Software Engineering

Introduction to Software Engineering

Assoc. Prof. Marenglen Biba
MSc in Computer Science, UoG-UNYT
Foundation Programme

Material

- Slides
  - http://www.marenglenbiba.net/foundprog-se/
  - Sufficient for FP exam purposes
- Reference book
  - *B. Bruegge & A. H. Dutoit.* Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition. (in library)
- Other useful material
  - I. Sommerville. Software Engineering (in library)
  - R. Pressman. Software Engineering: A Practitioner's Approach (in library)
  - *B. Bruegge & A. H. Dutoit.* Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition. (in library)

Intro to Software Engineering

- The Software Engineering Discipline
- The Software Process

Why Software Engineering?

- The new Airbus A380 uses a substantial amount of software to create a "paperless“ cockpit.
- Software engineering successfully maps and plans the millions of lines of code comprising the plane's software

Software Disasters

- Software errors cost the U.S. economy $60 billion annually in rework, lost productivity and actual damages.

- We all know software bugs can be annoying, but faulty software can also be expensive, embarrassing, destructive and deadly.

Software Disasters

Mariner Bugs Out (1962)

- **Cost:** $18.5 million
- **Disaster:** The Mariner 1 rocket with a space probe headed for Venus diverted from its intended flight path shortly after launch. Mission Control destroyed the rocket 293 seconds after liftoff.
- **Cause:** A programmer incorrectly transcribed a handwritten formula into computer code, missing a single superscript bar. Without the smoothing function indicated by the bar, the software treated normal variations of velocity as if they were serious, causing faulty corrections that sent the rocket off course.

Software Disasters

Ariane Rocket Goes Boom (1996)

- **Cost:** $500 million
- **Disaster:** Ariane 5, Europe's newest unmanned rocket, was intentionally destroyed seconds after launch on its maiden flight. Also destroyed was its cargo of four scientific satellites to study how the Earth's magnetic field interacts with solar winds.
- **Cause:** Shutdown occurred when the guidance computer tried to convert the sideways rocket velocity from 64-bits to a 16-bit format. The number was too big, and an overflow error resulted. When the guidance system shut down, control passed to an identical redundant unit, which also failed because it was running the same algorithm.

Software Disasters

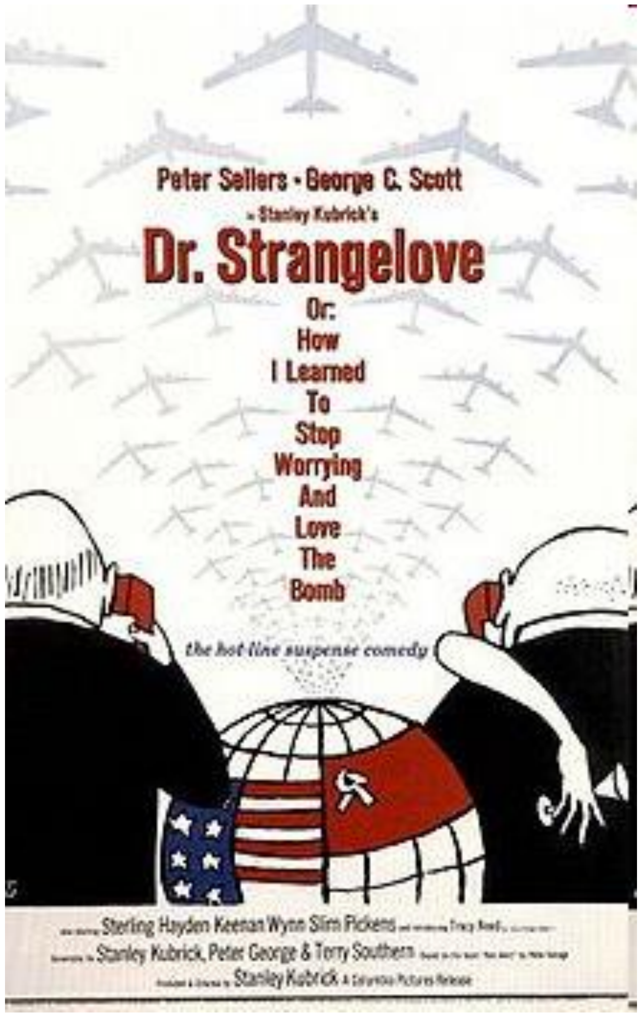
Hartford Coliseum Collapse (1978)

- **Cost:** $70 million, plus another $20 million damage to the local economy
- **Disaster:** Just hours after thousands of fans had left the Hartford Coliseum, the steel-latticed roof collapsed under the weight of wet snow.
- **Cause:** The programmer of the CAD software used to design the coliseum incorrectly assumed the steel roof supports would only face pure compression. But when one of the supports unexpectedly buckled from the snow, it set off a chain reaction that brought down the other roof sections like dominoes.

Software Disasters

World War III… Almost (1983)

- **Cost:** Nearly all of humanity
- **Disaster:** The Soviet early warning system falsely indicated the United States had launched five ballistic missiles. Fortunately the Soviet duty officer had a “funny feeling in my gut” and reasoned if the U.S. was really attacking they would launch more than five missiles, so he reported the apparent attack as a false alarm.
- **Cause:** A bug in the Soviet software failed to filter out false missile detections caused by sunlight reflecting off cloud-tops.

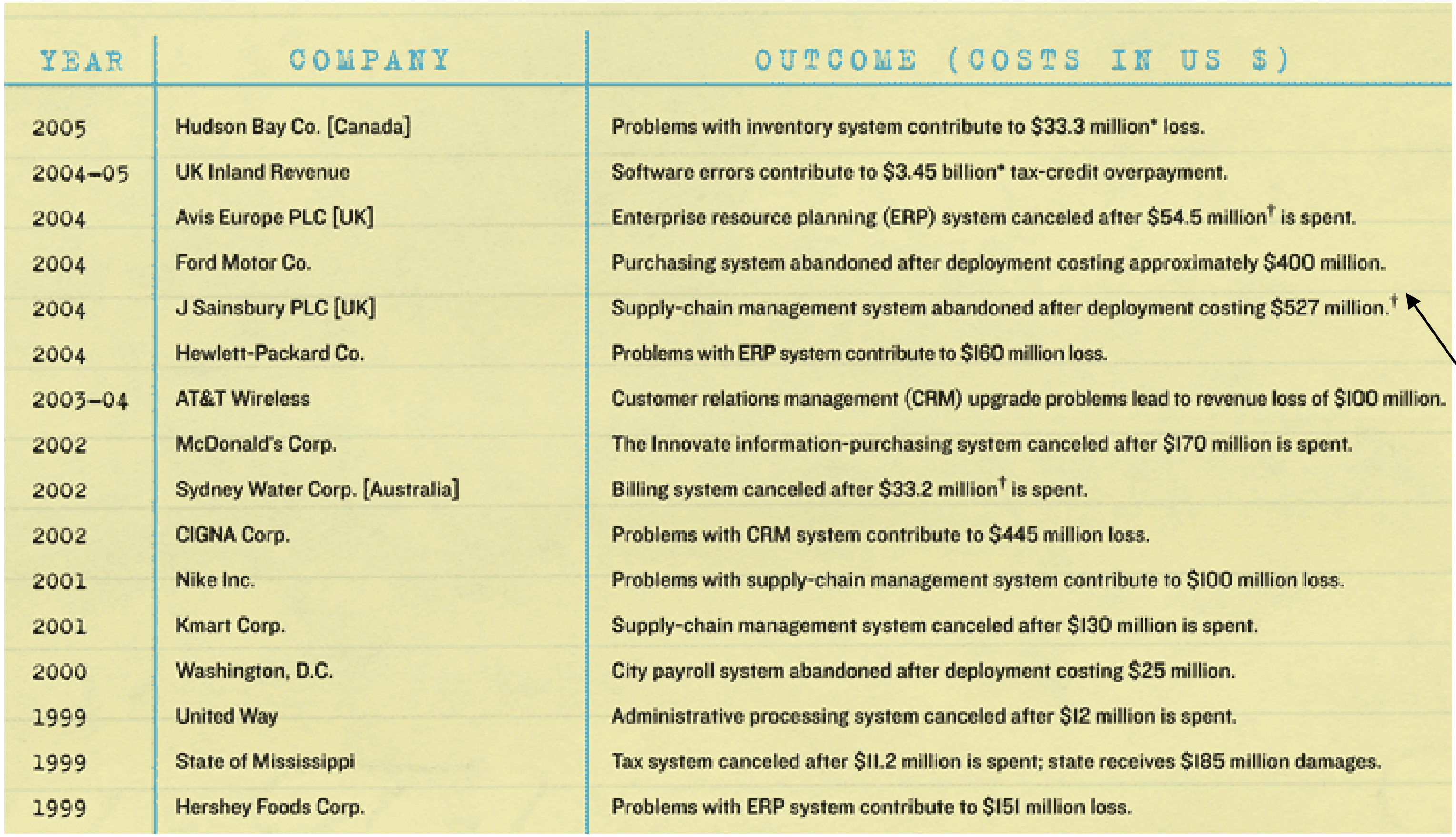

Peter Sellers · George C. Scott
Stanley Kubrick's
Dr. Strangelove
Or: How I Learned To Stop Worrying And Love The Bomb
the hot-line suspense comedy
Sterling Hayden Keenan Wynn Slim Pickens
Stanley Kubrick, Peter George & Terry Southern
Stanley Kubrick

Software Disasters

Wall Street Crash (1987)

- **Cost:** $500 billion in one day
- **Disaster:** On “Black Monday” (October 19, 1987), the Dow Jones Industrial Average plummeted 508 points, losing 22.6% of its total value. The S&P 500 dropped 20.4%. This was the greatest loss Wall Street ever suffered in a single day.
- **Cause:** A long bull market was halted by a rash of SEC investigations of insider trading and by other market forces. As investors fled stocks in a mass exodus, computer trading programs generated a flood of sell orders, overwhelming the market, crashing systems and leaving investors effectively blind.

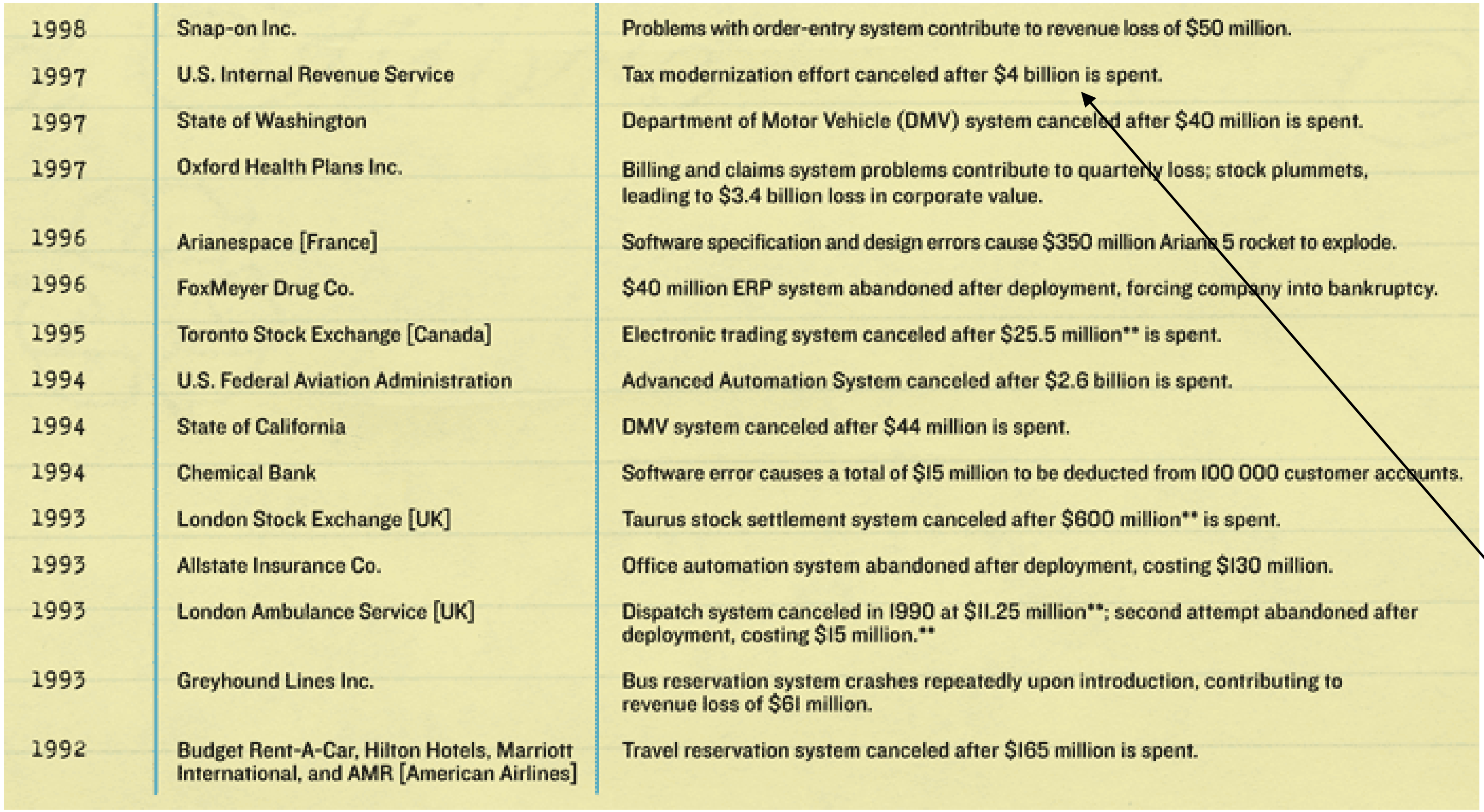
Software Engineering Disasters



YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system canceled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.

Oops ☺

Software Engineering Disasters



1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.
1996	Arianespace [France]	Software specification and design errors cause $350 million Ariane 5 rocket to explode.
1996	FoxMeyer Drug Co.	$40 million ERP system abandoned after deployment, forcing company into bankruptcy.
1995	Toronto Stock Exchange [Canada]	Electronic trading system canceled after $25.5 million** is spent.
1994	U.S. Federal Aviation Administration	Advanced Automation System canceled after $2.6 billion is spent.
1994	State of California	DMV system canceled after $44 million is spent.
1994	Chemical Bank	Software error causes a total of $15 million to be deducted from 100 000 customer accounts.
1993	London Stock Exchange [UK]	Taurus stock settlement system canceled after $600 million** is spent.
1993	Allstate Insurance Co.	Office automation system abandoned after deployment, costing $130 million.
1993	London Ambulance Service [UK]	Dispatch system canceled in 1990 at $11.25 million**; second attempt abandoned after deployment, costing $15 million.**
1993	Greyhound Lines Inc.	Bus reservation system crashes repeatedly upon introduction, contributing to revenue loss of $61 million.
1992	Budget Rent-A-Car, Hilton Hotels, Marriott International, and AMR [American Airlines]	Travel reservation system canceled after $165 million is spent.

Oops ☺

Software Engineering in Economy

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Origins

- The term *software engineering* first appeared in the 1968 **NATO Software Engineering Conference** and was meant to provoke thought regarding the current "software crisis" at the time.

- Since then, it has continued as a profession and **field of study dedicated to creating software** that is of higher quality, more affordable, maintainable, and quicker to build.

- Although it is questionable what impact it has had on actual software development over the last more than 40 years, the field's future looks bright.
  - According to important rating companies who have rated "software engineering" as the **best job** in America in different years.
  - http://money.cnn.com/magazines/moneymag/best-jobs/.
  - Best job in America for many years

Why is Software Development 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**
- Software offers **extreme** flexibility

David Lorge Parnas - an early pioneer in software engineering who developed the concepts of modularity and information hiding in systems which are the foundation of object oriented methodologies.

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** and models
  - Notations for abstractions
- It is **knowledge management**
  - Elicitation, analysis, design, validation of the system and the solution process
- It is **rationale management**
  - Making the design and development decisions explicit to all stakeholders involved.

Techniques, Methodologies and Tools

- **Techniques**:
  - Formal procedures for producing results using some well-defined notation
- **Methodologies**:
  - Collection of techniques applied across software development and unified by a philosophical approach
- **Tools**:
  - Instruments or automated systems to accomplish a technique
  - Interactive Development Environment (**IDE**)
  - Computer Aided Software Engineering (**CASE**)

Computer Science vs. Software Engineering

- Computer Scientist
  - Assumes techniques and tools have to be developed.
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…☺
- Engineer
  - Develops a solution for a problem formulated by a client
  - Uses computers & languages, techniques and tools
- Software Engineer
  - Works in multiple application domains
  - Has only 3 months...
  - …while changes occurs in the problem formulation (requirements) and also in the available technology.

Software Engineering: A Working Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

A high quality software system developed with a given *budget* before a given *deadline* while *change* occurs

Challenge: Dealing with complexity and change

Software Engineering: A Problem Solving Activity

- **Analysis**:
  - Understand the nature of the problem and break the problem into pieces
- **Synthesis**:
  - Put the pieces together into a large structure

For problem solving we use techniques, methodologies and tools.

Frequently asked questions about software engineering

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Frequently asked questions about software engineering

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Software engineering: not just an engineering discipline

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- All aspects of software production
  - Not just technical process of development.
  - Also project management and the development of tools, methods etc. to support software production.

General issues that affect most software

- Heterogeneity
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- Business and social change
  - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Application types

- Stand-alone applications
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

Application types

- Batch processing systems
  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- Entertainment systems
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- Systems for modeling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

Application types

- Data collection systems
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- Systems of systems
  - These are systems that are composed of a number of other software systems.

Software engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

Software engineering and the web

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Web services allow application functionality to be accessed over the web.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the ‘cloud’.
  - Users do not buy software buy pay according to use.

Web software engineering

- Software reuse is the dominant approach for constructing web-based systems.
  - When building these systems, you think about how you can assemble them from pre-existing software components and systems.
- Web-based systems should be developed and delivered incrementally.
  - It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.
- User interfaces are constrained by the capabilities of web browsers.
  - Technologies such as AJAX allow rich interfaces to be created within a web browser but are still difficult to use. Web forms with local scripting are more commonly used.

Web-based software engineering

- Web-based systems are complex distributed systems but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.

- The fundamental ideas of software engineering, discussed here, apply to web-based software in the same way that they apply to other types of software system.

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Intro to Software Engineering

- The Software Engineering Discipline
- The Software Process

Software Process

- Software process models
- Process activities
- Coping with change
- The Rational Unified Process
  - An example of a modern software process.

Inherent Problems with Software Development

- Requirements are constantly changing
  - The client might not know all the requirements in advance
- Frequent changes are difficult to manage
  - Identifying checkpoints for planning and cost estimation is difficult
- There is more than one software system
  - New system must often be backward compatible with existing system (“legacy system”)

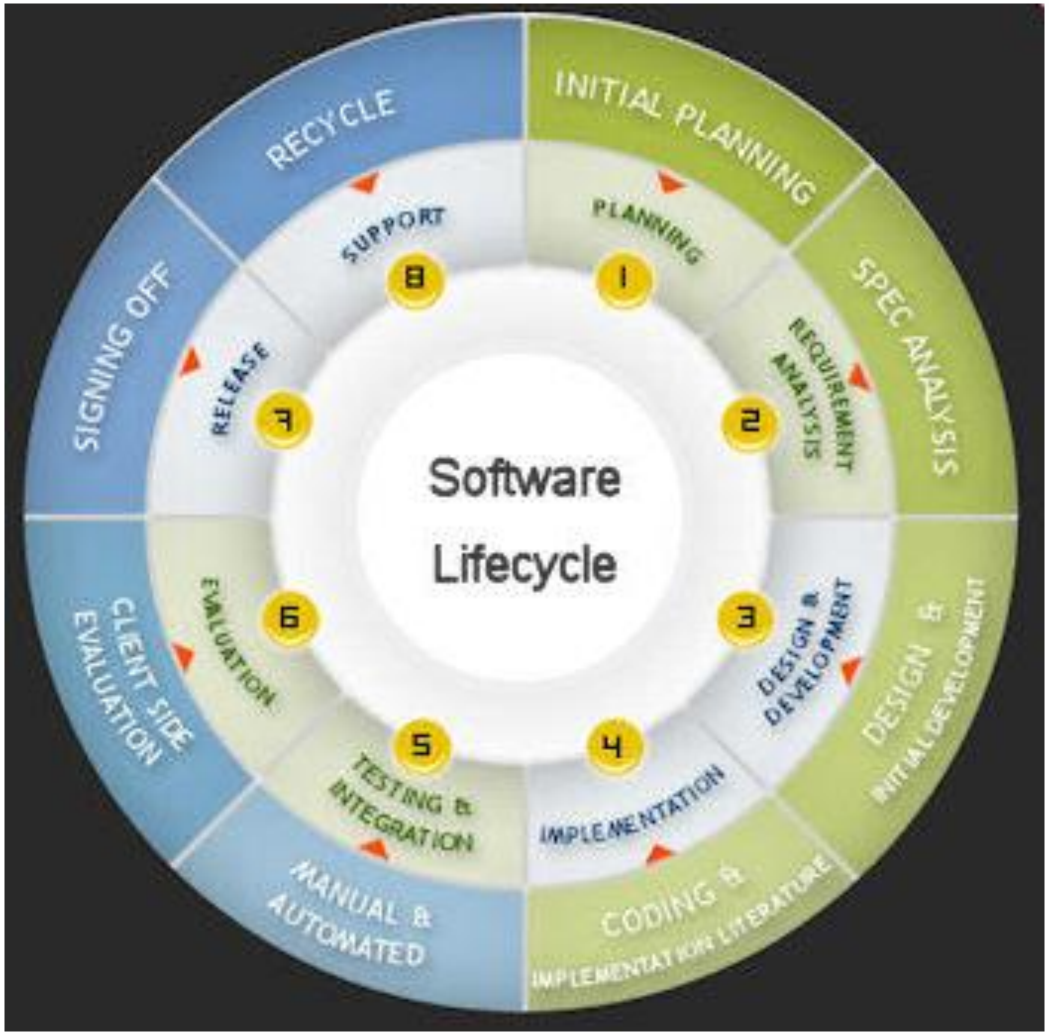
The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design – defining the organization of the system
  - Implementation –implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process descriptions

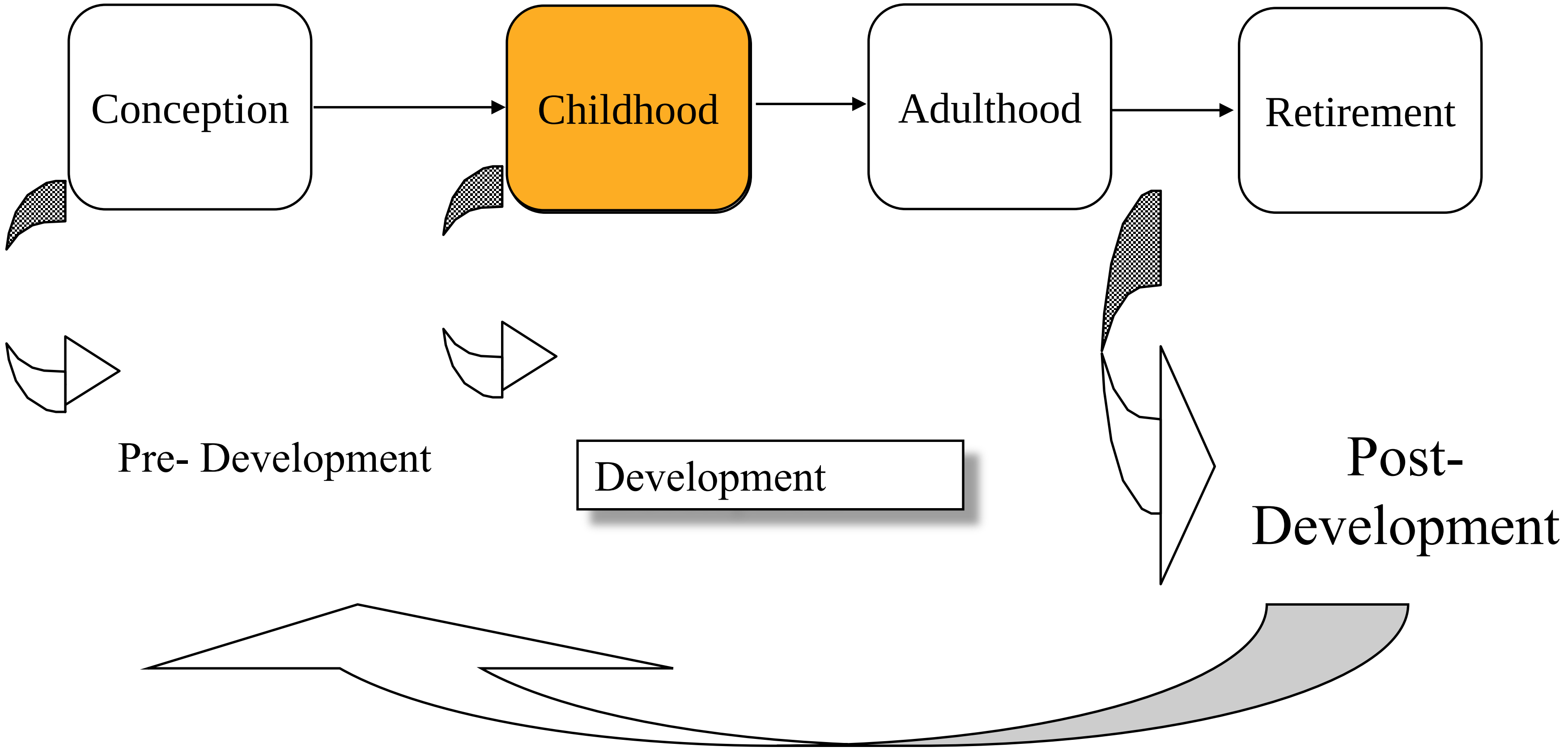
- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Software lifecycle



Software Life Cycle

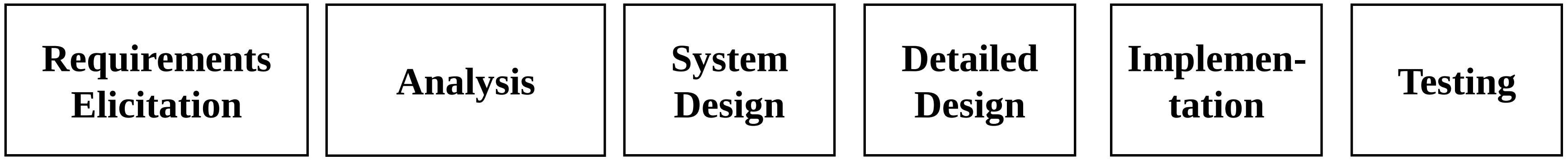
- The term "Lifecycle" is based on the metaphor of the life of a person:



Software Lifecycle Definition

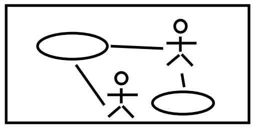
- Software lifecycle
  - Models for the development of software
    - **Set of activities** and their **dependency relationship**s to each other to support the development of a software system
    - Examples:
      - Analysis, design, implementation, testing
      - Design depends on analysis

A Typical Example of Software Lifecycle Activities



Software Lifecycle Activities

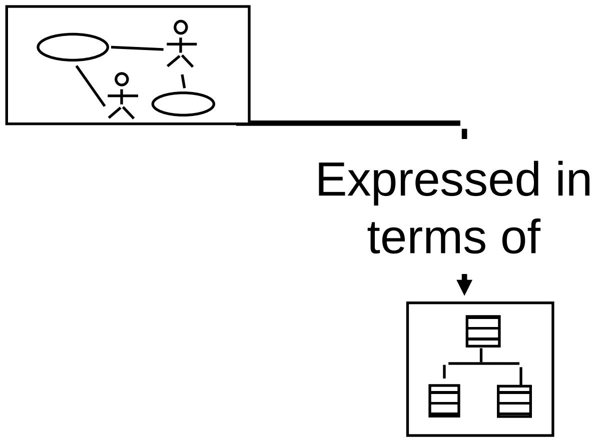
Requirements Elicitation | Analysis | System Design | Detailed Design | Implemen-tation | Testing



Use Case Model

Software Lifecycle Activities

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing

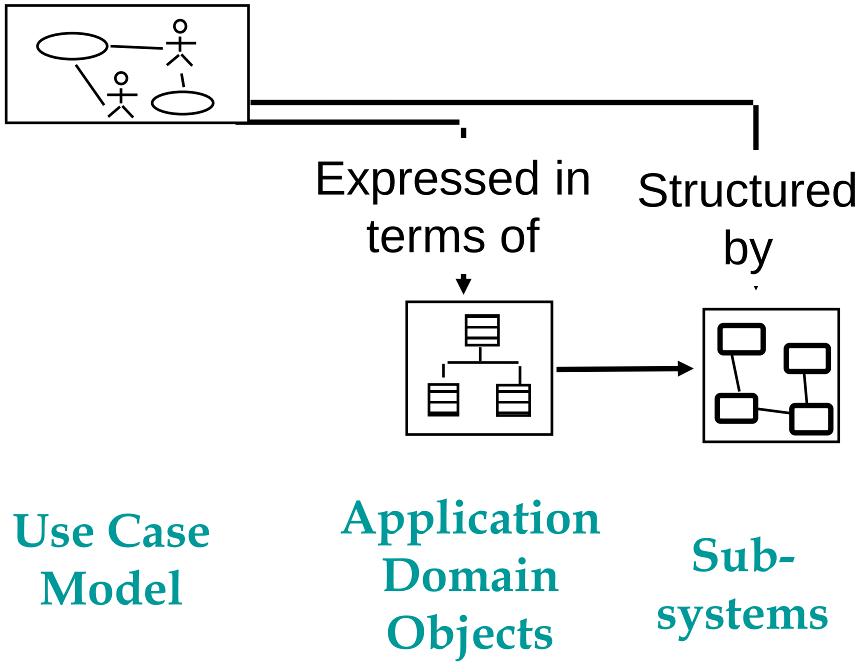


Use Case Model

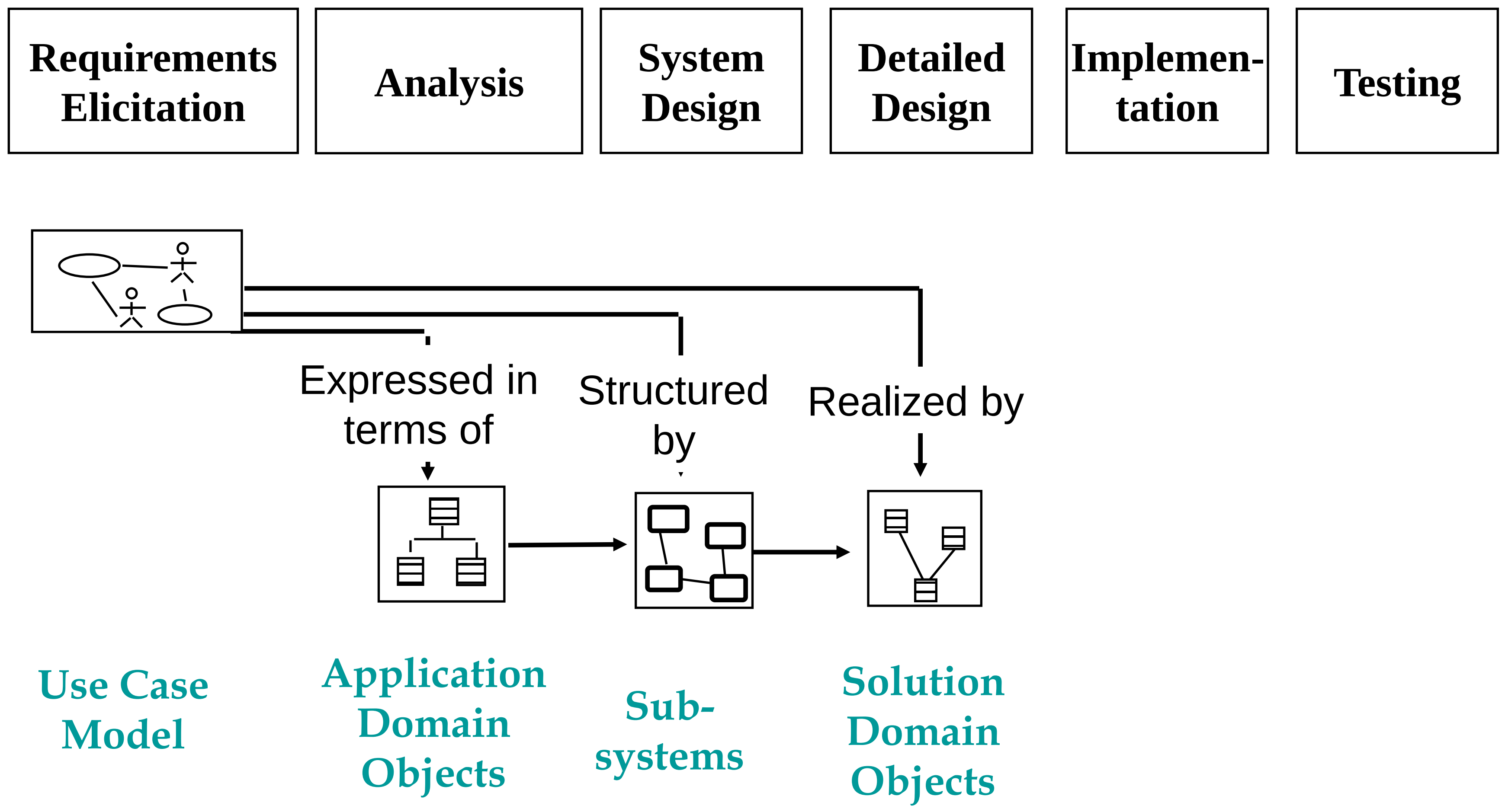
Application Domain Objects

Software Lifecycle Activities

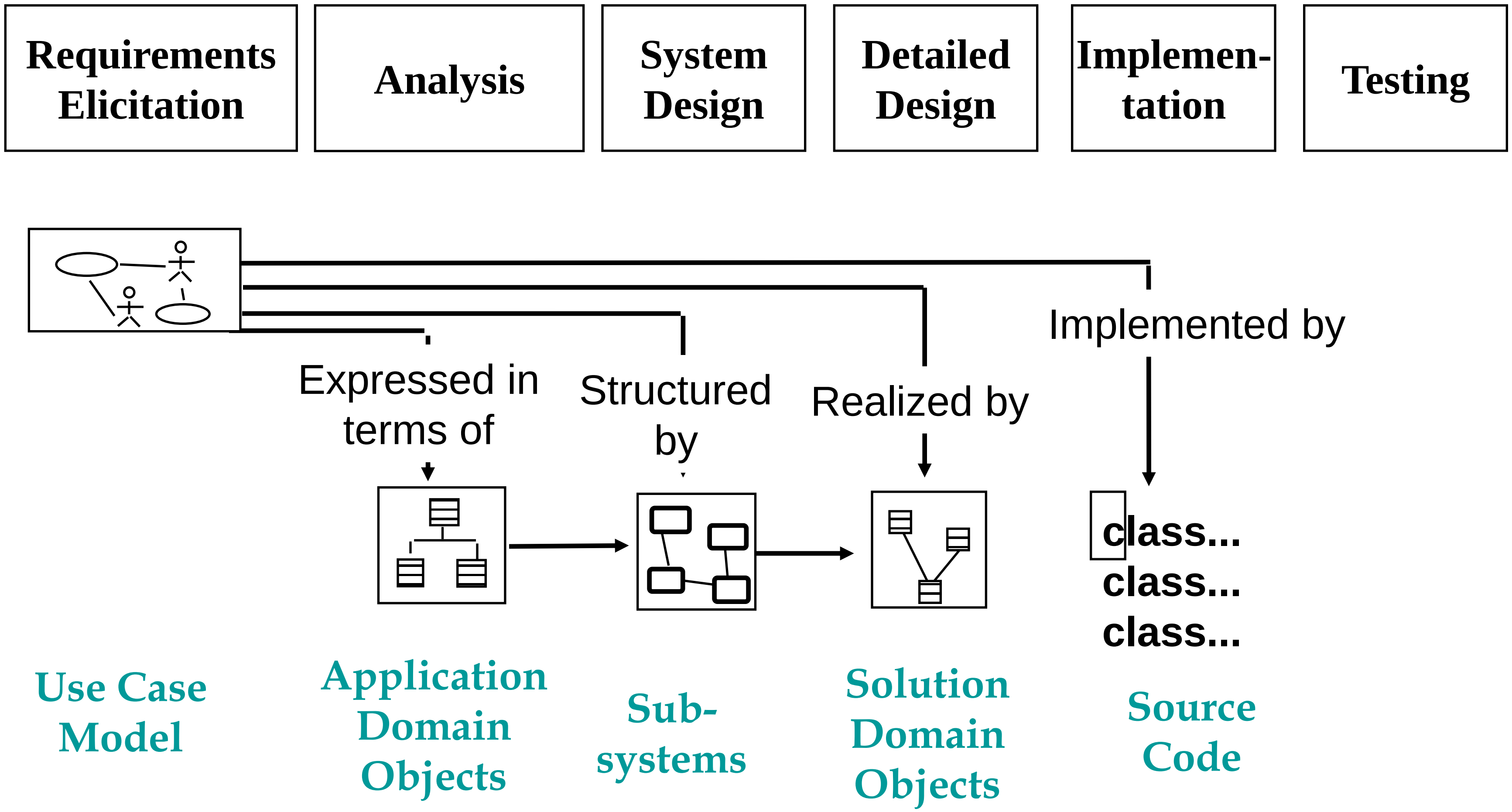
Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



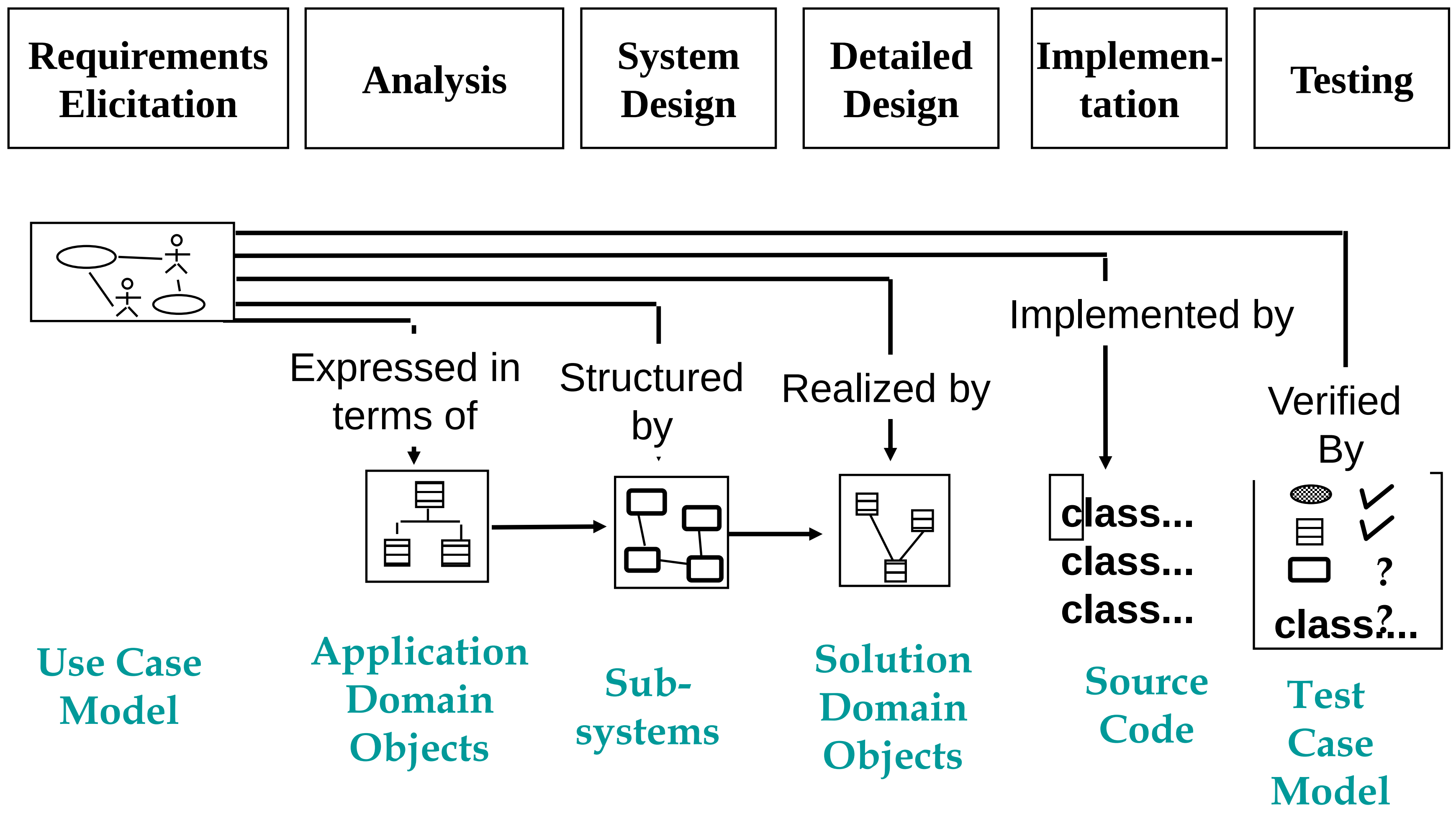
Software Lifecycle Activities



Software Lifecycle Activities



Software Lifecycle Activities



Plan-driven and agile processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

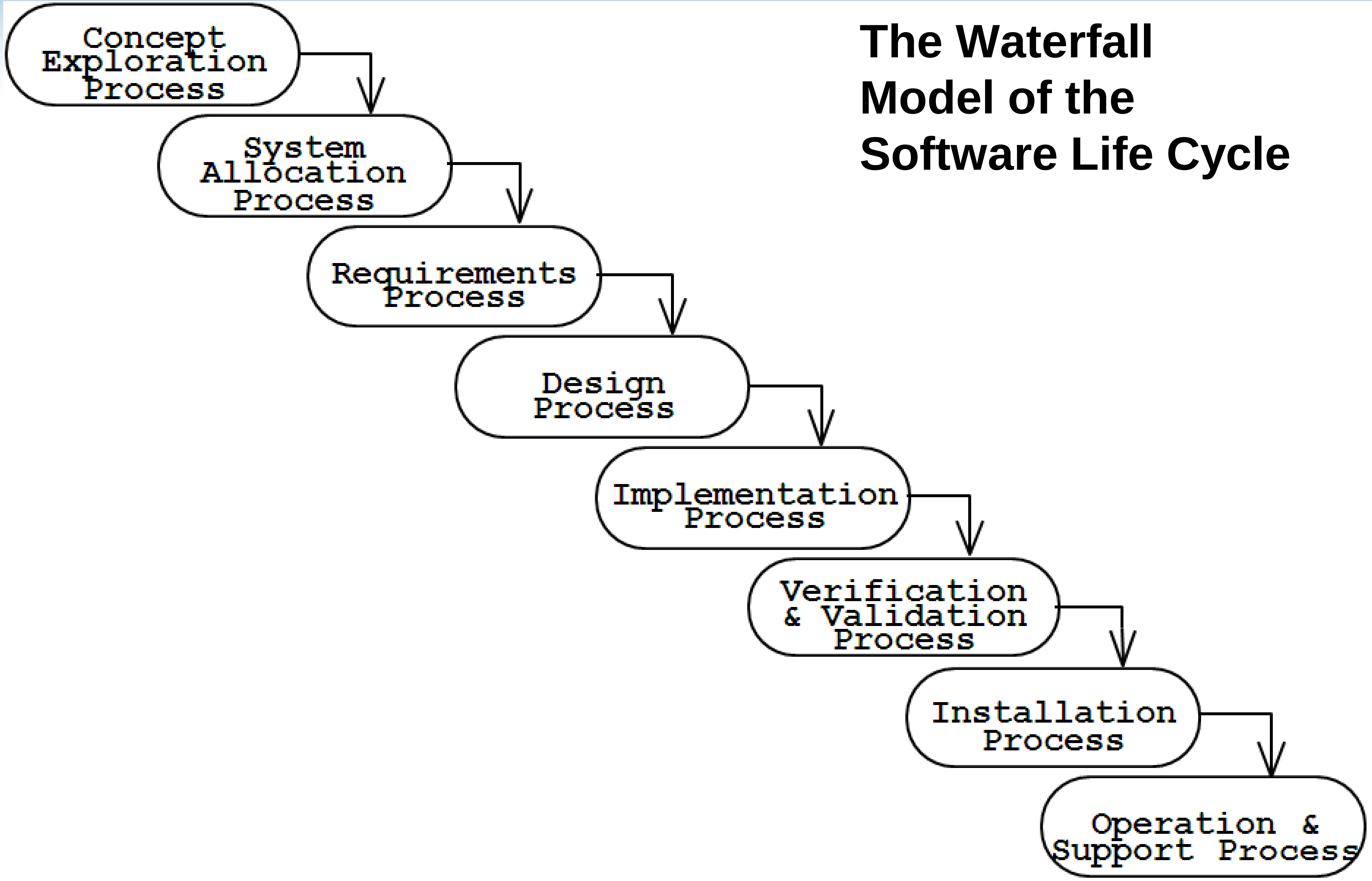
Reuse-oriented software engineering

- Reuse-oriented software engineering
  - The system is assembled from existing components.
  - May be plan-driven or agile.

Software Lifecycles Standards

- Software Development as Application Domain
  - Modeling the software lifecycle
- IEEE Standard 1074 for Software Lifecycles
- Modeling the software life cycle
  - Sequential models
    - Pure waterfall model
  - Iterative models
    - Boehm’s spiral model
    - Unified Process

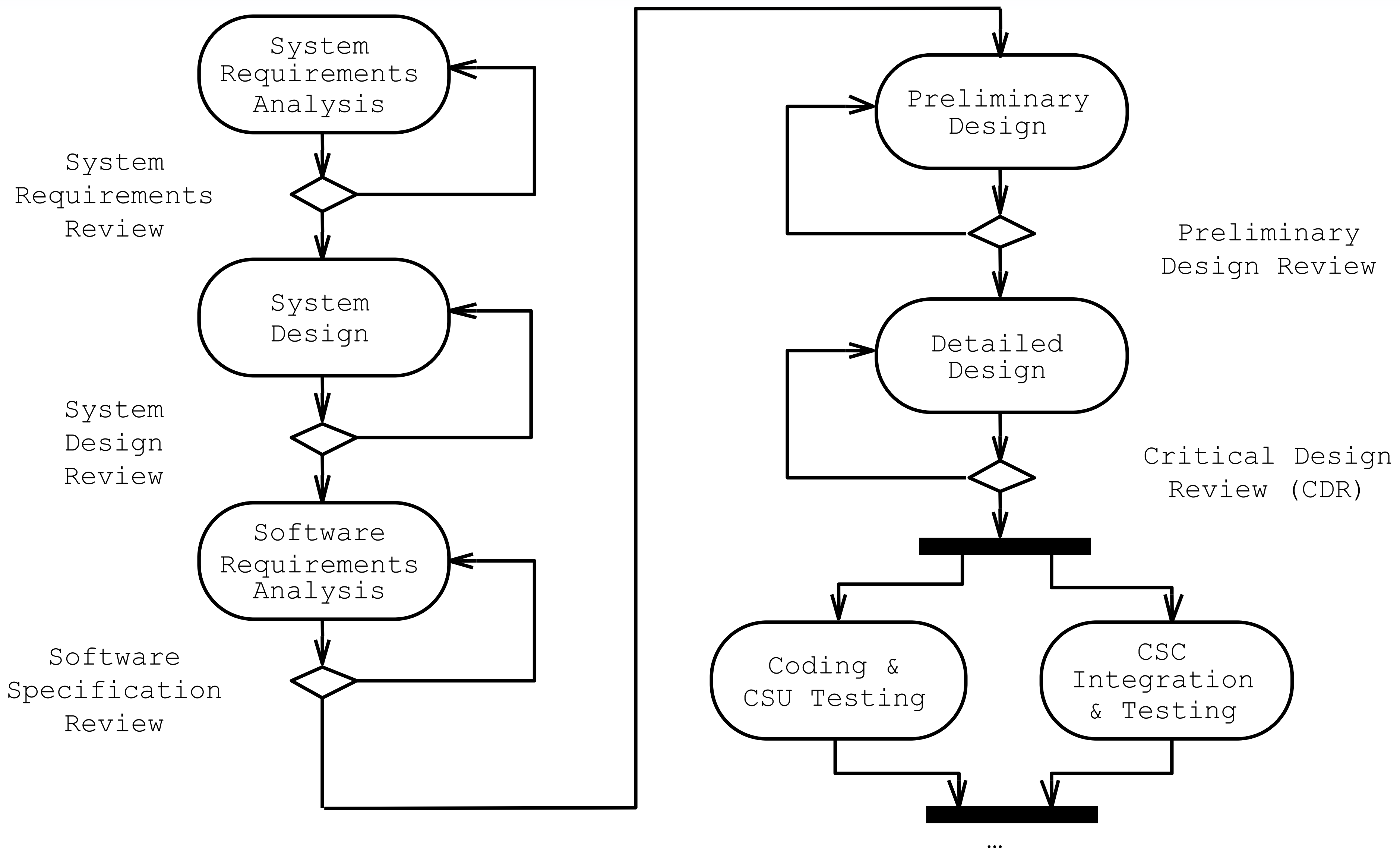
The Waterfall Model of the Software Life Cycle


Concept Exploration Process
System Allocation Process
Requirements Process
Design Process
Implementation Process
Verification & Validation Process
Installation Process
Operation & Support Process

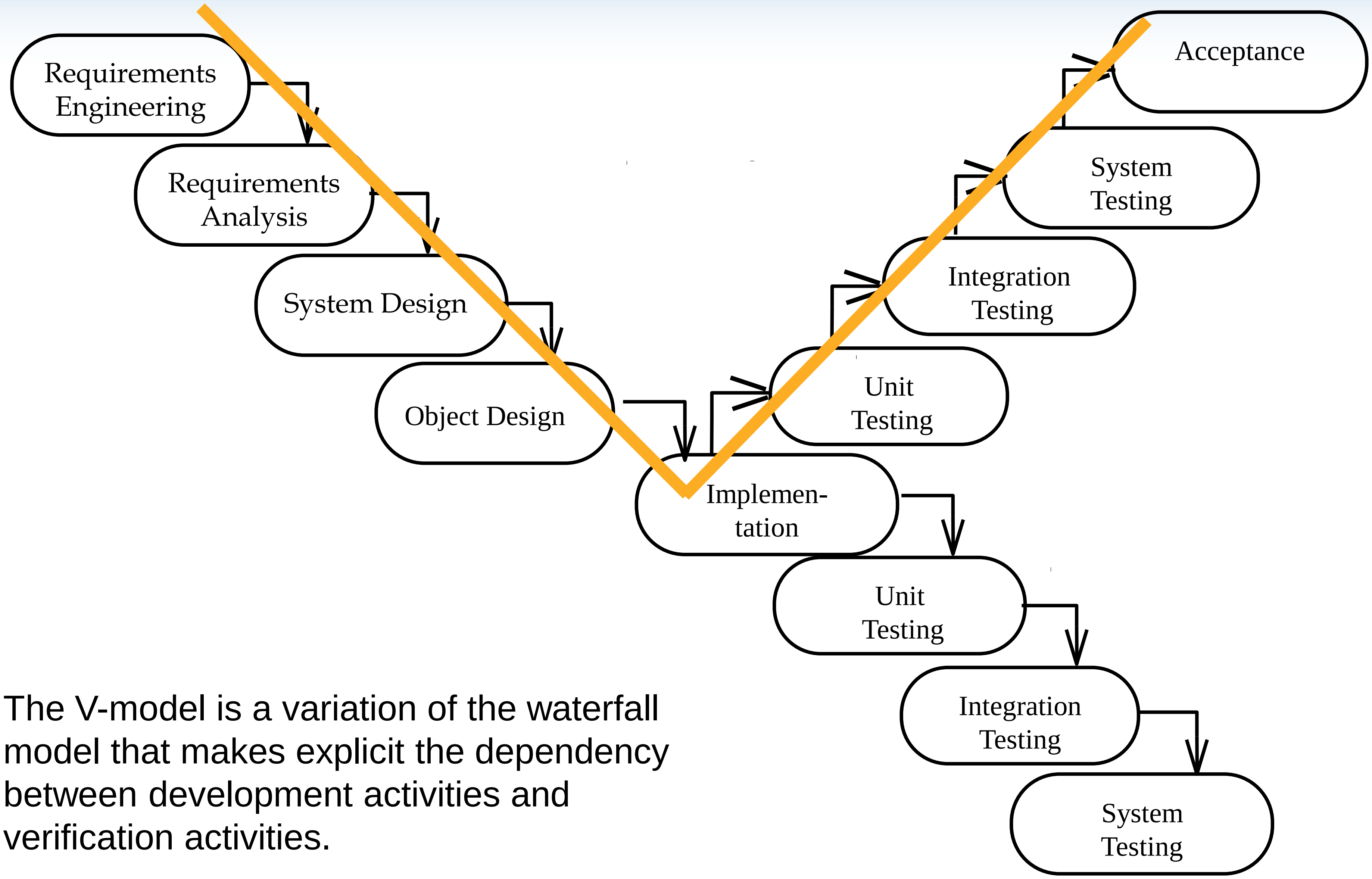
Example of a waterfall model : DOD Standard 2167A

- Software development activities:
  - System Requirements Analysis/Design
  - Software Requirements Analysis
  - Preliminary Design and Detailed Design
  - Coding and CSU testing
  - CSC Integration and Testing
  - CSCI Testing
  - System integration and Testing
- Required by the U.S. Department of Defense for all software contractors in the 1980-90’s.

Activity Diagram of MIL DOD-STD-2167A

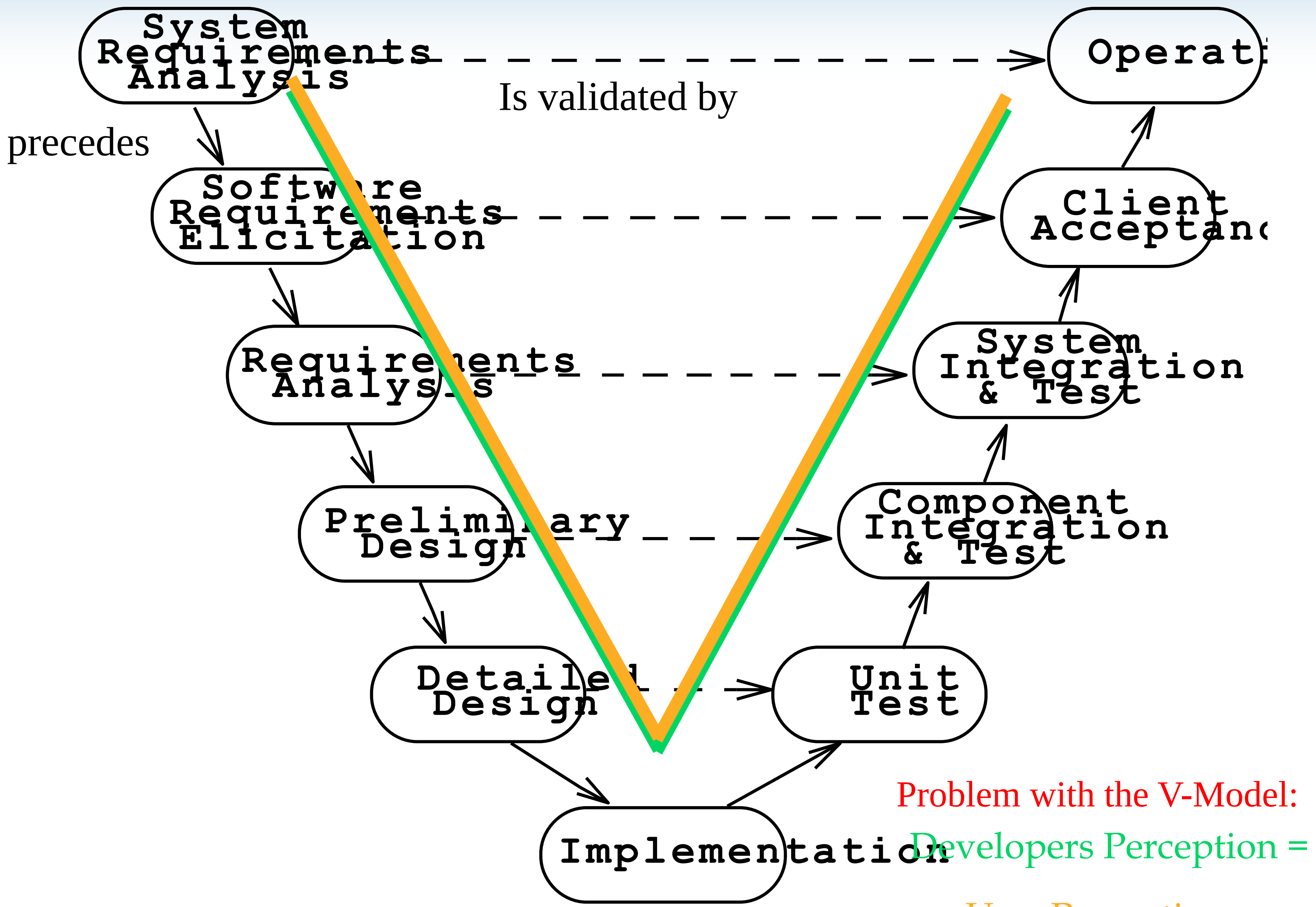

System Requirements Analysis
System Requirements Review
System Design
System Design Review
Software Requirements Analysis
Software Specification Review
Preliminary Design
Preliminary Design Review
Detailed Design
Critical Design Review (CDR)
Coding & CSU Testing
CSC Integration & Testing
...

From the Waterfall Model to the V Model



The V-model is a variation of the waterfall model that makes explicit the dependency between development activities and verification activities.

Activity Diagram of the V Model



Problem with the V-Model:

Developers Perception =

User Perception

Properties of Waterfall-based 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.

Spiral Model

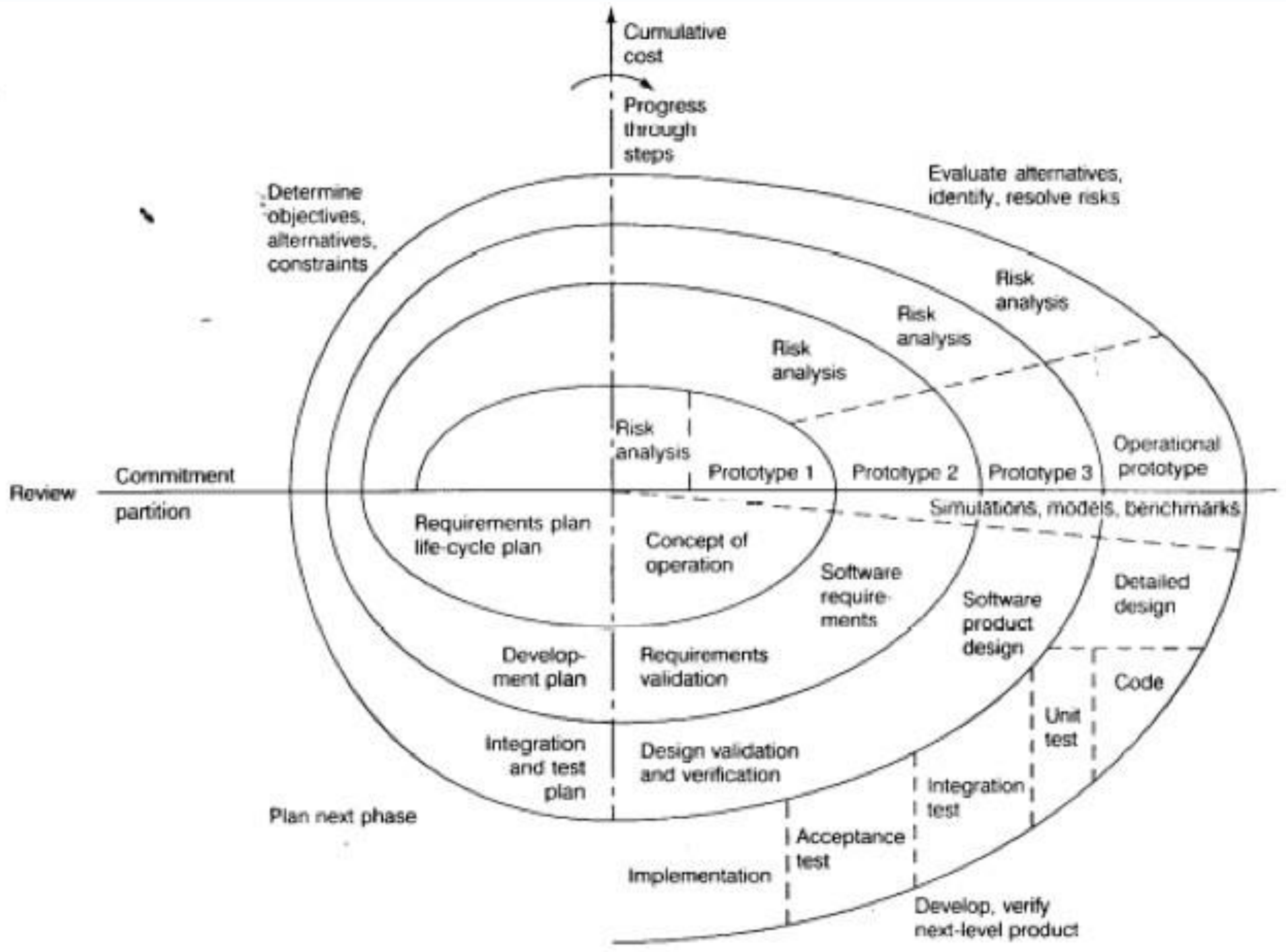
- The spiral model proposed by Boehm has the following set of activities
  - Determine objectives and constraints
  - Evaluate alternatives
  - Identify risks
  - Resolve risks by assigning priorities to risks
  - Develop a series of prototypes for the identified risks starting with the highest risk
  - 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.

Rounds in Boehm’s Spiral Model

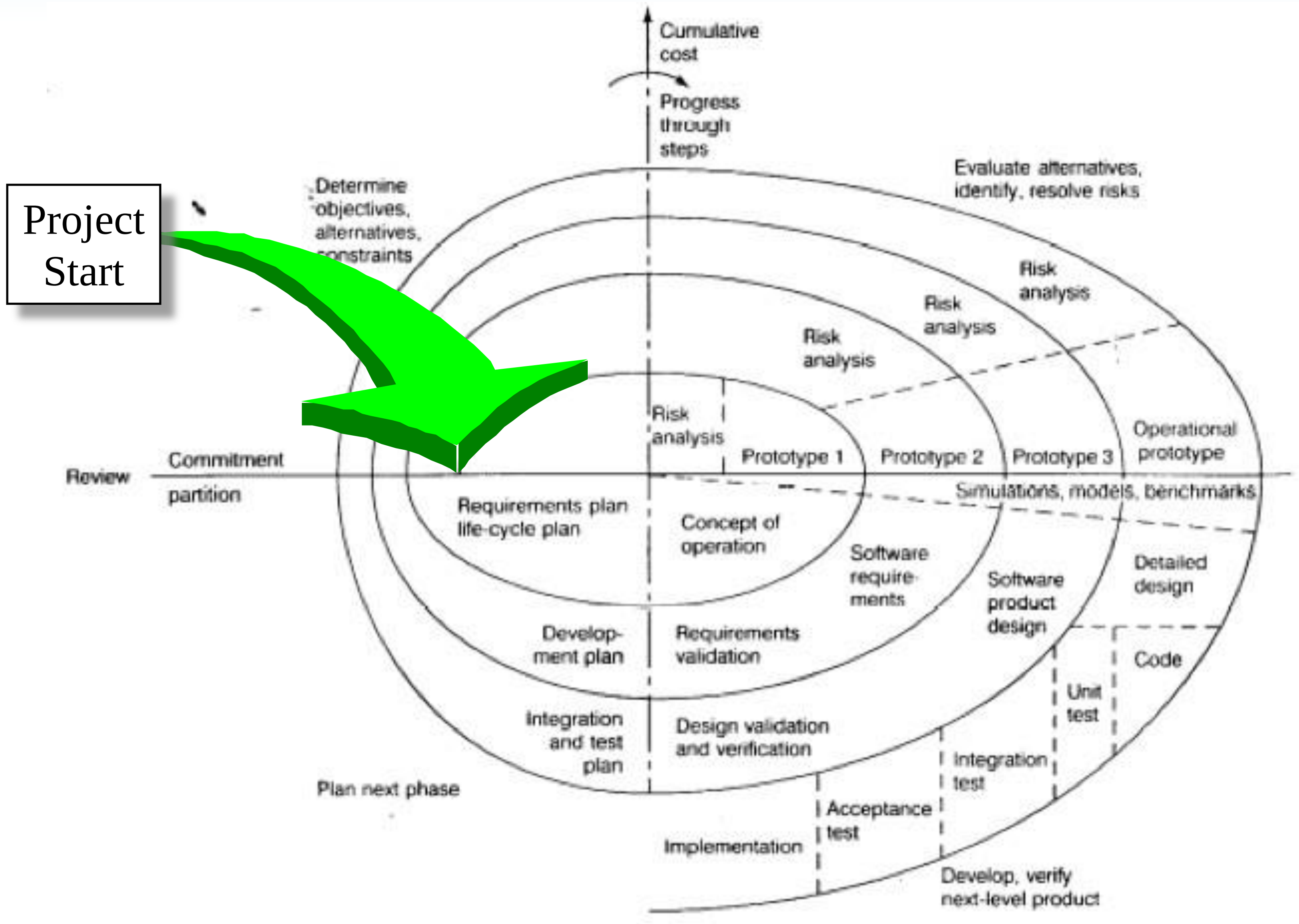
- Concept of Operations
- Software Requirements
- Software Product Design
- Detailed Design
- Code
- Unit Test
- Integration and Test
- Acceptance Test
- Implementation

- For each round go through these activities:
  - Define objectives, alternatives, constraints
  - Evaluate alternatives, identify and resolve risks
  - Develop and verify a prototype
  - Plan the next round.

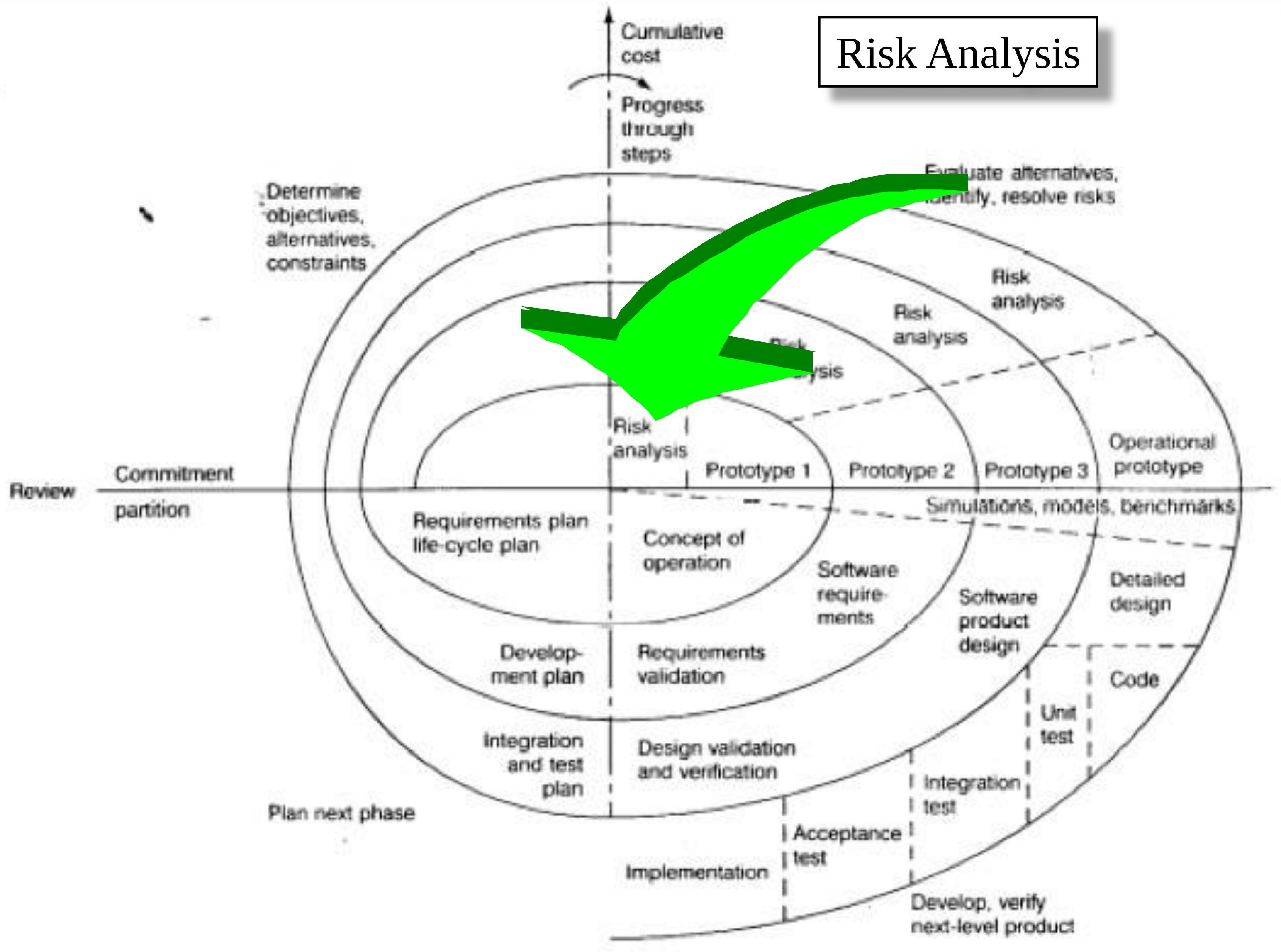
Diagram of Boehm’s Spiral Model



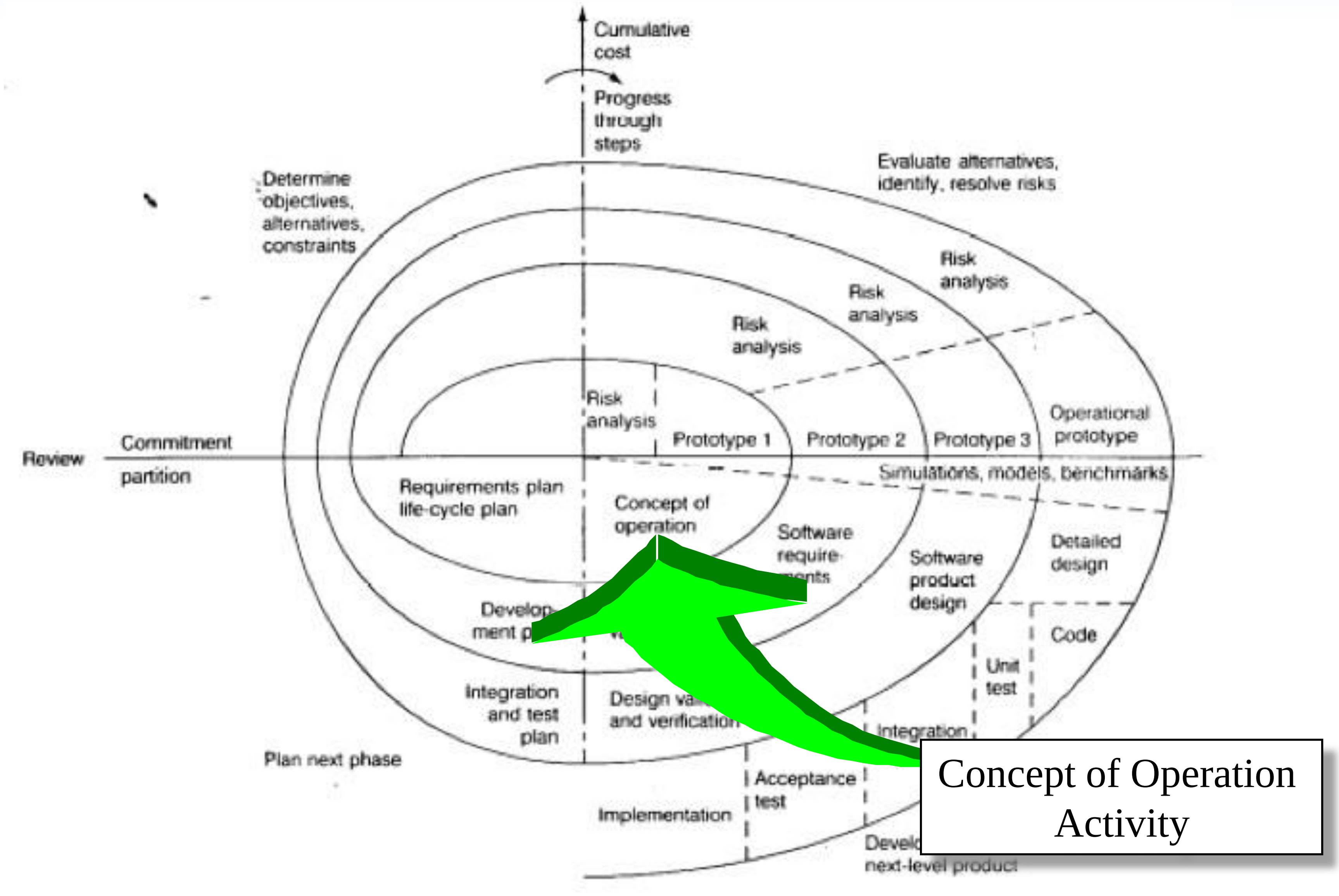
Round 1, Concept of Operations, Quadrant IV: Determine Objectives,Alternatives & Constraints



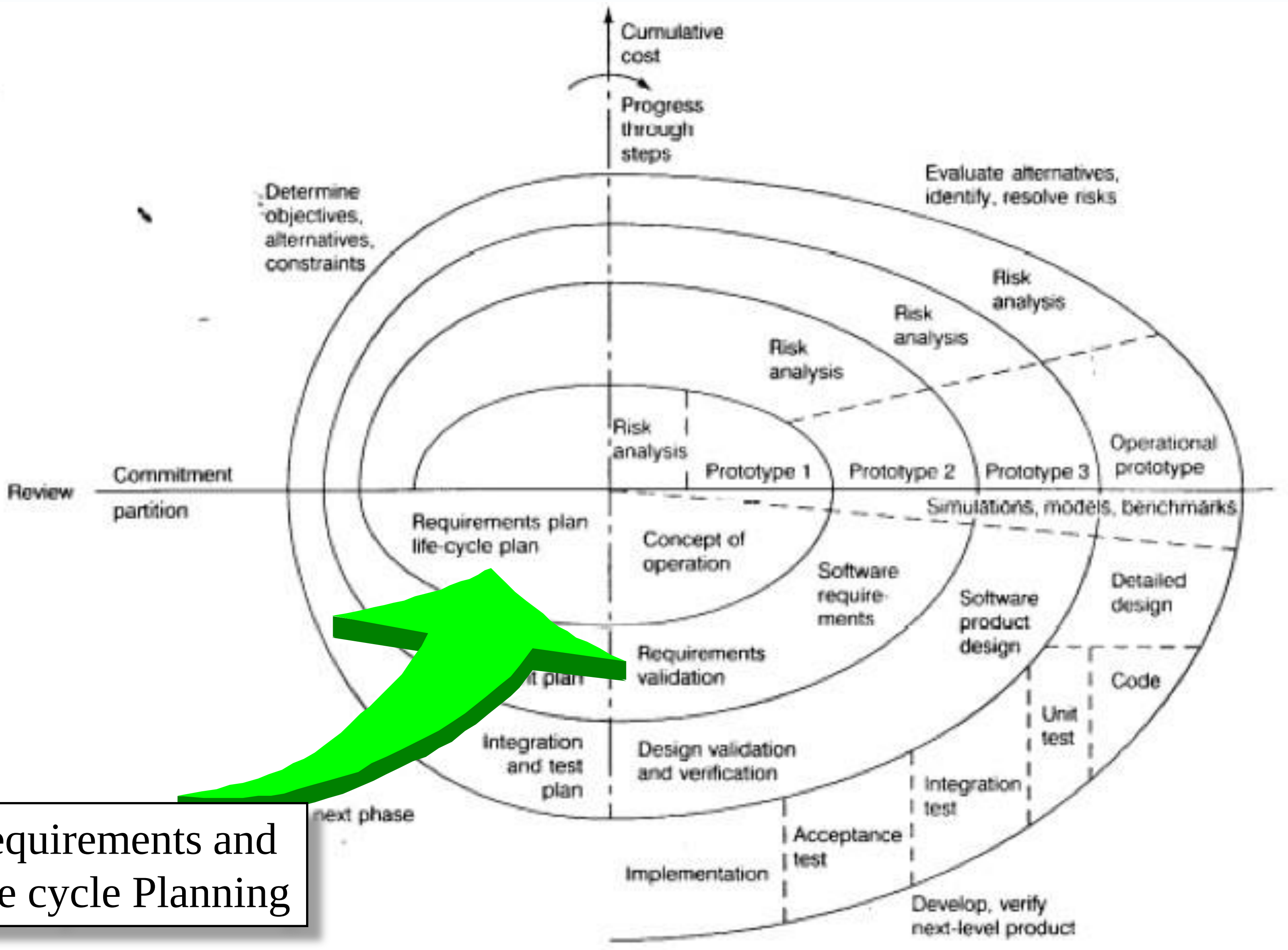
Round 1, Concept of Operations, Quadrant I: Evaluate Alternatives, identify & resolve Risks



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



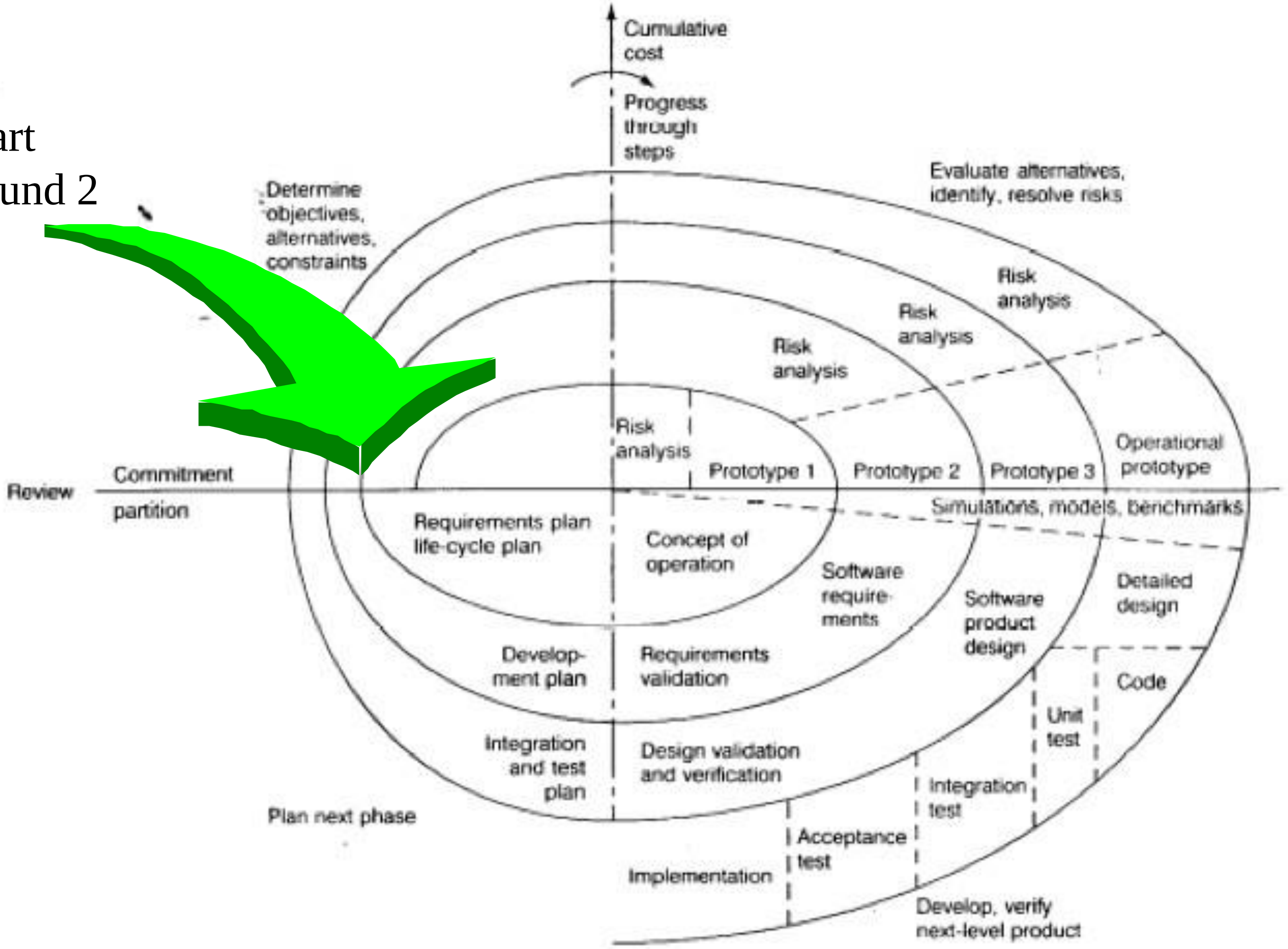
Round 1, Concept of Operations, Quadrant III: Prepare for Next Activity



Requirements and Life cycle Planning

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

Start
of Round 2



Limitations of Waterfall and Spiral Models

- Neither of these models deal well with frequent change
  - The Waterfall model assumes 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 does not allow change within a phase
- What do you do if change is happening more frequently?
  - In software development *"The only constant is the change"* ☺

Coping with change

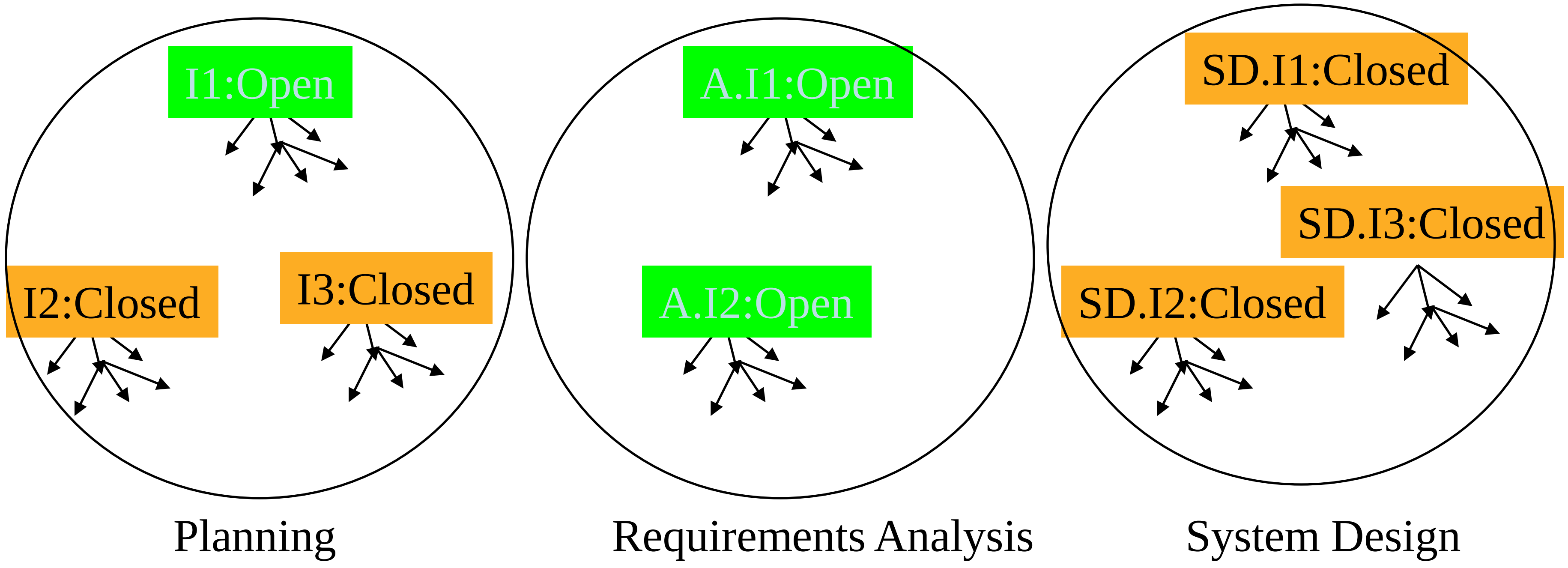
- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- Change leads to rework so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

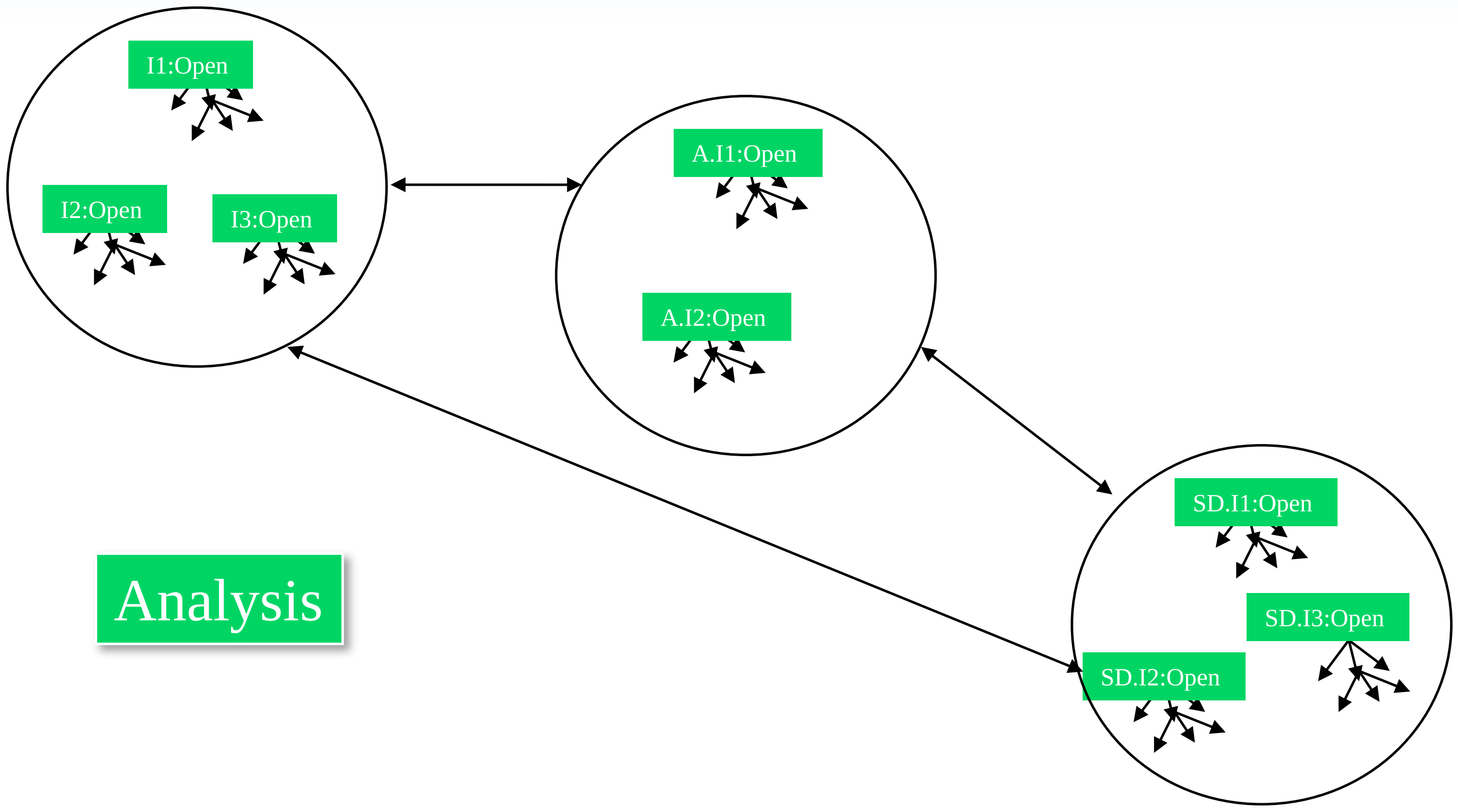
- Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.
- Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development.
  - Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

An Alternative: Issue-Based Development

- A system is described as a collection of issues
  - Issues are either closed or open
  - Closed issues have a resolution
  - Closed issues can be reopened (Iteration!)
- The set of closed issues is the basis of the system model

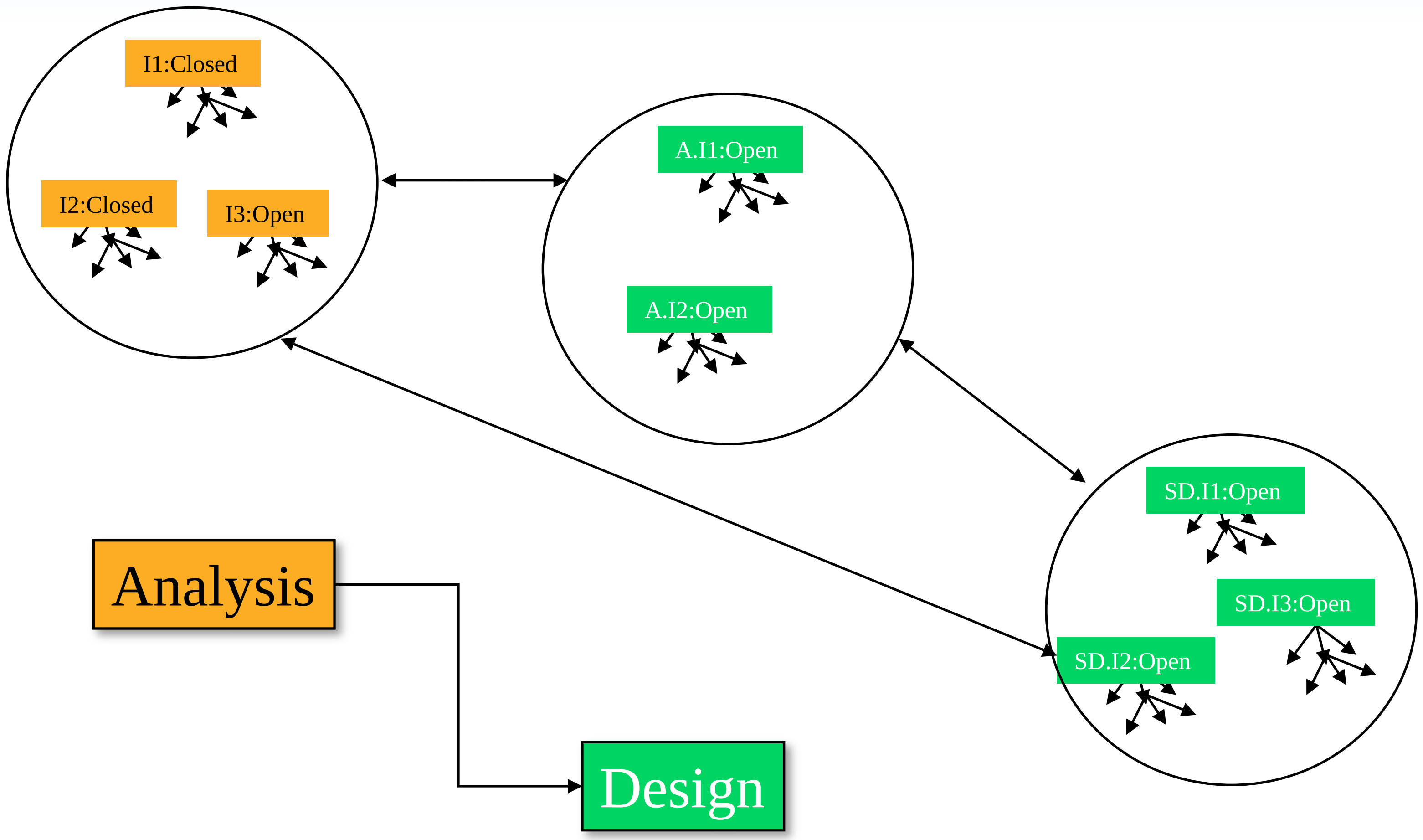


Waterfall Model: Analysis Phase

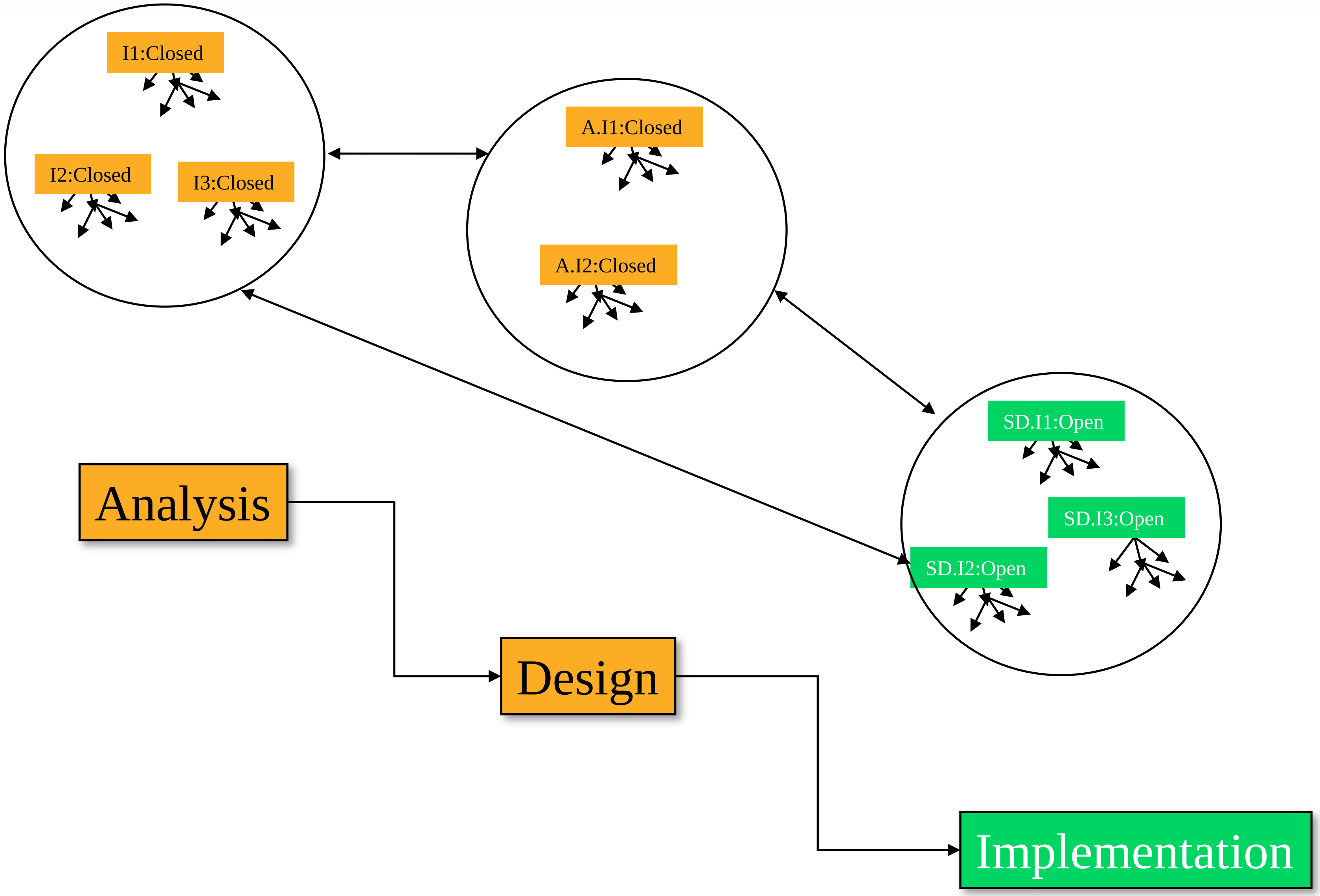


Analysis

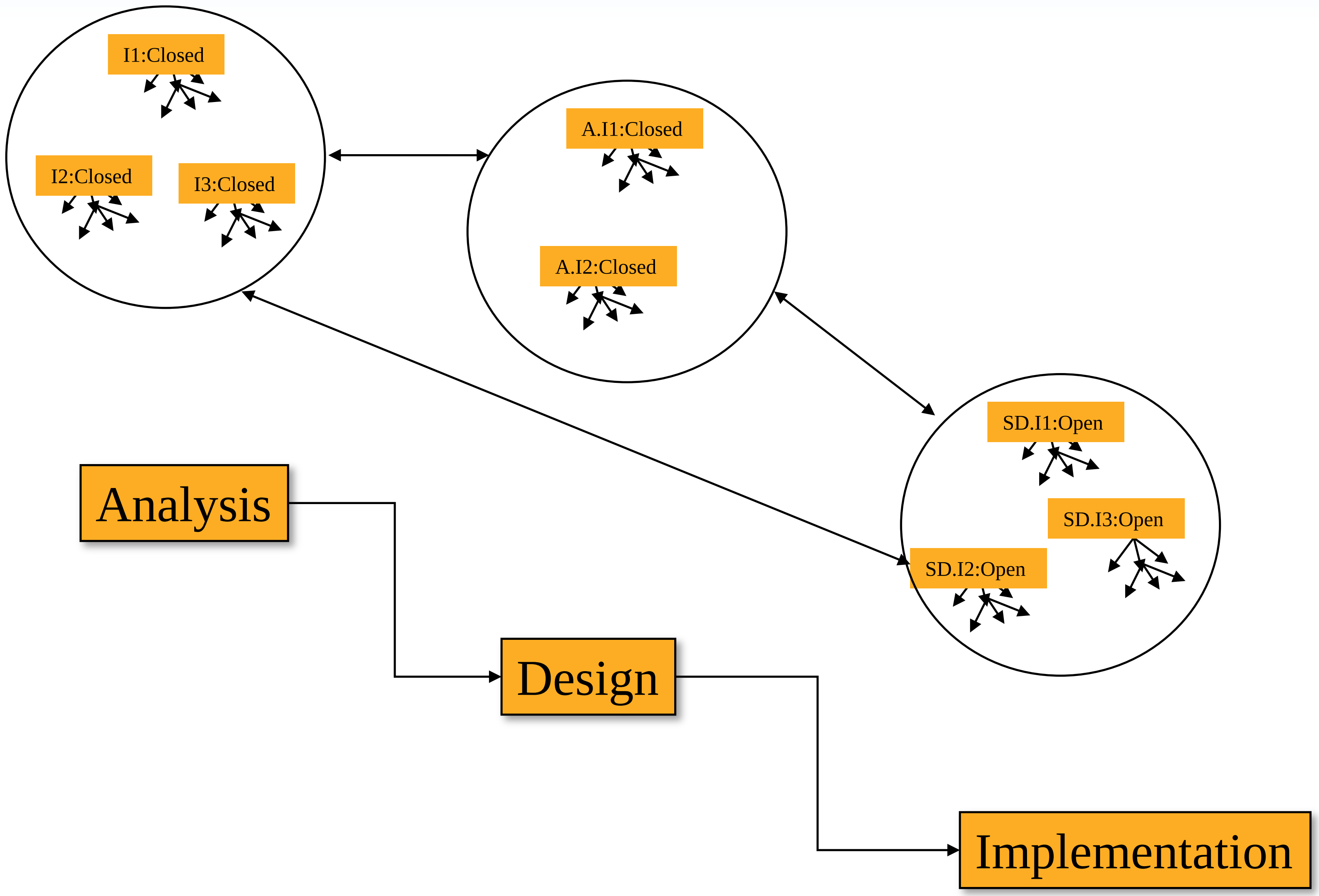
Waterfall Model: Design Phase



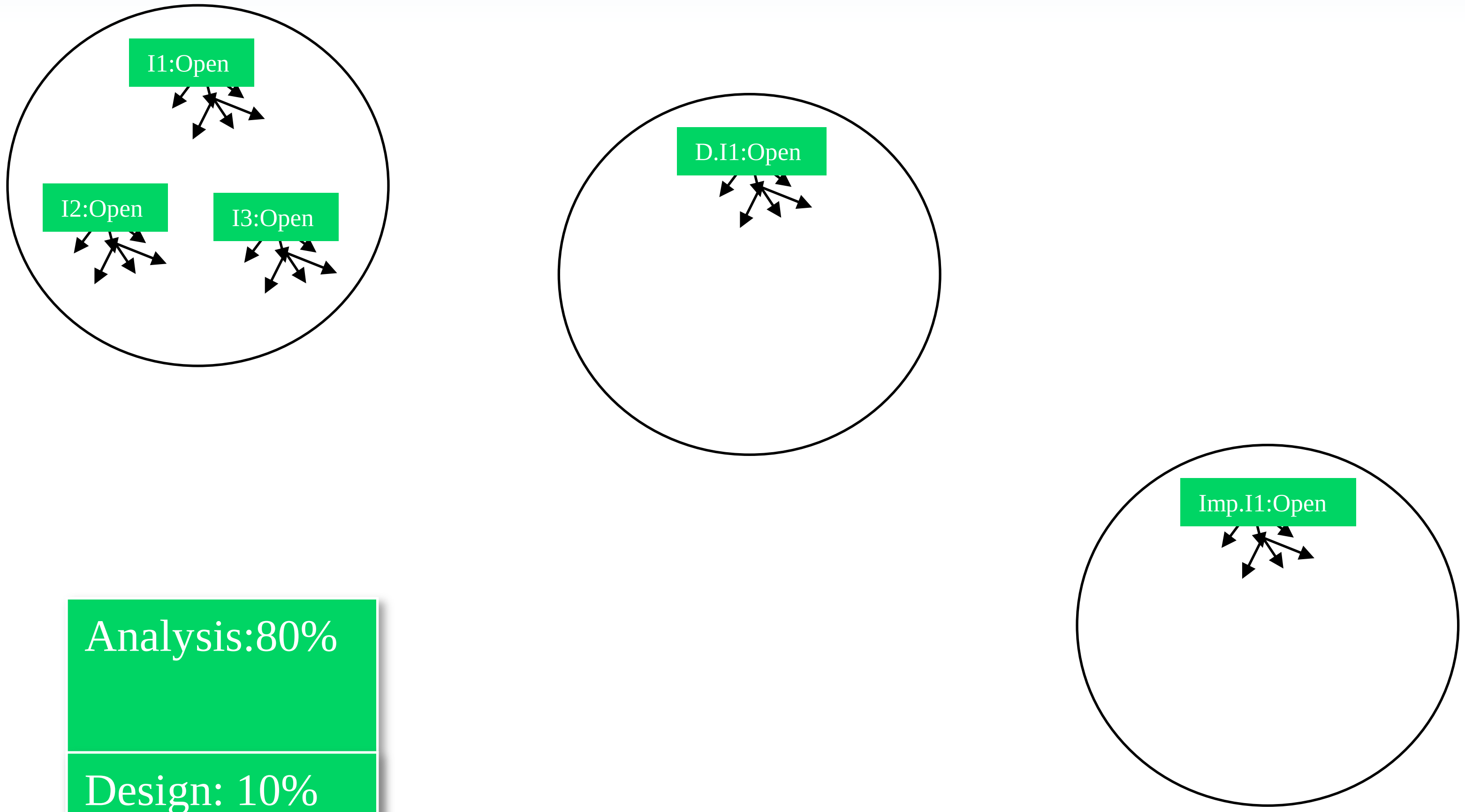
Waterfall Model: Implementation Phase



Waterfall Model: Project is Done



Issue-Based Model: Analysis Phase

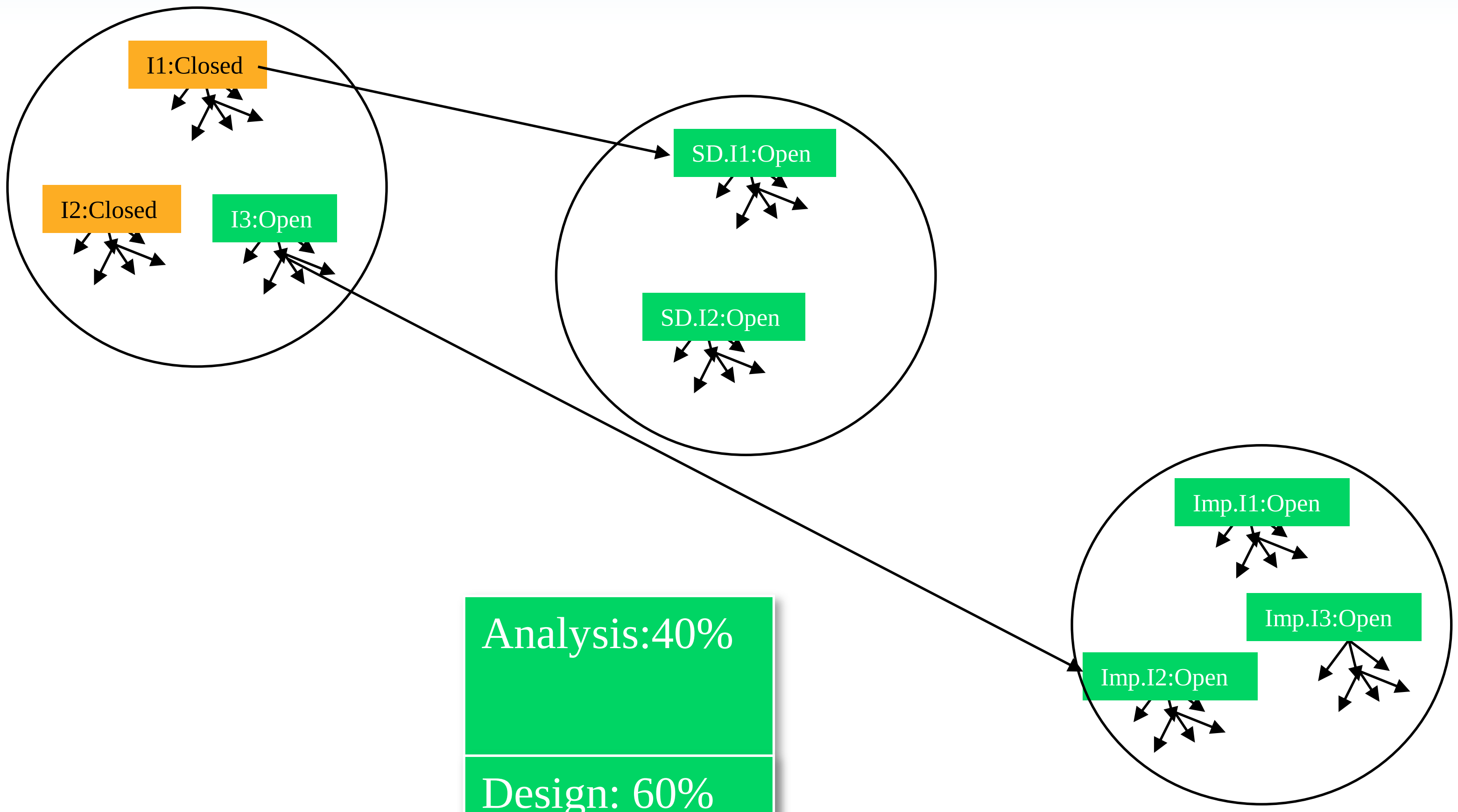


Analysis:80%

Design: 10%

Implemen-
tation: 10%

Issue-Based Model: Design Phase

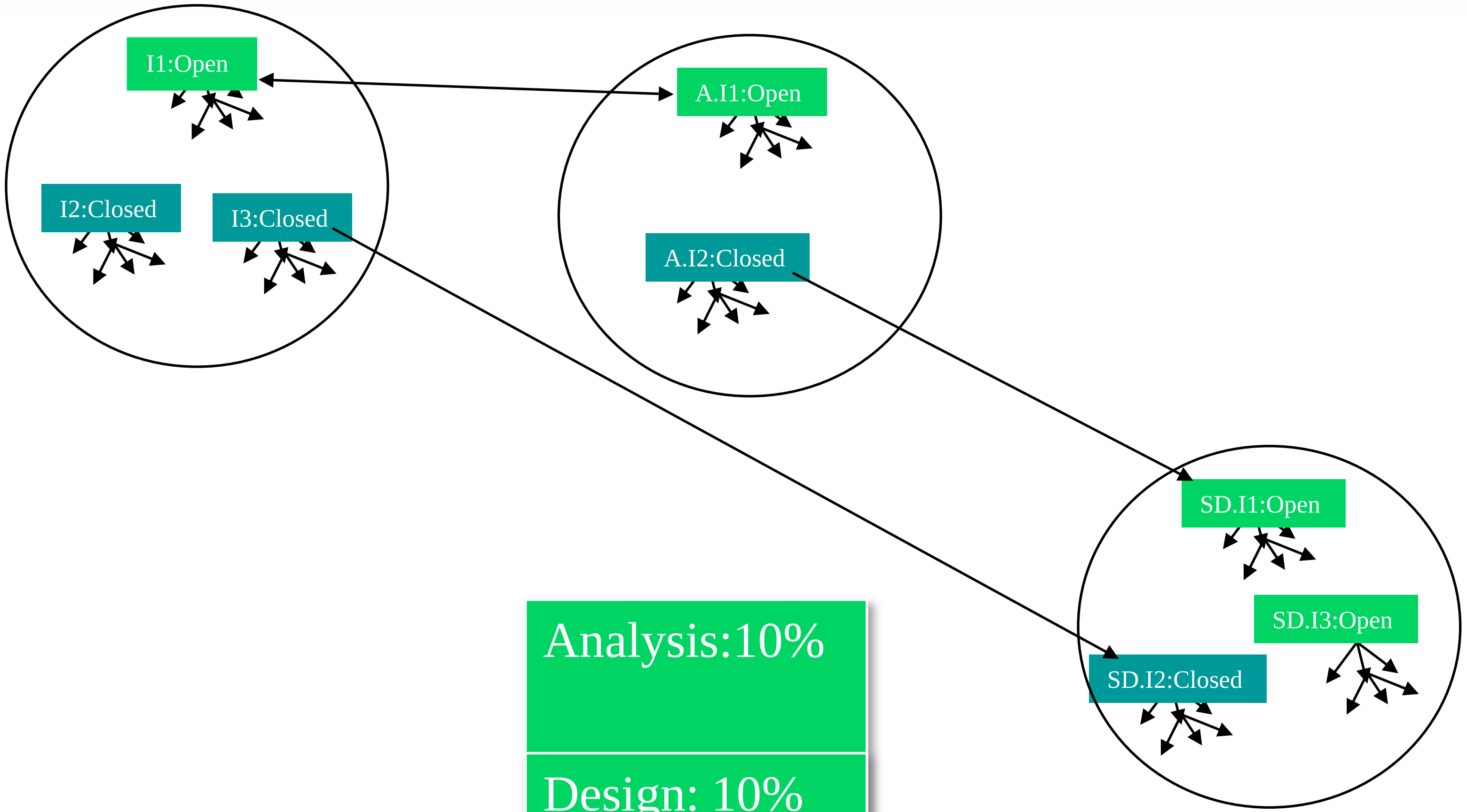


Analysis:40%

Design: 60%

Implemen-
tation: 0%

Issue-Based Model: Implementation Phase

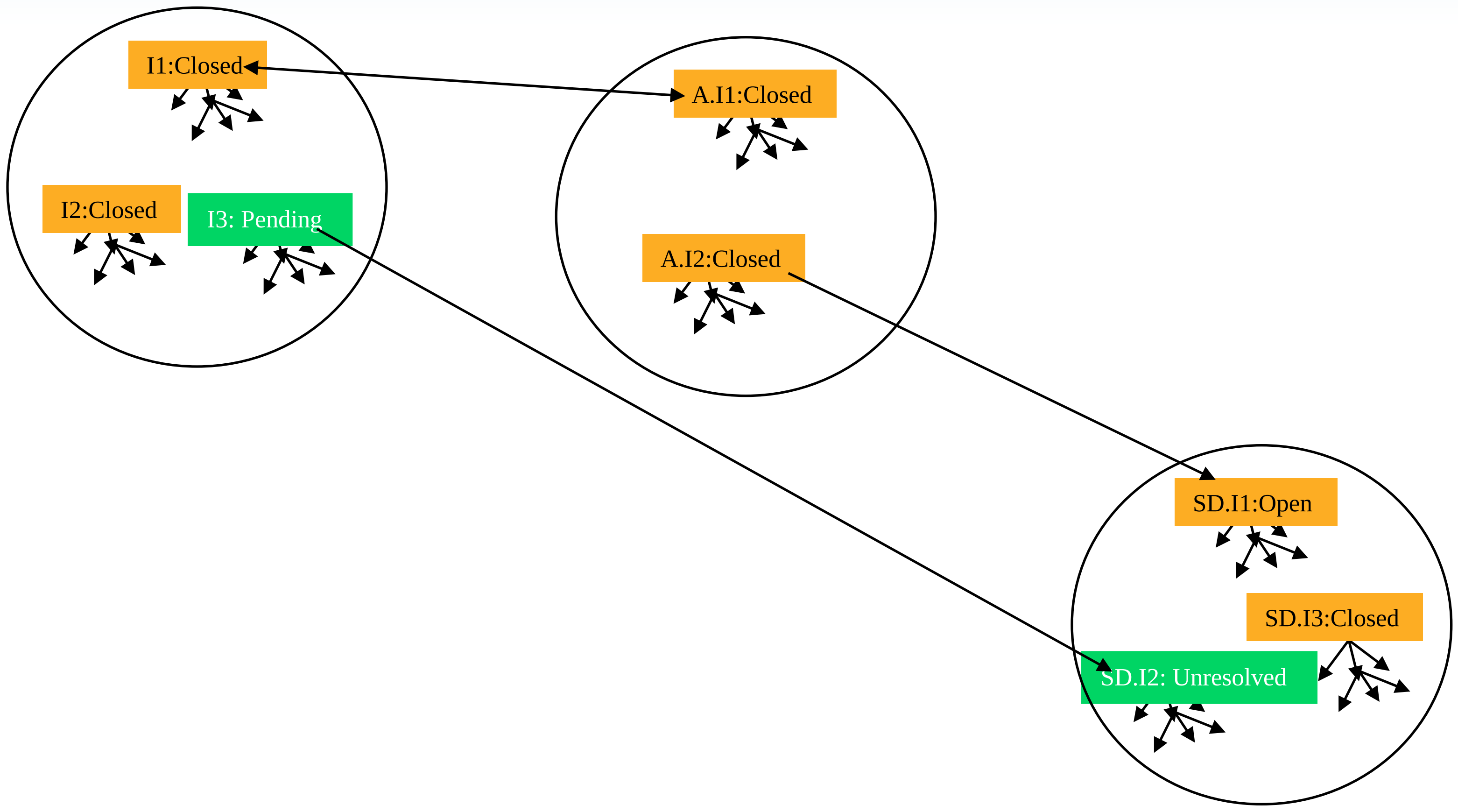


Analysis:10%

Design: 10%

Implemen-
tation: 60%

Issue-Based Model: Prototype is Done



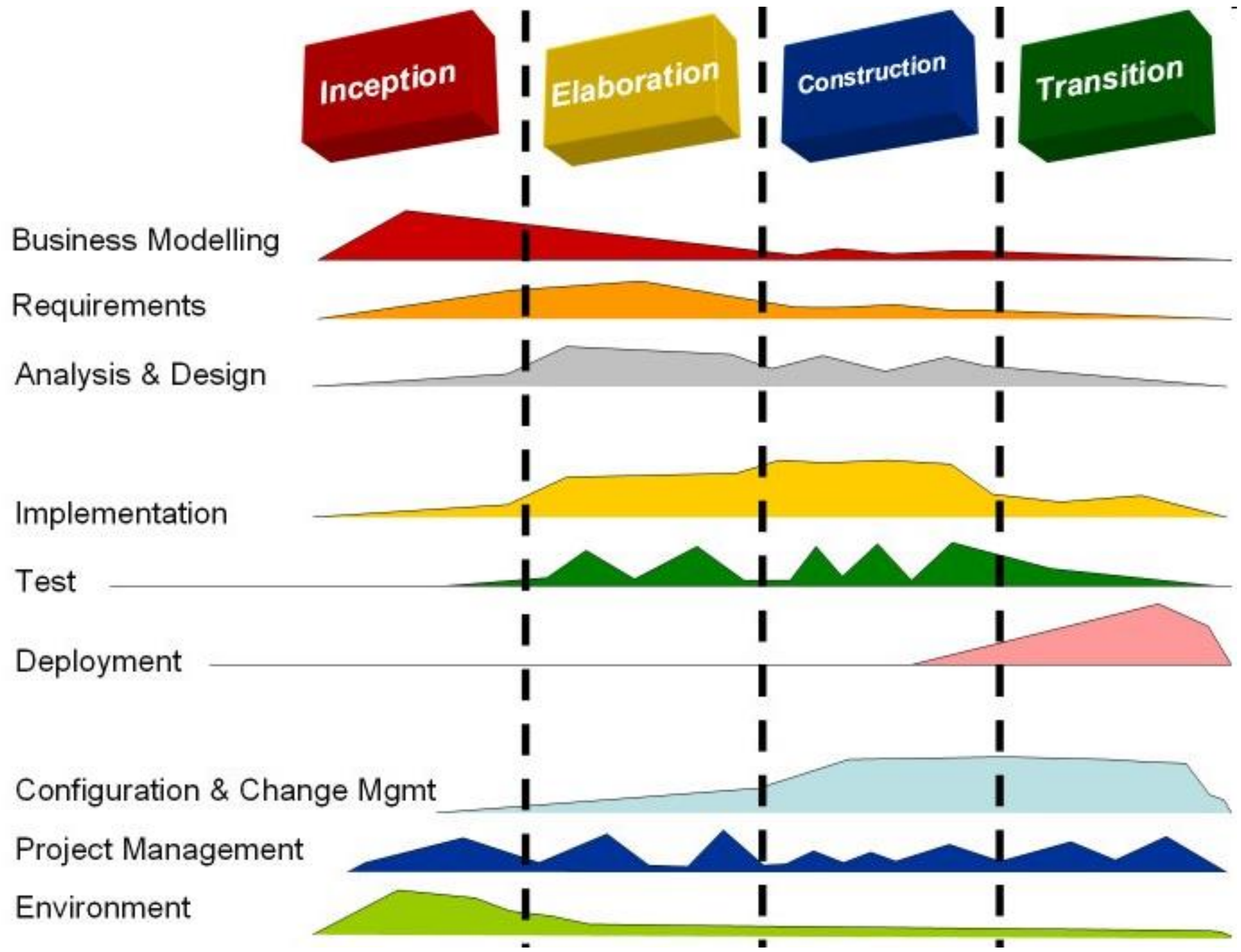
The Rational Unified Process

- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practice perspective that suggests good practice.

RUP phases

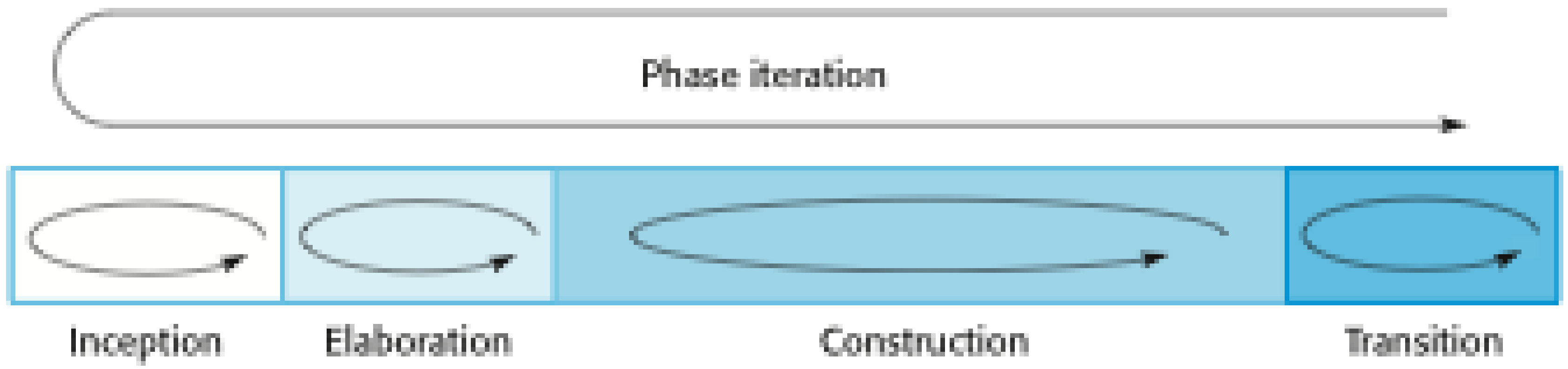
- Inception
  - Establish the business case for the system.
- Elaboration
  - Develop an understanding of the problem domain and the system architecture.
- Construction
  - System design, programming and testing.
- Transition
  - Deploy the system in its operating environment.

RUP



Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment

Phases in the Rational Unified Process



RUP iteration

- In-phase iteration
  - Each phase is iterative with results developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

- At the end of each repetition some kind of prototype or artefact are produced.
  - The phases can be repeated many times (i.e. iterations), producing one or many prototypes or artefacts.
  - In each of the iterations other workflows can be addressed and involved.

Static workflows in the Rational Unified Process

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.

Static workflows in the Rational Unified Process

Workflow	Description
Testing	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 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP good practice

- Develop software iteratively
  - Plan increments based on customer priorities and deliver highest priority increments first.
- Manage requirements
  - Explicitly document customer requirements and keep track of changes to these requirements.
- Use component-based architectures
  - Organize the system architecture as a set of reusable components.

RUP good practice

- Visually model software
  - Use graphical UML models to present static and dynamic views of the software.
- Verify software quality
  - Ensure that the software meet’s organizational quality standards.
- Control changes to software
  - Manage software changes using a change management system and configuration management tools.

End of class

- Get the material from
  - http://www.marenglenbiba.net/foundprog-se/
  - Sufficient for FP exam purposes