (Dependency Graph)



Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

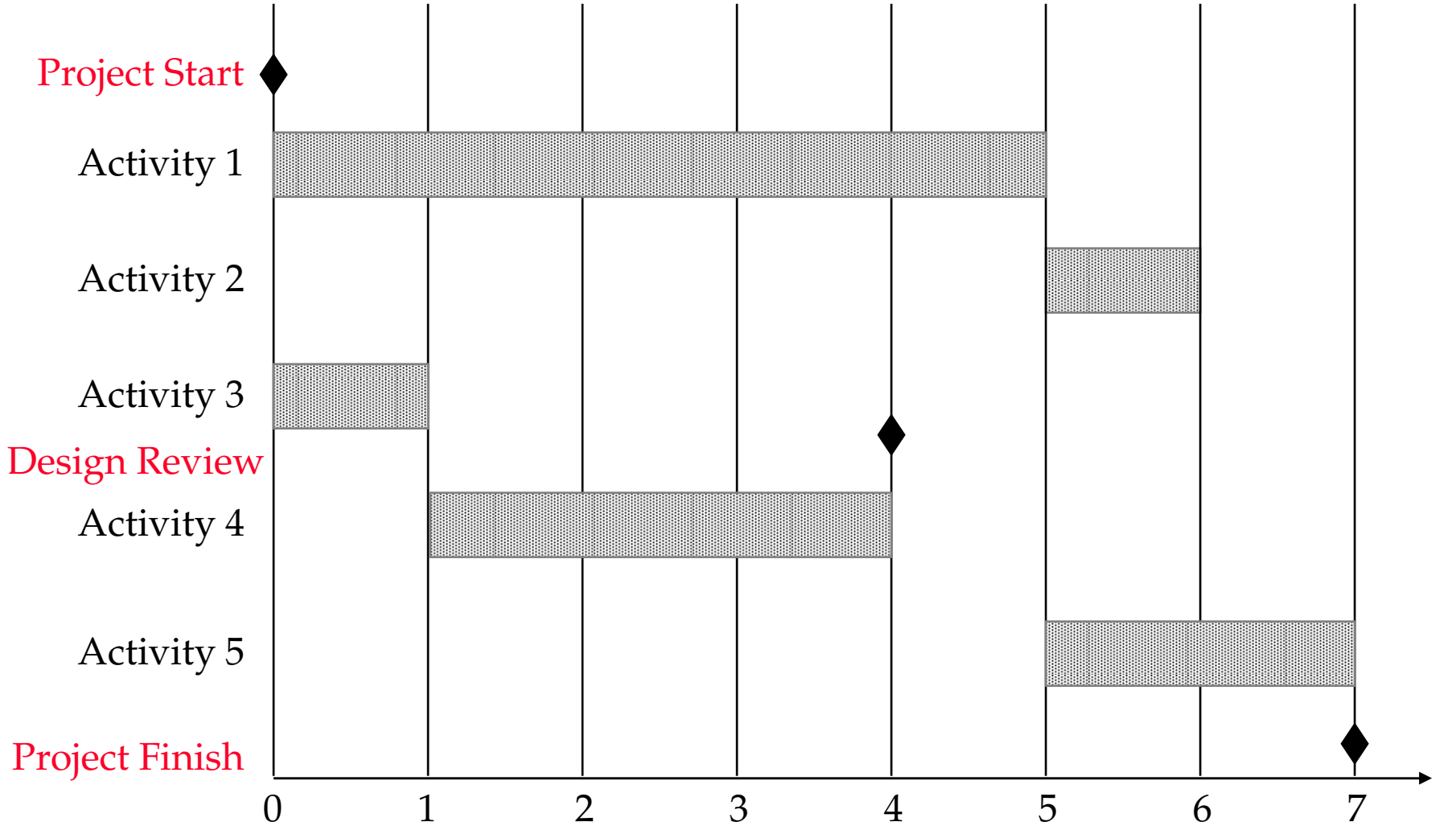
Gantt Chart



Easy to read

Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Gantt Chart with milestones

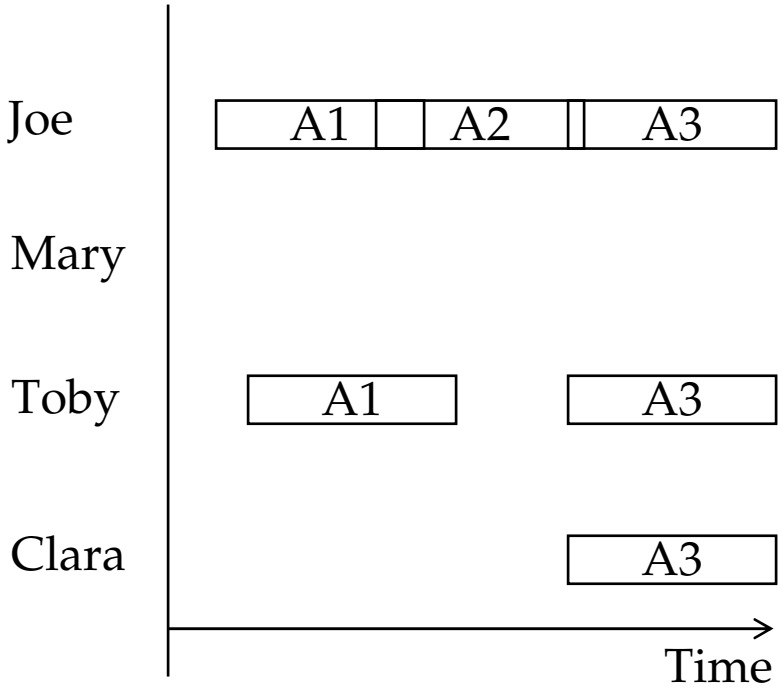


Good for reviews.

Types of Gantt Charts

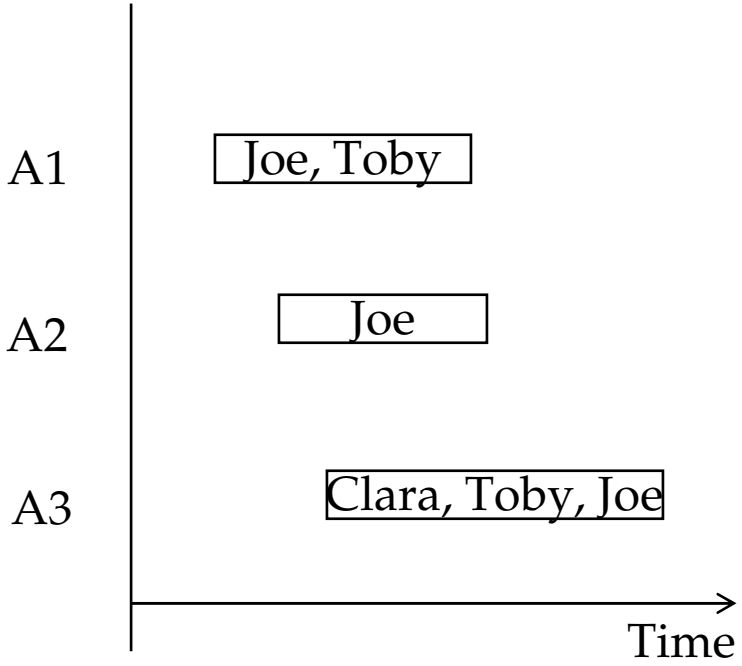
Person-Centered View

To determine people's load

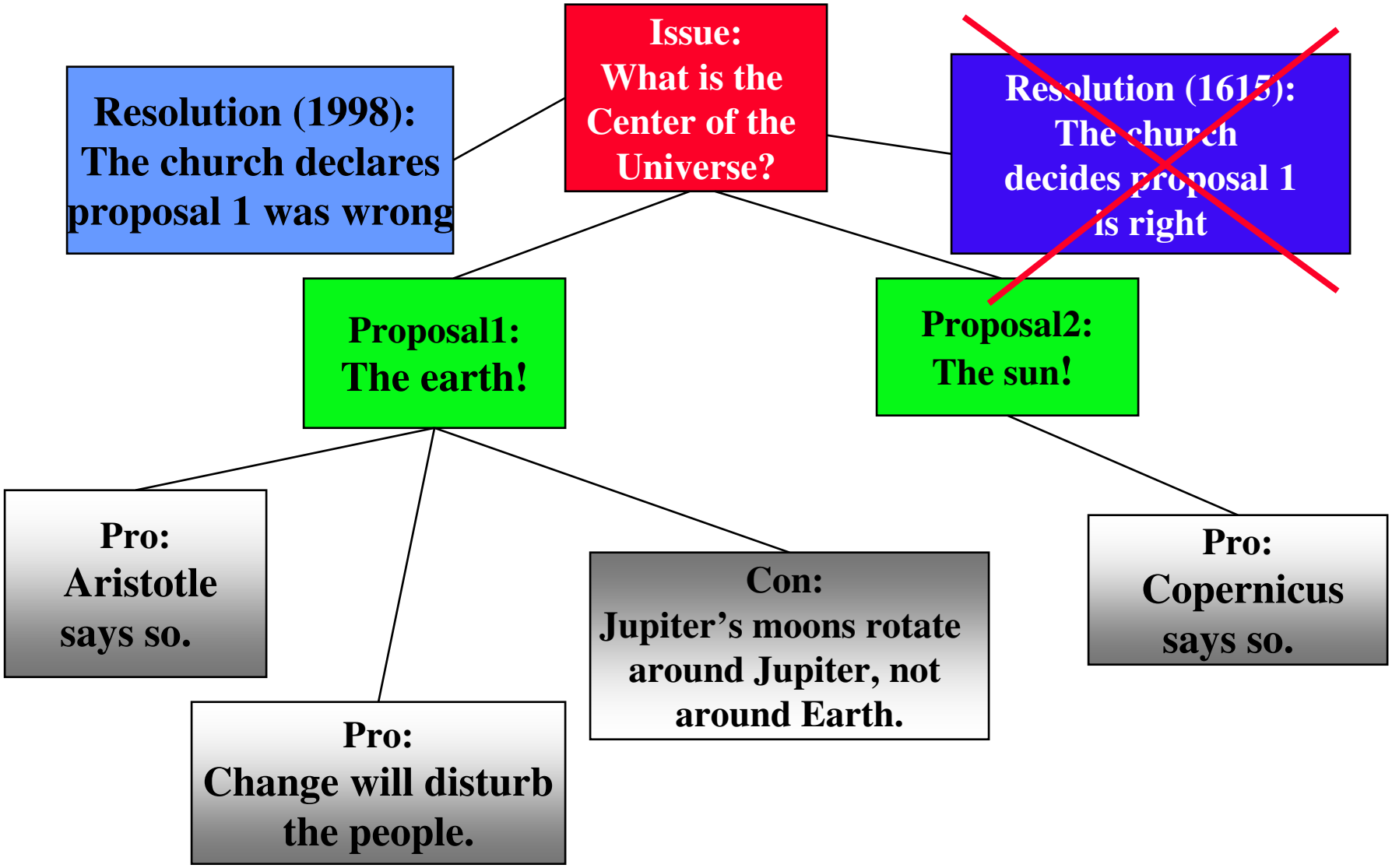


Activity-Centered View

To identify teams working together on the same tasks



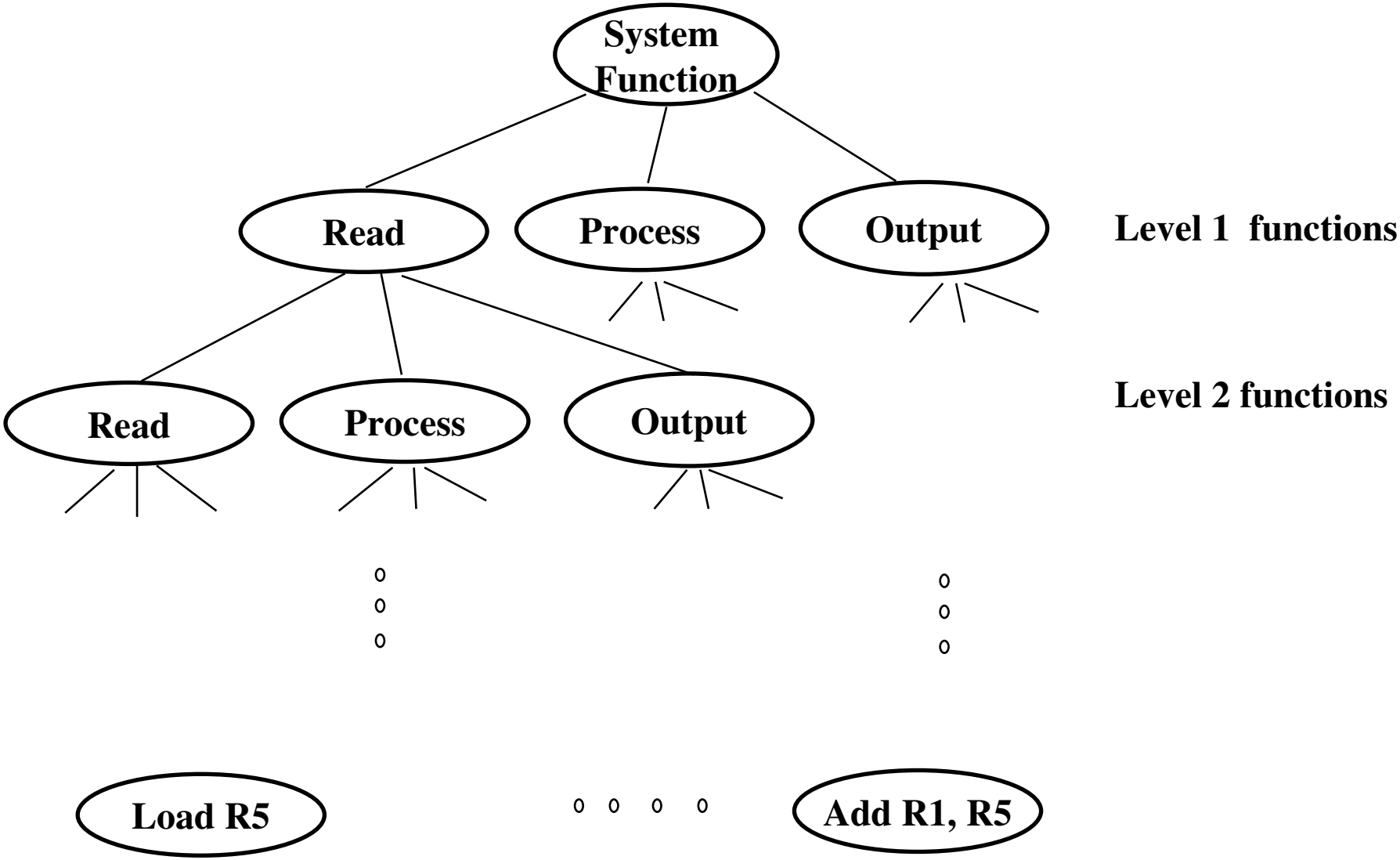
Issue Model


Issue:
What is the Center of the Universe?
Resolution (1998):
The church declares proposal 1 was wrong
Resolution (1615):
The church decides proposal 1 is right
Proposal1:
The earth!
Proposal2:
The sun!
Pro:
Aristotle says so.
Pro:
Change will disturb the people.
Con:
Jupiter's moons rotate around Jupiter, not around Earth.
Pro:
Copernicus says so.

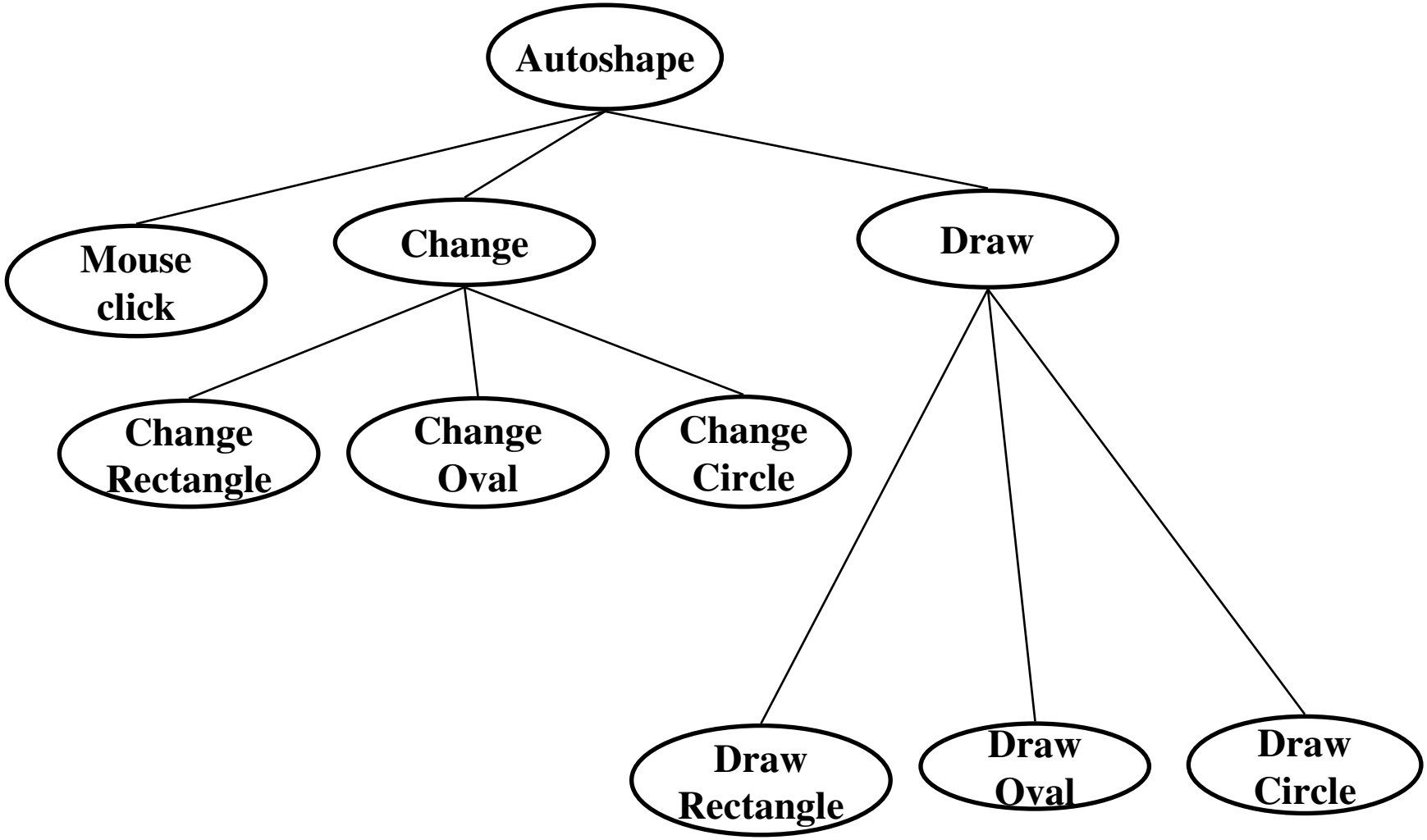
Decomposition

- Functional decomposition - emphasizes the ordering of operations
  - The system is decomposed into functional modules
  - Each module is a processing step (function) in the application domain
  - Modules can be decomposed into smaller modules
- Object-oriented decomposition - emphasizes the agents that cause the operations
  - The system is decomposed into classes (“objects”)
  - Each class is a major abstraction in the application domain
  - Classes can be decomposed into smaller classes

Functional Decomposition

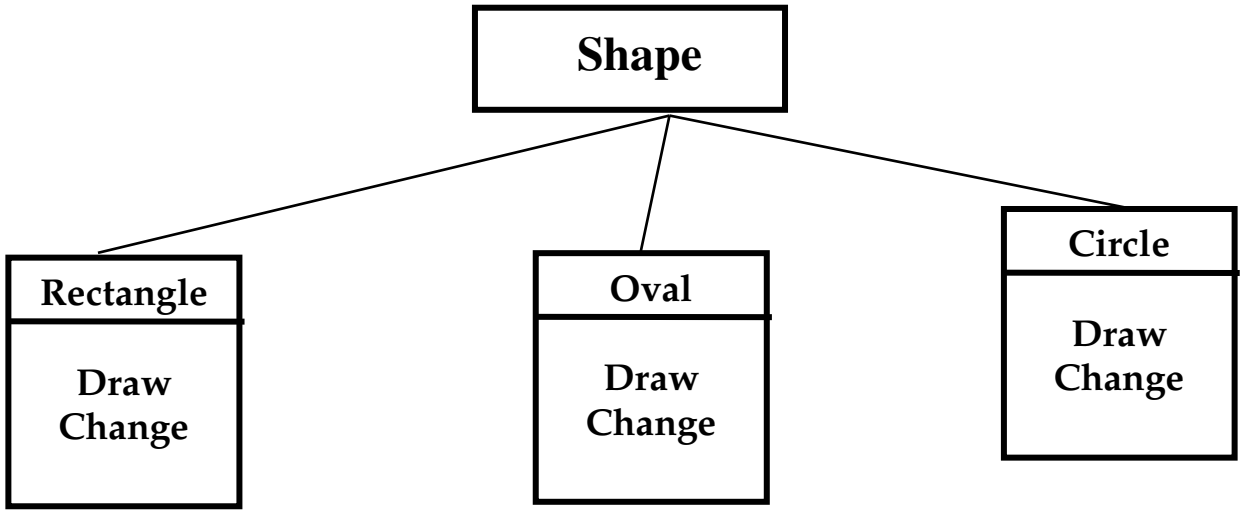


Functional decomposition

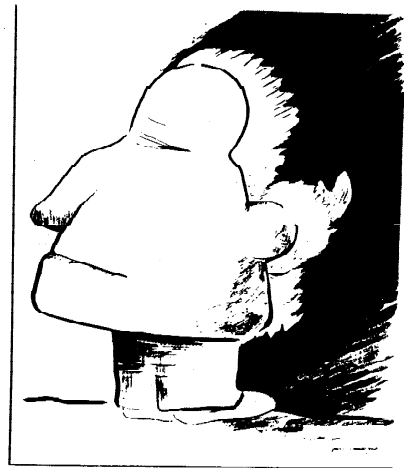


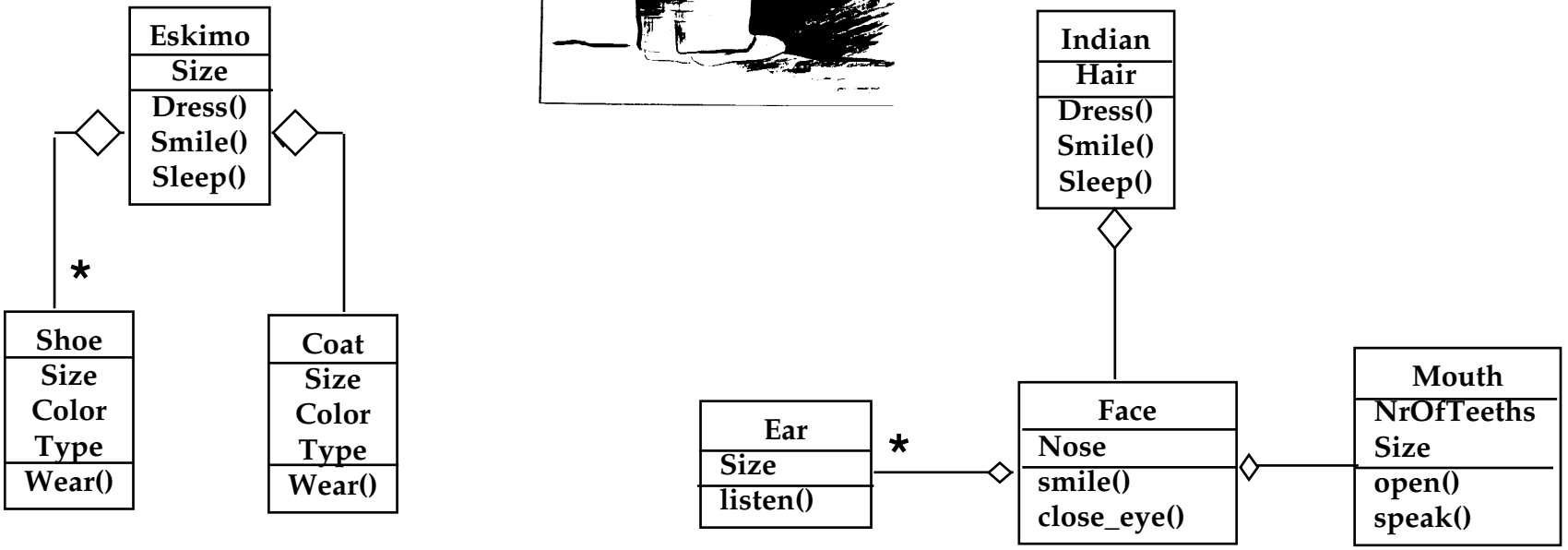
Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Object decomposition

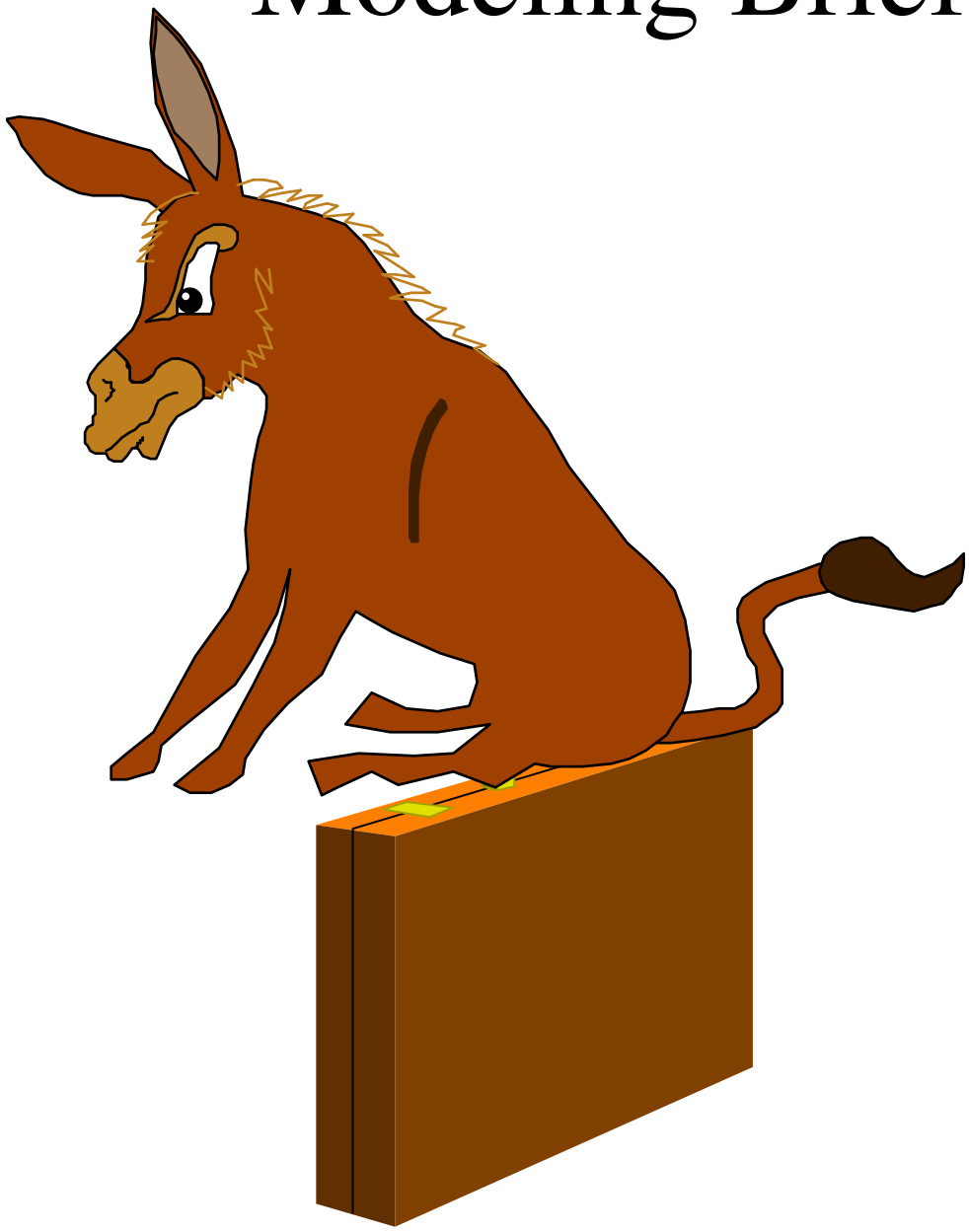


Object-Oriented Decomposition





Modeling Briefcase



BriefCase

Capacity: Integer
Weight: Integer

Open()
Close()
Carry()

SitOnIt()

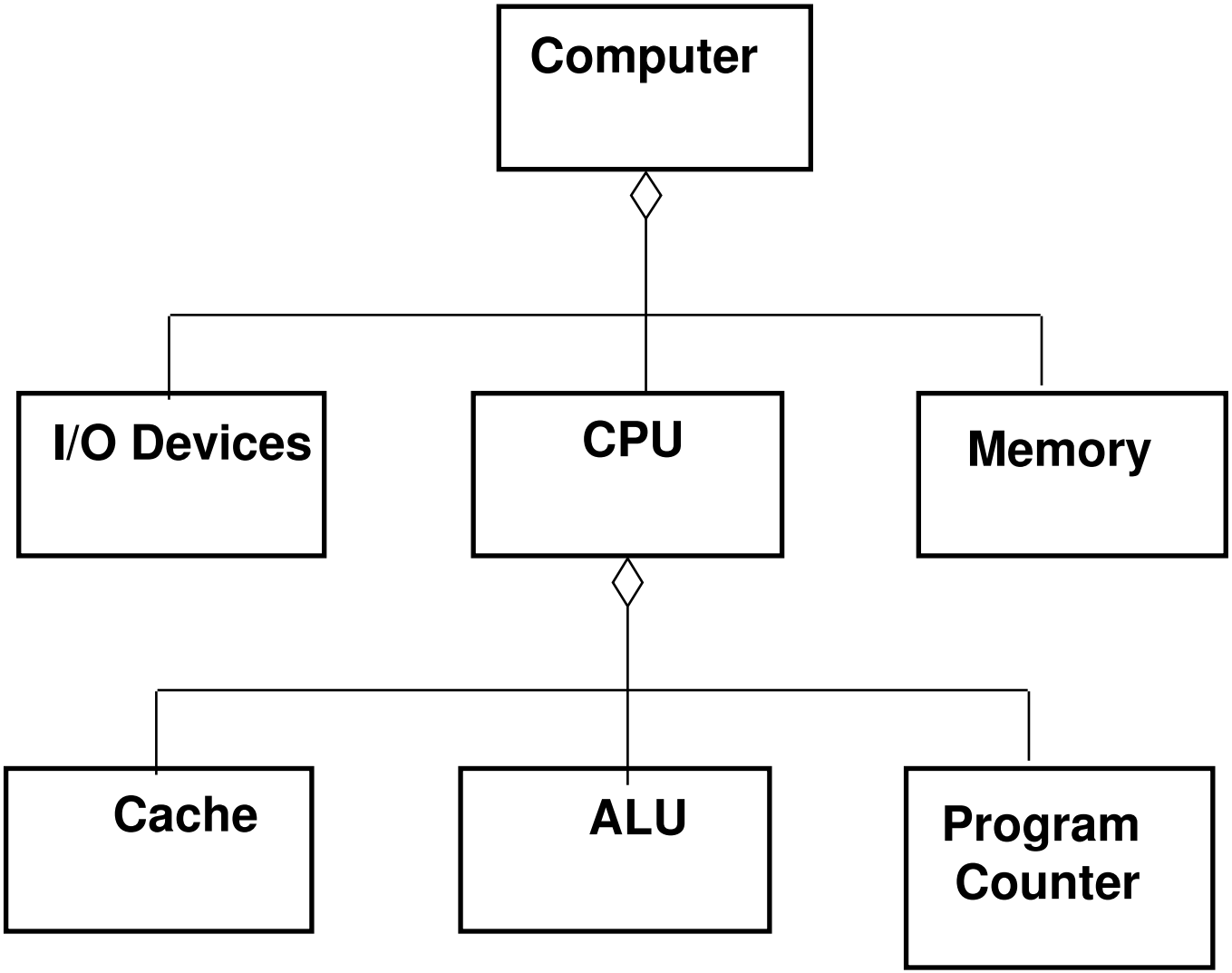
How to identify classes?

- Greenfield Engineering
  - we can find the classes for a new software system
- Reengineering
  - we can identify the classes in an existing system
- Interface Engineering
  - we can create a class-based interface to any system

Hierarchy

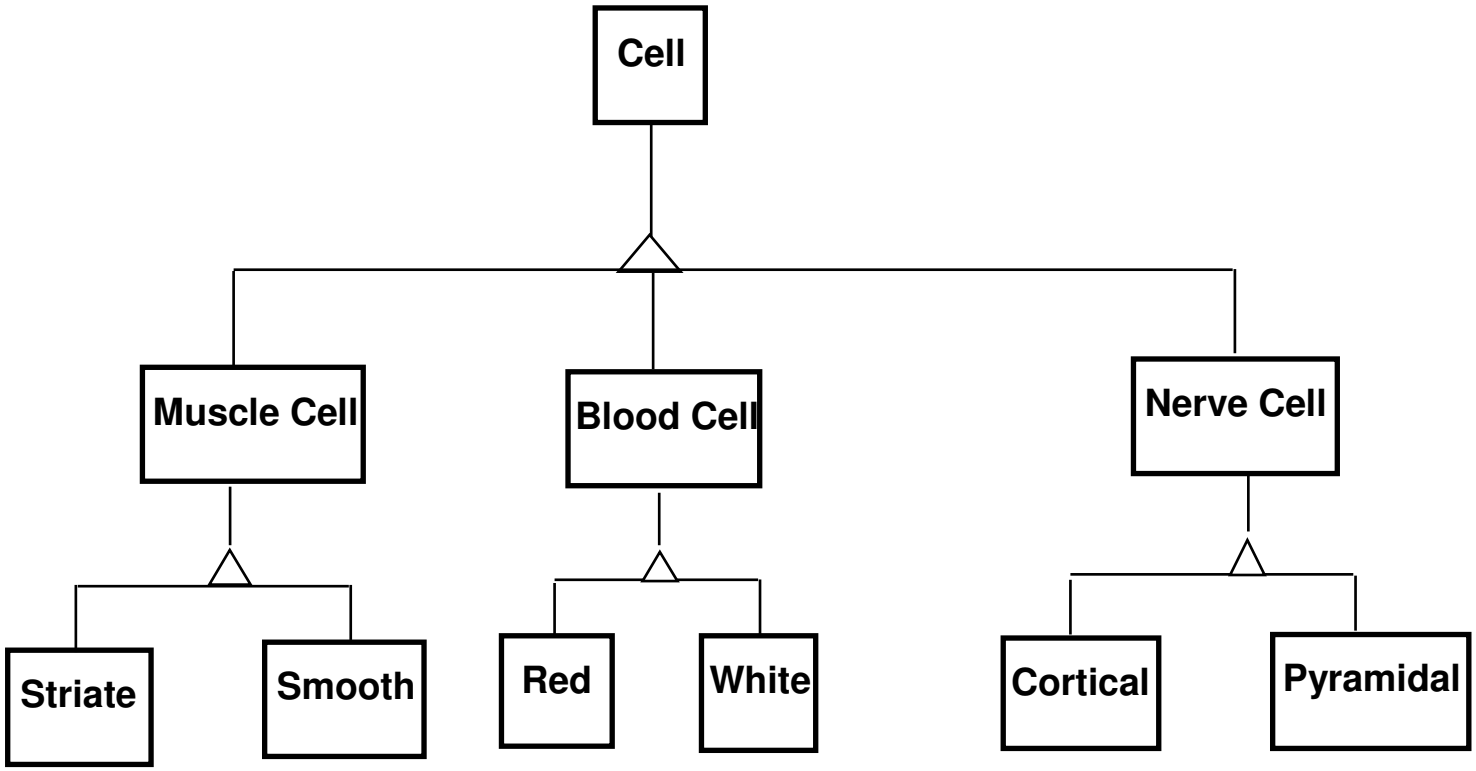
- Relationships between components obtained from abstraction and decomposition
- Hierarchy is one of ways to provide simple relationships
- Part-of hierarchy
- Is-kind-of hierarchy

Part-of hierarchy



Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Is-a-kind-of hierarchy



Where we are now?

- Three ways to deal with complexity:
  - Abstraction
  - Decomposition
  - Hierarchy
- Object-oriented decomposition is a good methodology
  - Unfortunately, depending on the purpose of the system, different objects can be found
- How can we do it right?
  - Many different possibilities
  - Our current approach: Start with a description of the functionality, then proceed to the object model
  - This leads us to the software lifecycle

Software Engineering Concepts (resources)

- Resources include time, equipment and labor
- Participants – all actors involved in the project
- Roles – set of responsibilities in the project
  - associated with a set of tasks and they are assigned to participants
  - participant can fill multiple roles
- Example of roles: client, user, manager, developer, technical writer
- Example of participants: traveler, train company, John, Alice, Zoe
- Example of role assignments: Alice – manager, John – technical writer and analyst
- Other resources: Tariff Database

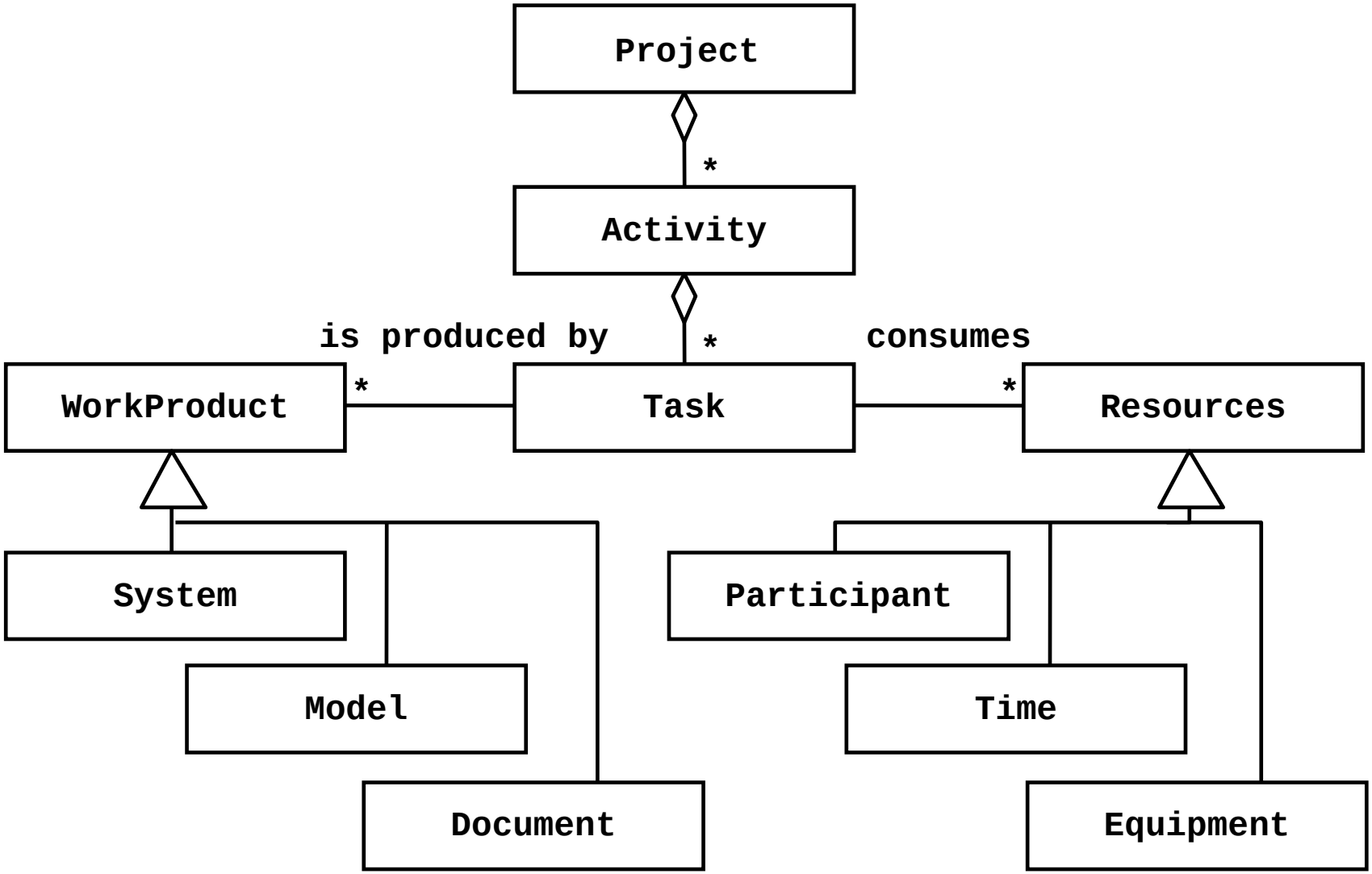
Software Engineering Concepts (Work Products)

- System – collection of interconnected parts
  - TicketDistributor system
- Model - abstraction of a system
  - Schematics of electrical wiring, object model
- Document
  - description
- Work product can be
  - internal - for project consumption
    - test manual
    - workplan
  - deliverable – delivers to the client
    - specification
    - operation manual

Software Engineering Concepts (activities and tasks)

- Task – atomic unit of work to be managed
  - consumes resources
  - produces Work Products
  - depends on other tasks
  - F/E develop “Out of Change” test case
- Activity – set of tasks performed for achieving the project’s goals
  - F/E requirements elicitation
- Work on the project is broken into tasks and assigned to resources

Software Engineering Concepts' Hierarchy



Possible SE activities - Simplified view

Problem Domain

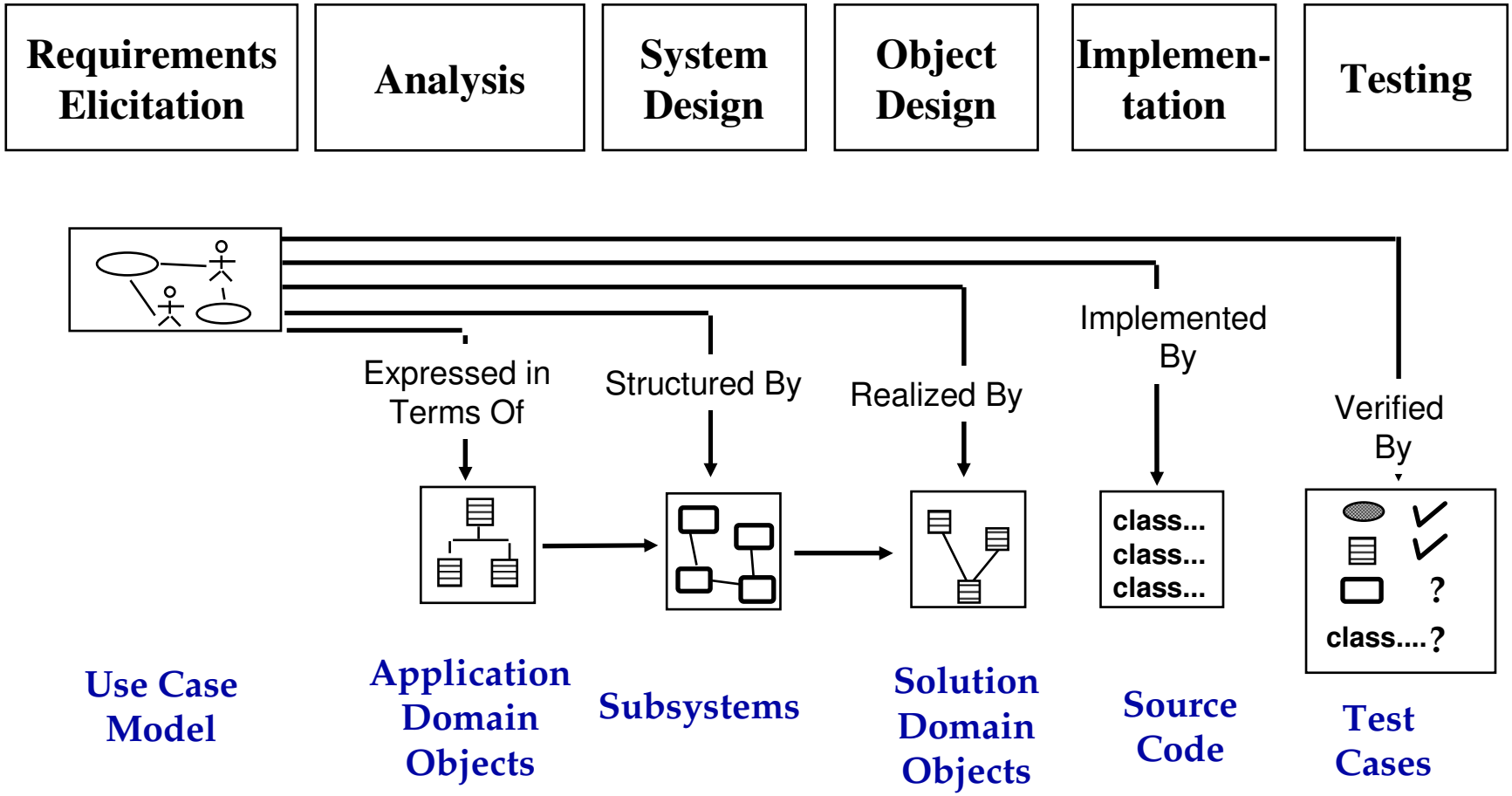
Requirements Analysis	**What is the problem?**
System Design	**What is the solution?**
Object Design	**What is the solution in the context of an existing hardware system?**
Implementation	**How is the solution constructed?**

Implementation Domain

Software Engineering Development Activities

- *Analysis* – concentrates on system requirements – definitions of a system from users' point of view
  - **Requirements elicitation** – determining functionality user needs and a way of its delivering
  - **Requirements analysis** – formalizing determined requirements and ensuring their completeness and consistency
- *Design* – constructing the system
  - **System design** – defining a system architecture in terms of design goals and a subsystem decomposition
  - **Object design** – modeling and construction activities related to the solution domain
- *Implementation* – translation of the solution domain model into code
- *Testing* – the goal is to discover faults in the system to be repaired before the system delivery

Activities and Models



Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Managing Software Development

- Management activities focus on
  - planning of the project
  - monitoring status of the project
  - tracking changes
  - coordinating resources
- Management activities include
  - communication – exchange of models, documents, decisions etc
  - rationale management – justification of decisions
  - software configuration management – change monitoring
  - project management – oversight activities that ensure the system delivery according to requirements
  - software life-cycle – set of activities and their relationships to each other to support the development of a system

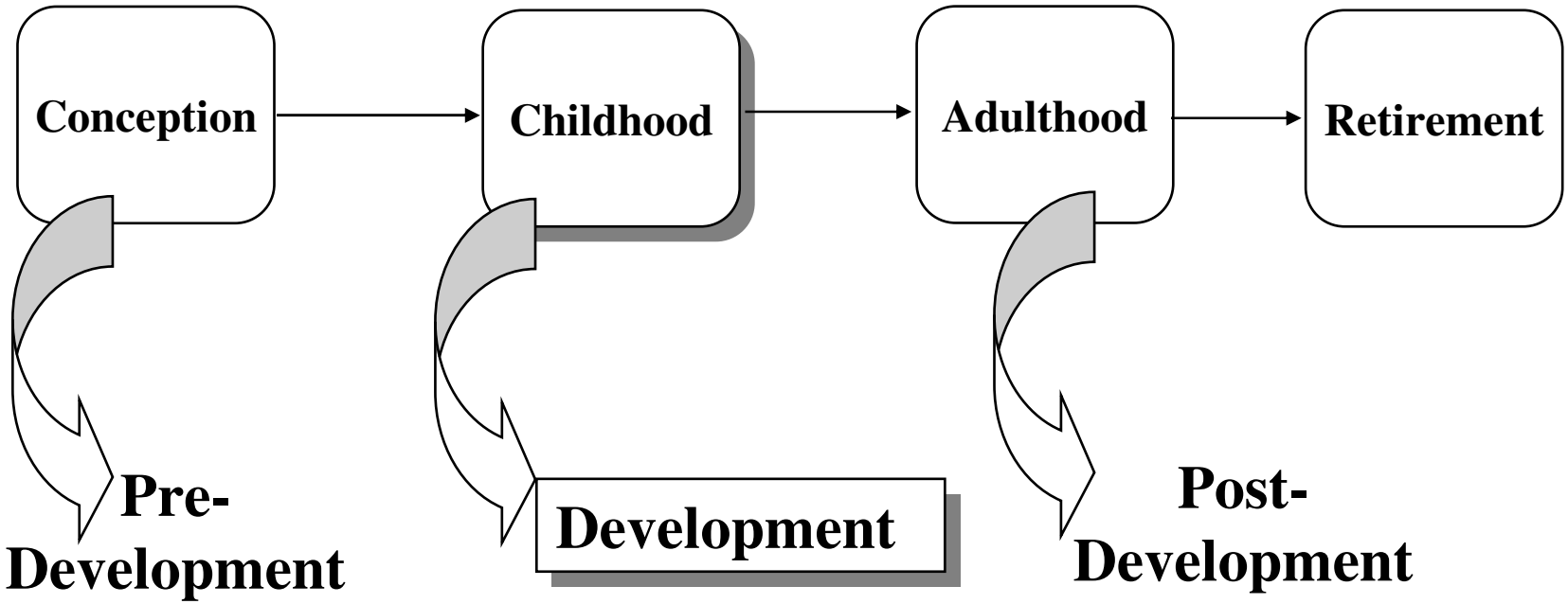
Software Life-Cycle

- Which activities should be selected for the software project?
- What are the dependencies between activities?
  - Does system design depend on analysis? Does analysis depend on design?
- How should the activities be schedule?
  - Should analysis precede design?
  - Can analysis and design be done in parallel?
  - Should they be done iteratively?

Software Life-cycle

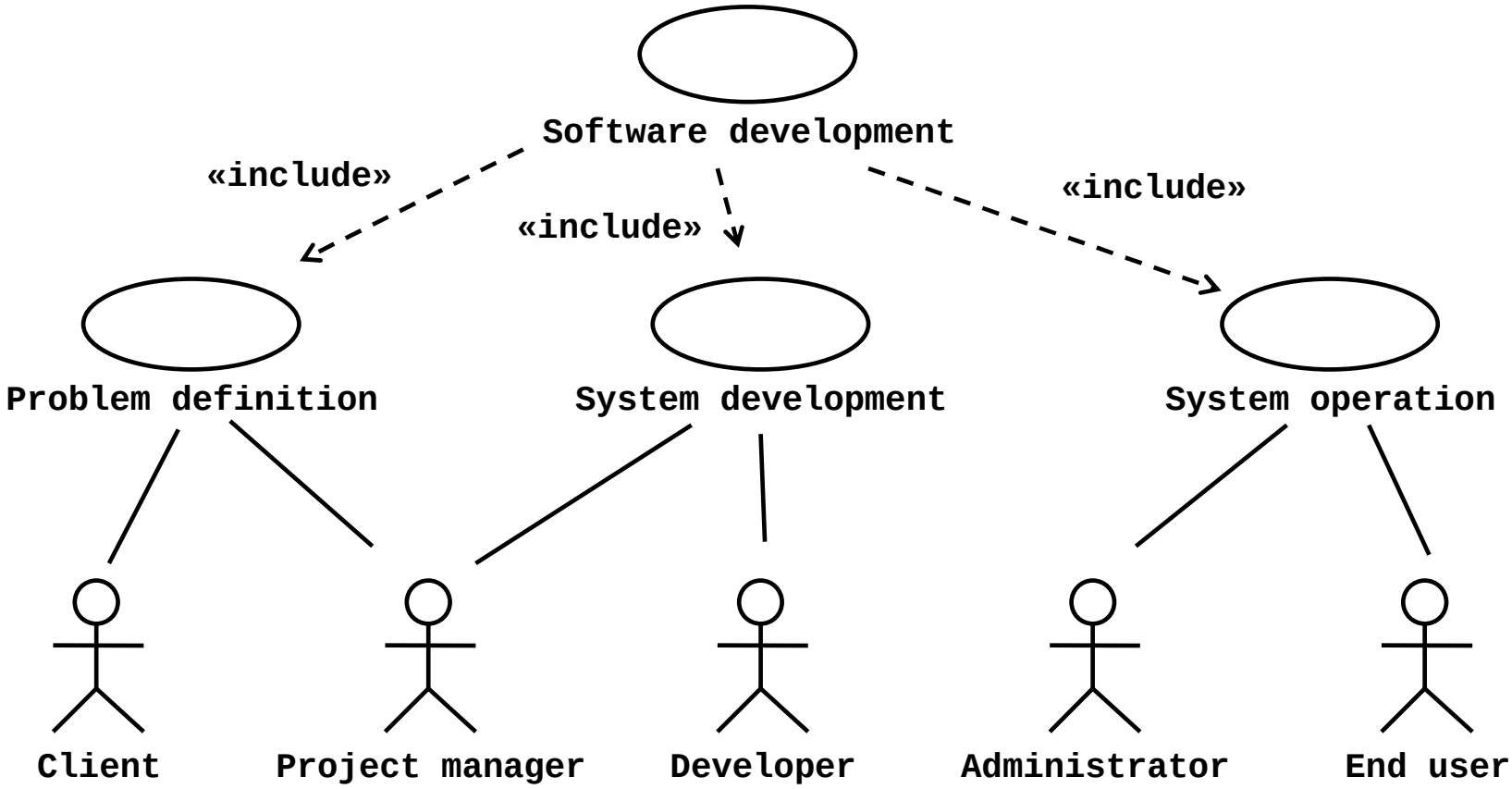
- Life-cycle differs between projects
- Different experience and skills
- Different application domains
- Environment changes
- Project size

Software Life-Cycle



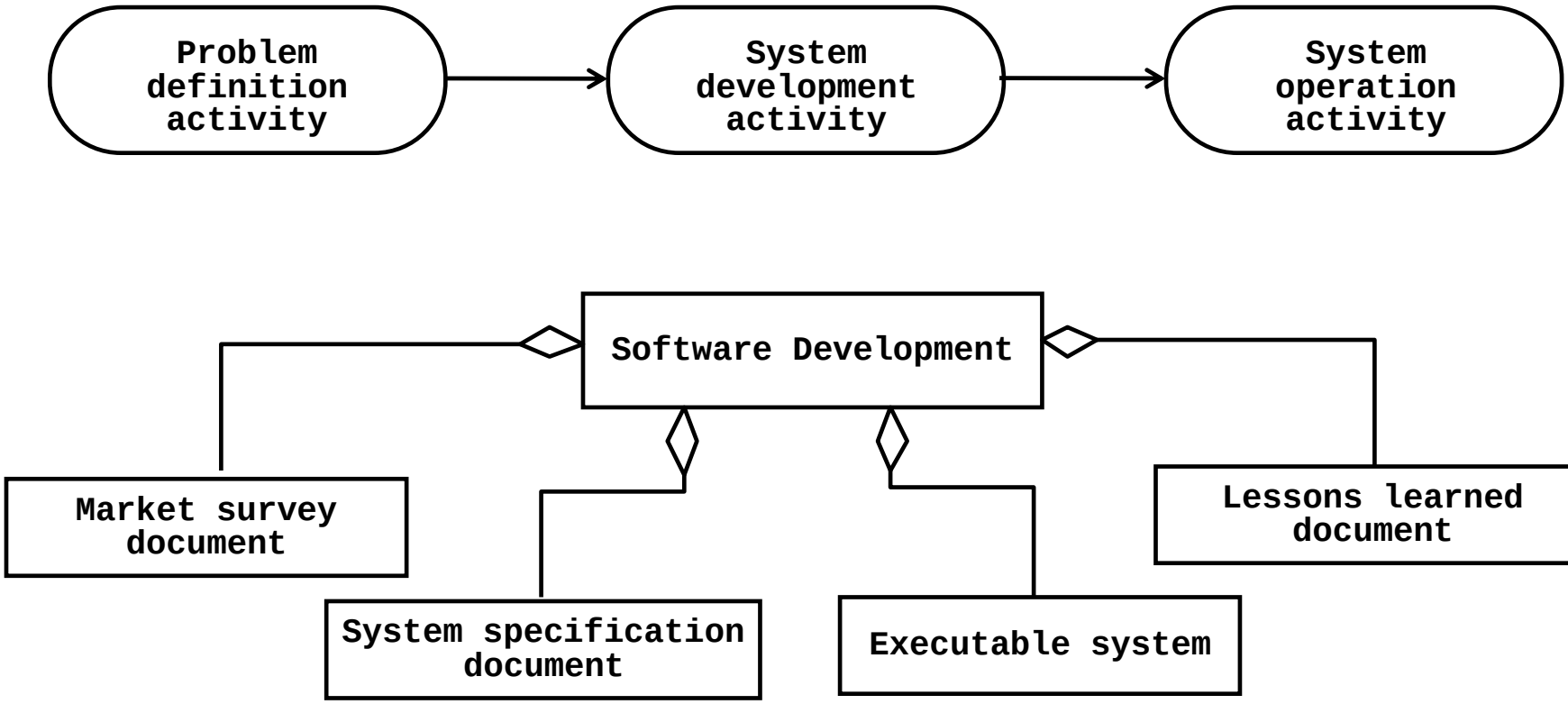
Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

A Simple lifecycle



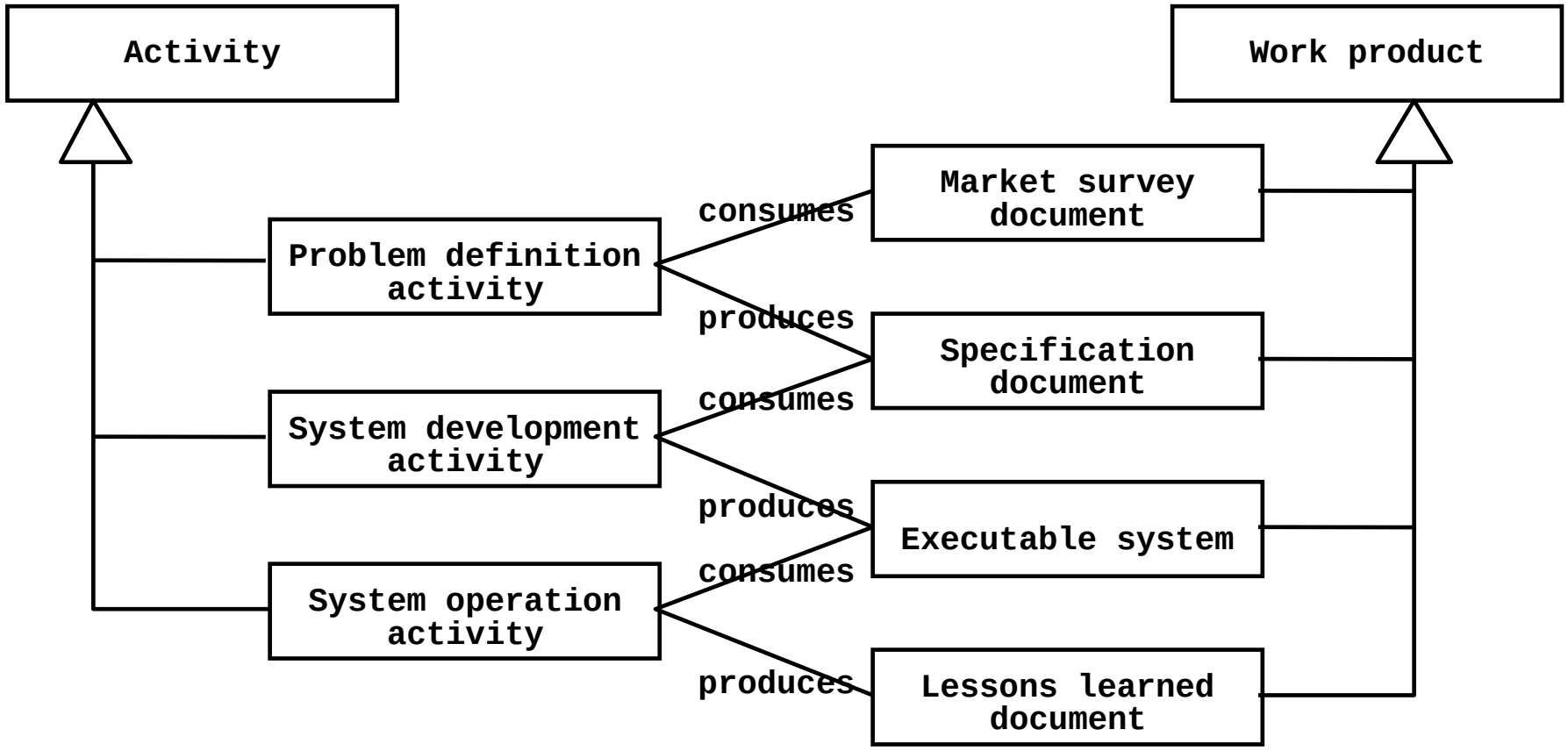
Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Activity and Entity Centered Software Life Cycle



Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Activity and Entity Centered Software Life Cycle



Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

IEEE 1074: Standard for Developing Software Lifecycle Processes

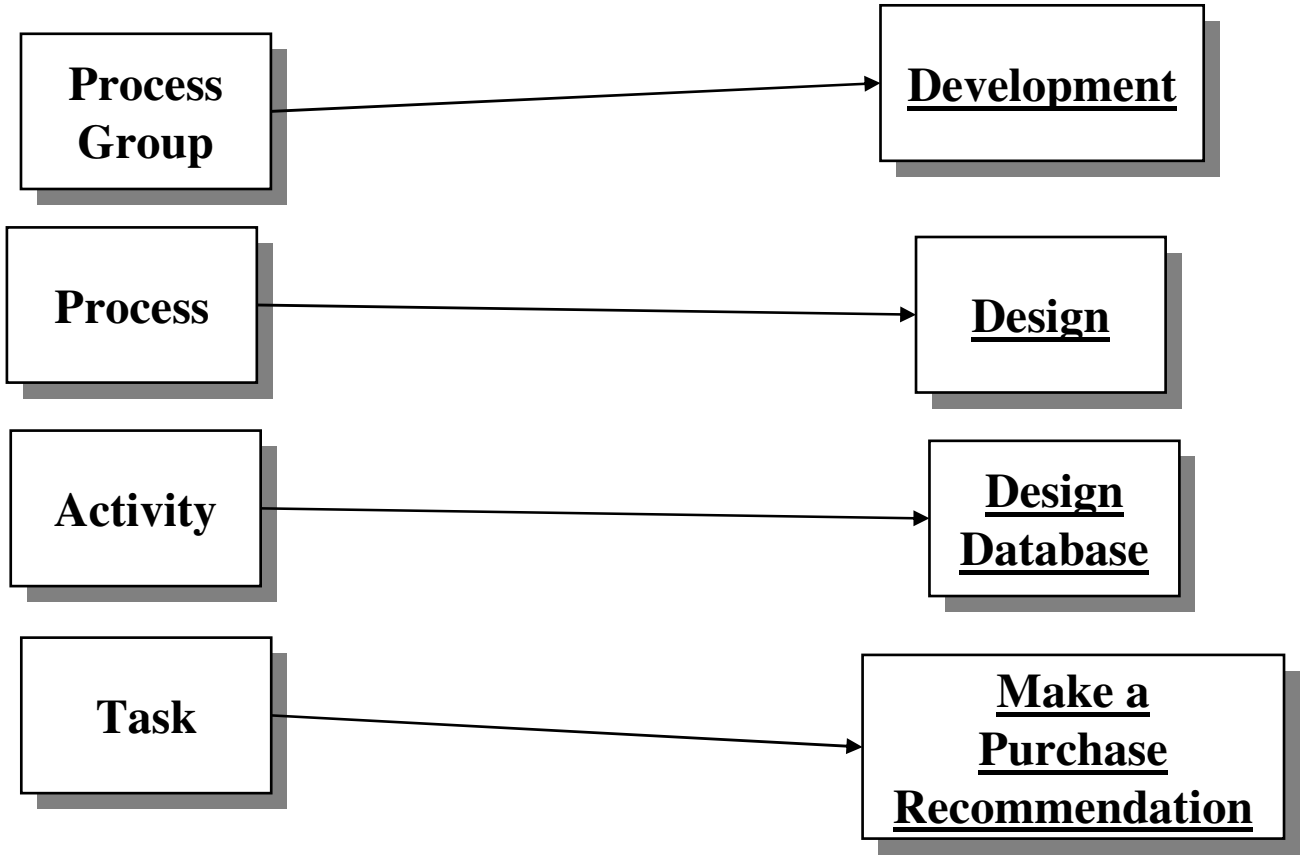
- Describes a set of activities and processes that are **mandatory** for development and maintenance of software
- Establishes a common framework for developing lifecycle models

IEEE 1074: Standard for Developing Software Lifecycle Processes

- Process Group - consists of Set of Processes
  - Design process is a part of Development group
- Process - consists of Activities. Design process may consist of:
  - Perform Architectural Design
  - Design Database (If Applicable)
  - Design Interfaces
  - Select or Develop Algorithms (If Applicable)
  - Perform Detailed Design (= Object Design)
- Activity - consists of subactivities and tasks. Design Database activity may consist of:
  - Review Relational Databases
  - Review Object-Oriented Databases
  - Make a Purchase recommendation
  - ....

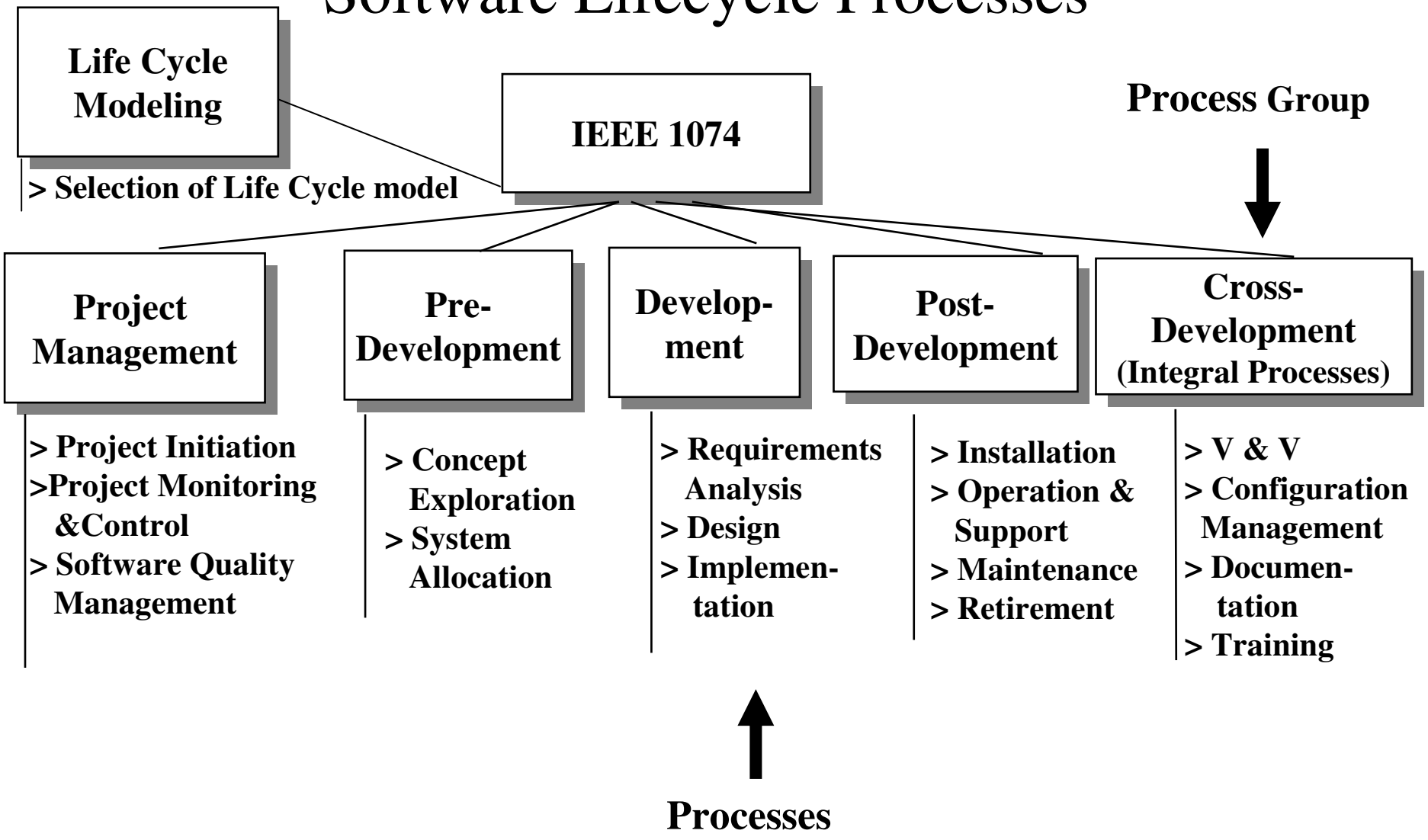
IEEE 1074: Standard for Developing Software Lifecycle Processes

- Process Group: Consists of Set of Processes
- Process: Consists of Activities
- Activity: Consists of sub activities and tasks



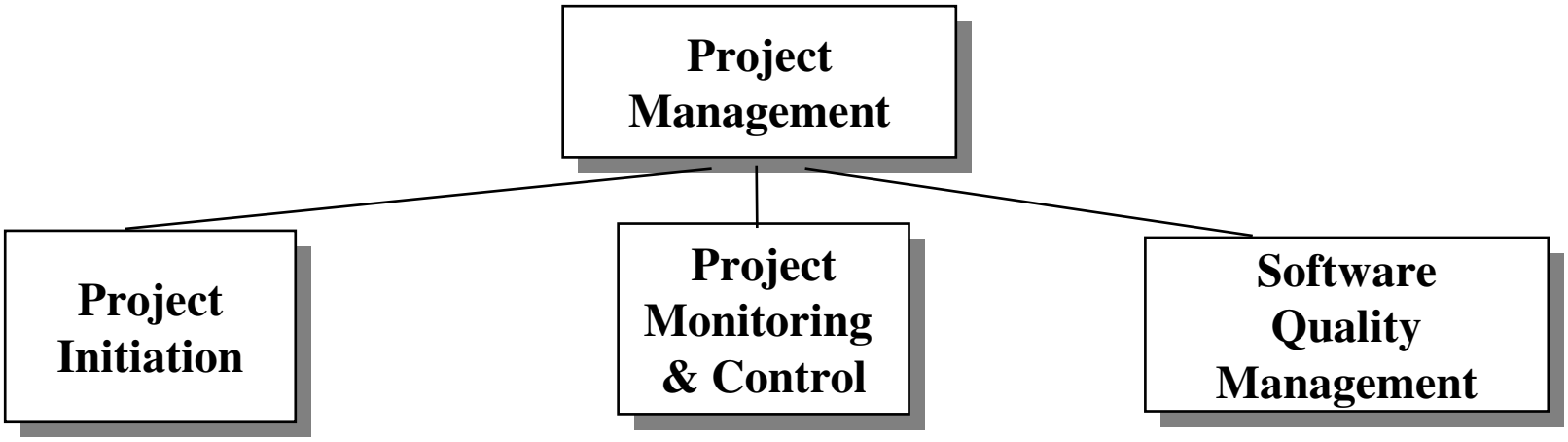
Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

IEEE 1074: Standard for Developing Software Lifecycle Processes



Adopted Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Project Management Process Group

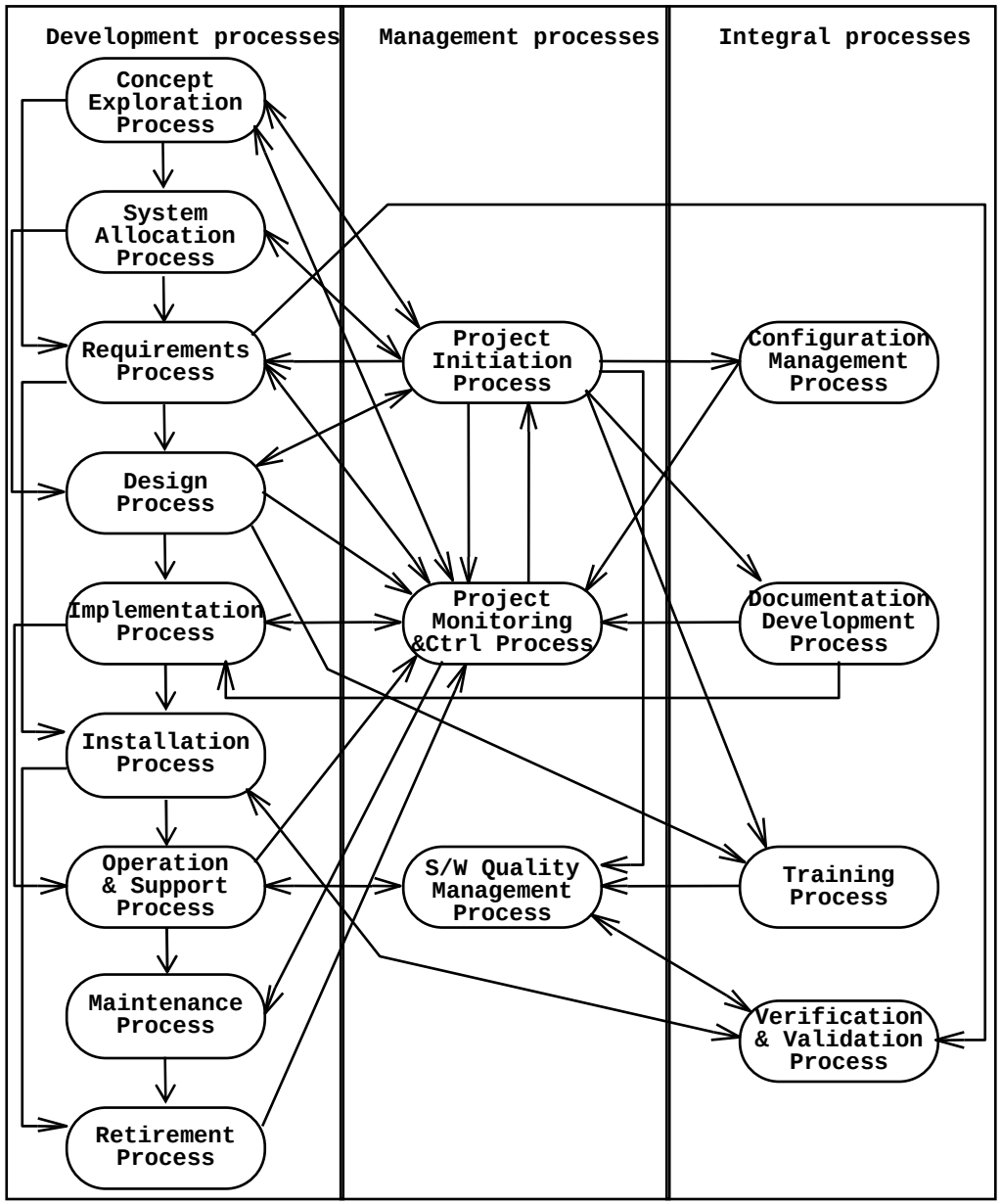


Map Activities to Software Life Cycle Model
Allocate Project Resources
Establish Project Environment
Plan Project Management

Analyze Risks
Perform Contingency Planning
Manage the Project
Retain Records
Implement Problem Reporting Model

Plan Software Quality Mesurement
Define Metrics
Manage Software Quality
Identify Quality Improvement Needs

IEEE 1074 standard.

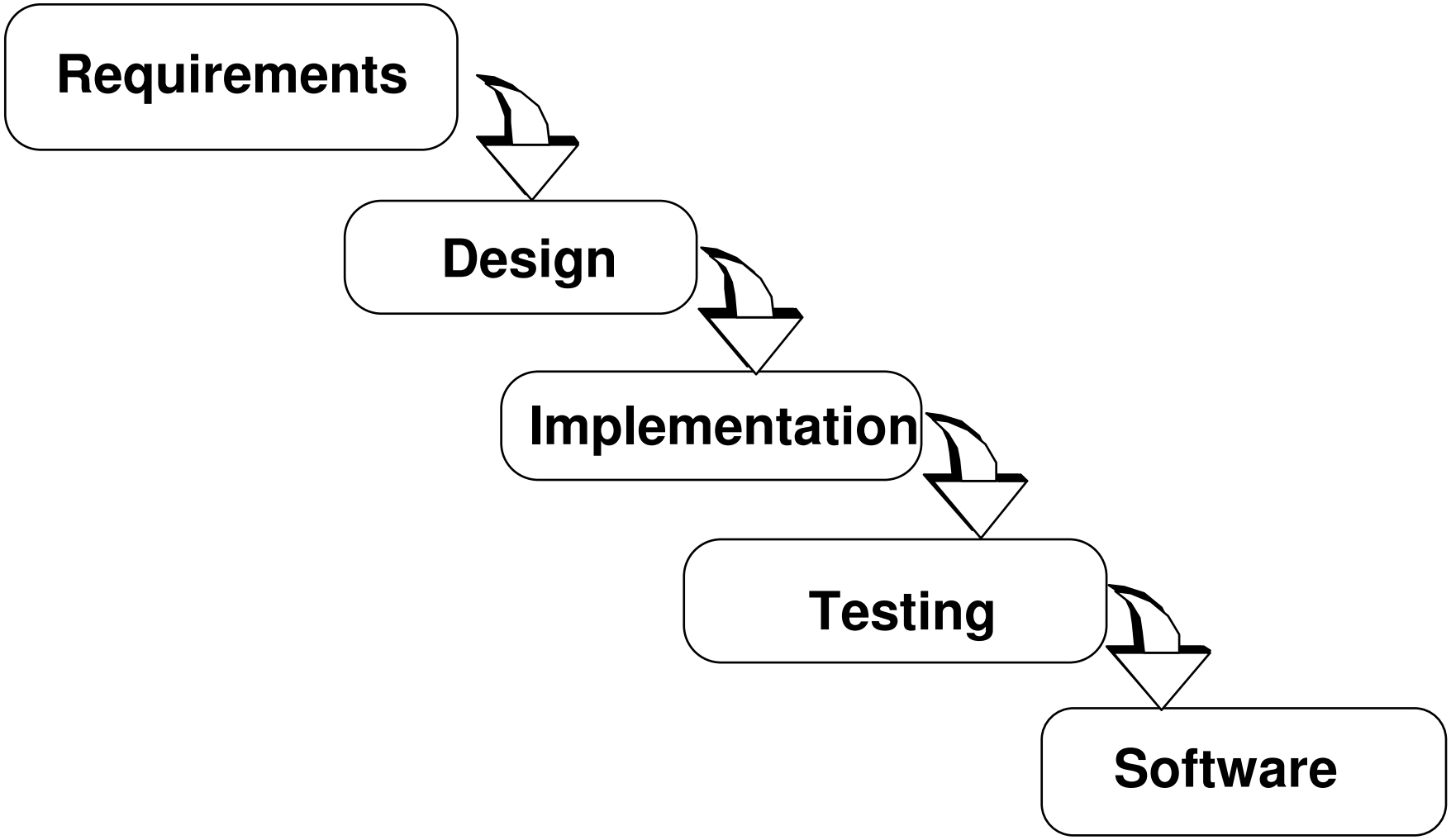


Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

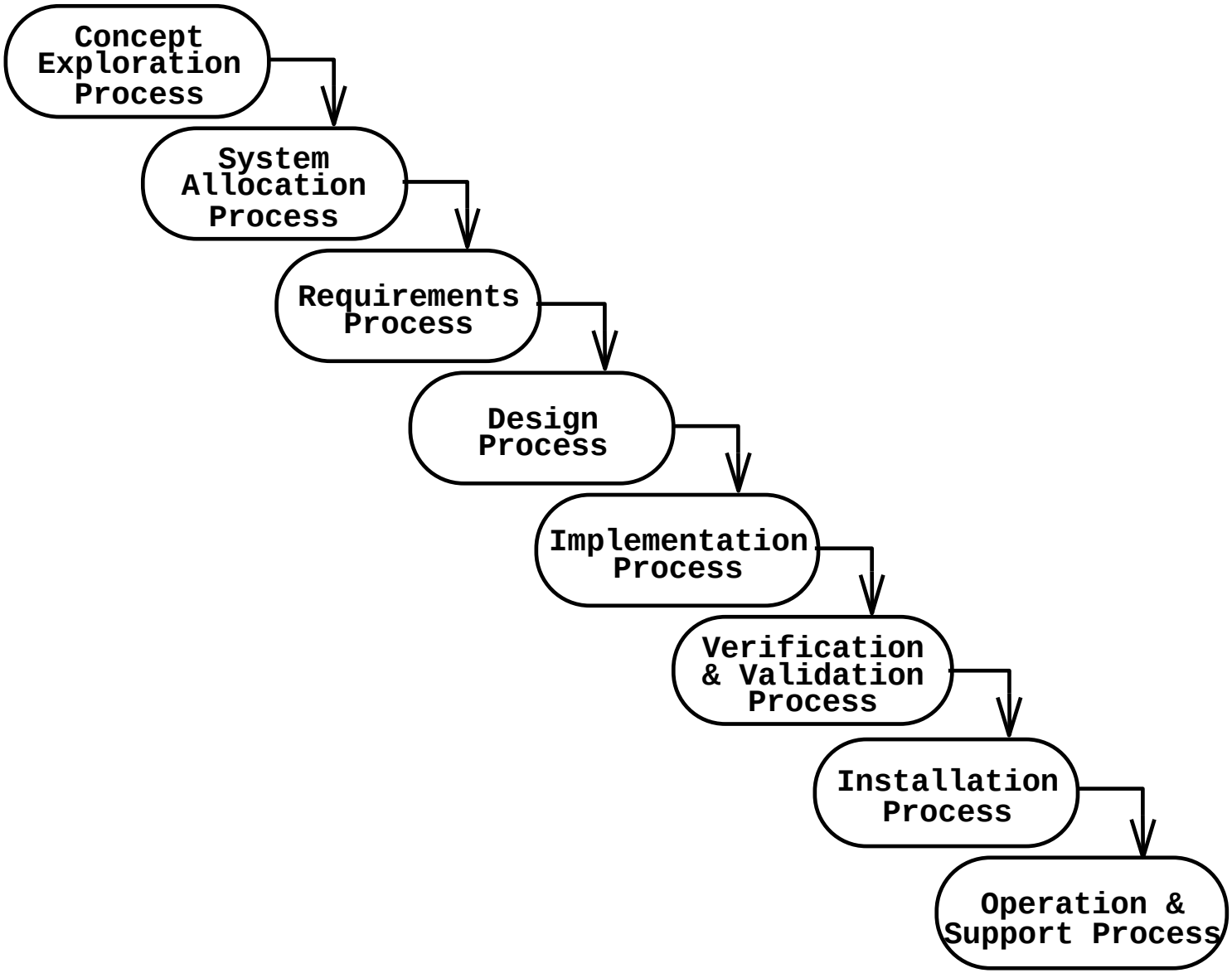
Capability Maturity Model (CMM)

- **Level 1: Initial**
  - Ad hoc activities applied
  - Success depends on skills of key individuals
  - System is black box for users
  - No interaction with user during the project
- **Level 2: Repeatable**
  - Each project has well-defined software life cycle
  - Models differ from project to project
  - Previous experience in similar projects allows predictability success
  - Interaction with user in defined points
- **Level 3: Defined**
  - Documented software life cycle model is used for all activities
  - Customized version of the model is produced for each project
  - User knows standard model and the model selected for the project
- **Level 4: Managed**
  - Metrics for activities and deliverables defined
  - Collecting data during project duration
  - The software life cycle model can be analyzed
  - User informed about the risks before the project and knows the measures
- **Level 5: Optimized**
  - The measurement data are used as a feedback to improve the software life cycle over lifetime of the organization
  - The user, mangers and developers communicate and work together during the whole project

What do we want?



Waterfall Model

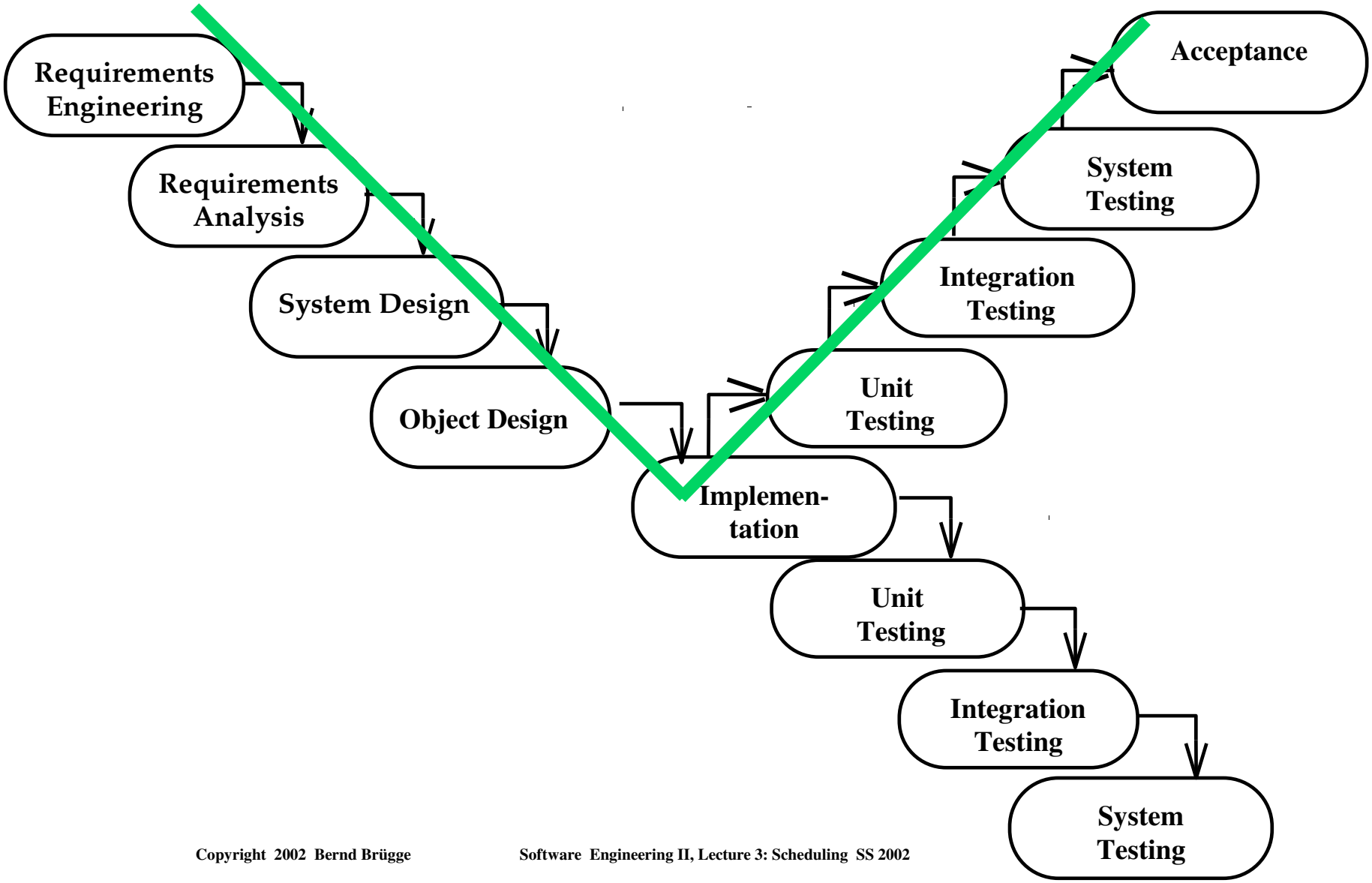


Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

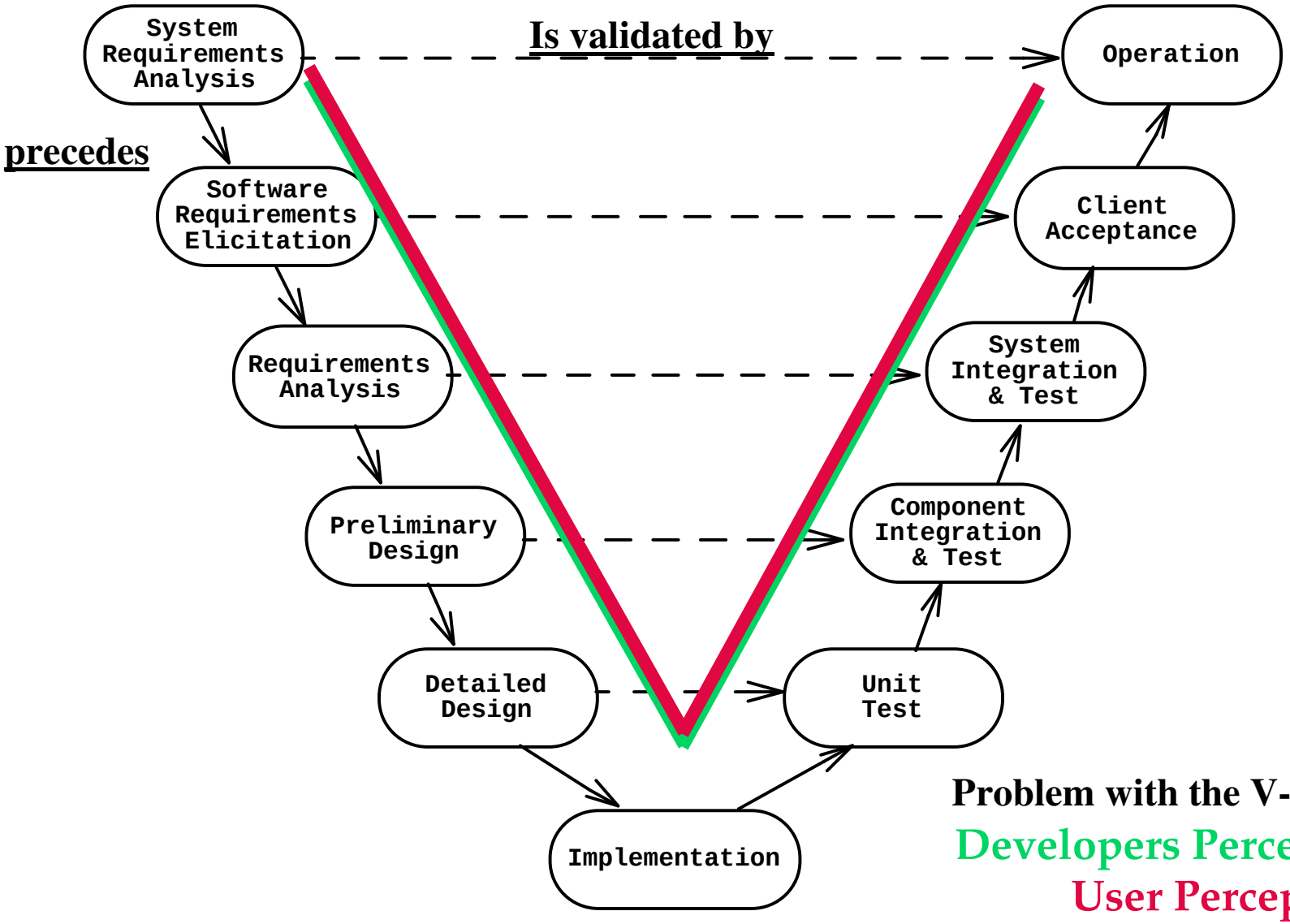
Waterfall model

- Managers like waterfall models:
  - Clear milestones
  - No need to look back (linear system), one activity at a time
  - Easy to check progress : 90% coded, 20% tested
- In practice, software development is not sequential
  - The development stages overlap
- Different stakeholders need different abstractions
- System development is a nonlinear activity

V- model


Requirements Engineering
Requirements Analysis
System Design
Object Design
Implemen-tation
Unit Testing
Integration Testing
System Testing
Acceptance
Unit Testing
Integration Testing
System Testing

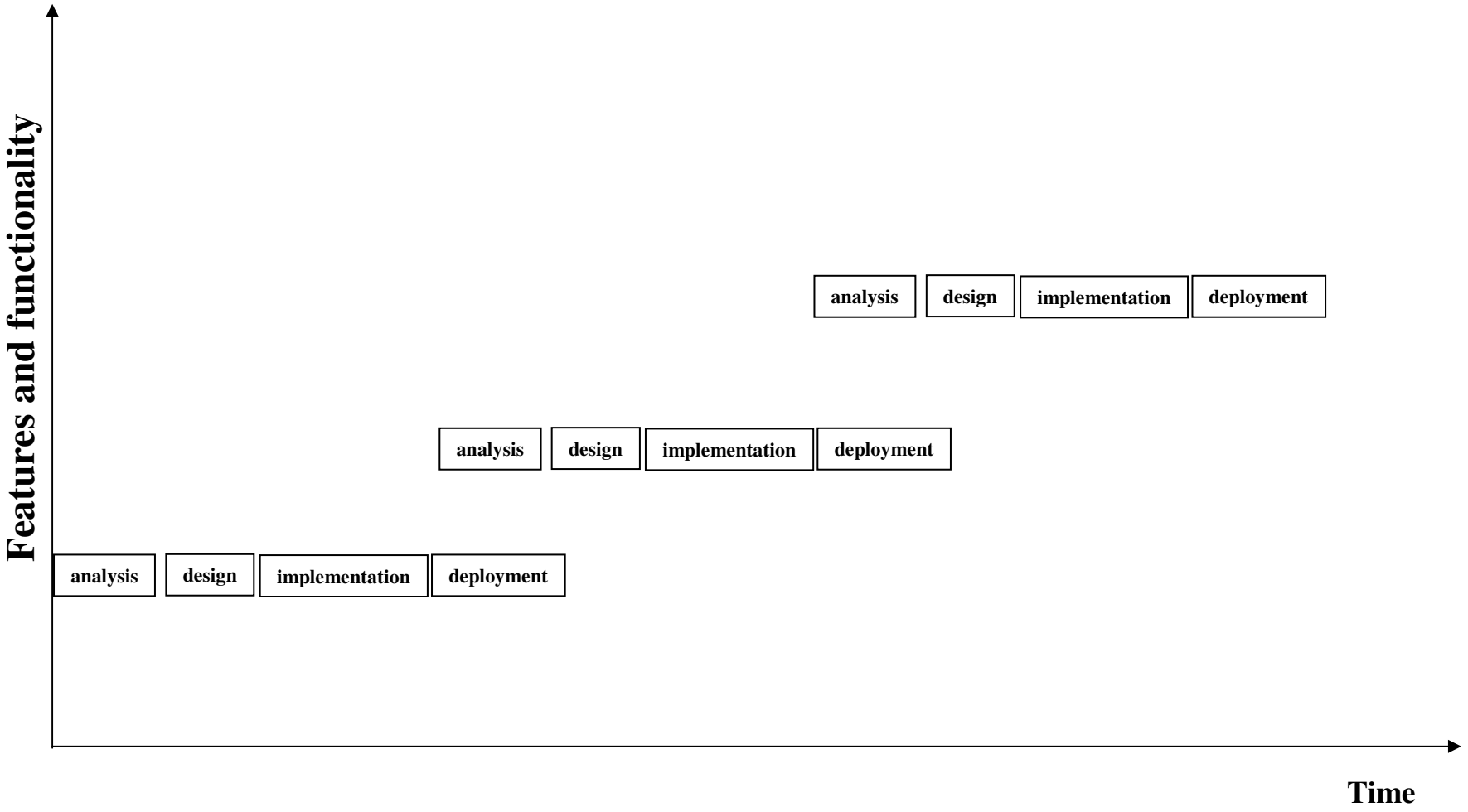
V-Model



Problem with the V-Model:

Developers Perception = User Perception

Incremental model



Iterative models

- Software development is iterative
  - During design problems with requirements are identified
  - During coding, design and requirement problems are found
  - During testing, coding, design & requirement errors are found

  => Spiral Model

Spiral model

- The spiral model proposed by Boehm is an iterative model with the following activities
  - Determine objectives and constraints
  - Evaluate Alternatives
  - Identify risks
  - Resolve risks by assigning priorities to risks
  - Develop a series of prototypes for the identified risks starting with the highest risk.
  - Use a waterfall model for each prototype development ("cycle")
  - If a risk has successfully been resolved, evaluate the results of the "cycle" and plan the next round
  - If a certain risk cannot be resolved, terminate the project immediately

Spiral model

- Concept of Operation,
- Software Requirements,
- Software Product Design,
- Detailed Design,
- Code,
- Unit Test,
- Integration and Test,
- Acceptance

- For each cycle go through these activities
  - Quadrant IV: Define objectives, alternatives, constraints
  - Quadrant I: Evaluate alternative, identify and resolve risks
  - Quadrant II: Develop, verify prototype
  - Quadrant III: Plan next “cycle”

IV Determine objectives

IV Specify constraints

IV Generate alternatives

I Identify risks

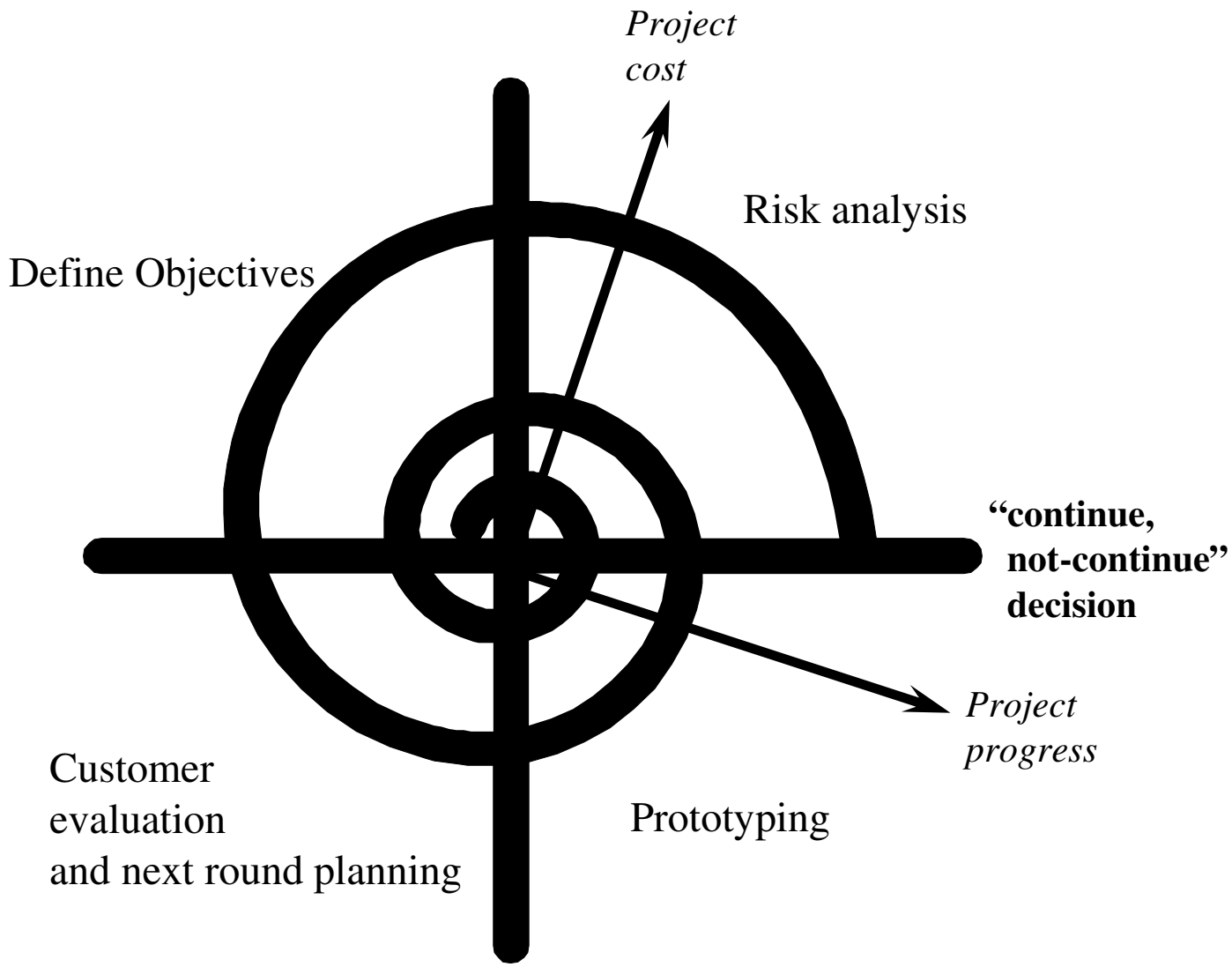
I Resolve risks

II Develop and verify prototype

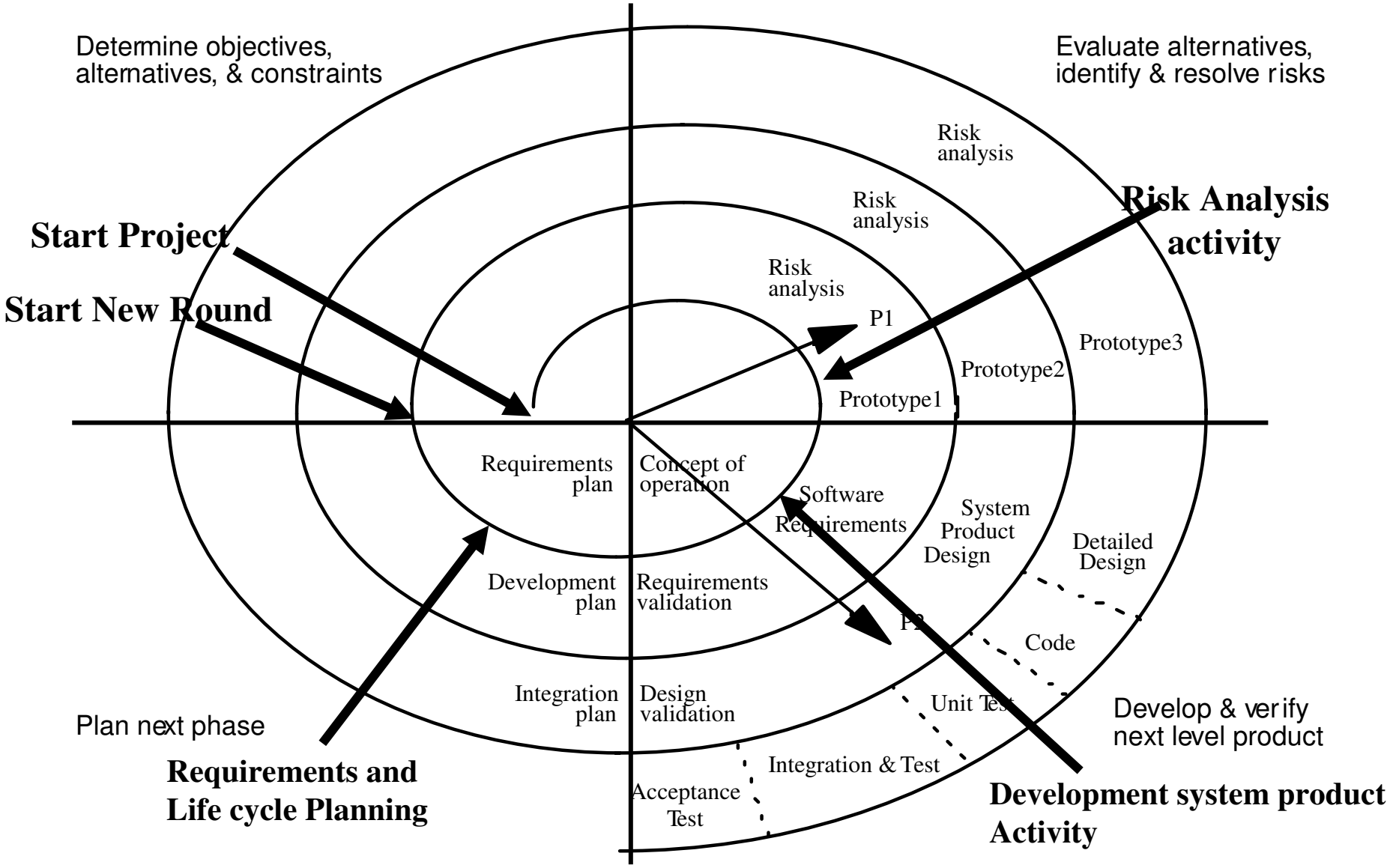
III Plan

Adopted from Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java

Spiral model



Spiral Model



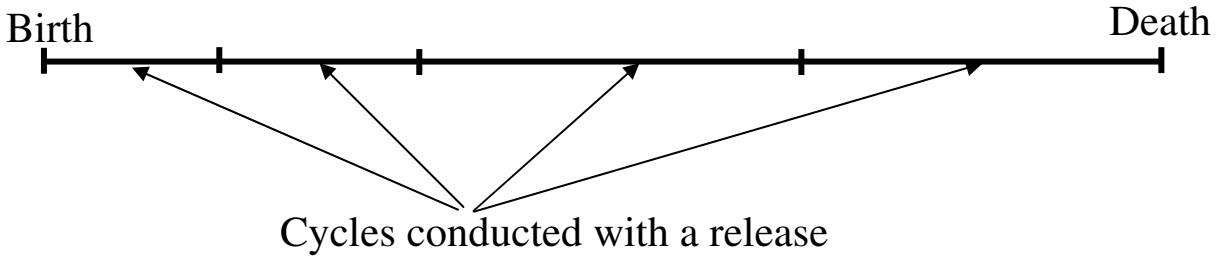
Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java Adopted from Boem

Limitations of Waterfall and Spiral Models

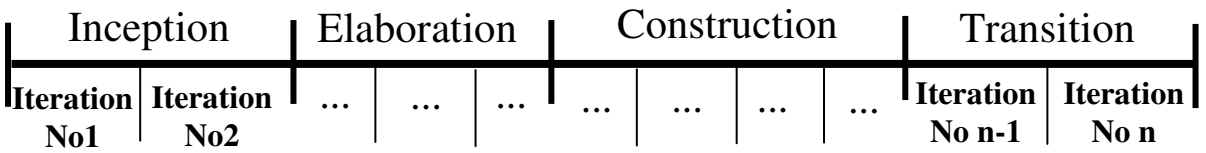
- Neither of these model deals well with frequent change
  - The Waterfall model assume that once you are done with a phase, all issues covered in that phase are closed and cannot be reopened
  - The Spiral model can deal with change between phases, but once inside a phase, no change is allowed
- What do you do if change is happening more frequently? (“The only constant is the change”)

Unified Software Development Process (UP)

- Repeats over a series of cycles



- Each cycle consists of four phases which are subdivided into iterations



Unified process

- Inception – establishes a business case for the system
- Elaboration – most of the product cases are specified in details, architecture is designed
- Construction – the product is built. The architectural baseline becomes a full-pledged system
- Transition – period when product moves to beta release

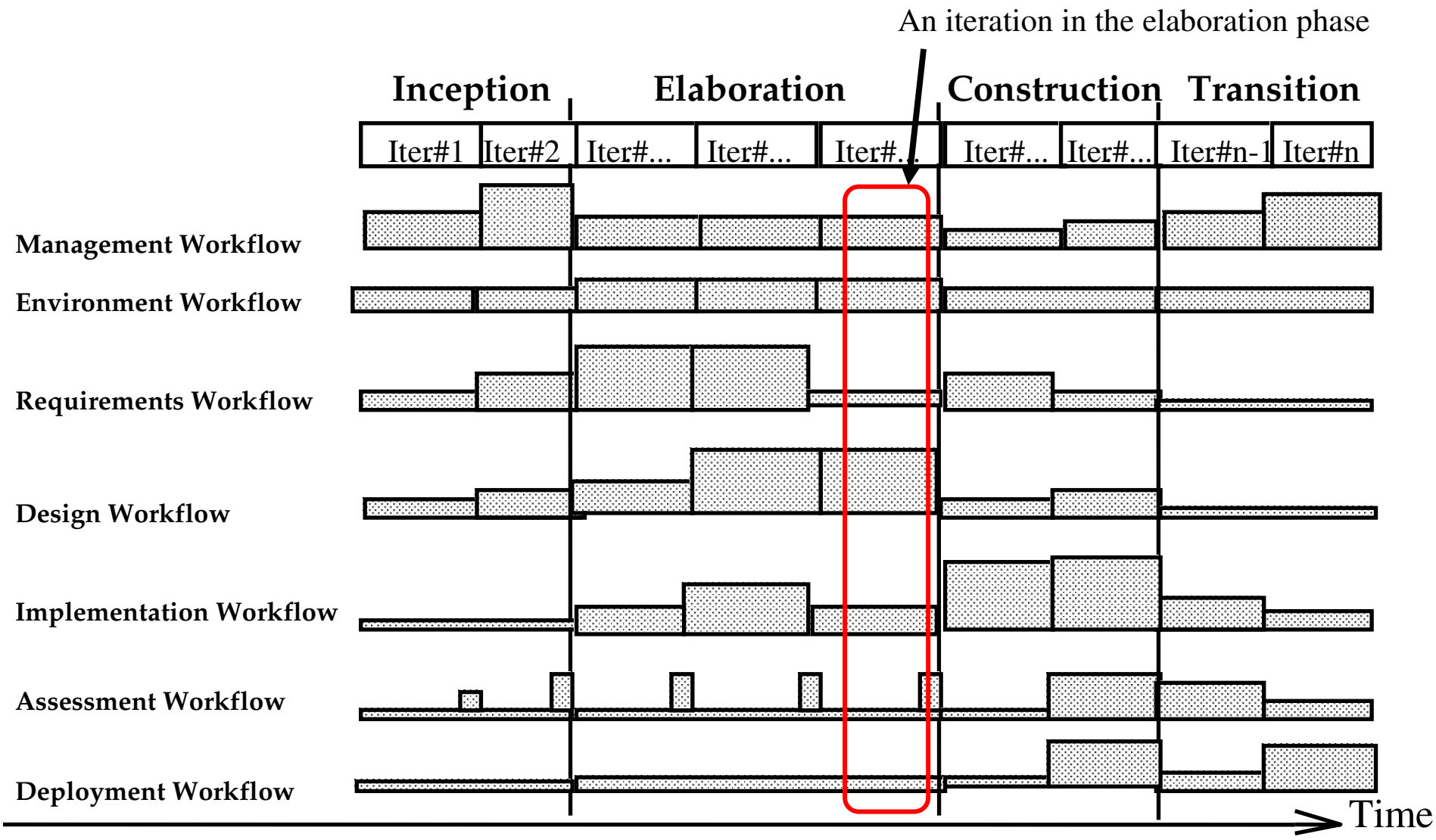
Unified Software Development Process (UP)

- UP organizes projects in two-dimensional terms
- The **horizontal dimension** represents the successive phases of each project iteration:
  - inception,
  - elaboration,
  - construction, and
  - transition.
- The **vertical dimension** represents software development disciplines and supporting activities of configuration and change management, project management, and environment.

Workflows

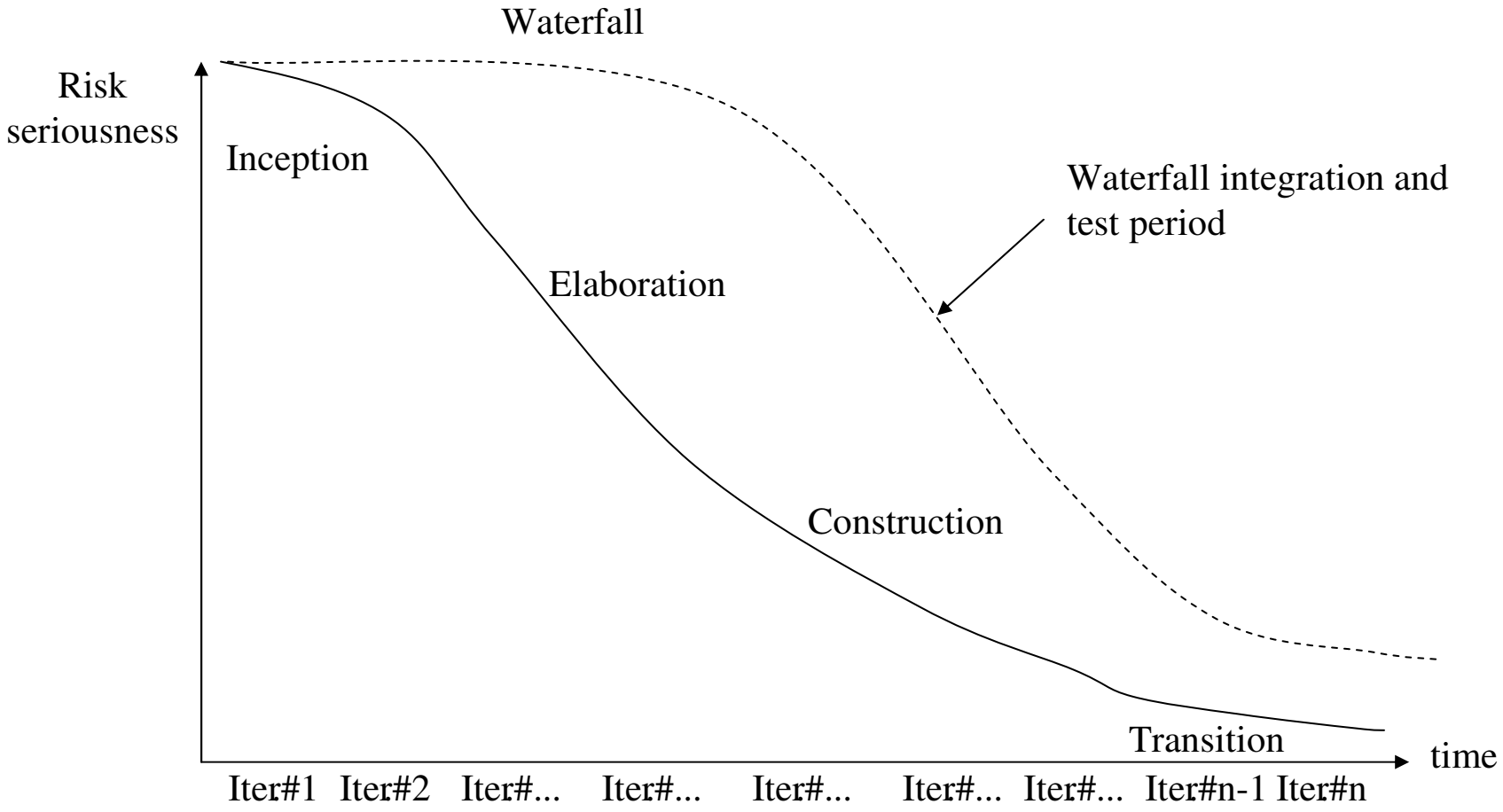
- Cross-functional
  - Management – planning aspects
  - Environment – automation of the process itself
  - Assessment – assesses processes and products needed for reviews
  - Deployment – transition of the system
- Engineering
  - Requirements
  - Design
  - Implementation
  - Testing

Unified Software Development Process

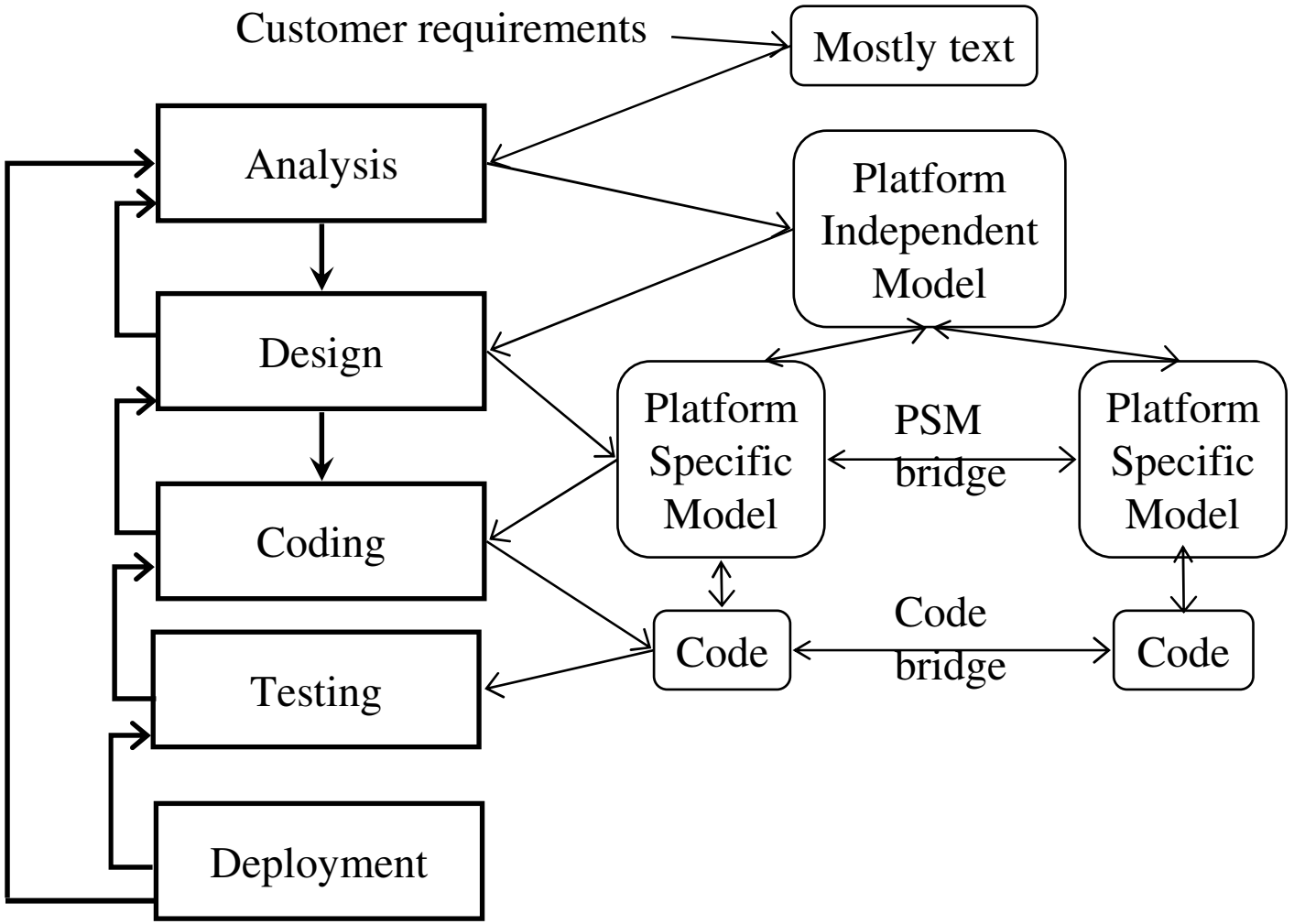


From Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java adopted from Jacobson

UP vs. Waterfall



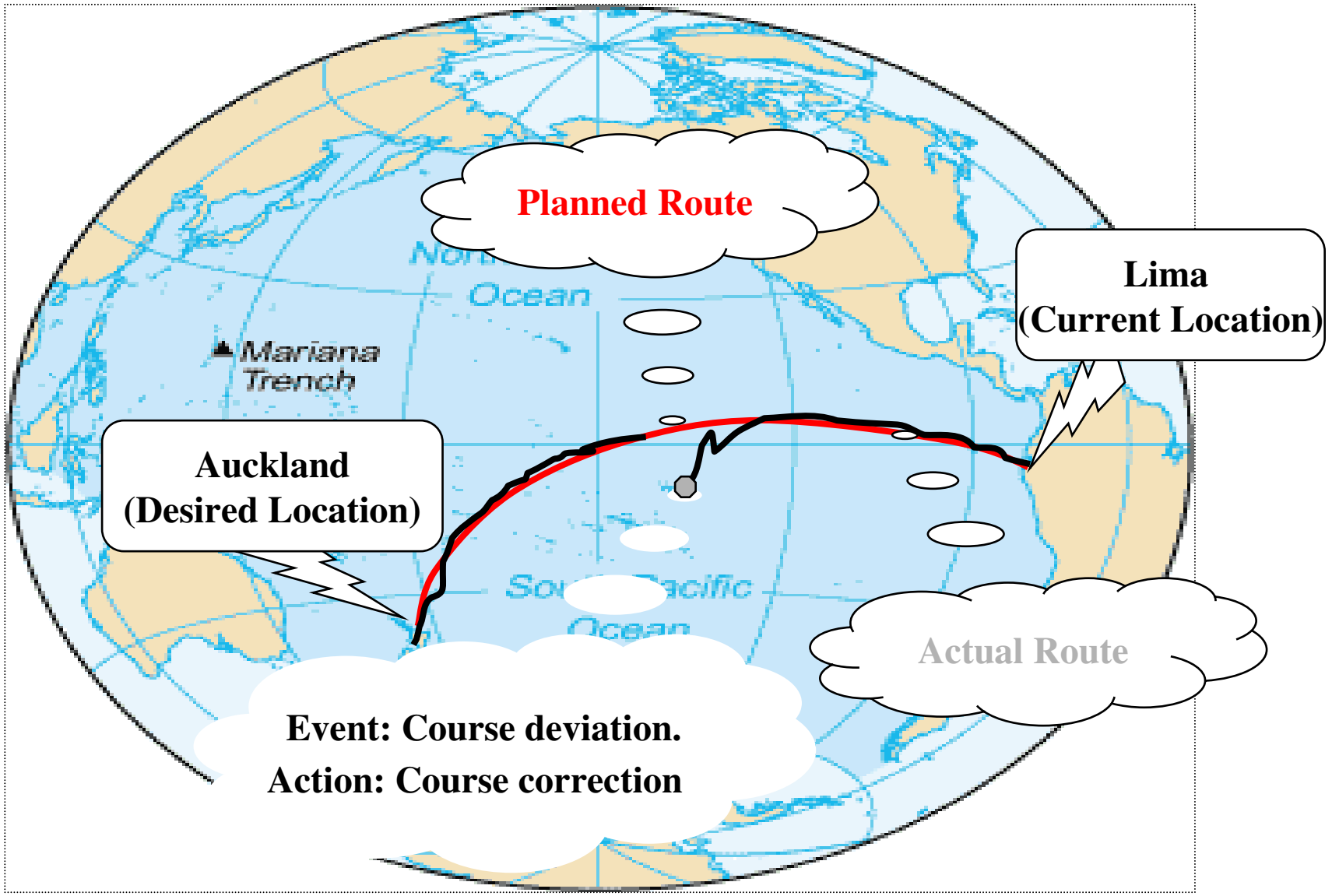
Model Driven Architecture



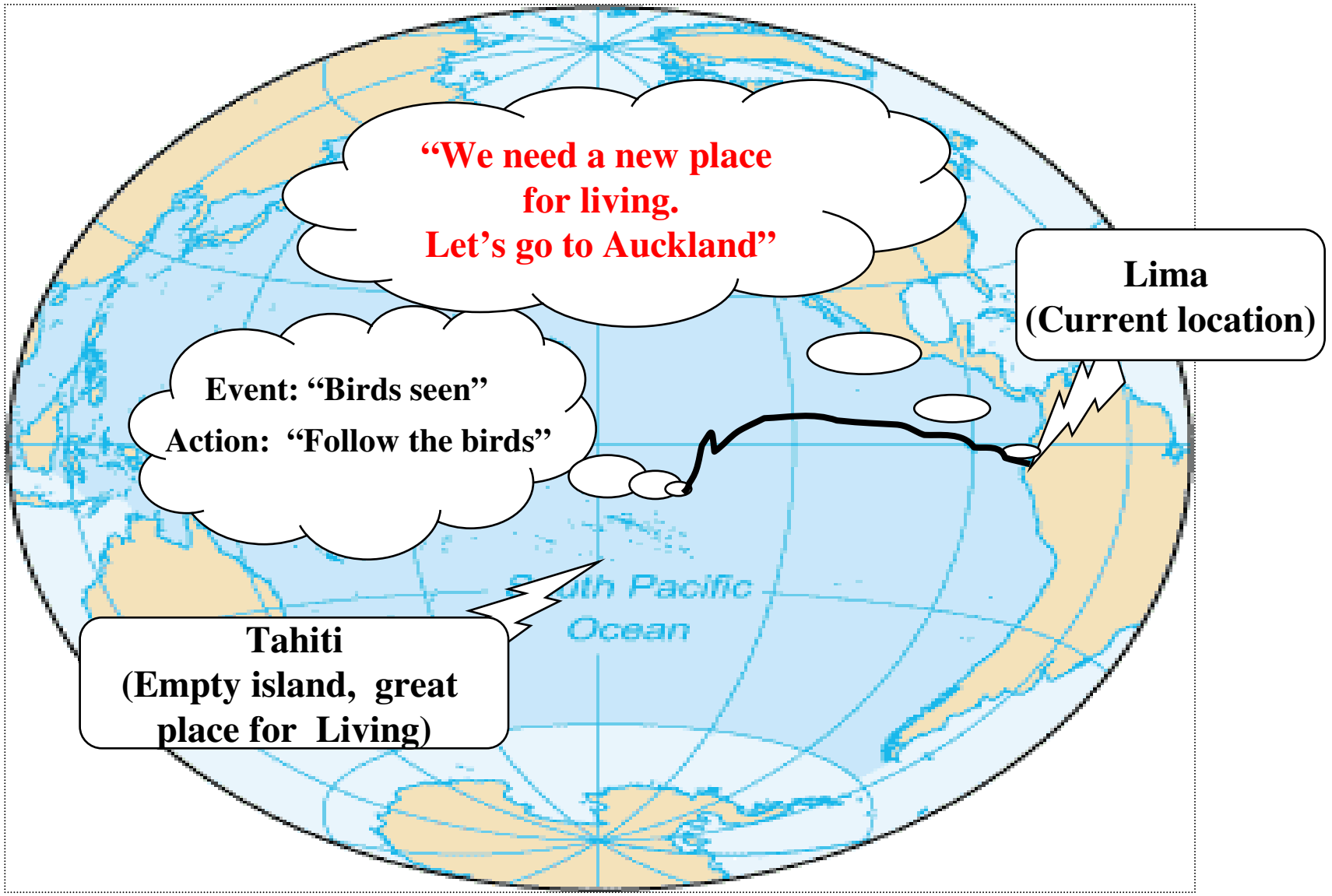
How much Planning?

- Two styles of navigation [Gladwin 1964]
  - European navigation
  - "Polynesian navigation"

“European Navigation”



Polynesian Navigation



Auckland Project Plan (European Navigation)

- Project Goal: Auckland
- Desired Outcome: Auckland is found
- Team: Captain and 50 sailors
- Organization: Flat hierarchy
- Tools: Compass, speed meter, map
- Methods: Determine planned course, write planned course before departure. Example: Start Lima. Sail West, keep the compass constantly at 97 degrees, stay at latitude 20 degrees)
- Work breakdown structure
    - Task T1 (Check direction): Determine current direction of ship
    - Task T2 (Compute deviation): Determine deviation from desired course
    - Task T3 (Course Correction): Bring ship back on course
- Process:
    - T1 and T2 are executed hourly. If there is a deviation, T3 is executed to bring the ship back on the planned course.
- Schedule: With good wind 50 days, if doldrums are encountered, 85 days.

Auckland Project Plan (Polynesian Navigation)

- Project Goal: Auckland
- Desired Outcome: A new place for living is found
- Team: Captain and 50 sailors
- Organization: Flat hierarchy
- Tools: Use stars for navigation, measure water temperature with hand
- Methods: Set up a set of event-action rules. When an event occurs, determine the action to be executed in the given context.
- Work breakdown structure
  - Task T1 (Determine direction): Set direction of ship to a certain course
  - Task T2 (Check Clouds): Look for non-moving clouds in the distance
  - Task T3 (Check Birds): Look for birds and determine their direction
  - Task T4 (Compute course): Determine new course for ship
  - Task T5 (Change course): Change direction to follow new course
- Process:
  - Start with T1. Tasks T2 and T3 are executed regularly. The result (cloud detected, birds detected, nothing happened) is interpreted in the current context. If the interpretation makes a new course more promising, execute task T4 and T5.
- Schedule: None

Situated action

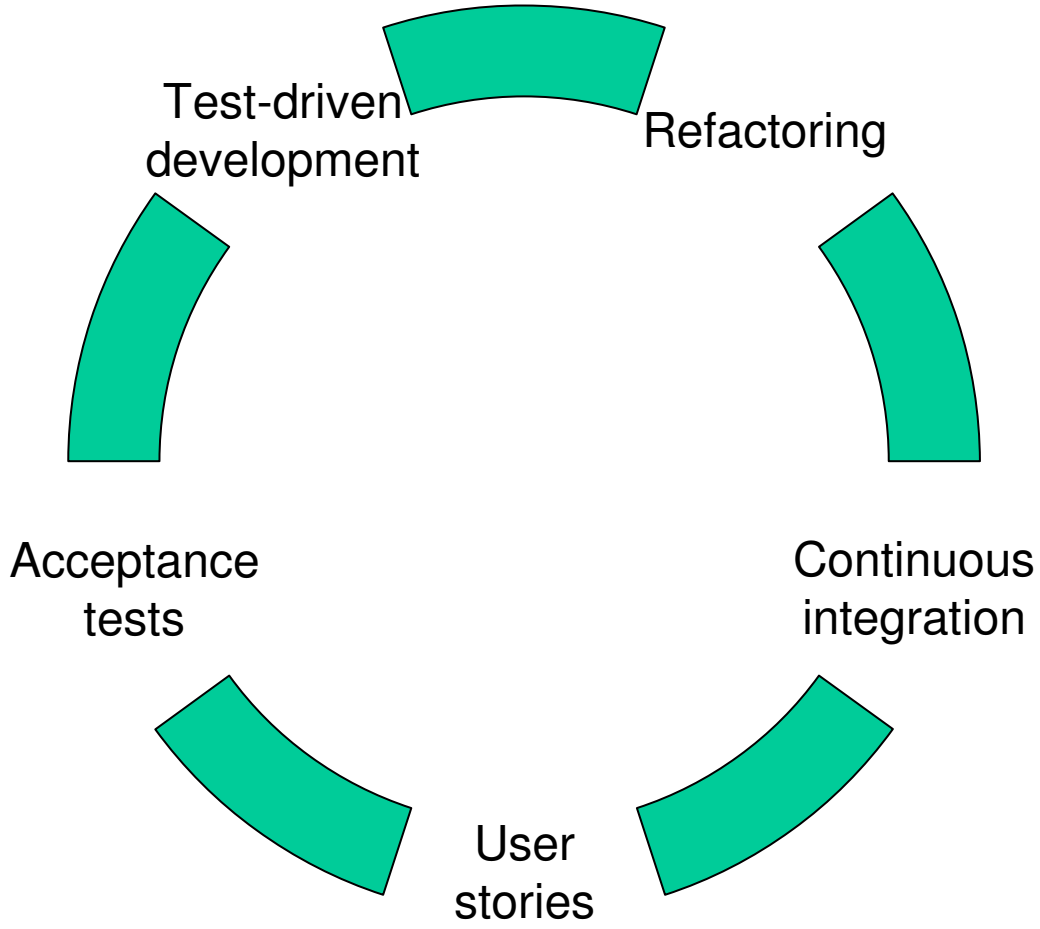
- Situated action [Suchman 1990]
  - Selection of action depends on the type of event, the situation and the skill of the developer. Also called context-dependent action.
- Examples of navigation events: “Course deviation”, “Birds seen”, “Clouds seen”.
- European Navigation is context independent:
  - Event: “Course deviation in the morning”
    - Action: “Course correction towards planned route”
  - Event: “Course deviation in the evening”
    - Action: “Course correction towards planned route”
- Polynesian Navigation is context dependent:
  - Event: “Birds seen”, Context: Morning
    - Action: “Sail opposite to the direction the birds are flying”
  - Event: “Birds seen”, Context: Evening
    - Action: “Sail in the direction the birds are flying

Agile software development

Key points of agility in software production:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Modeling



Summary

- Software engineering is a modeling, problem-solving activity, knowledge acquisition activity and rationale-driven activity.
- Dealing with complexity:
  - Abstraction
  - Decomposition
  - Hierarchy
- Software Engineering activities
  - Analysis
    - **Requirements elicitation**
    - **Requirements analysis**
  - Design
    - **System design** –
    - **Object design**
  - Implementation
  - Testing
- Software Development Life Cycle
  - Waterfall
  - Incremental
  - Spiral/UP
  - Agile methods

Next lecture

- UML

Chapter 2 in the text-book.