THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

SENG2130
Systems Analysis and Design

Week 1 Course Introduction

Dr. Joe Ryan
Room: ES234
Phone: x16071
email: joe.ryan@newcastle.edu.au

Timetable

COURSE ACTIVITY	CLASS TYPE	DAY	START	END	DURATION	DATES	ROOM	CAMPUS	COMMENTS
SENG2130/S1_CA/C01/01	Computer Lab.	Friday	10:00am	12:00pm	2:00	16 Mar 18 - 30 Mar 18, 20 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/C01/02	Computer Lab.	Friday	12:00pm	2:00pm	2:00	16 Mar 18 - 30 Mar 18, 20 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/C01/03	Computer Lab.	Friday	2:00pm	4:00pm	2:00	16 Mar 18 - 30 Mar 18, 20 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/C01/04	Computer Lab.	Thursday	5:00pm	7:00pm	2:00	15 Mar 18 - 29 Mar 18, 19 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/C01/05	Computer Lab.	Friday	5:00pm	7:00pm	2:00	16 Mar 18 - 30 Mar 18, 20 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/C01/06	Computer Lab.	Thursday	8:00am	10:00am	2:00	15 Mar 18 - 29 Mar 18, 19 Apr 18	ES205	CALLAGHAN	
SENG2130/S1_CA/L01/01	Lecture	Wednesday	3:00pm	5:00pm	2:00	28 Feb 18 - 28 Mar 18, 18 Apr 18 - 6 Jun 18	MCTH100	CALLAGHAN	
SENG2130/S1_CA/W01/01	Workshop.	Friday	10:00am	12:00pm	2:00	2 Mar 18 - 30 Mar 18, 20 Apr 18 - 8 Jun 18	AVG14	CALLAGHAN	
SENG2130/S1_CA/W01/02	Workshop.	Friday	12:00pm	2:00pm	2:00	2 Mar 18 - 30 Mar 18, 20 Apr 18 - 8 Jun 18	HE28	CALLAGHAN	
SENG2130/S1_CA/W01/03	Workshop.	Friday	2:00pm	4:00pm	2:00	2 Mar 18 - 30 Mar 18, 20 Apr 18 - 8 Jun 18	AVG14	CALLAGHAN	
SENG2130/S1_CA/W01/04	Workshop.	Thursday	5:00pm	7:00pm	2:00	1 Mar 18 - 29 Mar 18, 19 Apr 18 - 7 Jun 18	HC01	CALLAGHAN	
SENG2130/S1_CA/W01/05	Workshop.	Friday	5:00pm	7:00pm	2:00	2 Mar 18 - 30 Mar 18, 20 Apr 18 - 8 Jun 18	GP212	CALLAGHAN	only run if required
SENG2130/S1_CA/W01/06	Workshop.	Thursday	9:00am	11:00am	2:00	1 Mar 18 - 29 Mar 18, 19 Apr 18 - 7 Jun 18	HA145	CALLAGHAN	only run if required

Workshops/Computer Labs start at Week 3

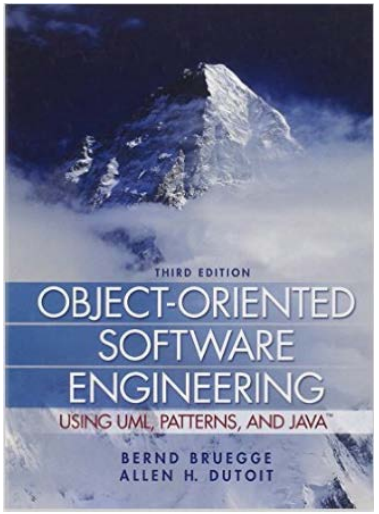
References

- Object-Oriented Software Engineering

  using UML, Patterns, and JAVA

  By Bernd Bruegge, Allen H. Dutoit

  3rd edition, Prentice Hall

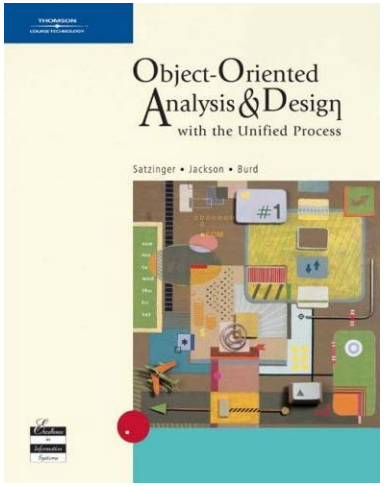


- Object-Oriented Analysis & Design

  with the Unified Process

  By Satzinger, Jackson and Burd

  Thomson Publishing Co.



Lecture Material

- Note that this is NOT a remote delivery course and that students are expected to attend lectures and to work through case studies presented at lectures and in workshops. Worked solutions to these case studies will be discussed but will not necessarily be placed on Blackboard.
- The reference books contain examinable material and should be viewed as an extension of the material presented in lectures.
- Examination material may be taken from presented lecture/tutorial material.

Assessment

- Group Project:
  - Part 1 (20%)
    - Requirements Modelling
  - Part 2 (35%)
    - Design
    - Design Evaluation and Software Testing
- Final Exams (45%)
  - Note: Essential Criterion applies (You must obtain 50% overall and at least 40% in the final exam to pass the course)

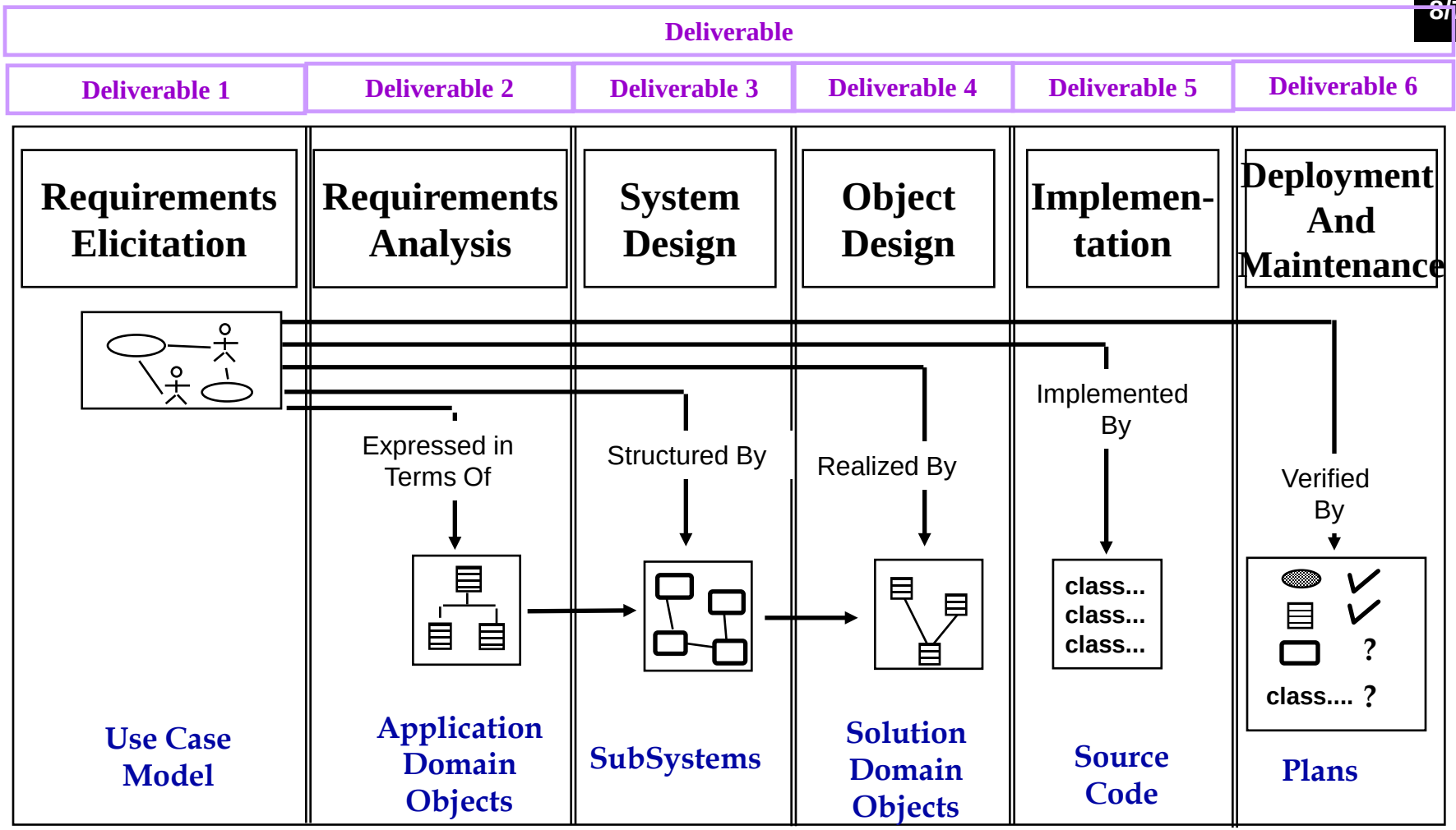
Assessment Penalties

- As per Faculty rules (see Course Outline) Late submissions will be penalised
  - 10% of the possible maximum mark for the assessment item will be deducted for each day or part day that the item is late (this applies equally to week days and weekend days).
- Assessment items submitted more than five days after the due date will be awarded zero marks.

Course Introduction

- System analysis and design deals with complexity by constructing and validating models of the application domain or the system. These include following software lifecycle activities:
  - Requirements Elicitation
  - Requirements Analysis
  - System Design
  - Object Design
  - Implementation
  - Testing
  - Deployment
  - Post Delivery Maintenance

Course Introduction



Each activity produces one or more models

Course Introduction

Week 1. Introduction of Software Engineering and lifecycles
Week 2. Requirements Elicitation: Introduction to UML and Use Case diagram
Week 3. Requirements Elicitation: Use Case description and Activity diagram
Week 4. UML diagrams – Class and Object diagrams
Week 5. UML diagrams – Sequence & Collaboration diagrams (no labs)
Week 6. Analysis (week 5 labs)
Week 7. Anzac Day – no lecture (week 6 labs)
Week 8. System design
Week 9. Object Design
Week 10. User Interface, Introductory Testing, Deployment
Week 11. Maintenence, Project Management, Ethics
Week 12. Review of the course

Outline for today

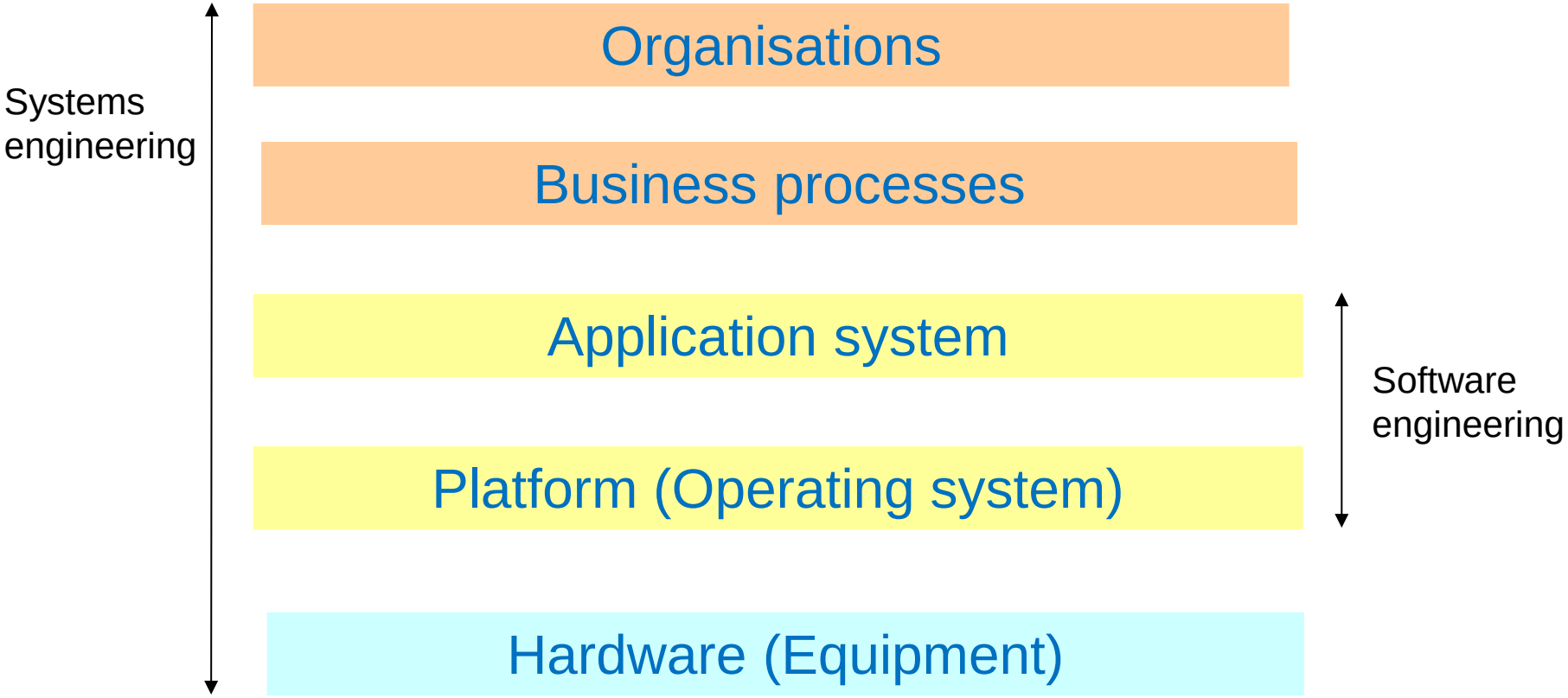
- Systems Engineering
- Development Lifecycle

System/Software Engineering

What is a system?

- A purposeful collection of inter-related components working together to achieve some common objective.
- A system may include software, mechanical, electrical and electronic hardware and be operated by people.
- System components are dependent on other system components
- The properties and behaviour of system components are inextricably inter-mingled

The systems stack



Software engineering

- Software engineering is an engineering discipline which is concerned with all aspects of software production from the early stages of system requirements through to maintaining the system after it has gone into use.
- The main objective is to support software production in order to deliver software that is “fit for purpose”, e.g., good enough (functionally, non-functionally), meets constraints (e.g., time and financial) of the environment, law, ethics and work practices.

Software engineering

- Engineering discipline – Engineers make things work. They apply theories, methods and tools that are appropriate and always try to discover solutions to problems even when there are not applicable theories and methods to support them.
- All aspects of – Software engineering is not just concerned with the technical process of software development but also with activities such as software project management and the development of tools, methods and theories to support software production.

Why software engineering?

- Software development is very difficult.
  - The problem is usually ambiguous
  - The requirements are usually unclear and changing when they become clearer
  - The problem domain (called application domain) is complex, and so is the solution domain
  - The development process is difficult to manage

Why software engineering?

- Software development is more than just writing code
  - It is problem solving
    - Understanding a problem
    - Proposing a solution and plan
    - Engineering a system based on the proposed solution using a ***good*** design

  - It is about dealing with complexity
    - Creating abstractions (models)
    - Notations for abstractions
  - It is knowledge management
    - Elicitation, analysis, design, validation of the system and the solution process

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law.

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The code contains eight principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.
- have a look now at:
  - http://www.acm.org/about/se-code
  - http://www.computer.org/cms/Computer.org/Publications/code-of-ethics.pdf
  - https://www-rohan.sdsu.edu/faculty/giftfire/

Eight Principles

1. **Public**. Software engineers shall act consistently with the public interest.
2. **Client and employer**. Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
3. **Product**. Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **Judgment**. Software engineers shall maintain integrity and independence in their professional judgment.

Eight Principles

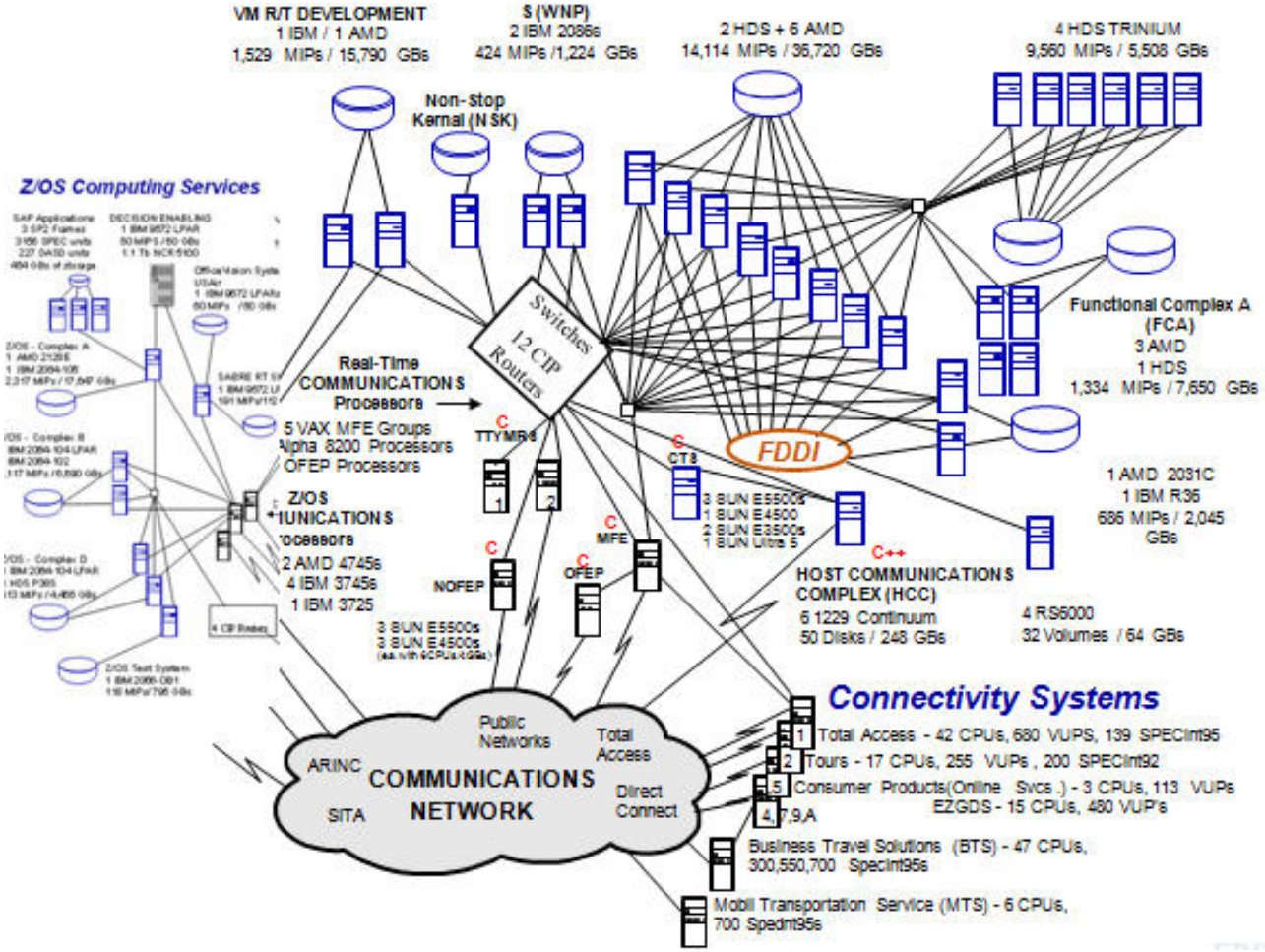
5. **Management**. Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. **Profession**. Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

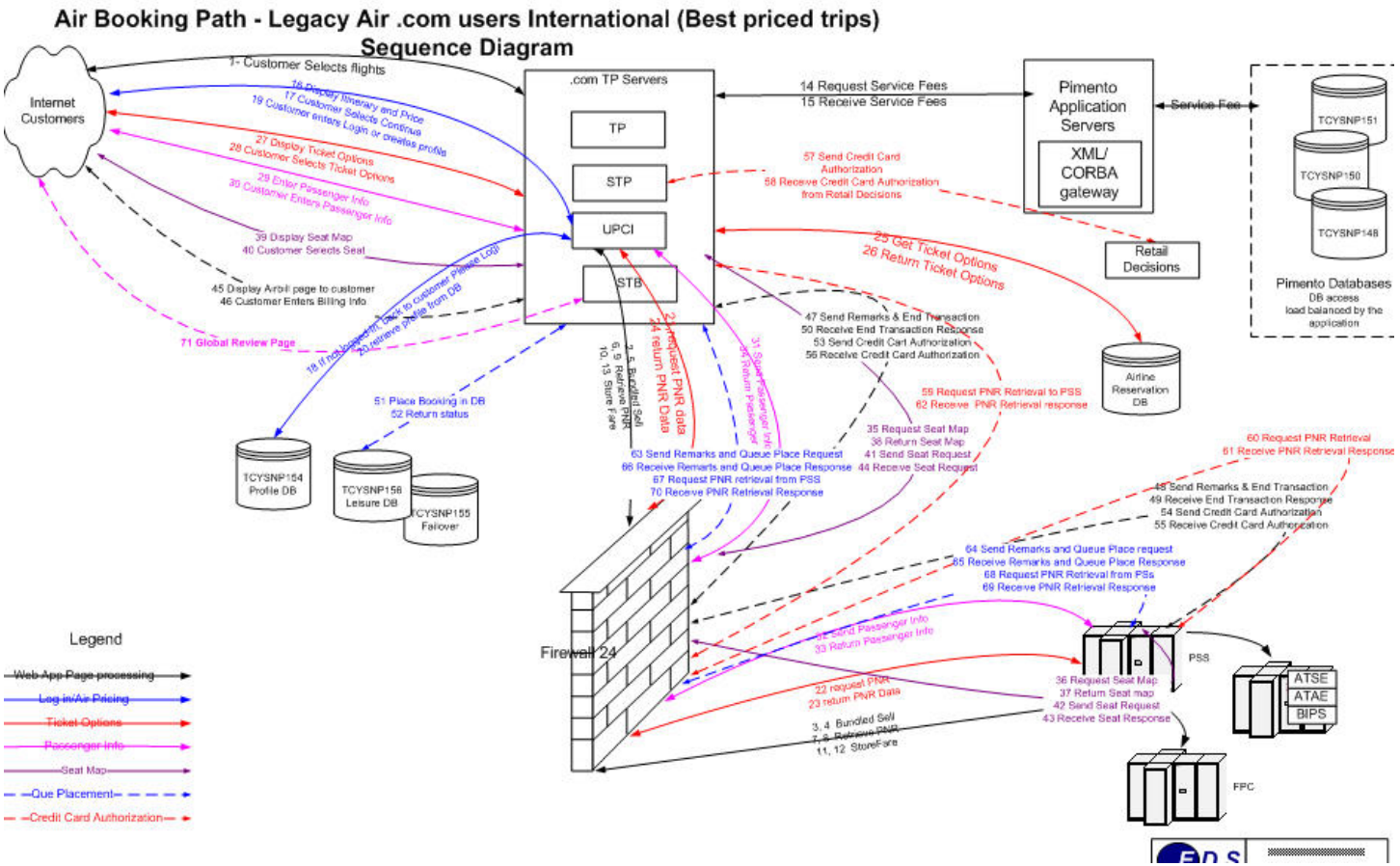
7. **Colleagues**. Software engineers shall be fair to and supportive of their colleagues.

8. **Self**. Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Complex Server Connections



Complex Message Flow



Dealing with complexity

- Abstraction
- Decomposition
- Hierarchy

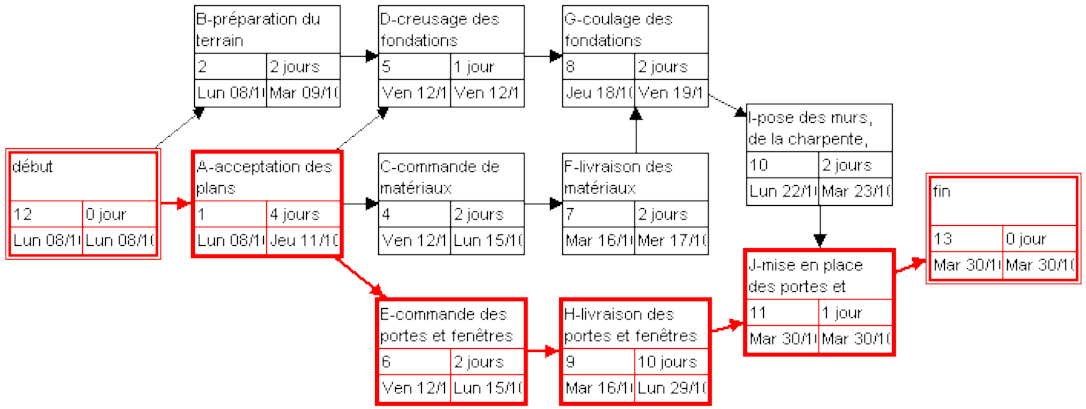
Dealing with complexity: Abstraction

- We use the term **model** to refer to any abstraction of the system
- 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 ?

UML

Dealing with complexity: Abstraction

- Task Model
  - PERT Chart: What are the dependencies between the tasks?



  - Schedule: How can this be done within the time limit?

Dealing with complexity: Abstraction

- Model-based software Engineering
- Code is derivation of object model

Problem statement: a stock exchange lists many companies. Each company is identified by a ticker symbol

Analysis phase results in object model (UML Class Diagram):

Implementation phase results in code

```
public class StockExchange
{
      public ArrayList m_Company = new Company();
};
public class Company
{
      private int m_tickerSymbol
};
```

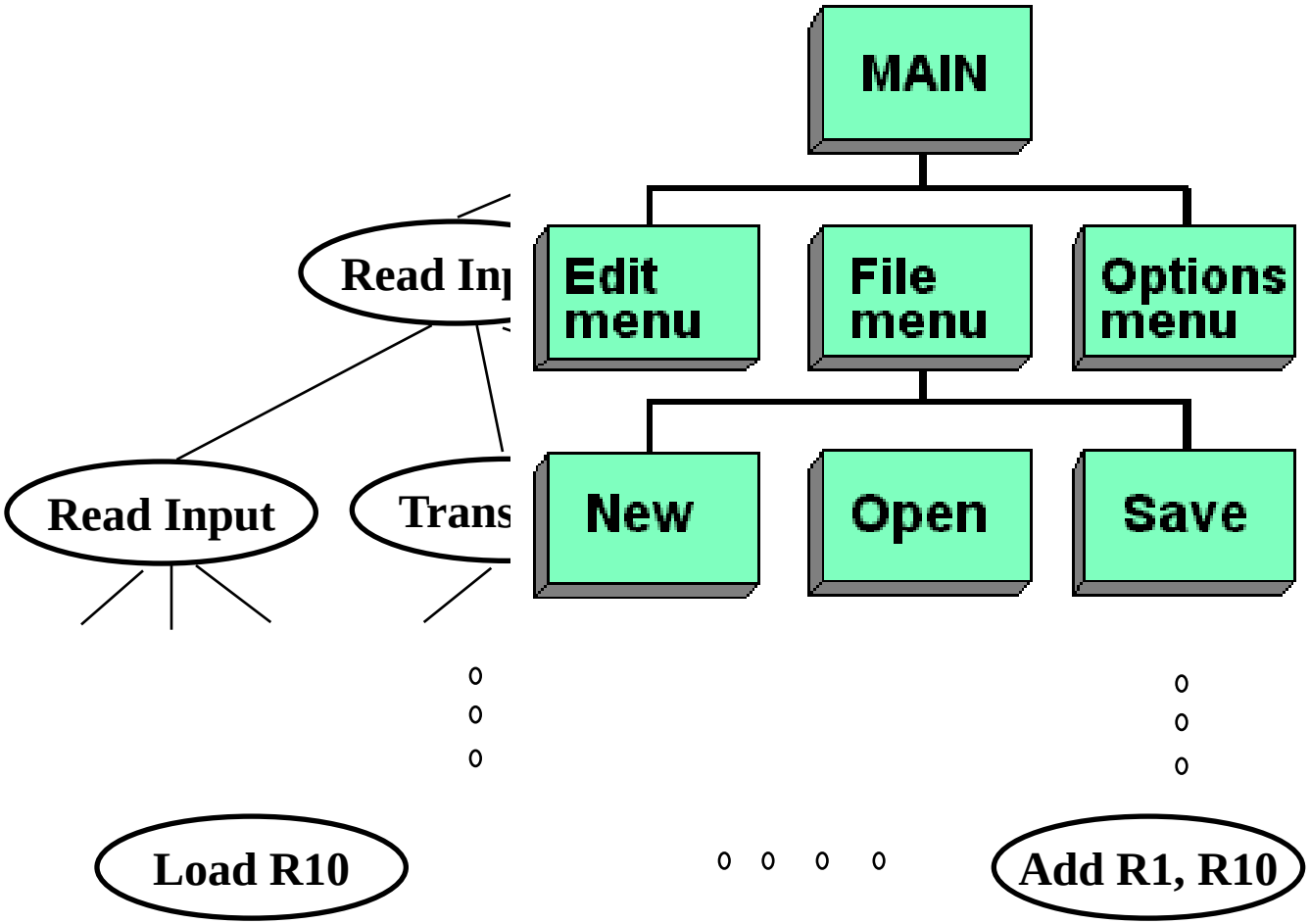
Dealing with complexity: Decomposition

A technique used to master complexity ("divide and conquer")

- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major processing step (function) in the application domain
  - Modules can be decomposed into smaller modules
- Object-oriented decomposition
  - The system is decomposed into classes ("objects")
  - Each class is a major abstraction in the application domain
  - Classes can be decomposed into smaller classes

Which decomposition is the right one?

Functional decomposition



Top Level functions

Level 1 functions

Level 2 functions

Machine Instructions

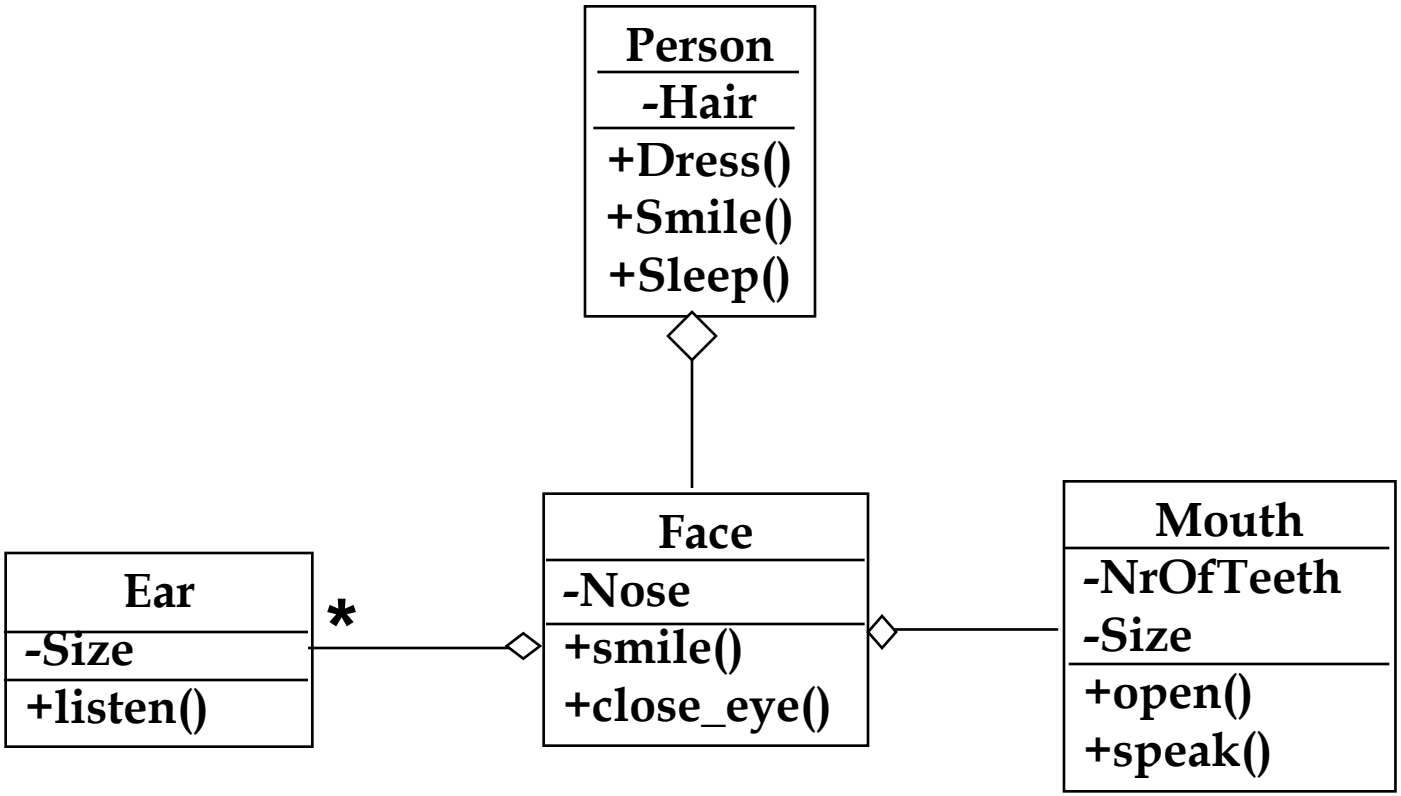
Functional decomposition

- Functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - Codes are hard to understand
  - Code that is complex and impossible to maintain
  - User interface is often awkward and non-intuitive

Object-oriented decomposition

- Object-oriented decomposition needs Class identification
- Basic assumption:
  1. We can find the classes for a new software system
  2. We can identify the classes in an existing system
  3. We can create a class-based interface to any system

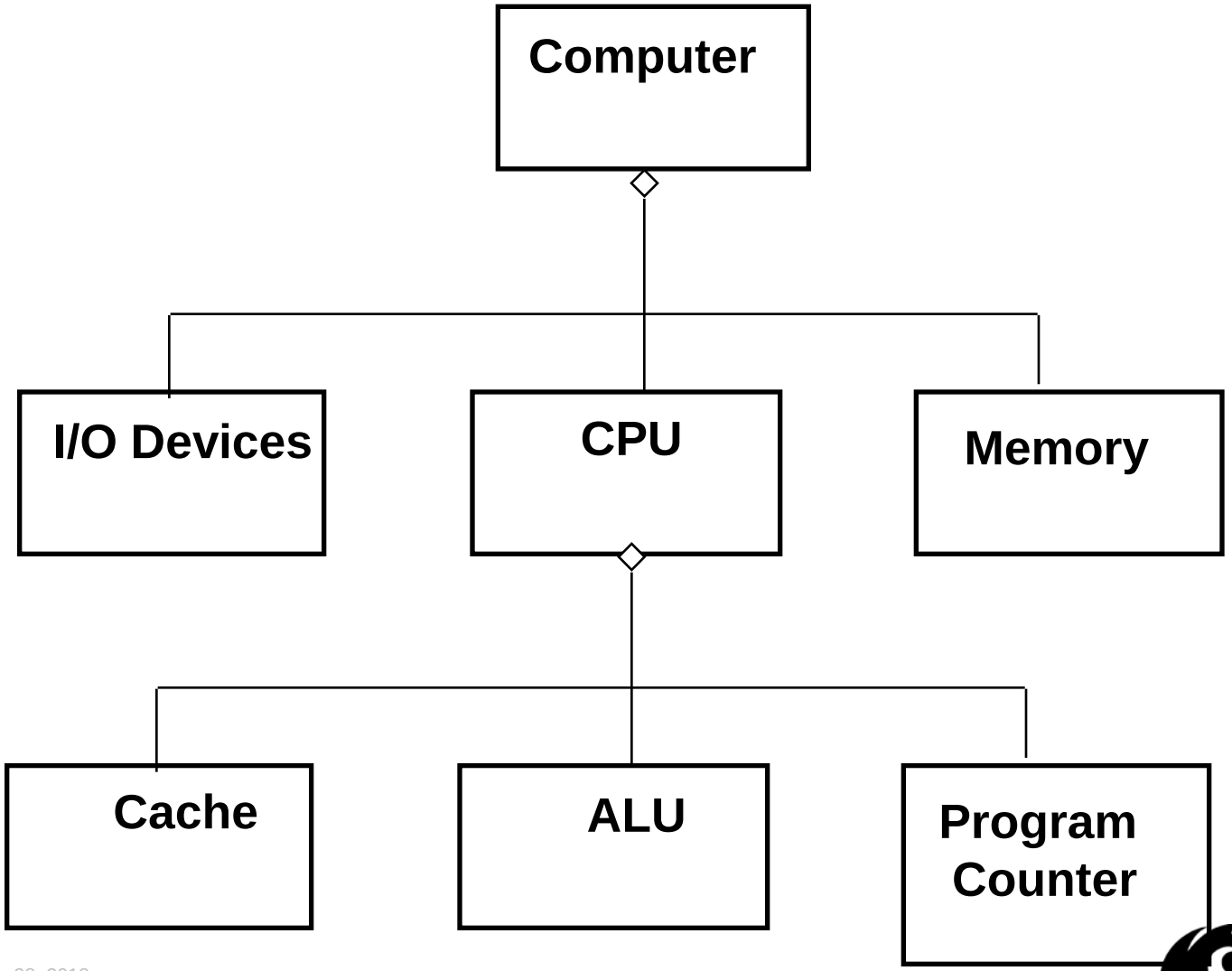
Object-oriented decomposition



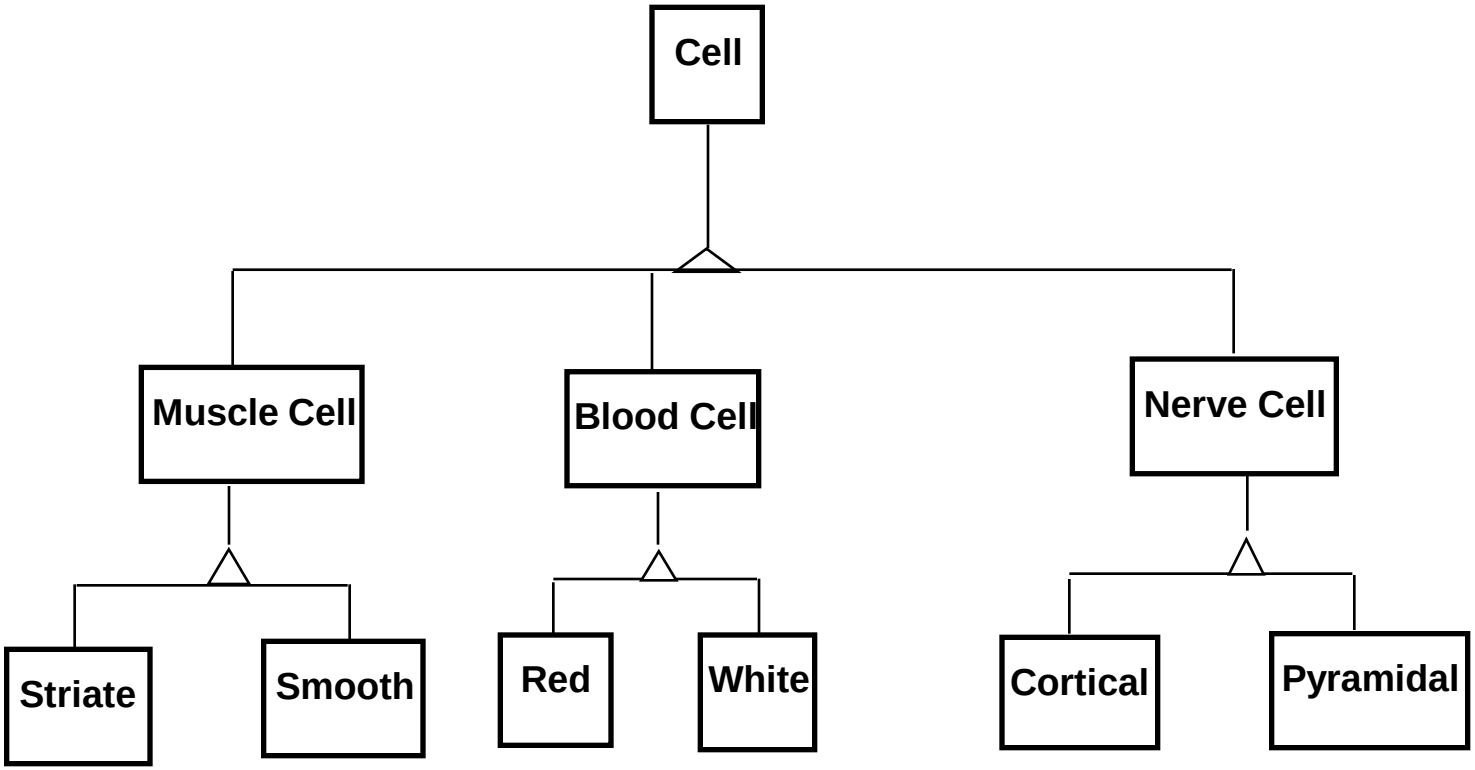
Dealing with complexity: Hierarchies

- 2 important hierarchies
  - Part-of
  - Is-Kind-of

Part-of Hierarchy



Is-Kind-of Hierarchy



So? Our current approach

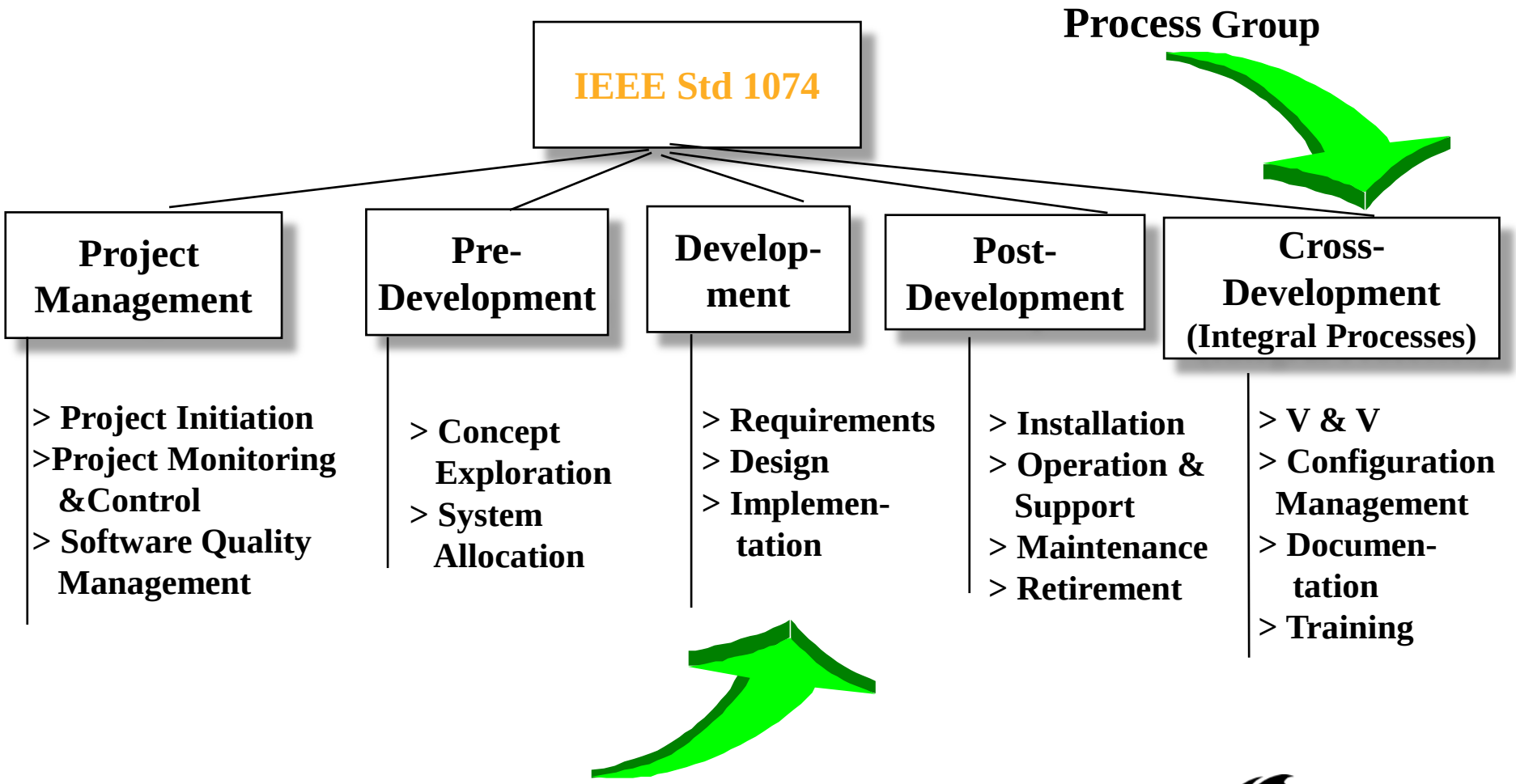
- Start with a description of the functionality (Use case diagram, Activity diagram), then proceed to the object model (Class diagram, Sequence diagram)
- This leads us to the software lifecycle

System/Software Lifecycle

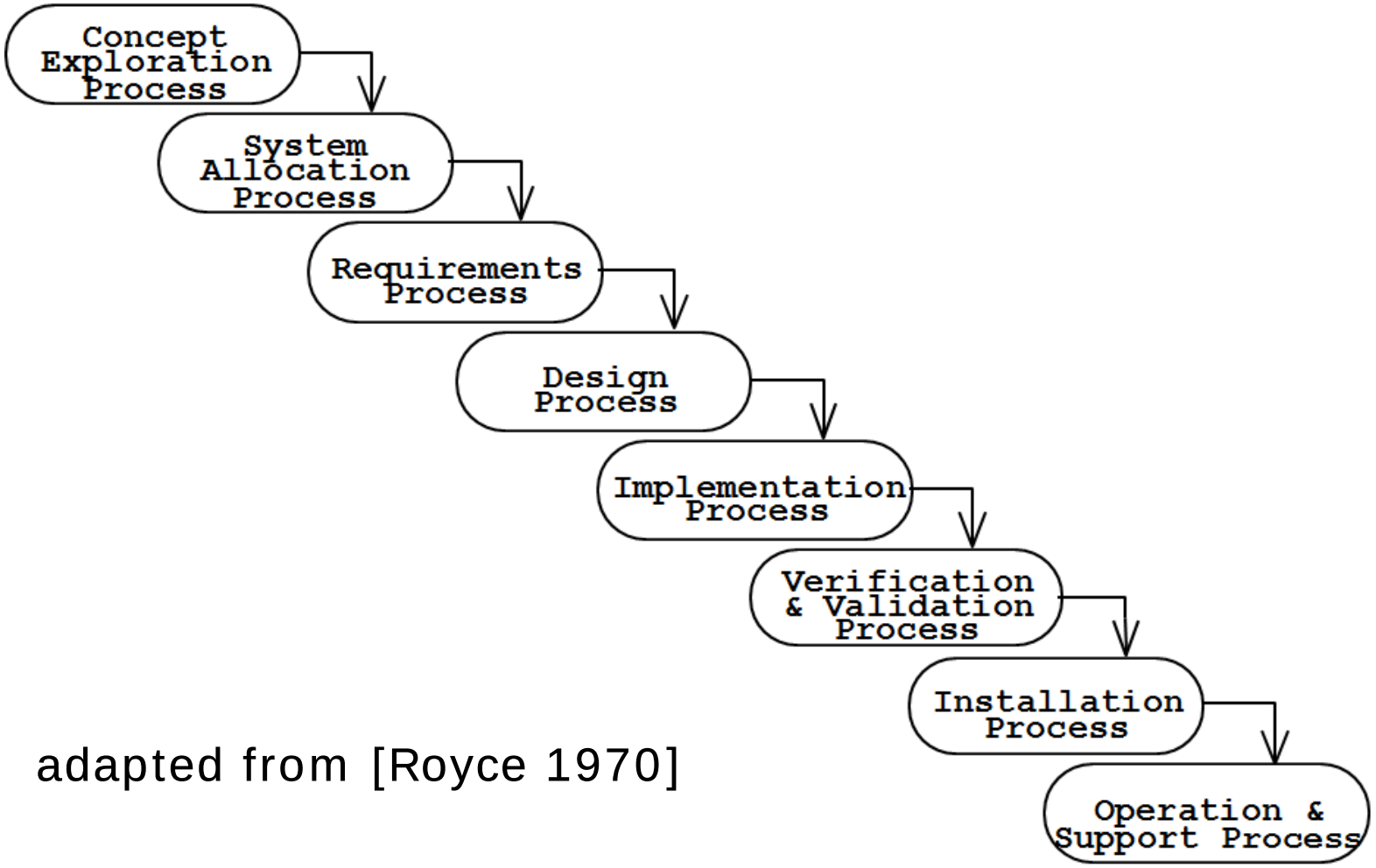
Software lifecycle

- Definition: Set of activities and their relationships to each other to support the development of a software system
- Typical Software Activities
  - Requirements Elicitation
  - Requirements Analysis
  - System Design
  - Object Design
  - Implementation
  - ~~Testing~~ Not in SENG2130
  - Deployment
  - Post Delivery Maintenance

IEEE Std 1074: Standard for Software lifecycle Activities

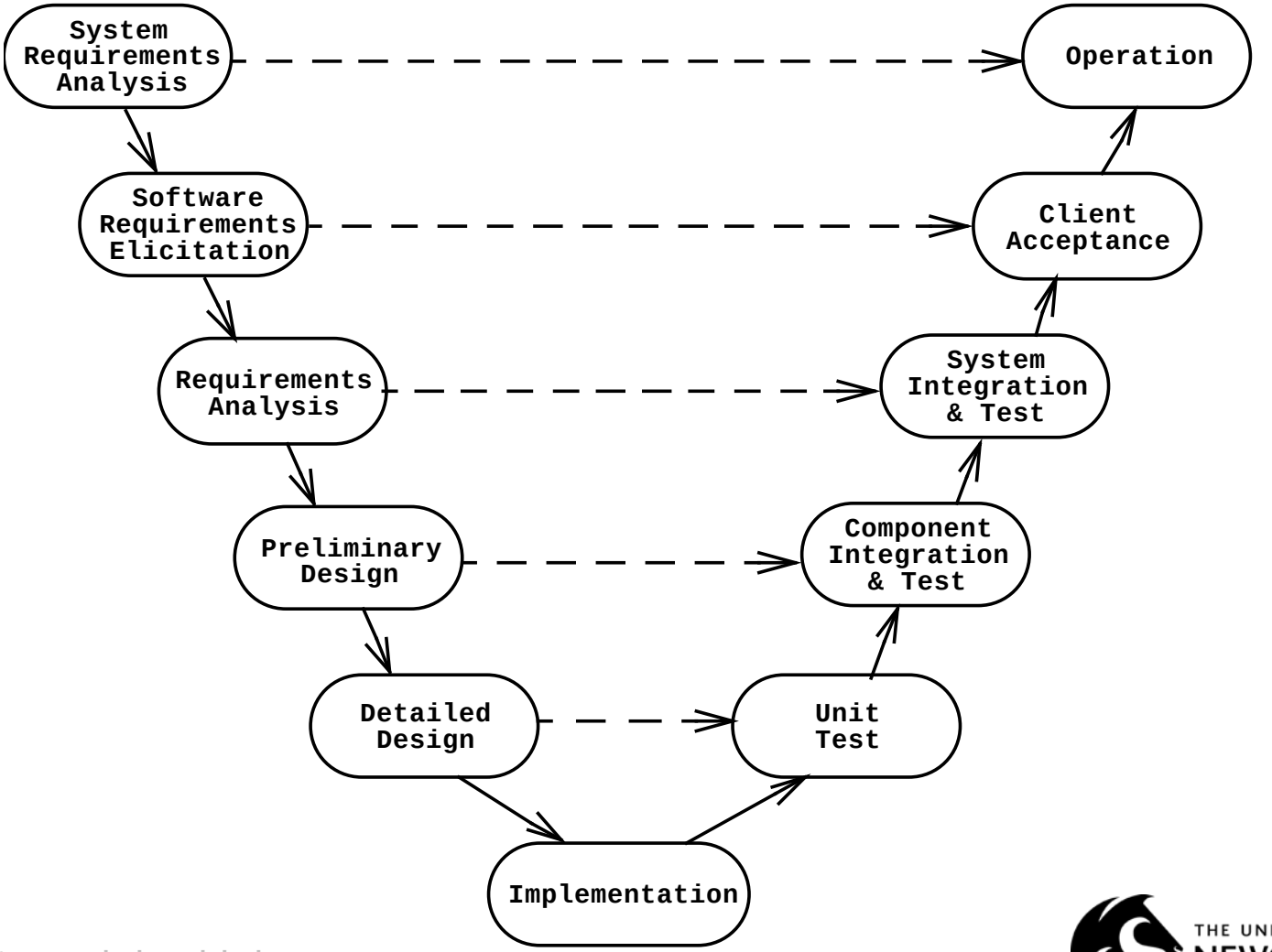


Software lifecycle model: Waterfall



adapted from [Royce 1970]

Software lifecycle model: V model



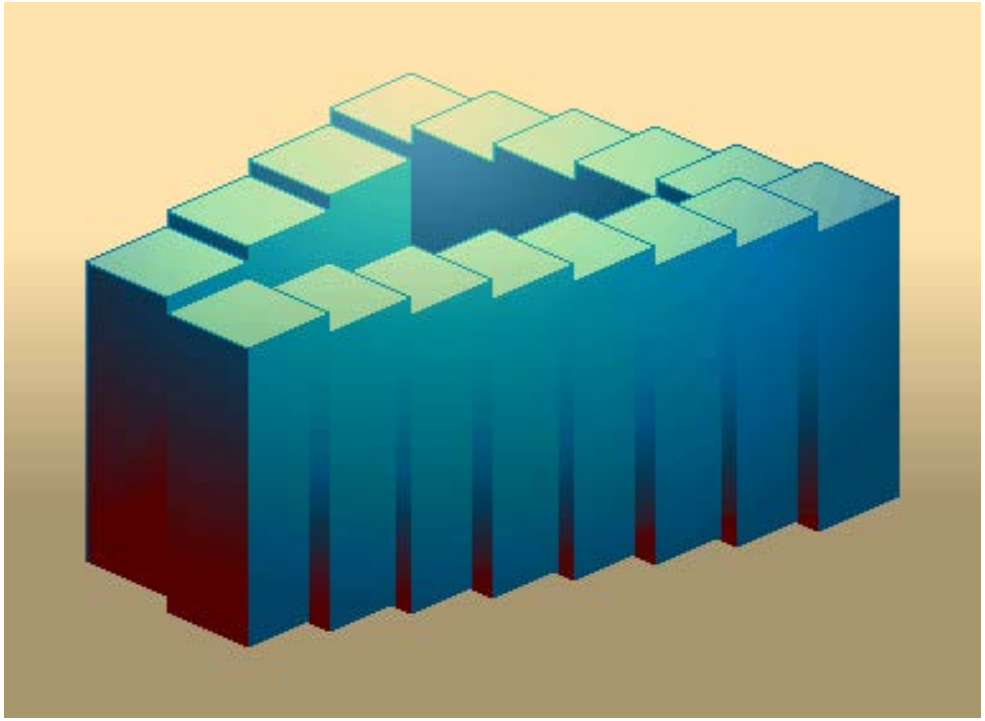
Properties of Waterfall and V Models

- Managers love waterfall models
  - Nice milestones
  - No need to look back (linear system)
  - Always one activity at a time
  - Easy to check progress during development: 90% coded, 20% tested
- However, software development is non-linear
  - While a design is being developed, problems with requirements are identified
  - While a program is being coded, design and requirement problems are found
  - While a program is tested, coding errors, design errors and requirement errors are found.

The Alternative: Allow Iteration

Escher was the first:-)

See his famous waterfall drawing on
http://en.wikipedia.org/wiki/File:Escher_Waterfall.jpg

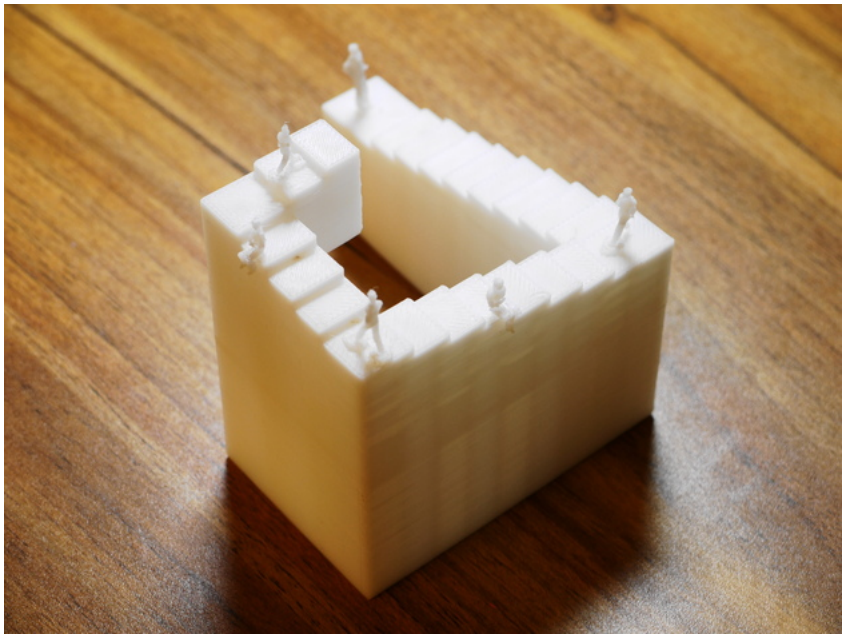


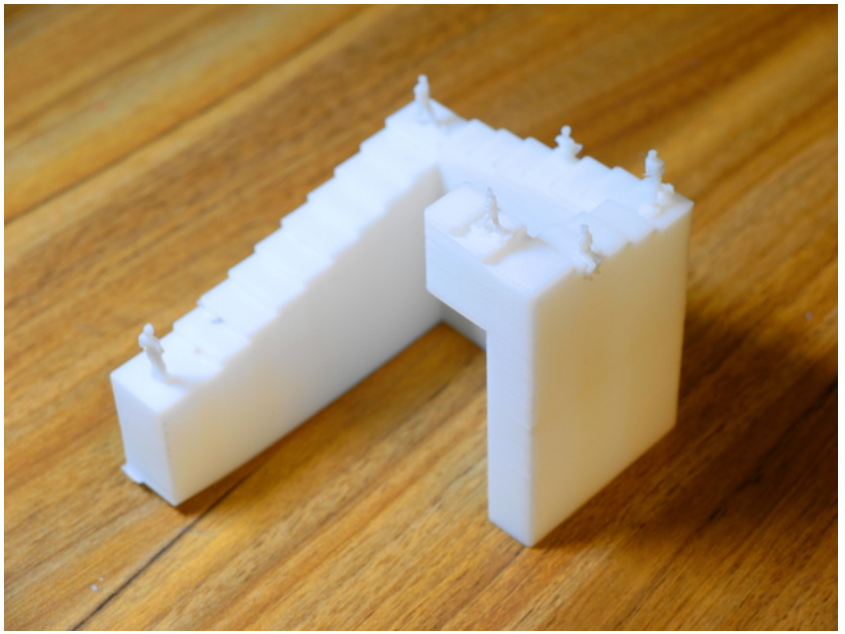
Source: http://farm4.static.flickr.com/3243/2967961102_250e4bdecb.jpg

The Alternative: Allow Iteration

Escher was the first:-)

Images courtesy of
http://www.thingiverse.com/thing:210267

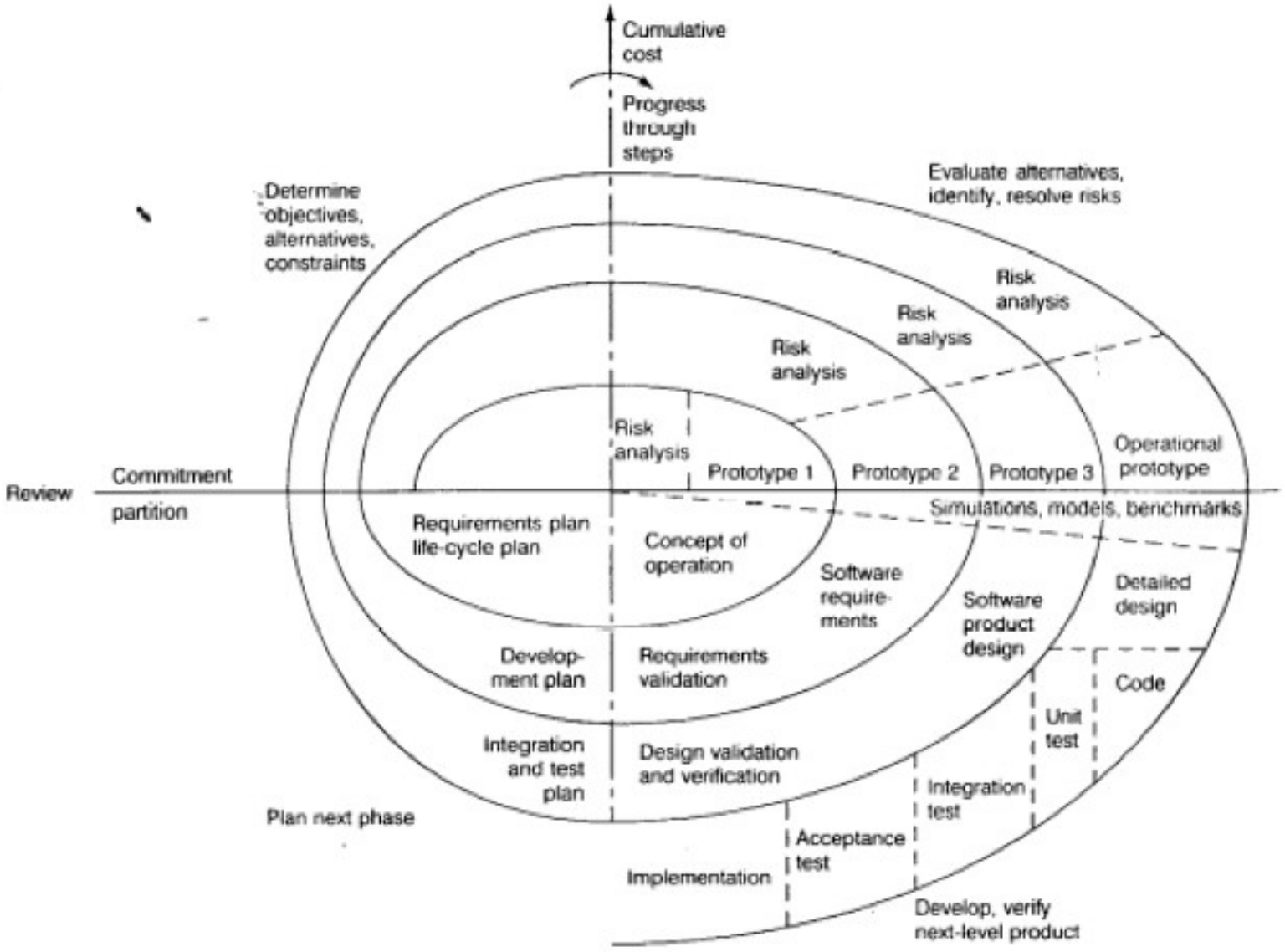




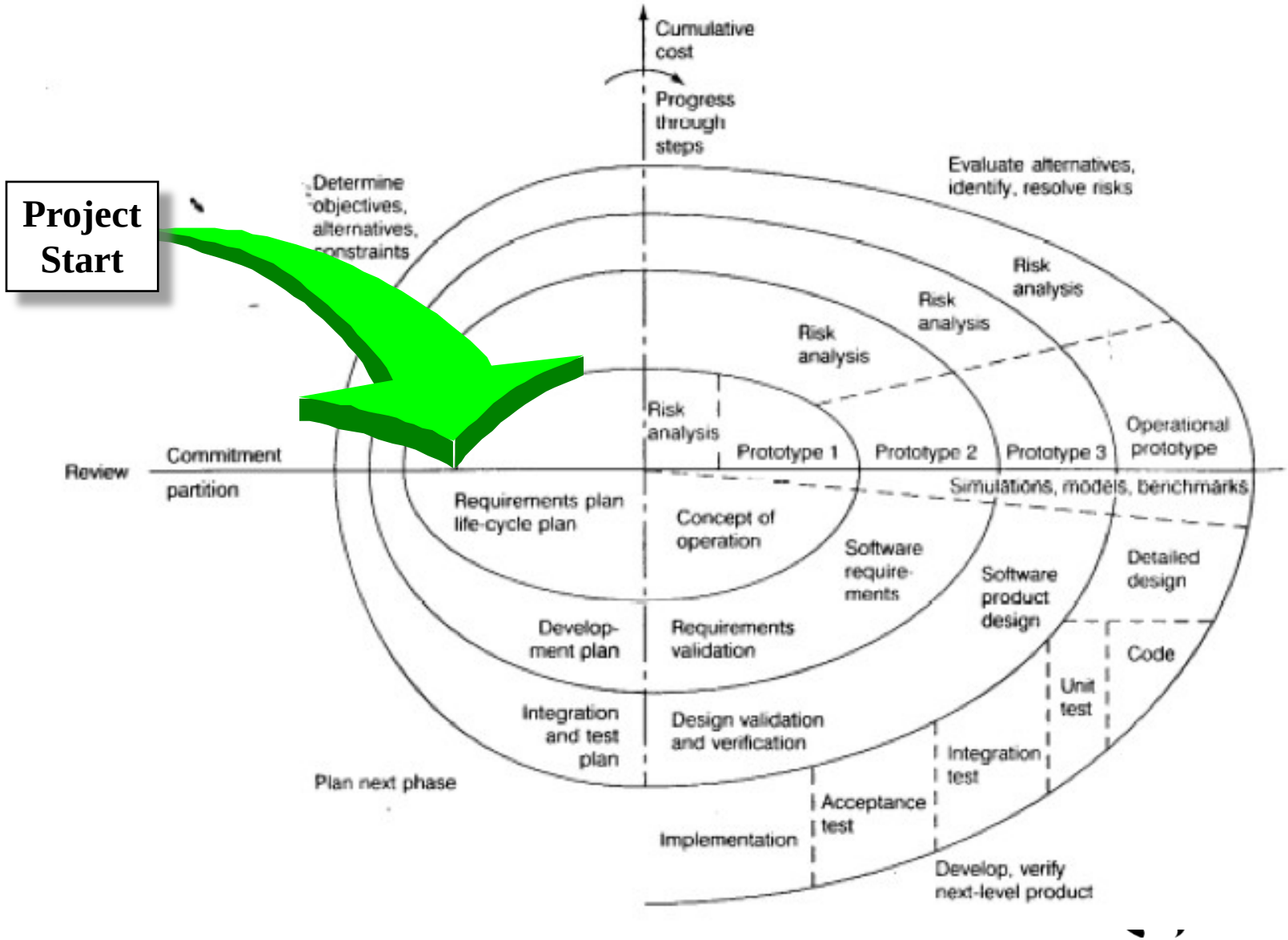
Software lifecycle model: Spiral

- The spiral model proposed by Boehm has the following set of 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
  - If a risk has successfully been resolved, evaluate the results of the round and plan the next round
  - If a certain risk cannot be resolved, terminate the project immediately
- This set of activities is applied to a couple of so-called rounds.

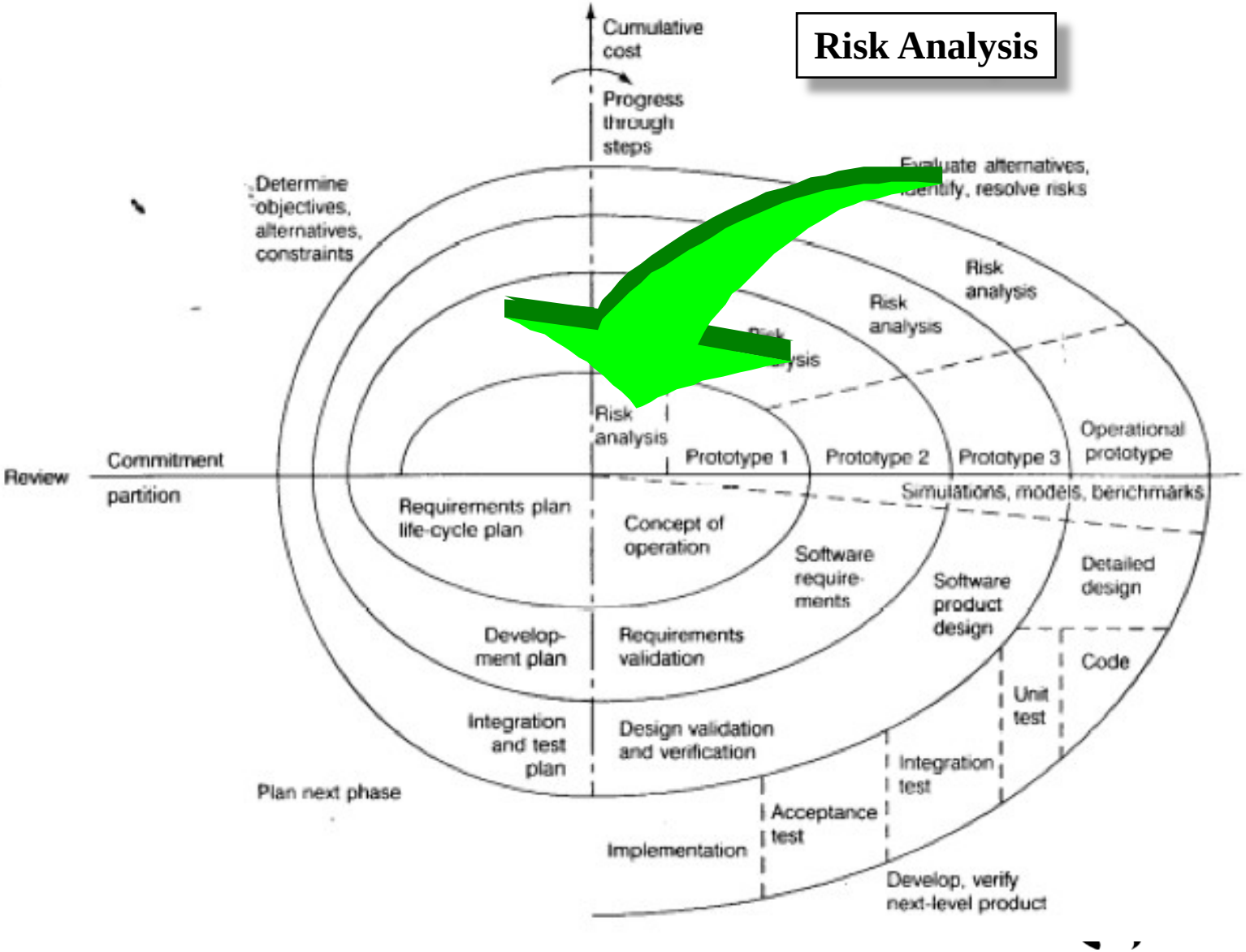
Diagram of Boehm's Spiral Model



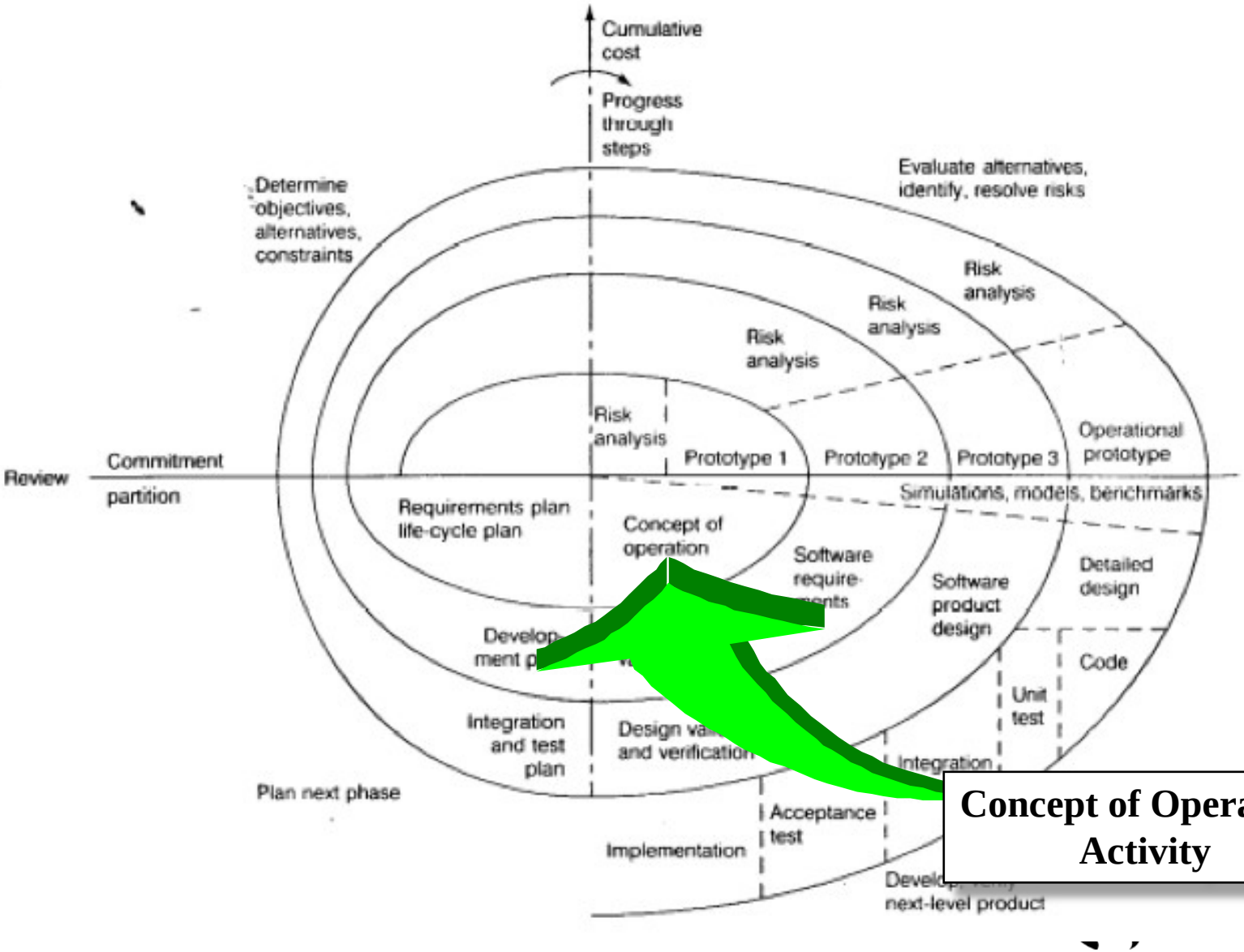
Round 1, Concept of Operations, Quadrant IV: Determine Objectives,Alternatives & Constraints



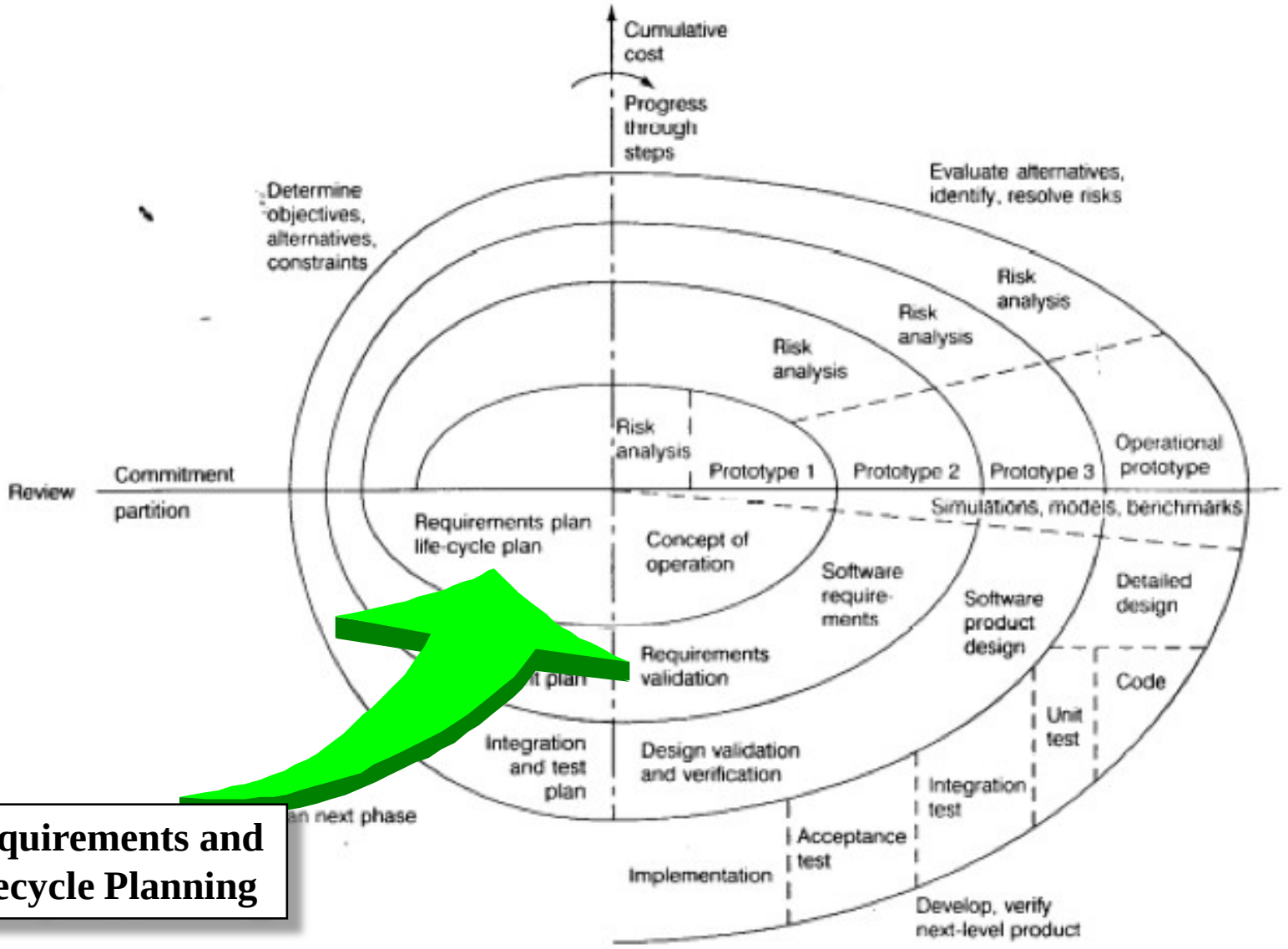
Round 1, Concept of Operations, Quadrant I: Evaluate Alternatives, identify & resolve Risks



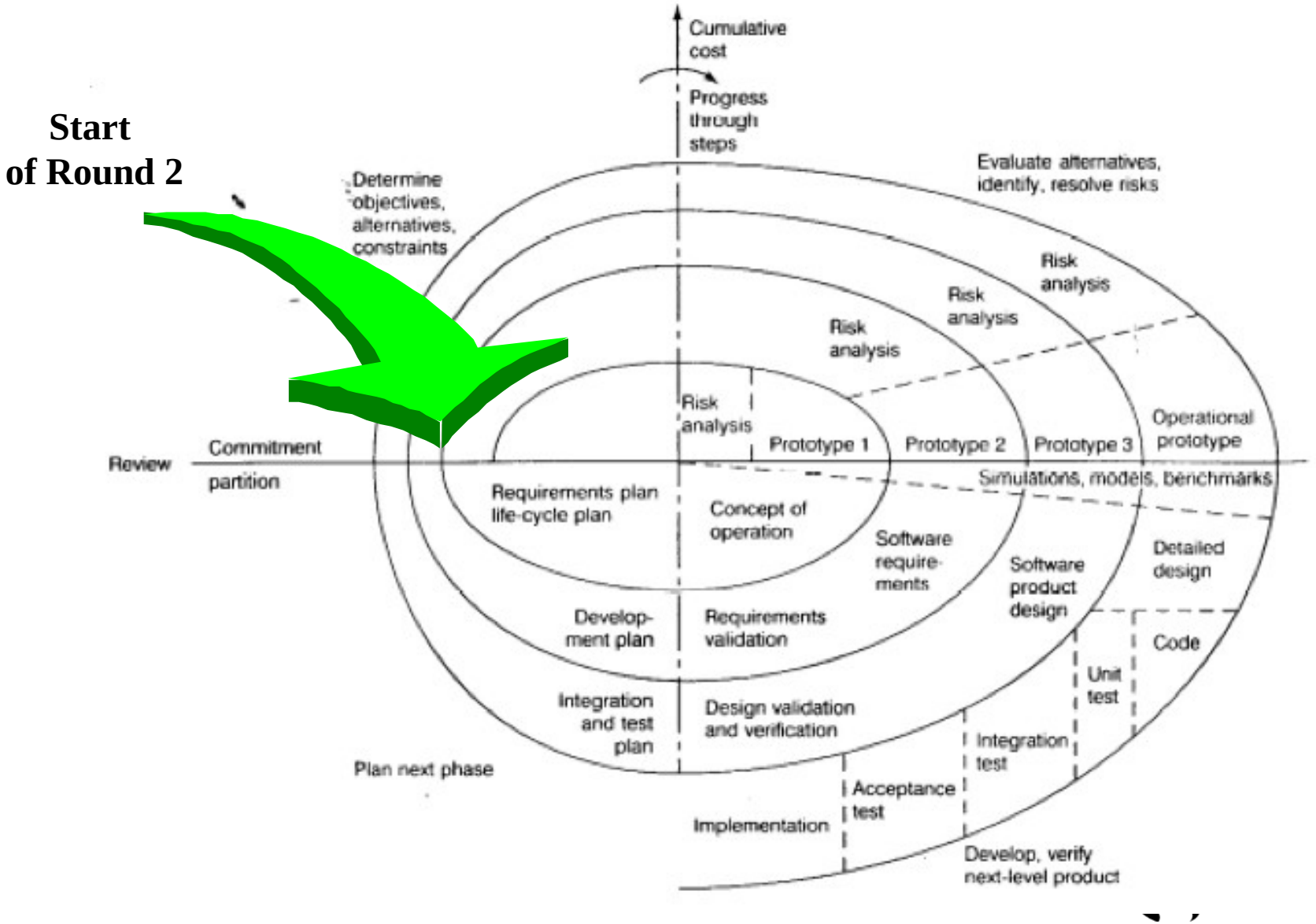
Round 1, Concept of Operations, Quadrant II: Develop and Verify



Round 1, Concept of Operations, Quadrant III: Prepare for Next Activity

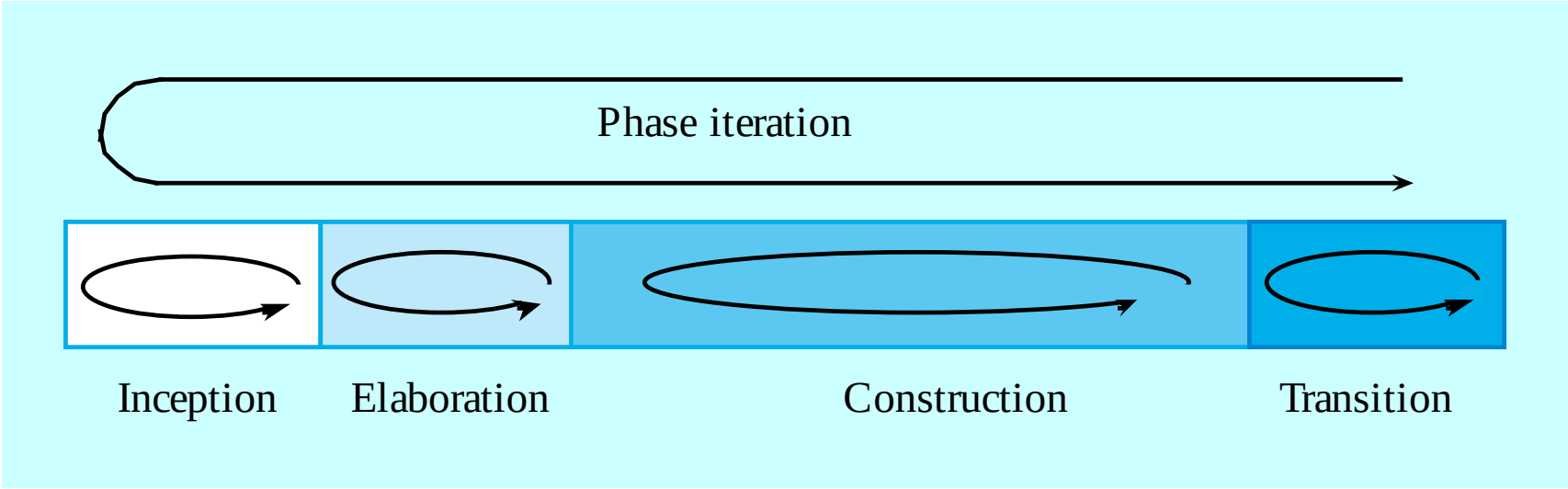


Round 2, Software Requirements, Quadrant IV: Determine Objectives,Alternatives & Constraints

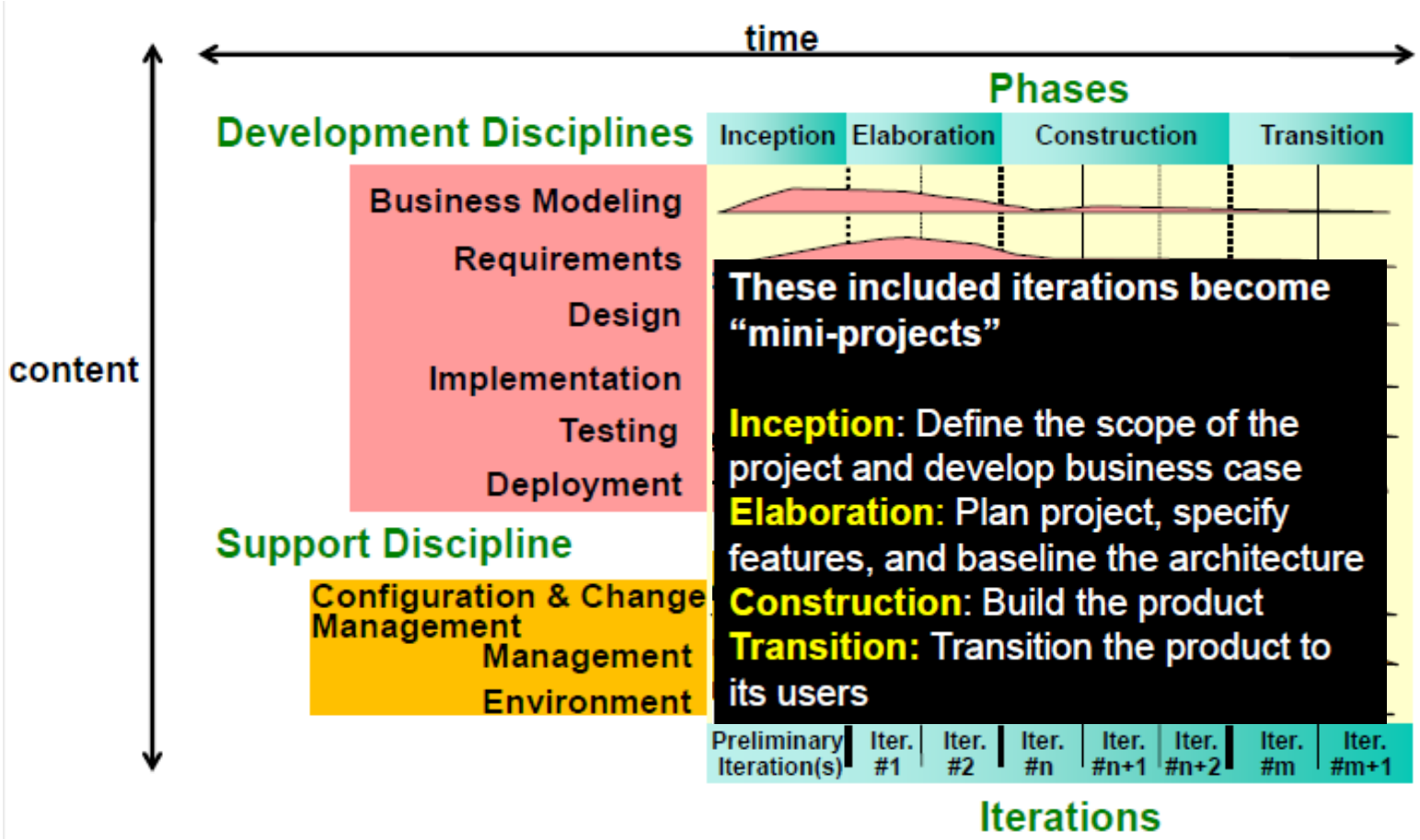


Software lifecycle model: Unified Process

- It is an adaptive approach
- Iterative and incremental software development process framework
- A dynamic perspective that shows phases over time



Software lifecycle model: Unified Process



Limitations of linear (Waterfall, V) and Iterative (Spiral, UP) Models

- None of these models deal well with frequent change
  - The linear models assume that once you are done with a phase, all issues covered in that phase are closed and cannot be reopened
  - The iterative models can deal with change between phases, but does not allow change within a phase
- What do you do if change is happening more frequently?
  - “The only constant is the change”

The Problem is how to Deal with Change and uncertainty

- Controlling Software Development with a Process through agility (Ken Schwaber: https://www.youtube.com/watch?v=8WXT7_cHsXI)
  - Large parts of software development is empirical in nature; they cannot be modeled with a defined process
  - There is a difference between defined and empirical process

Software lifecycle model: Agile

- Classical software development methodologies have some disadvantages:
  - Huge effort during the planning phase
  - Poor requirements conversion in a rapid changing environment
  - Treatment of staff as a factor of production

- Agile Software Development Methodologies
  - Minimize risk → short iterations
  - Real-time communication (preferable face-to-face) → very little written documentation
  - www.agilealliance.org

Software lifecycle model: Agile

- The process is imperfectly defined, not all pieces of work are completely understood
- Deviations are seen as opportunities that need to be investigated
  - The empirical process “expects the unexpected”
- Control is exercised through frequent inspection
- Conditions when to apply this model:
  - Frequent change, unpredictable and unrepeatable outputs.

Software lifecycle model: Agile - Scrum

- Definition (Rugby): A Scrum is a way to restart the game after an interruption,
  - The forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is tossed in among them
- Definition (Software Development): Scrum is an agile, lightweight process
  - To manage and control software and product development with rapidly changing requirements
  - Based on improved communication and maximizing cooperation.

Why Scrum?

Traditional methods are like relay races

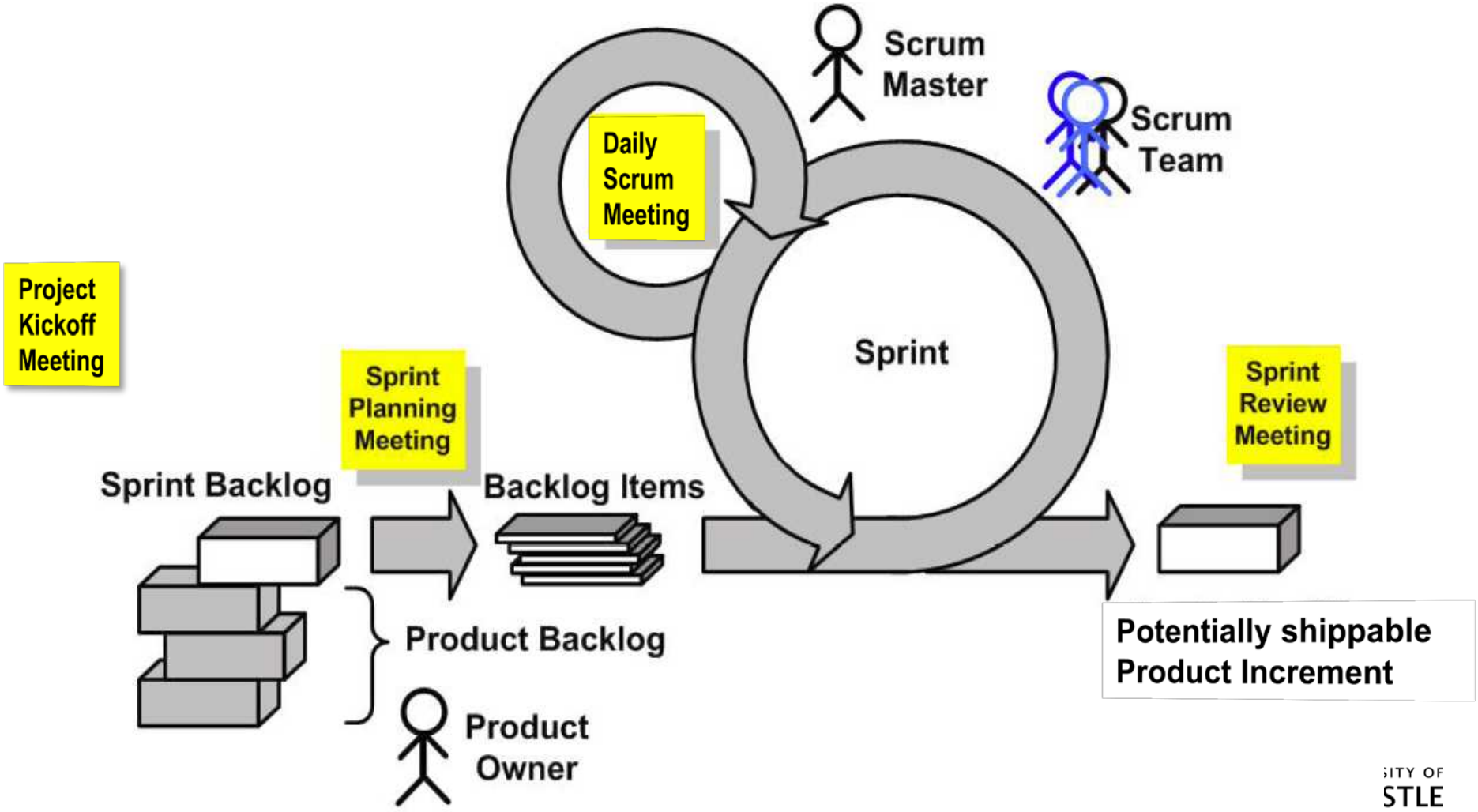
Agile methods are like rugby

Image sources: http://en.wikipedia.org/wiki/File:Relay_race_baton_pass.jpg
http://upload.wikimedia.org/wikipedia/commons/b/bf/Rugby_ST.F-ST.T_27022007-19.JPG
For reuse see http://commons.wikimedia.org/wiki/Commons:Reusing_content_outside_Wikimedia

Components of Scrum

- Scrum Roles
  - Scrum Master, Scrum Team, Product Owner
- Process
  - Kickoff Meeting
  - Sprint Planning Meeting
  - Sprint (~~ Iteration in a Unified Process)
  - Daily Scrum Meeting
  - Sprint Review Meeting
- Scrum Artifacts
  - Product Backlog, Sprint Backlog, Backlog Items
  - Potentially shippable Product

Overview of Scrum



Scrum

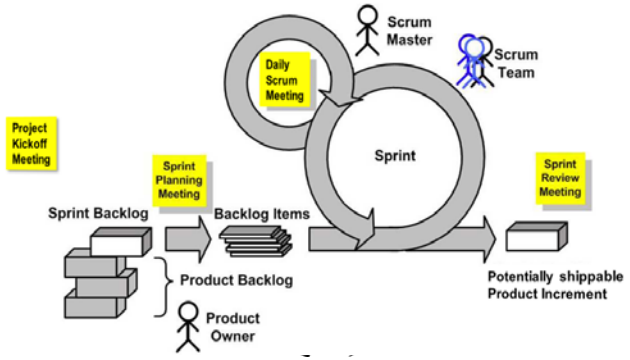
- Scrum master
  - Represents management to the project
  - Typically filled by a project manager or team leader
- The scrum team
  - Typically 5-6 people
  - Cross-functional (programmers, UI designers, etc.)
- Product Owner
  - Knows what needs to be build and in what sequence this should be done

Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting

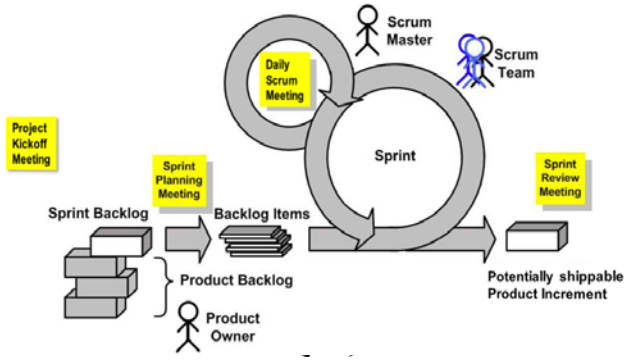
Scrum Process Activities

- Project-Kickoff Meeting
  - A collaborative meeting in the beginning of the project
    - Participants: Product Owner, Scrum Master
    - Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)
  - Goal: Create the Product Backlog
- Sprint Planning Meeting
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting



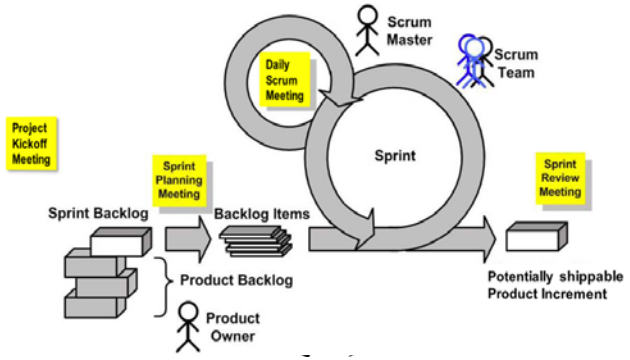
Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
  - A collaborative meeting in the beginning of each Sprint
    - Participants: Product Owner, Scrum Master and Scrum Team
    - Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)
  - Goal: Create the Sprint Backlog
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting



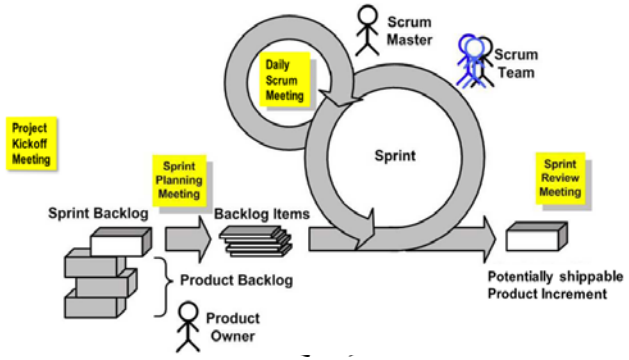
Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
- Sprint
  - A month-long iteration, during which is incremented a product functionality
  - No outside influence can interfere with the Scrum team during the Sprint
  - Each day in a Sprint begins with the Daily Scrum Meeting
  - ~~ Iteration in a Unified Process
- Daily Scrum Meeting
- Sprint Review Meeting



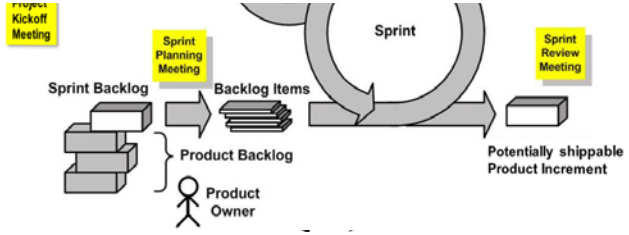
1. Status:
   What did I do since the last Scrum meeting?
2. Issues:
   What is stopping me getting on with the work?
3. Action items:
   What am I doing until the next Scrum meeting?

- Daily Scrum Meeting
  - Is a short (15 minutes long) meeting, which is held every day before the Team starts working
  - Participants:
    - Scrum Master (which is the chairman), Scrum Team
  - Every Team member should answer on 3 questions
- Sprint Review Meeting



Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting
  - At the end of each sprint, a sprint review meeting is held.
  - Participants: Product Owner, Scrum Master and Scrum Team
  - During this meeting the scrum team shows which scrum product backlog items they completed during the sprint.
  - This might take place in the form of a demo of the new features.



Summary

- Software engineering
  - It is to support software production in order to deliver software that is “fit for purpose”.
  - Dealing with complexity
    - Abstraction: =Model
      - Object Model: It defines the objects and their relationships
      - Functional Model: It defines the functions and data flowing through the system
      - Dynamic Model: it defines how the system react to external event and how the event flow in the system
    - Decomposition: divide and conquer
      - Functional decomposition: the system is decomposed into modules and each module is a major processing step (function) in the application domain
      - Object-oriented decomposition: the system is decomposed into classes (“objects”) and each class is a major abstraction in the application domain
    - Hierarchy: part-of, is-kind-of

Summary

- Our approach is to start with a description of the functionality then proceed to the object model
- Software lifecycle
  - Software lifecycle activities:
    - requirements elicitation
    - requirement analysis
    - system design
    - object design
    - implementation
    - testing
    - deployment
    - post delivery maintenance
  - Software lifecycle models: Waterfall/V model, Spiral, Unified Process, Agile

Next week

- Requirements
- What is UML?
- A more detailed view on
  - Use case diagrams
  - Activity diagrams (week 3)
  - Class diagrams (week 4)
  - Sequence diagrams (week 5)