15-413

Software Life Cycle

Bernd Bruegge

Department of Computer Science

Carnegie Mellon University

5 October 1999

Outline of Lecture

- **Software Life cycle**
  - **Waterfall model and its problems**
    - **Pure Waterfall Model**
    - **V-Model**
    - **Sawtooth Model**
  - **Alternative process models**
    - **Boehm’s Spiral Model**
    - **Issue-based Development Model (Concurrent Development)**
- **Software Development Activities**
- **Software Development Roles**

Odds and Ends

- **The client is not responding. What is going on?**
- **We are approaching system design. Managerial and technical challenges**
- **Major issues in managing system design**
  - **The initial subsystem decomposition is usually wrong**
  - **The planned design window for technology enablers is usually wrong**
- **Two phenomena**
  - **Some of the initial subsystems don't have enough "meat" => Subsystem merger**
  - **Important subsystems have been overlooked => Identification of new subsystems**

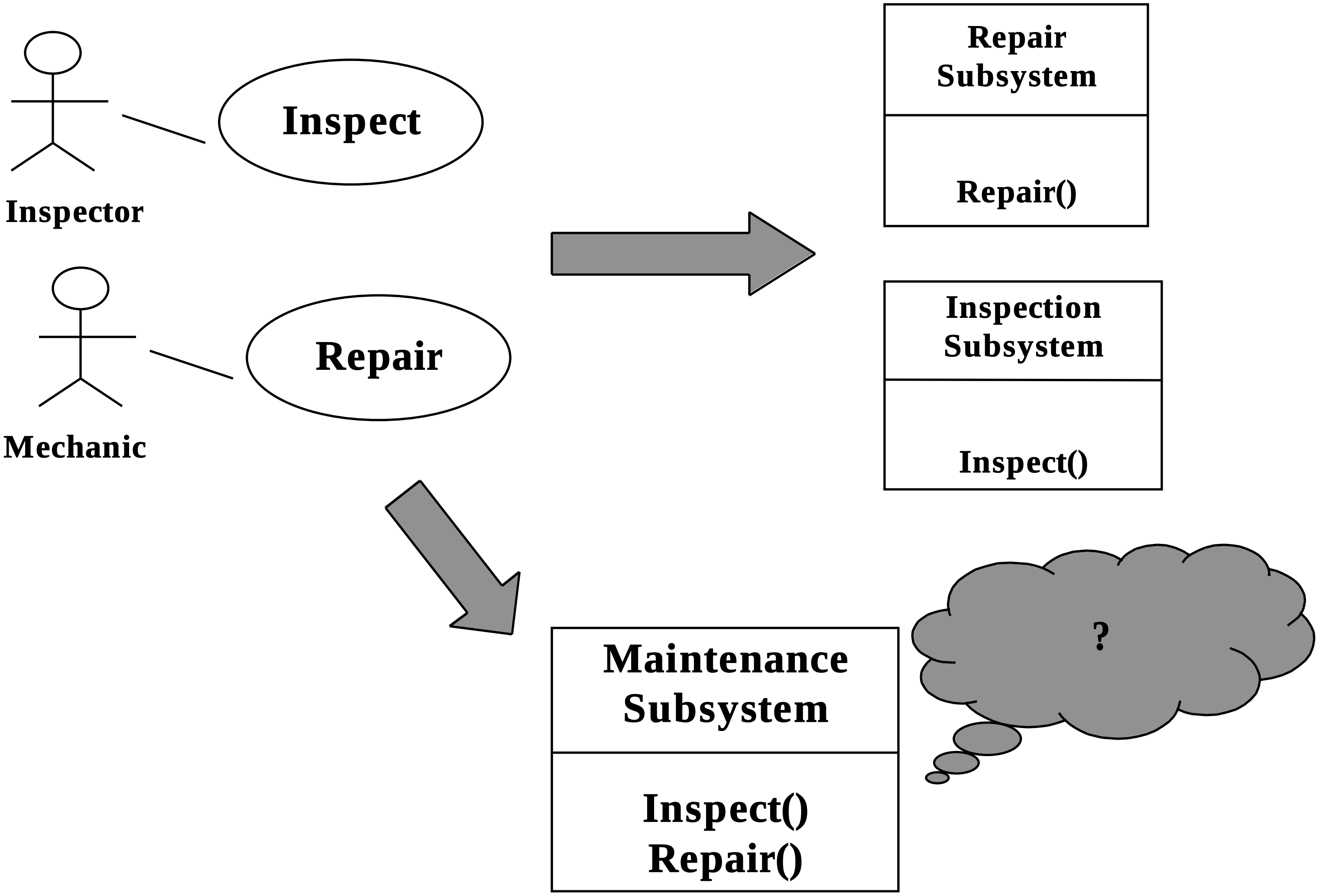
"Leftover team Syndrome"

"Orphan Object Syndrome"

The leftover team syndrome

- Two subsystems are merged (collapsed) into one
- However, the team structure stays the same
  - Too much organizational effort to redo the team organization
- STARS example:
  - The appearance of the maintenance subsystem from the rubble of inspection and repair subsystems
  - Inspection and repair team organization stays

Analysis Review leads to a Revision of the System Design (new Subsystem Decomposition)



Questions to be answered by the Project Manager

- Who is responsible for the new subsystem?
- How can the labor be assigned?
- Should there be a new team “Maintenance”
- What happens to “Inspection” and “Repair?
- Should we keep the old organization chart?
- Should there be one large team meeting, or shall we continue with two team meetings?
- Should we schedule additional team meetings?
- Should there be one or two API liaisons?

The Orphan subsystem syndrome

- **A new subsystem is appearing**
- **Who is responsible for the formulation of the use cases, for the identification of objects, who will implement it?**
- **STARS example:**
  - **Work order seem more and more to be a central issue in STARS?**
  - **Who is responsible for Workorder?**
  - **How extensible to we have to model Work Orders?**
    - **Management says, “Forget work orders for now, let’s do it in the next iteration”. Can we afford to be so sloppy?**

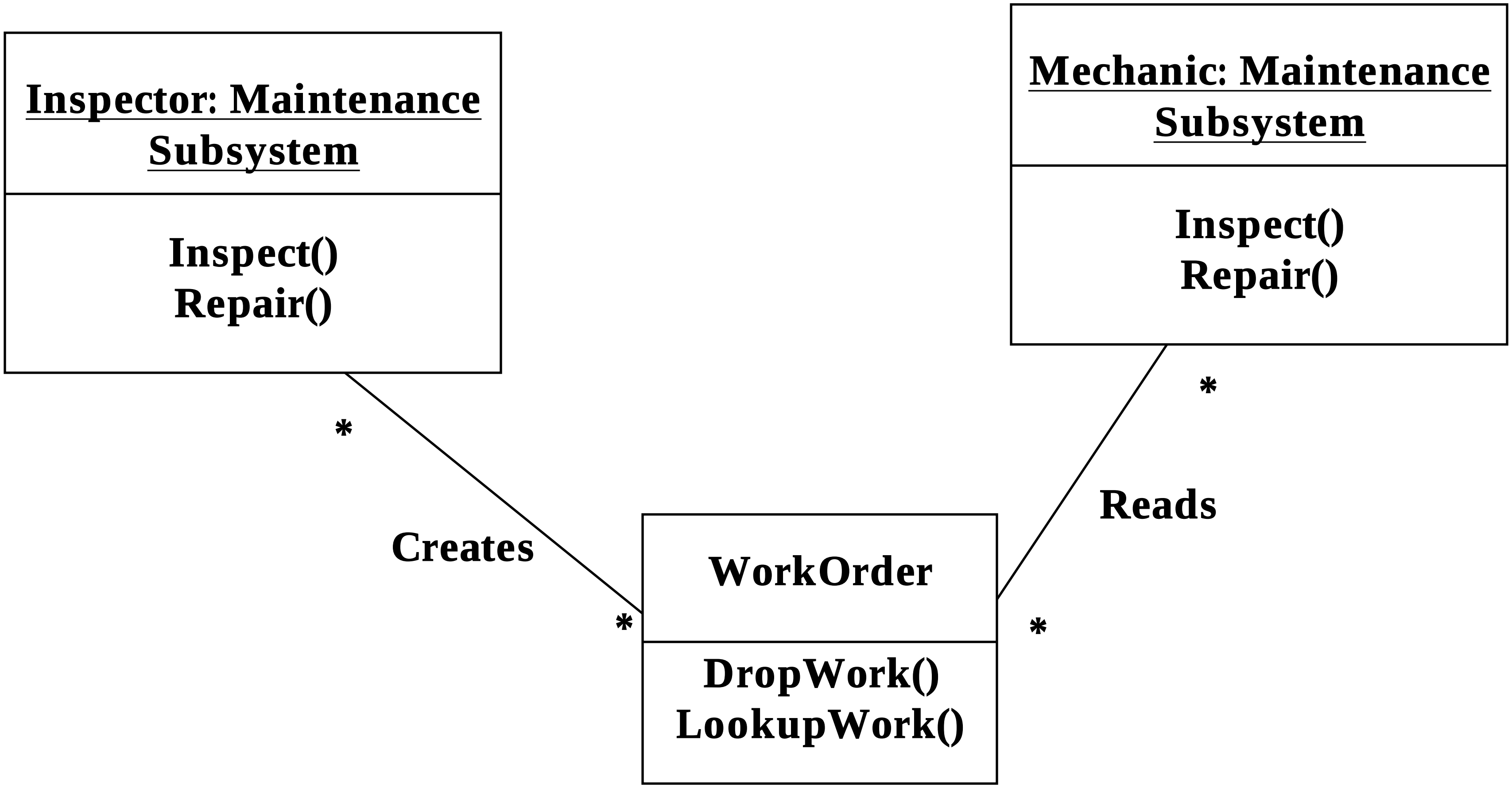
Workorder in STARS

- Keep it simple
- Identify different beasts in the zoo (Emergency Workorder, Scheduled Workorder, Preventive Workorder, Condition-Based Workorder)
- Establish a taxonomy with a superclass
- Concentrate on one subclass for the project
- Implement rudimentary methods for Dropping and Extracting Tasks into the Workorder
- Don’t do a calendar object

The shrinking design window syndrome

- Design window
  - The time interval after which design issues have to be resolved
- Products are promised but don’t materialize
  - The product turns out to be too complex
  - The product turns out to be useless
  - A critical subsystem provider goes bankrupt.
- Issues:
  - How long can we stretch the design window?
  - Buy or build?
    - Pros of building software
    - Cons of building software
  - How can we get buy without not building when the design window is closed but nothing got delivered?

Modeling WorkOrders



Manager Questions?

- **Should Workorder be a Subsystem or only a class?**
- **Should the creation of Workorders cause a push notification? Should Workorders be pulled?**
- **We have been told by the client, that we should not model a work order system:**
  - **What is the simplest scenario that we can concentrate on during 15-413?**
- **Use a observer pattern for the work order (patterns are discussed in the next lectures on system design)**

Inherent Problems with Software Development

- **Requirements are complex**
  - **The client usually does not know all the functional requirements in advance**
- **Requirements may be changing**
  - **Technology enablers introduce new possibilities to deal with nonfunctional requirements**
- **Frequent changes are difficult to manage**
  - **Identifying milestones and cost estimation is difficult**
- **There is more than one software system**
  - **New system must often be backward compatible with existing system ("legacy system")**
  - **Phased development: Need to distinguish between the system under development and already released systems**

Definitions

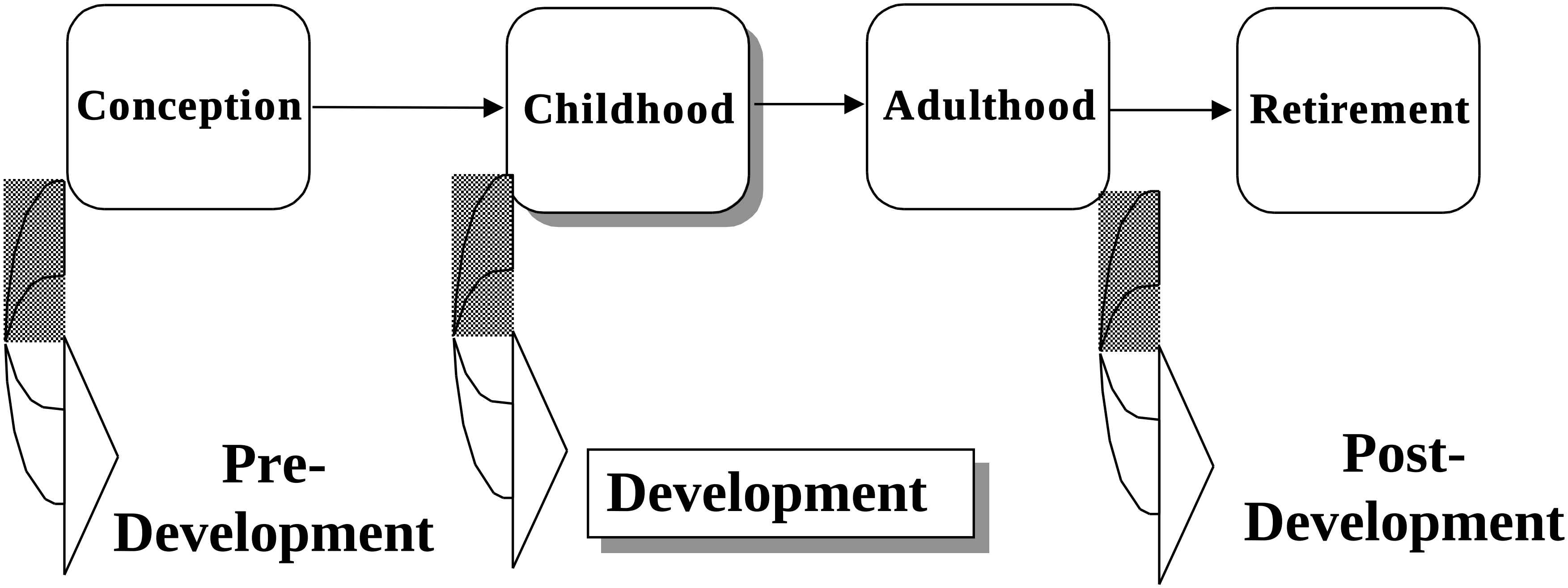
- Software lifecycle modeling: Attempt to deal with complexity and change

- Software lifecycle:
  - Set of activities and their relationships to each other to support the development of a software system

- Software development methodology:
  - A collection of techniques for building models - applied across the software lifecycle

Software Life Cycle

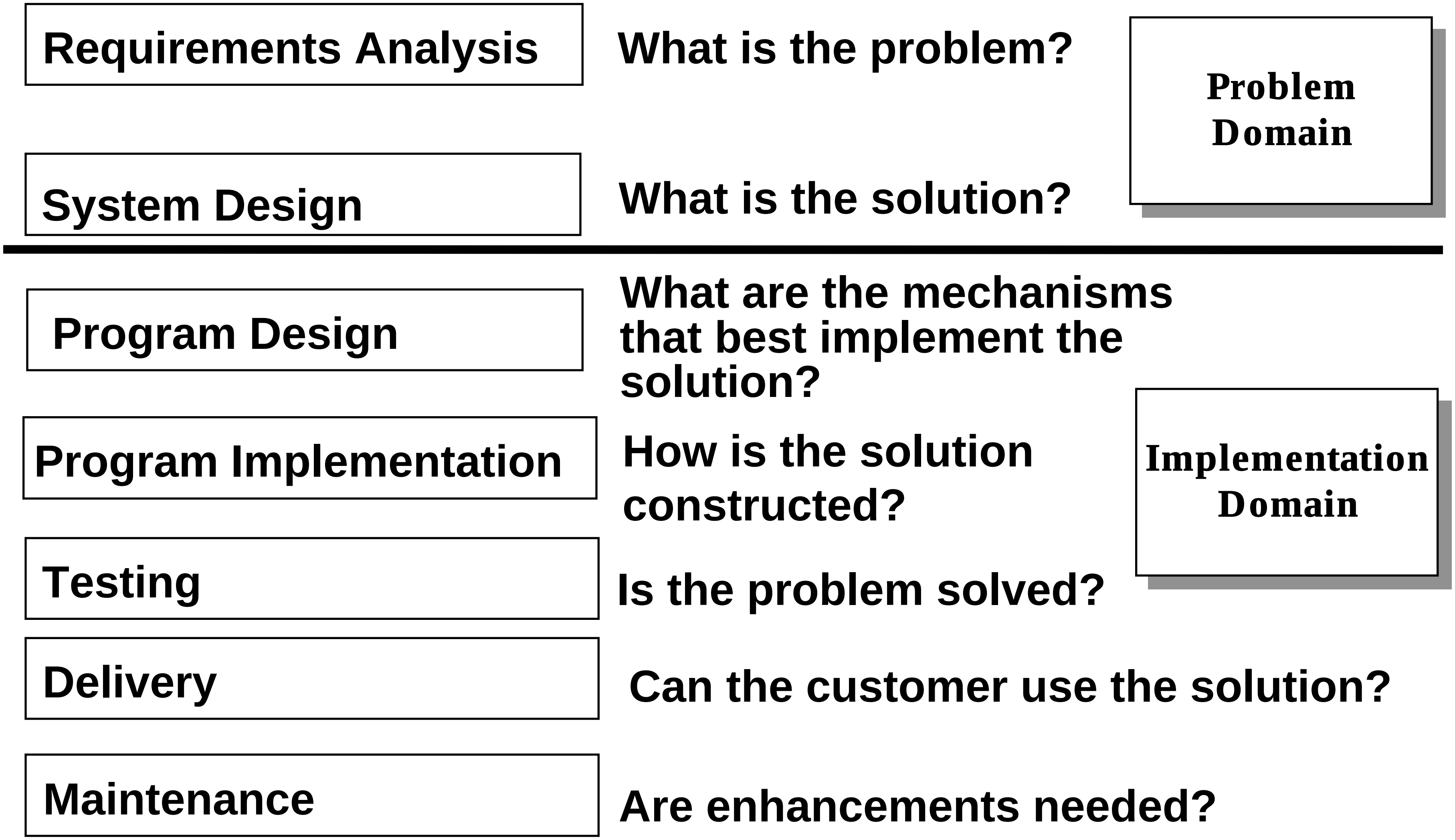
- **Software construction goes through a progression of states**



Typical Software Lifecycle Questions

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

Possible Identification of Software Development Activities



Requirements Analysis — What is the problem?

System Design — What is the solution?

Problem Domain

Program Design — What are the mechanisms that best implement the solution?

Program Implementation — How is the solution constructed?

Testing — Is the problem solved?

Implementation Domain

Delivery — Can the customer use the solution?

Maintenance — Are enhancements needed?

Alternative Identification of Software Development Activities

Requirements Analysis **What is the problem?**

Problem
Domain

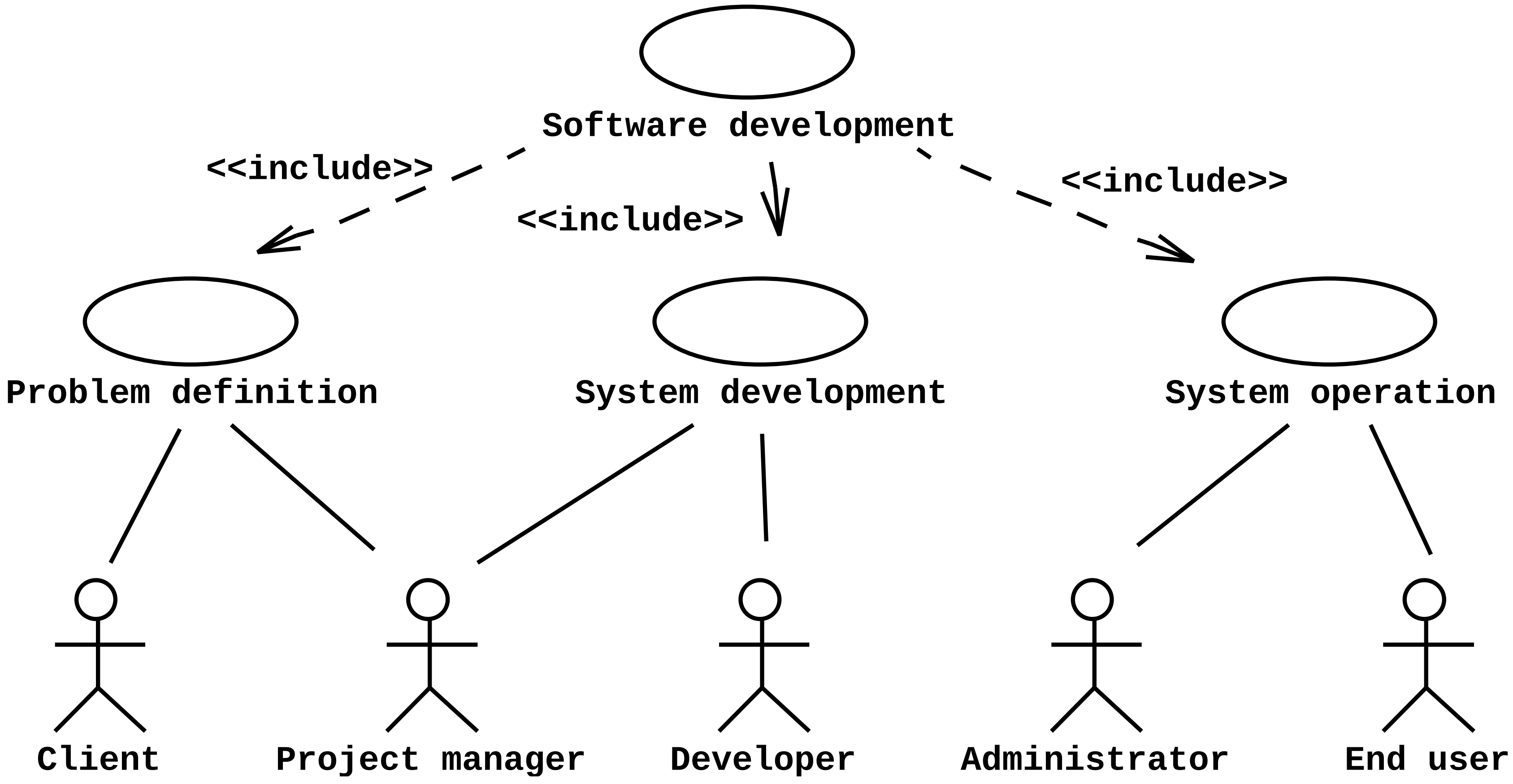
System Design **What is the solution?**

Object Design **What is the solution in the context of an existing hardware system?**

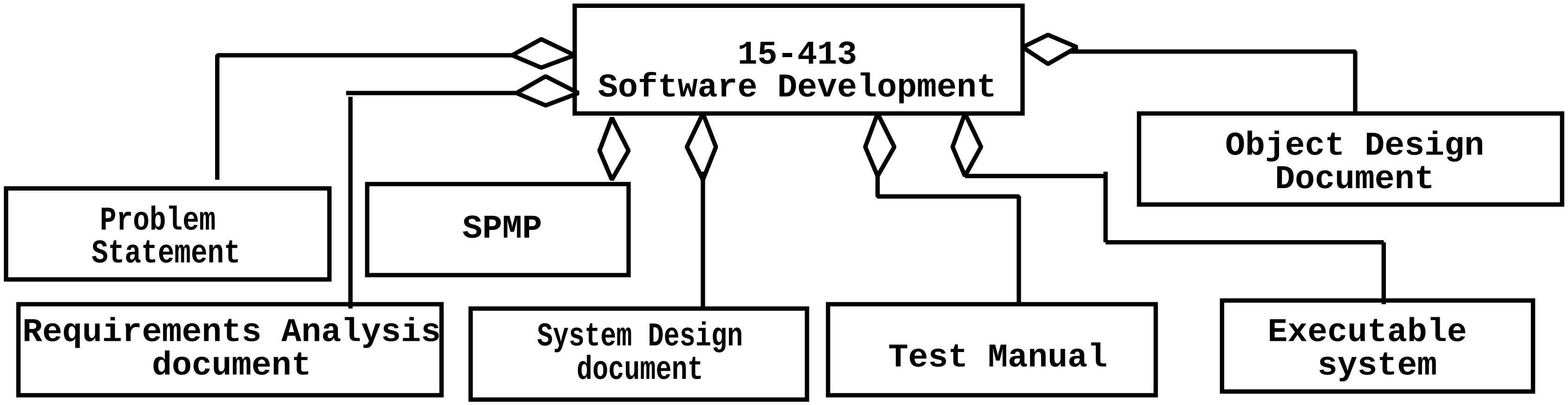
Implementation
Domain

Implementation **How is the solution constructed?**

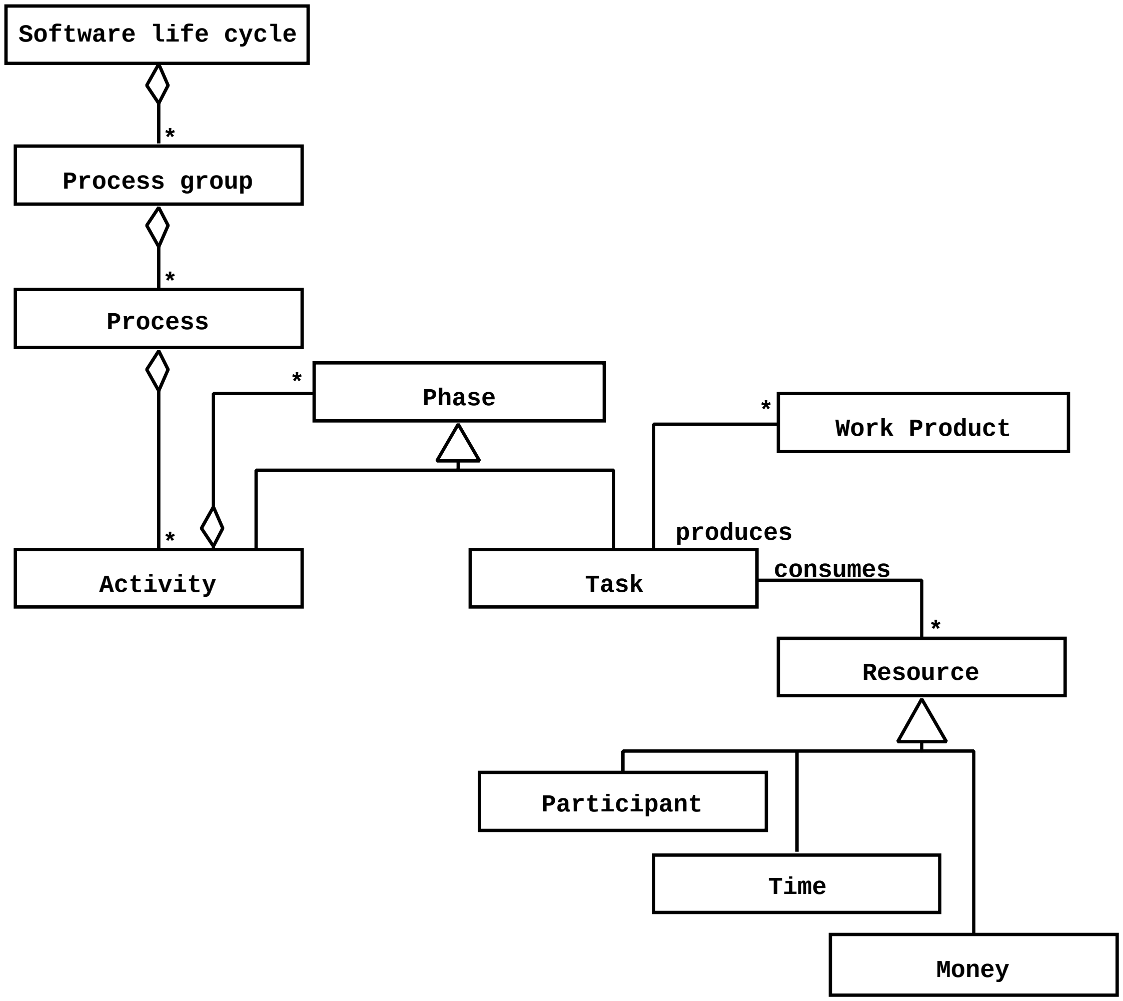
Software Development as Application Domain: A Use Case Model



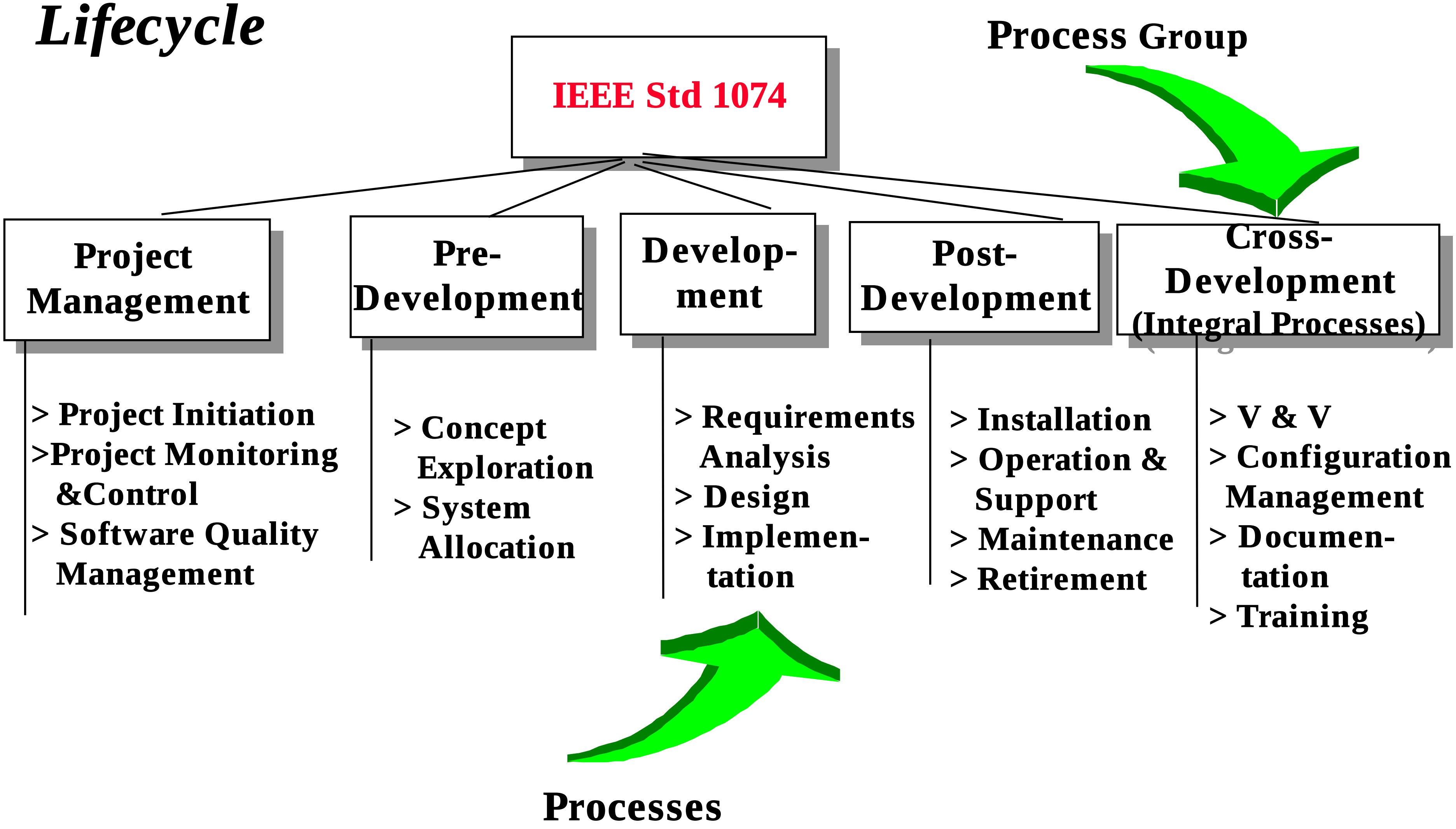
Software Development as Application Domain: Simple Object Model



General Object Model of the Software Lifecycle



IEEE Std 1074: Standard for Software Lifecycle



Process Group

IEEE Std 1074

Project Management
- > Project Initiation
- >Project Monitoring &Control
- > Software Quality Management

Pre-Development
- > Concept Exploration
- > System Allocation

Develop-ment
- > Requirements Analysis
- > Design
- > Implemen-tation

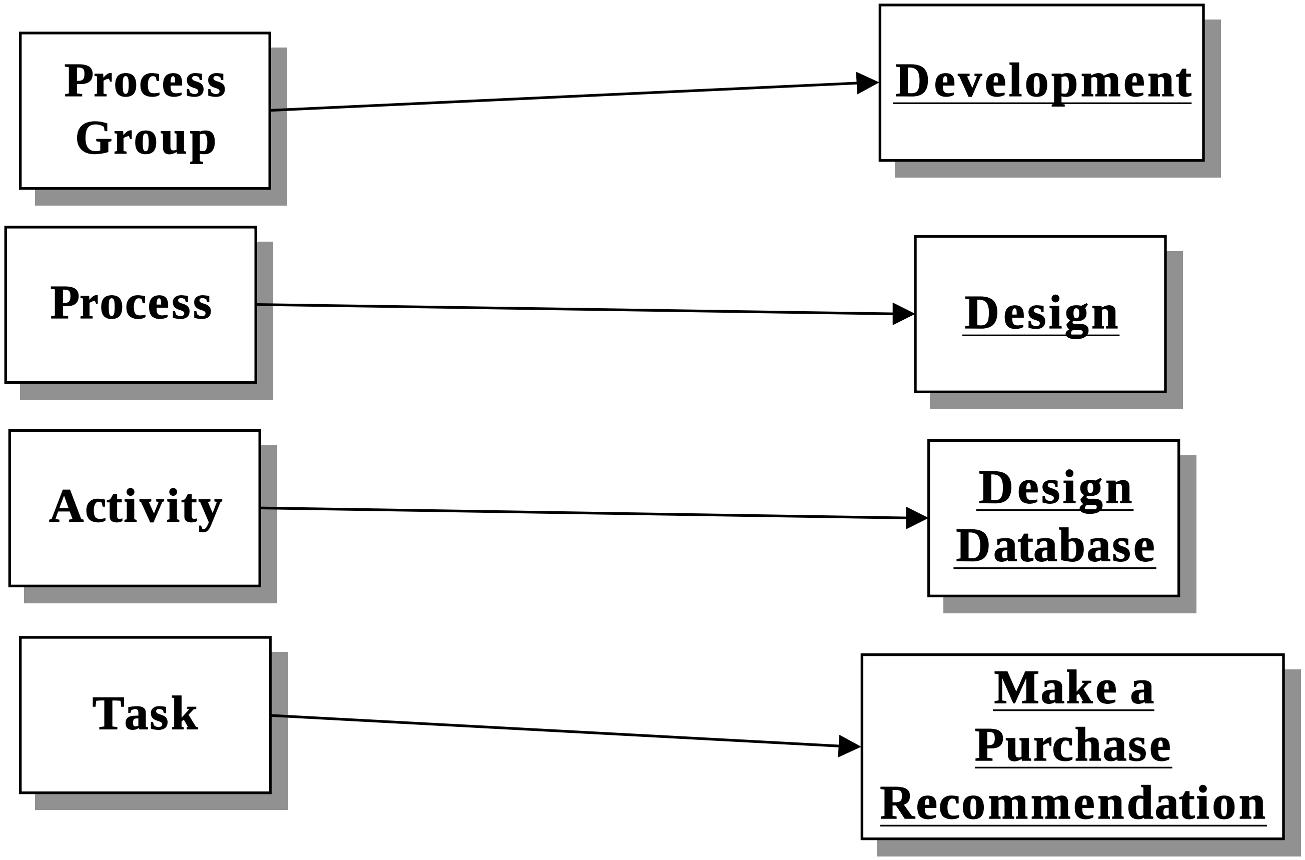
Post-Development
- > Installation
- > Operation & Support
- > Maintenance
- > Retirement

Cross-Development (Integral Processes)
- > V & V
- > Configuration Management
- > Documen-tation
- > Training

Processes

Processes, Activities and Tasks

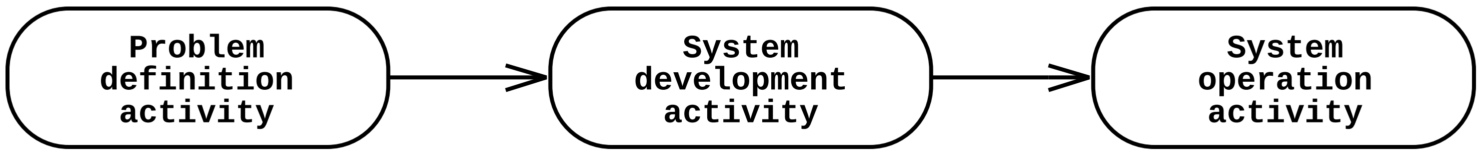
- **Process Group: Consists of Set of Processes**
- **Process: Consists of Activities**
- **Activity: Consists of sub activities and tasks**

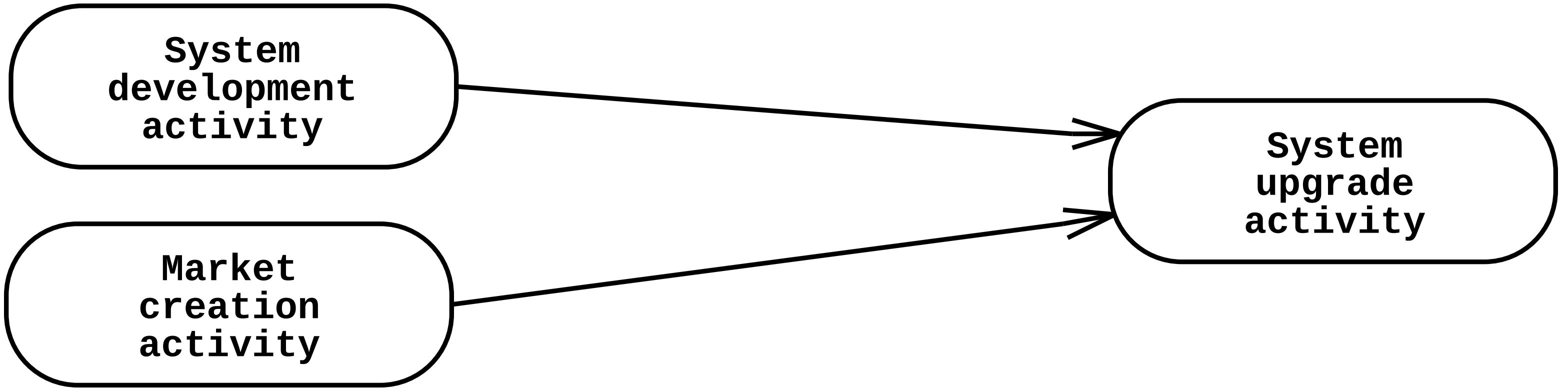


Example

- The Design Process is part of Development
- The Design Process consists of the following Activities
  - Perform Architectural Design
  - Design Database (If Applicable)
  - Design Interfaces
  - Select or Develop Algorithsm (If Applicable)
  - Perform Detailed Design (= Object Design)
- The Design Database Activity has the following Tasks
  - Review Relational Databases
  - Review Object-Oriented Databases
  - Make a Purchase recommendation
  - ....

Modeling Dependencies in a Software Lifecycle


Problem definition activity
System development activity
System operation activity

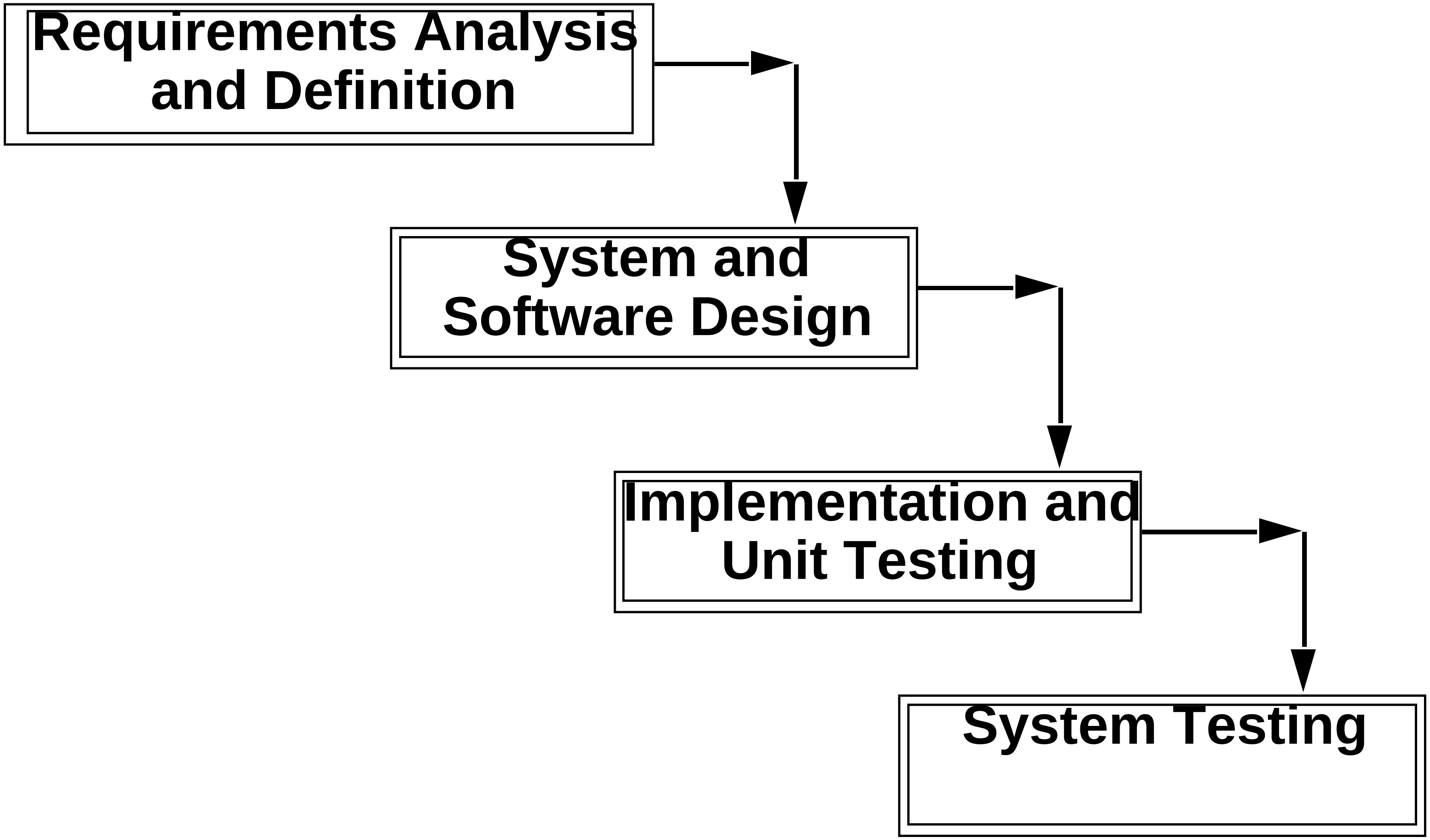


- **Note that the dependency association can mean one of two things:**
  - **Activity B depends on Activity A**
  - **Activity A must temporarily precede Activity B**
- **Which one is right?**

Life-Cycle Model: Variations on a Theme

- **Many models have been proposed to deal with the problems of defining activities and associating them with each other**
- **The waterfall model**
  - **First described by Royce in 1970**
- **There seem to be at least as many versions as there are authorities - perhaps more**

The Waterfall Model of the Software Life Cycle



Requirements Analysis and Definition

System and Software Design

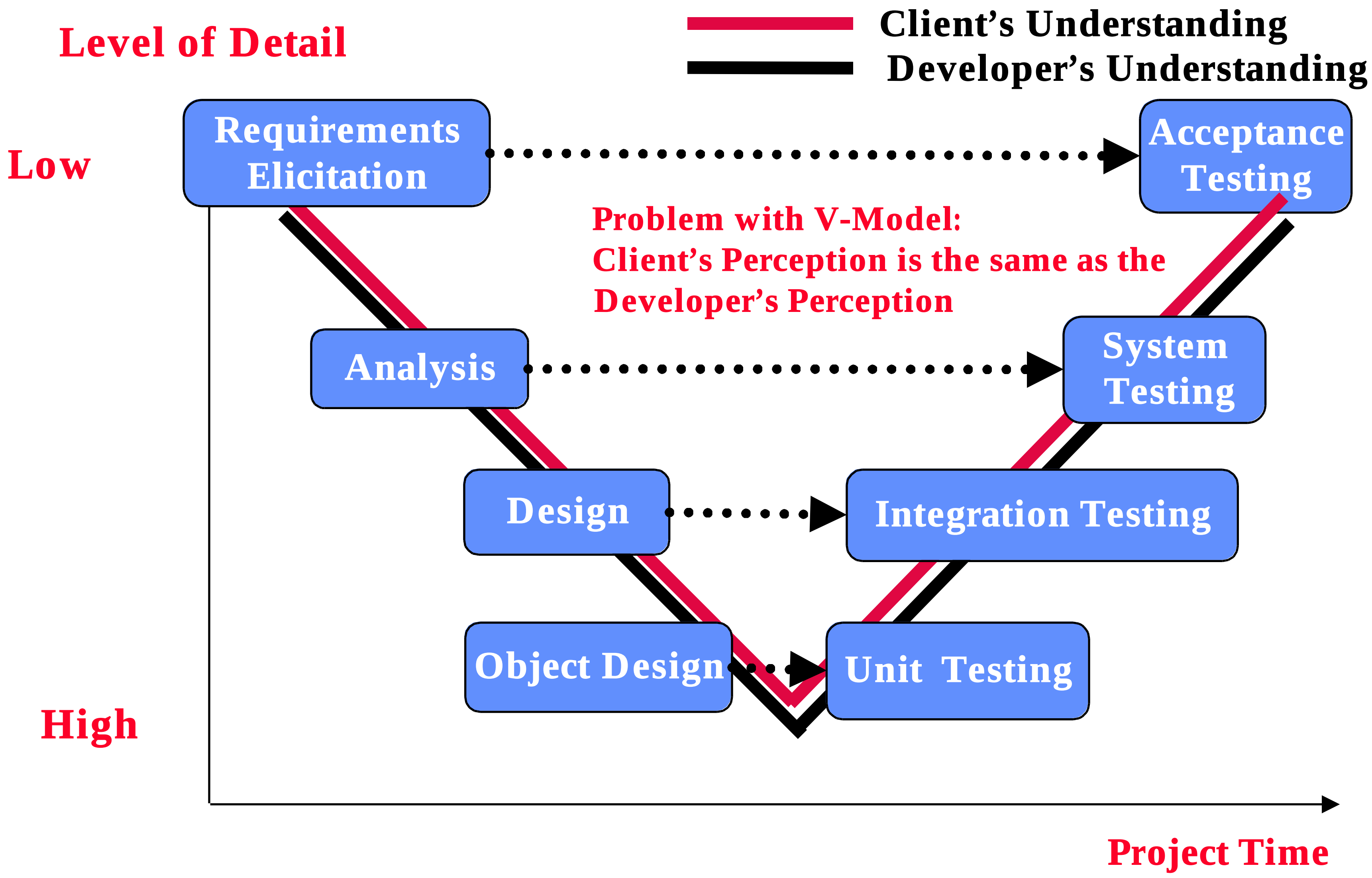
Implementation and Unit Testing

System Testing

Problems with Waterfall Model

- Managers love waterfall models:
  - Nice milestones
  - No need to look back (linear system), one activity at a time
  - Easy to check progress : 90% coded, 20% tested
- Different stakeholders need different abstractions
  - => V-Model
- 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
- System development is a nonlinear activity
  - => Issue-Based Model

V Model: Distinguishes between Development and Verification Activities

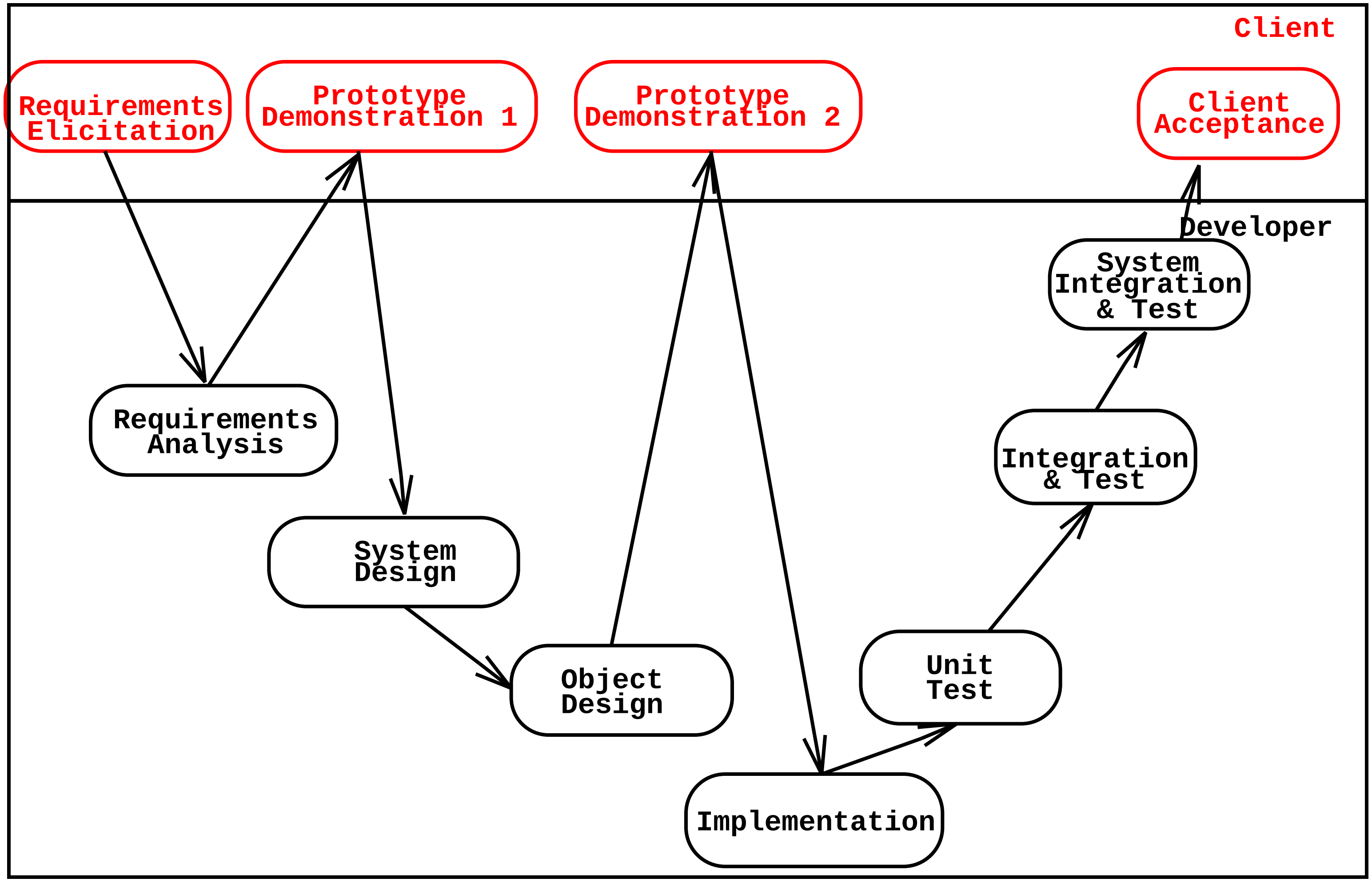


Problem with V-Model: Client's Perception is the same as the Developer's Perception

Sawtooth Model

🟥 Client's Understanding

⬛ Developer's Understanding



Client

Requirements Elicitation

Prototype Demonstration 1

Prototype Demonstration 2

Client Acceptance

Developer

Requirements Analysis

System Design

Object Design

Implementation

Unit Test

Integration & Test

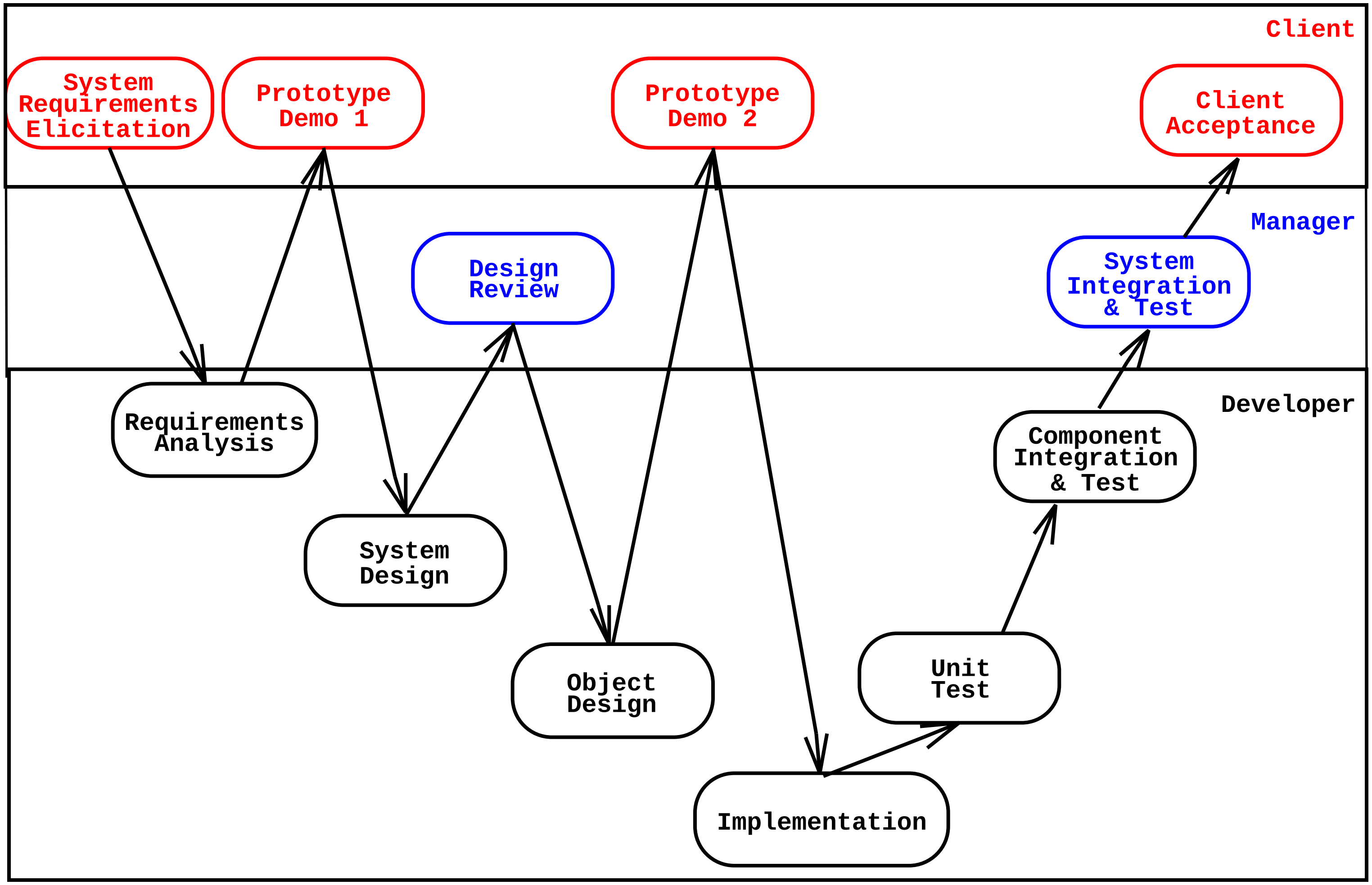
System Integration & Test

“Sharktooth” Model

🟥 **User’s Understanding**

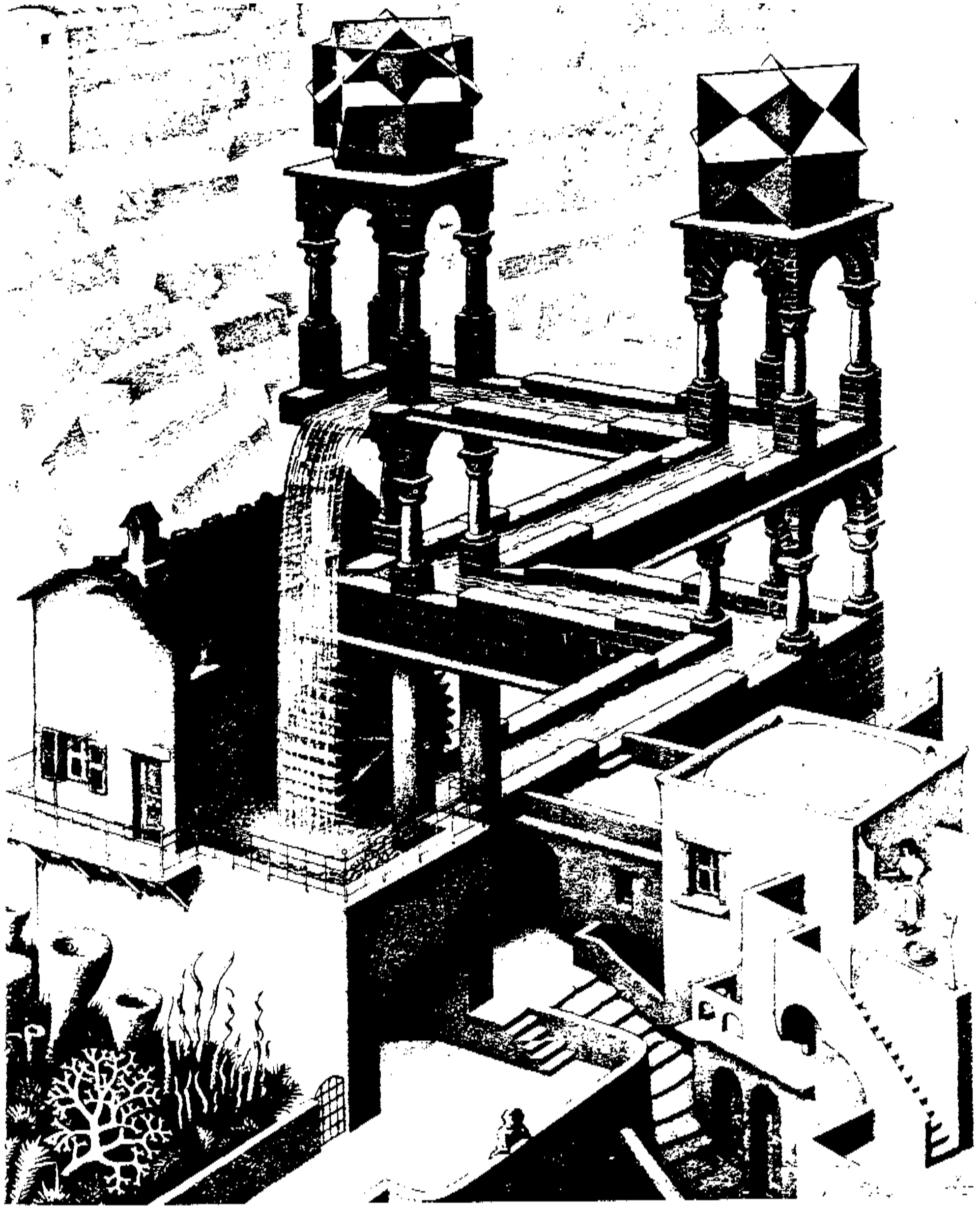
🟦 **Manager’s Understanding**

⬛ **Developer’s Understanding**



Problems with V Model

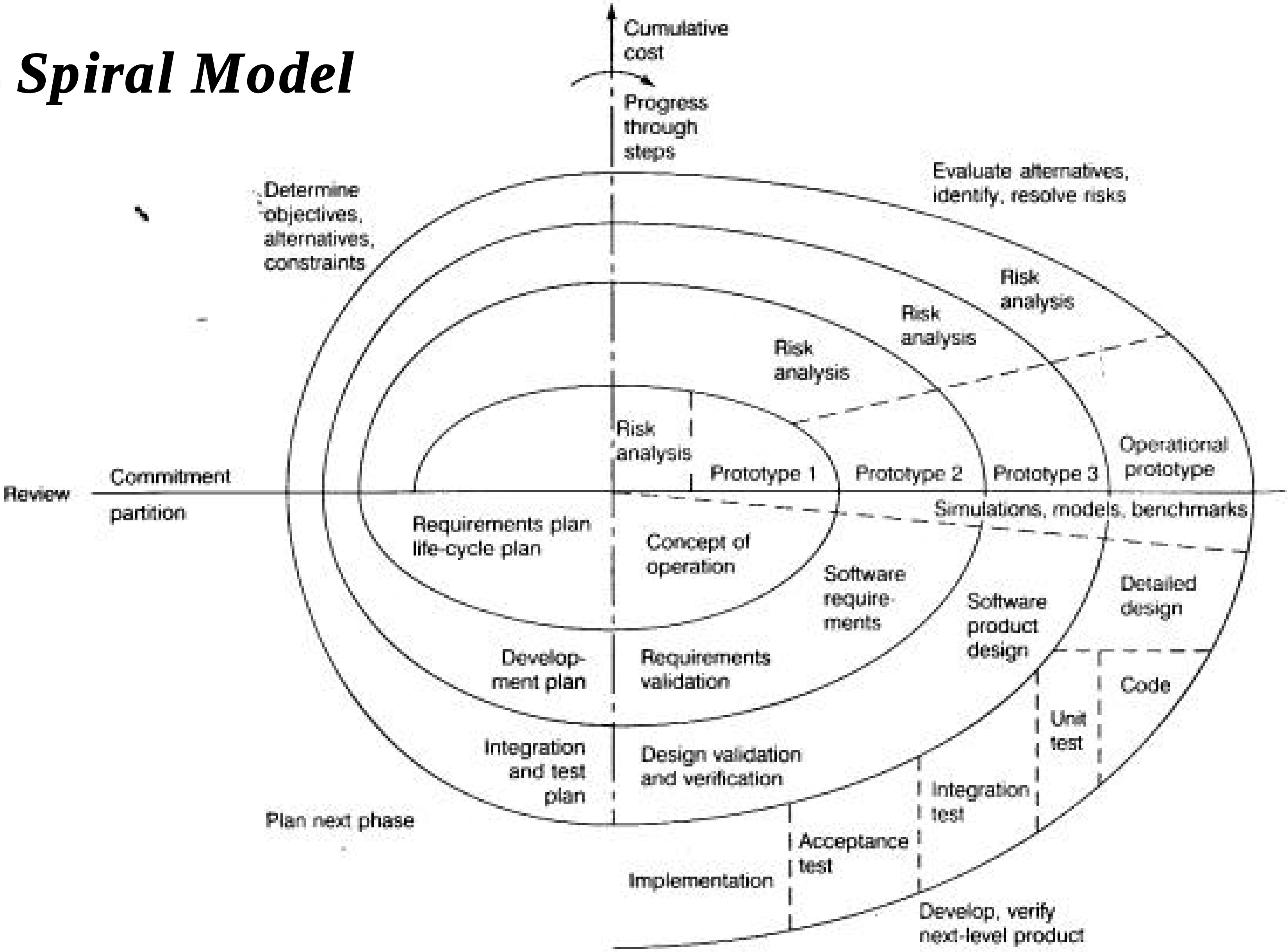
- **The V model and its variants do not distinguish temporal and logical dependencies, but fold them into one type of association**
- **In particular, the V model does not model iteration**



Spiral Model (Boehm) Deals with Iteration

- **Identify risks**
- **Assign 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

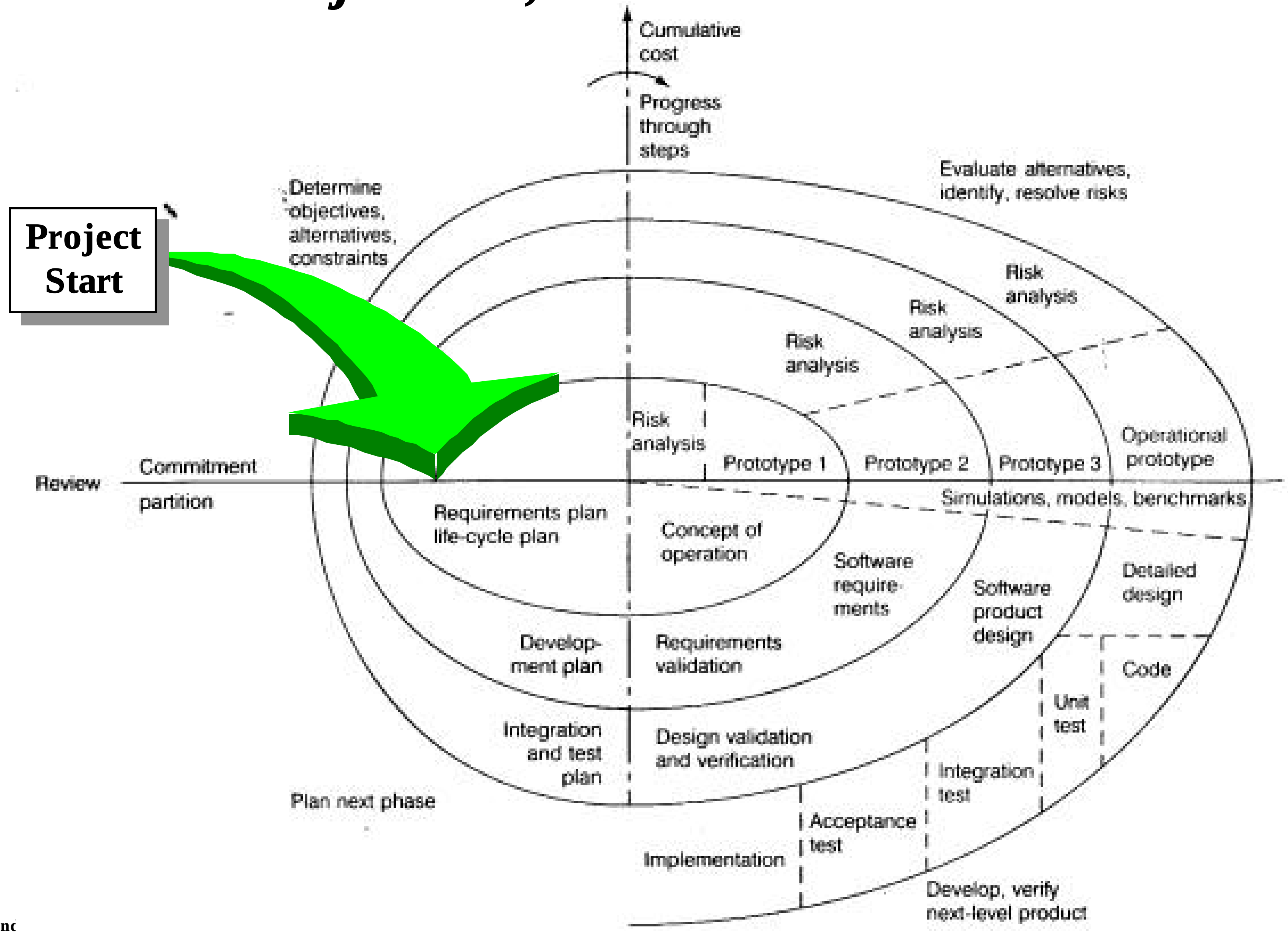


Activities ("Rounds") in Boehm's Spiral Model

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

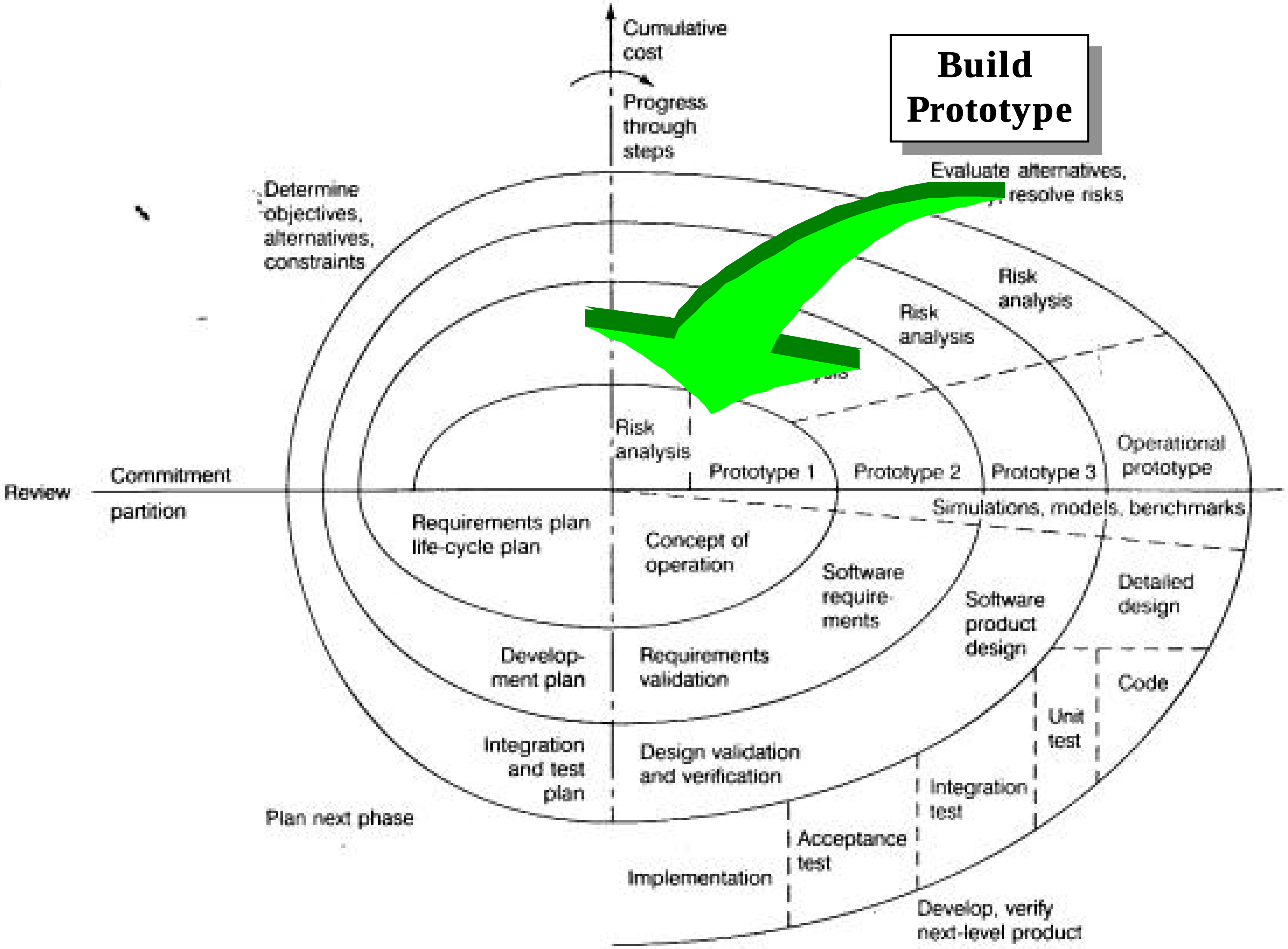
- **For each cycle go through these steps**
  - **Define objectives, alternatives, constraints**
  - **Evaluate alternative, identify and resolve risks**
  - **Develop, verify prototype**
  - **Plan next "cycle"**

Determine Objectives, Alternatives and Constraints



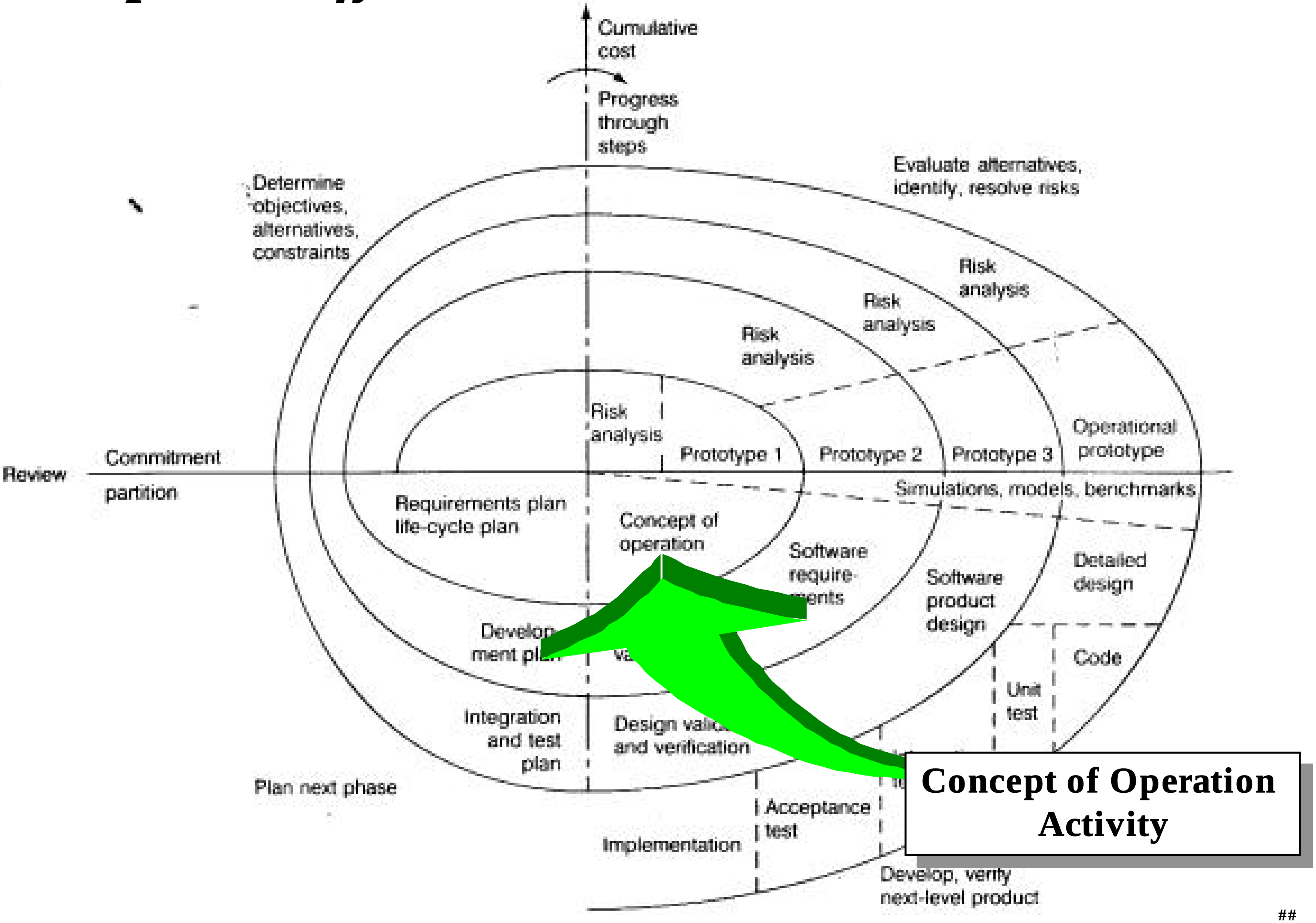
Project Start

Evaluate Alternatives, Identify, resolve risks



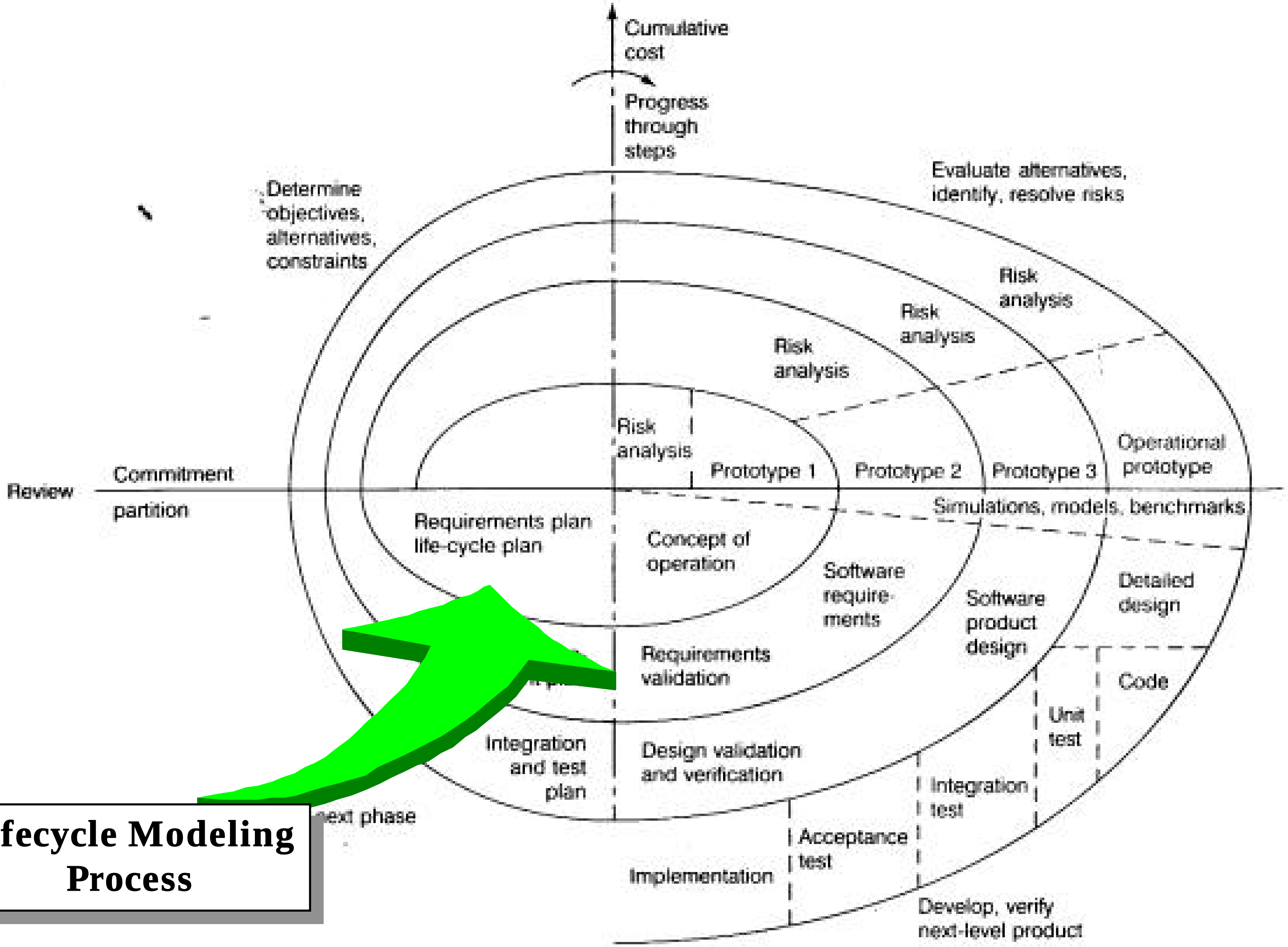
Build Prototype

Develop & Verify Product



Concept of Operation Activity

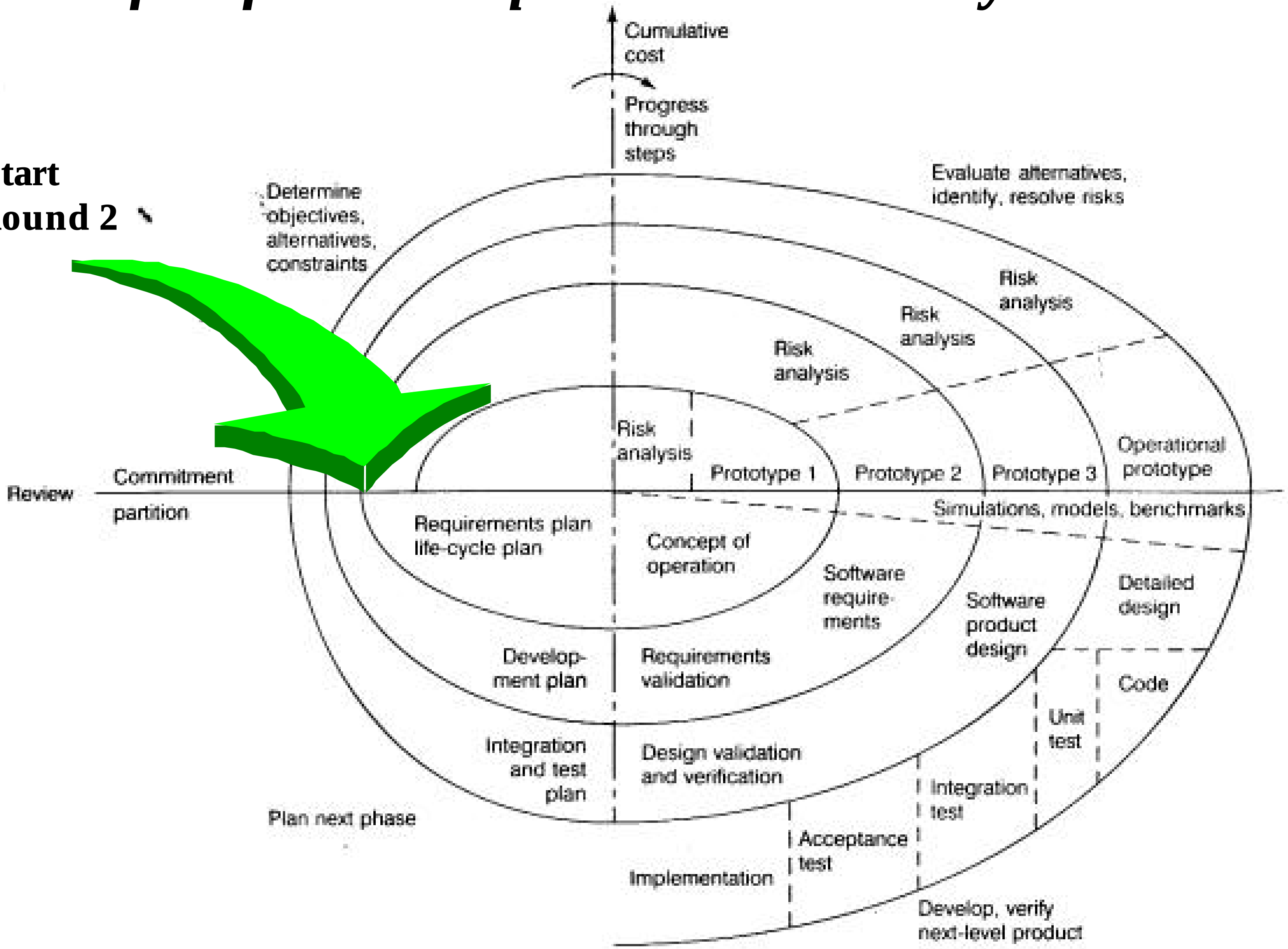
Prepare for Next Activity



Lifecycle Modeling Process

Start of Software Requirements Activity

Start of Round 2



Types of Prototypes used in the Spiral Model

- ***Illustrative Prototype***
  - **Develop the user interface with a set of storyboards**
  - **Implement them on a napkin or with a user interface builder (Visual C++, ...)**
  - **Good for first dialog with client**
- ***Functional Prototype***
  - **Implement and deliver an operational system with minimum functionality**
  - **Then add more functionality**
  - **Order identified by risk**
- ***Exploratory Prototype ("Hacking")***
  - **Implement part of the system to learn more about the requirements.**
  - **Good for paradigm breaks**

Types of Prototyping ctd

- **Revolutionary Prototyping**
  - Also called specification prototyping
  - Get user experience with a throwaway version to get the requirements right, then build the whole system
    - Disadvantage: Users may have to accept that features in the prototype are expensive to implement
    - User may be disappointed when some of the functionality and user interface evaporates because it can not be made available in the implementation environment
- **Evolutionary Prototyping**
  - The prototype is used as the basis for the implementation of the final system
  - Advantage: Short time to market
  - Disadvantage: Can be used only if target system can be constructed in prototyping language

Prototyping vs Rapid Development

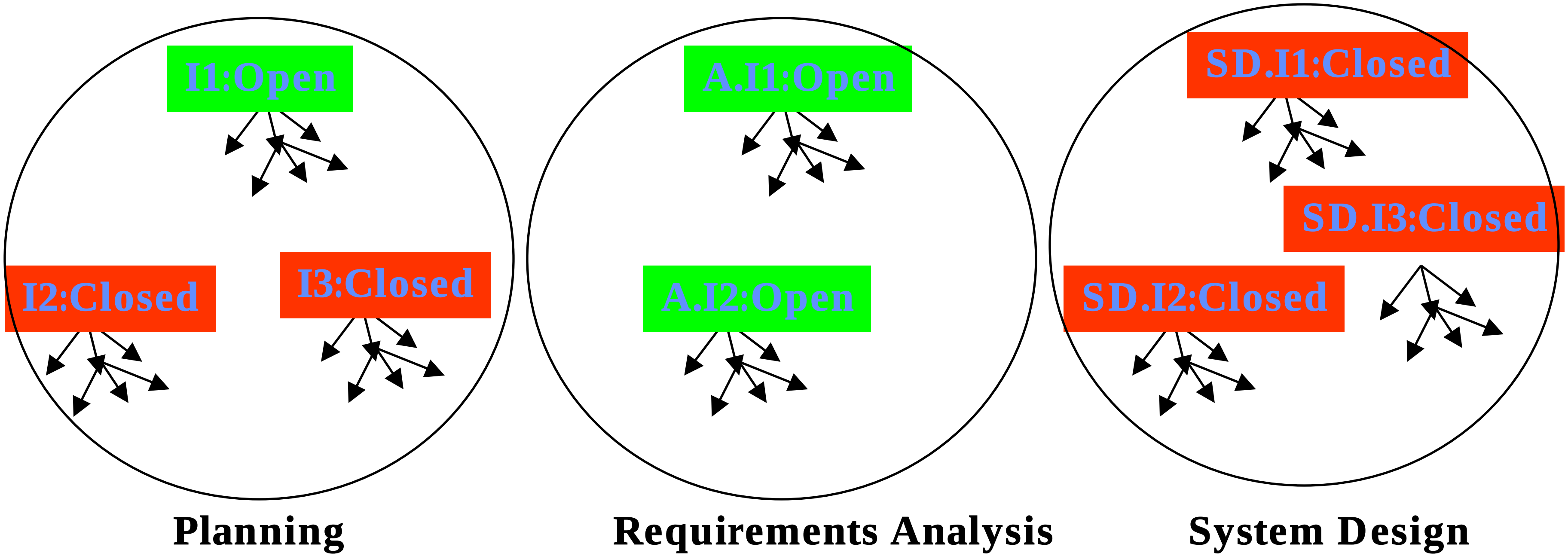
- **Revolutionary prototyping is sometimes called rapid prototyping**
- **Rapid Prototyping is not a good term because it confuses *prototyping* with *rapid development***
  - Prototyping is a technical issue: It is **a particular model in the life cycle process**
  - Rapid development is a management issue. It is a **particular way to control a project**
- **Prototyping can go on forever if it is not restricted**
  - "Time-boxed" prototyping

The Limitations of the 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”)

An Alternative: Issue-Based Development

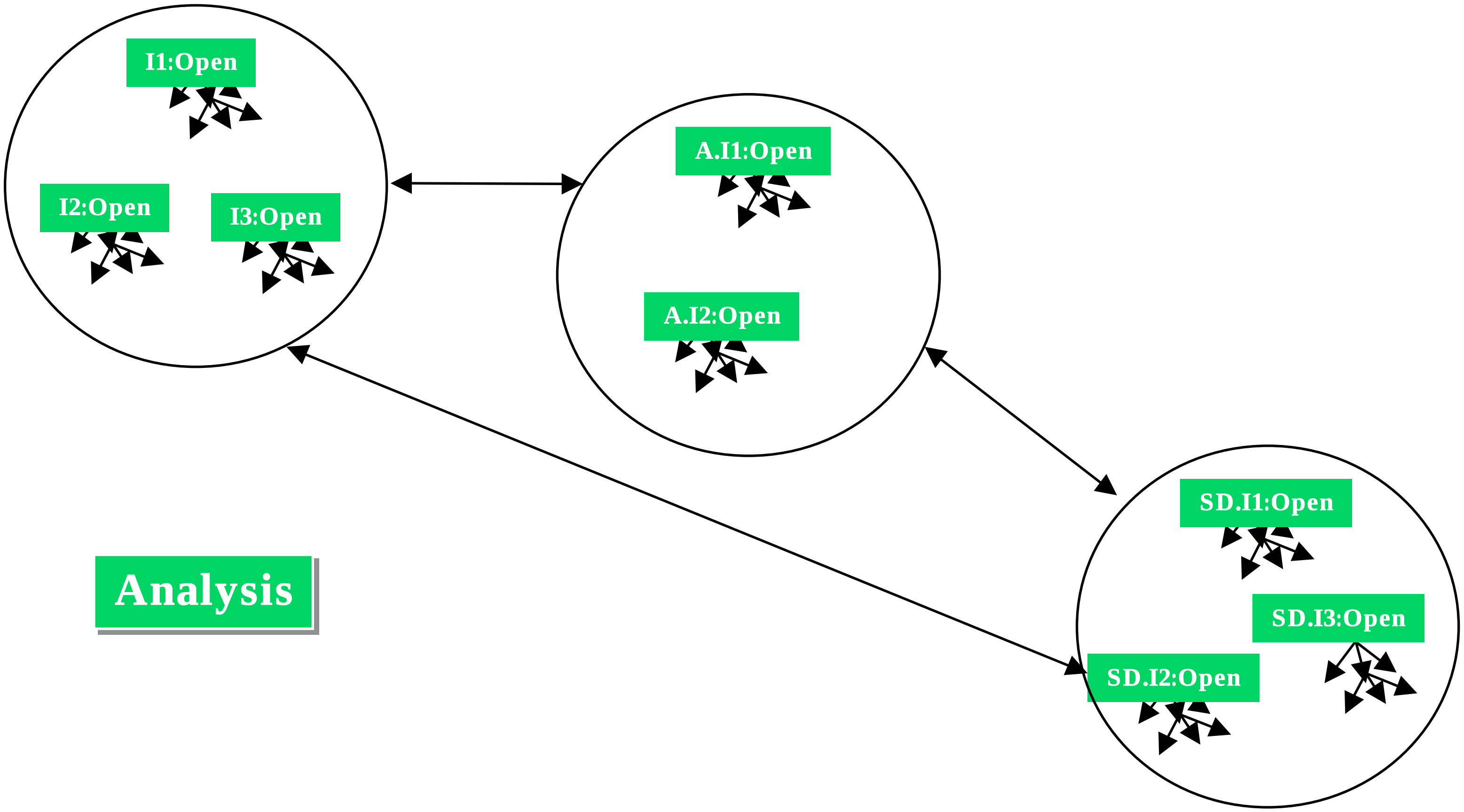
- A system is described as a collection of issues
  - Issues are either closed or open
  - Closed issues have a resolution
  - Closed issues can be reopened (Iteration!)
- The set of closed issues is the basis of the system model



Frequency Change and Software Lifeycle

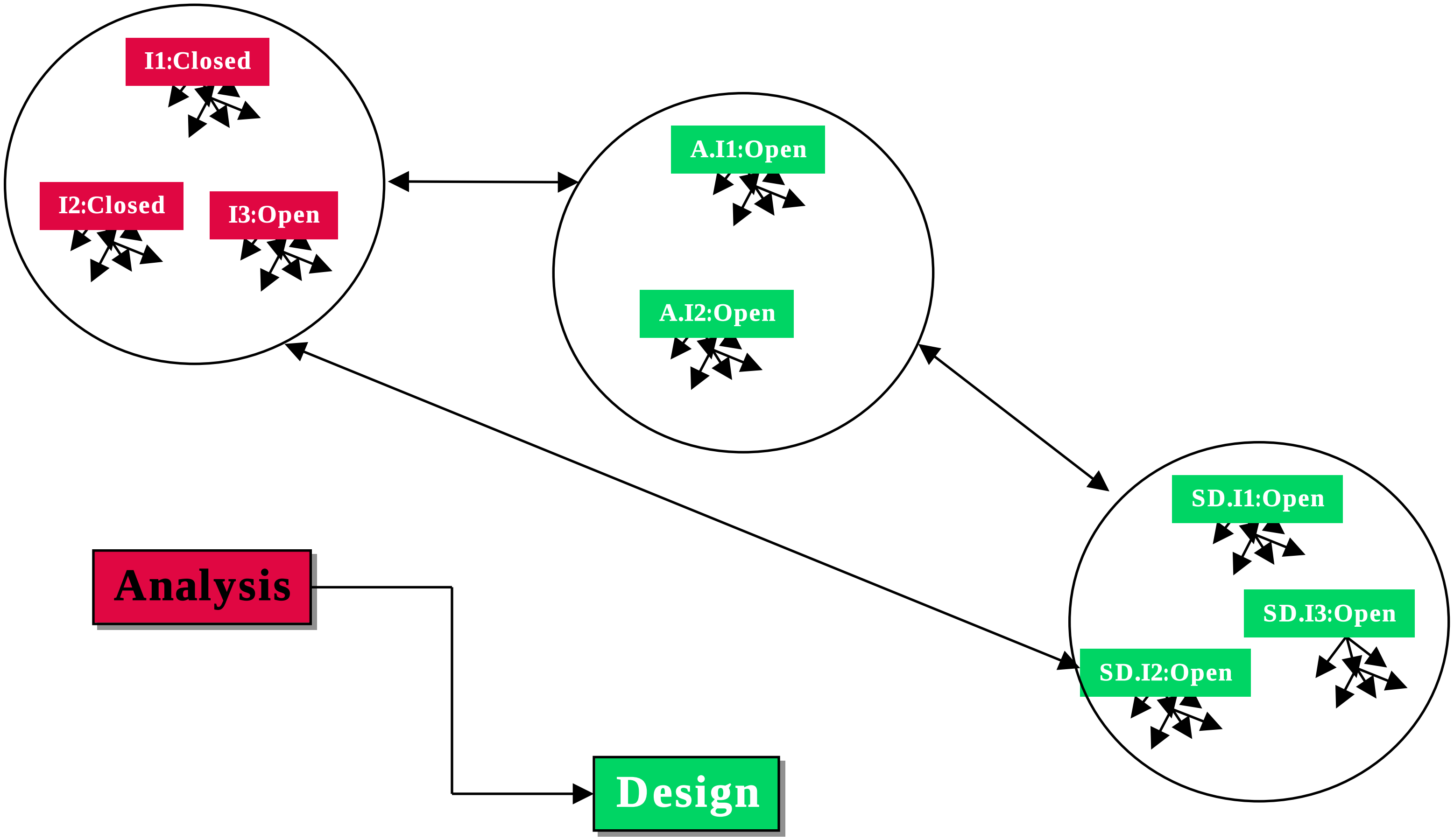
- **PT = Project Time, MTBC = Mean Time Between Change**
- **Change rarely occurs (MTBC >> PT):**
  - **Waterfall Model**
  - **All issues in one phase are closed before proceeding to the next phase**
- **Change occurs sometimes (MTBC = PT):**
  - **Boehm's Spiral Model**
  - **Change occuring during a phase might lead to an iteration of a previous phase or cancellation of the project**
- **"Change is constant" (MTBC << PT):**
  - **Issue-based Development (Concurrent Development Model)**
  - **Phases are never finished, they all run in parallel**
    - **Decision when to close an issue is up to management**
    - **The set of closed issues form the basis for the system to be developed**

Waterfall Model: Analysis Phase

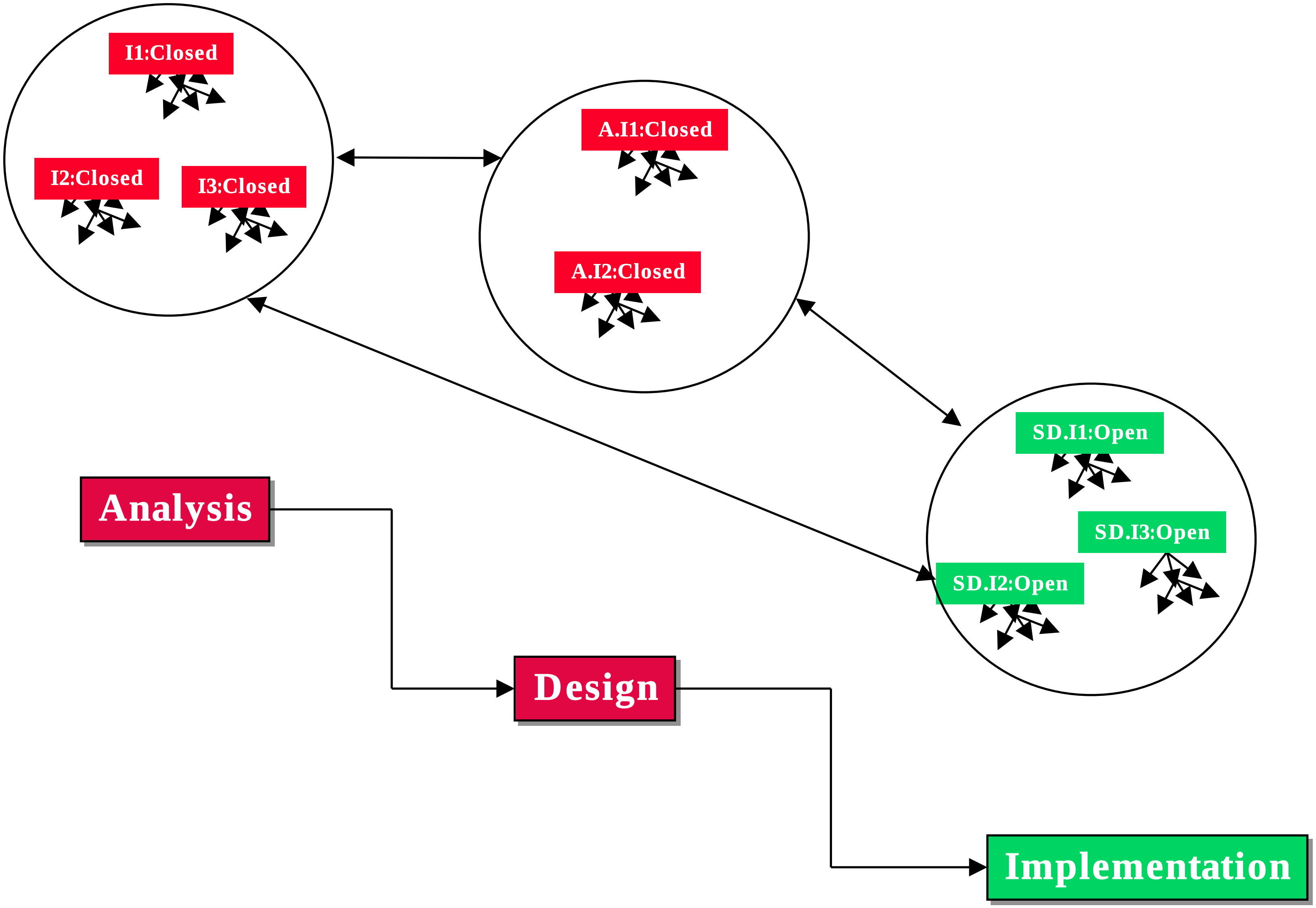


Analysis

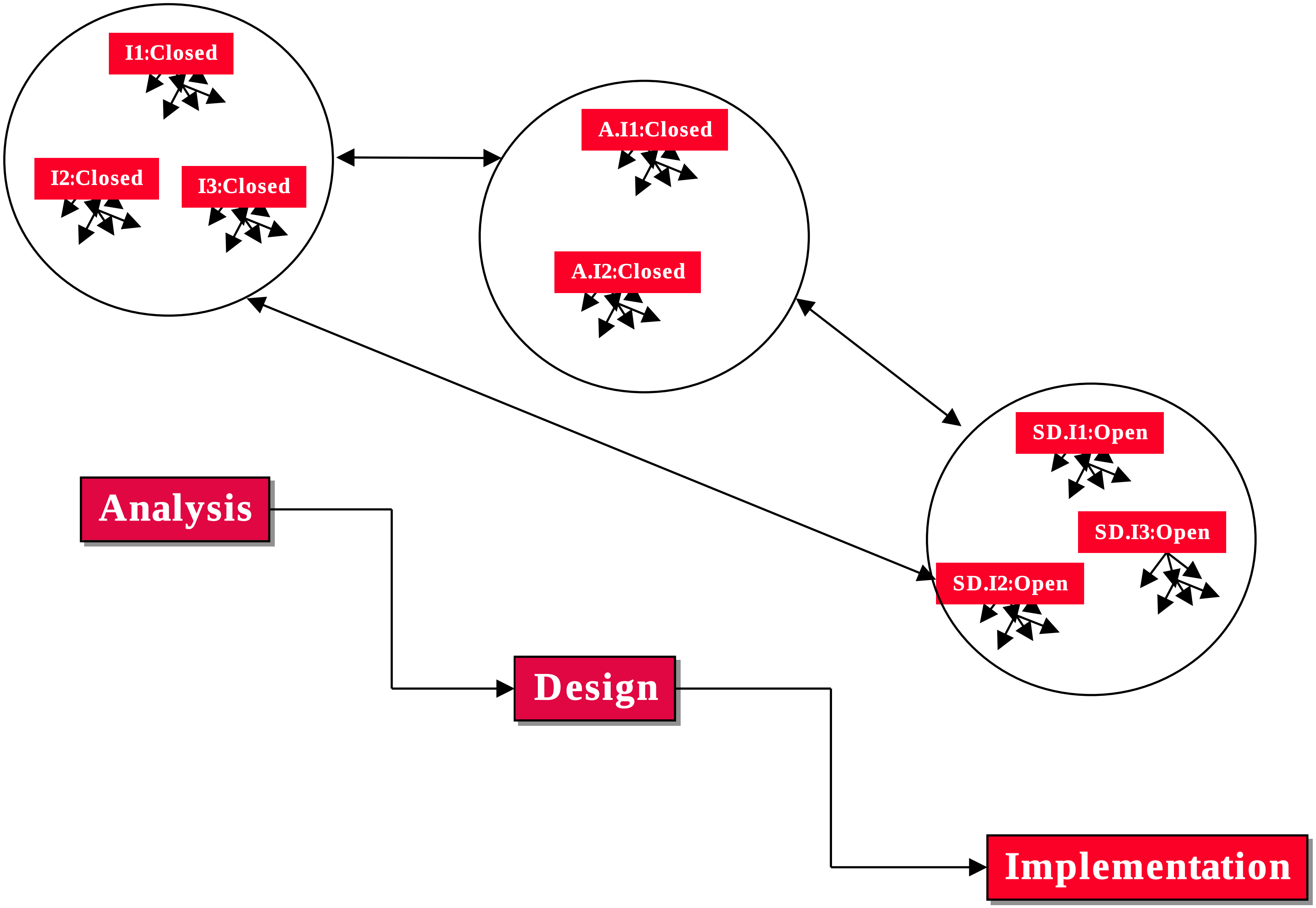
Waterfall Model: Design Phase



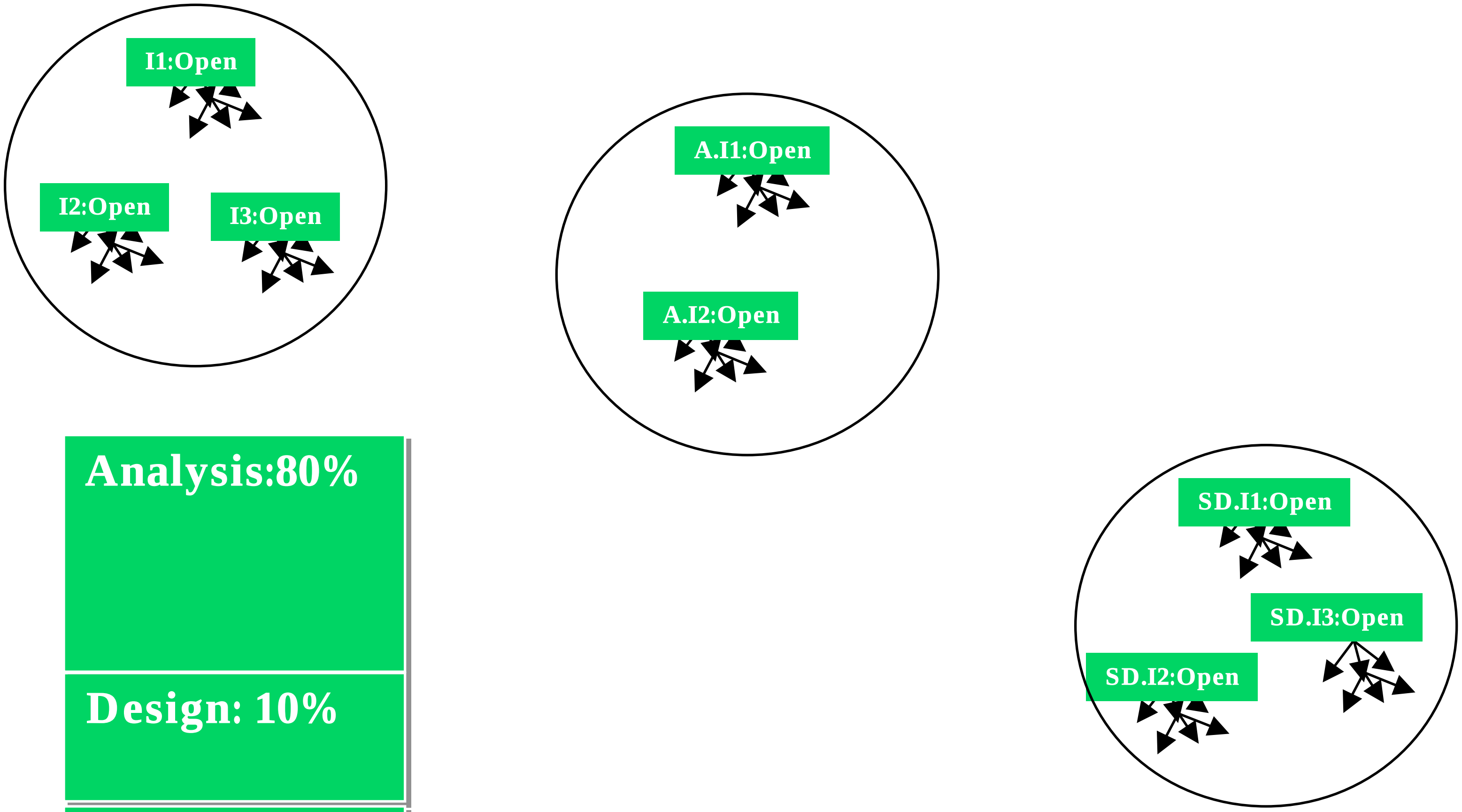
Waterfall Model: Implementation Phase



Waterfall Model: Project is Done



Issue-Based Model: Analysis Phase

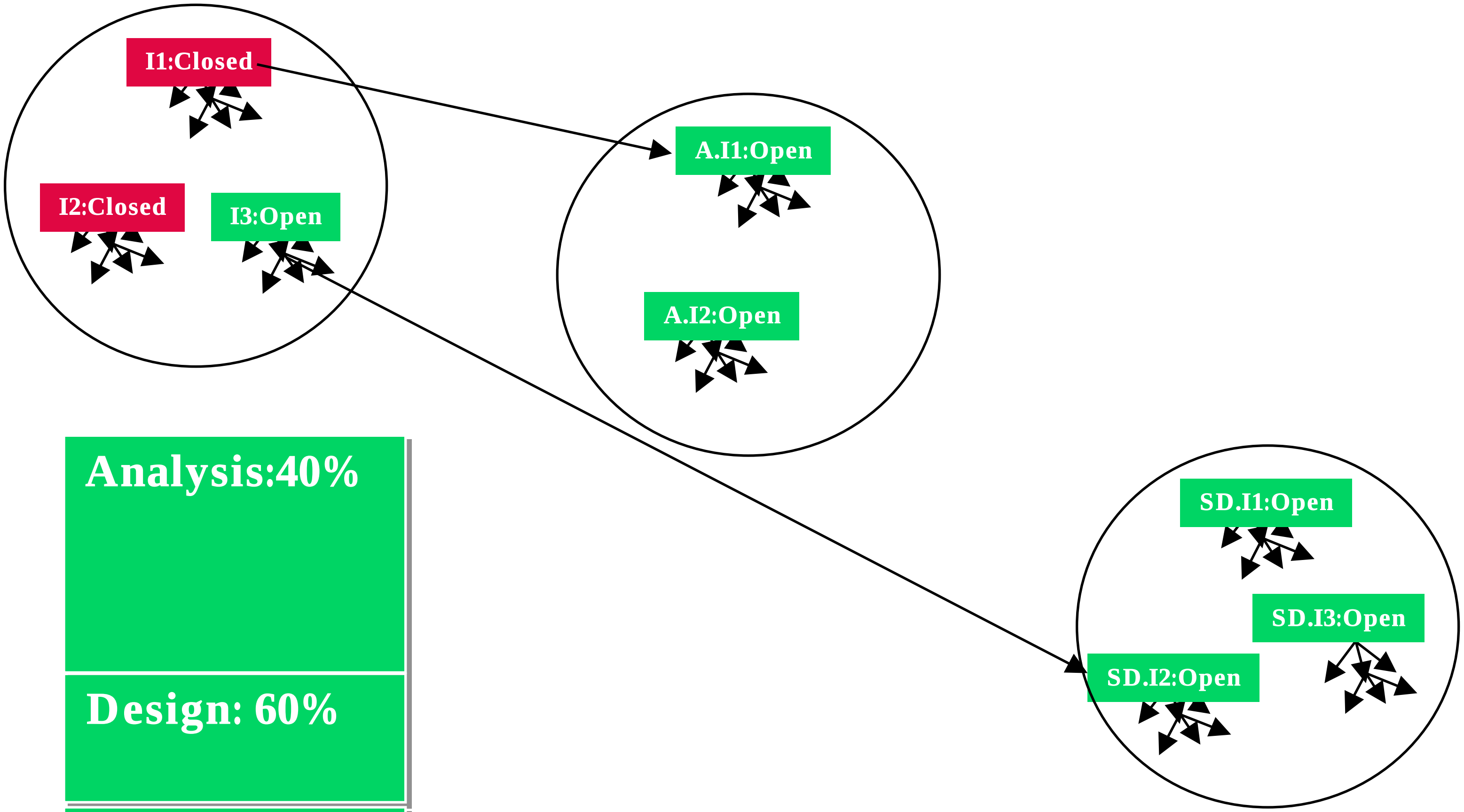


Analysis:80%

Design: 10%

Implementation: 10%

Issue-Based Model: Design Phase

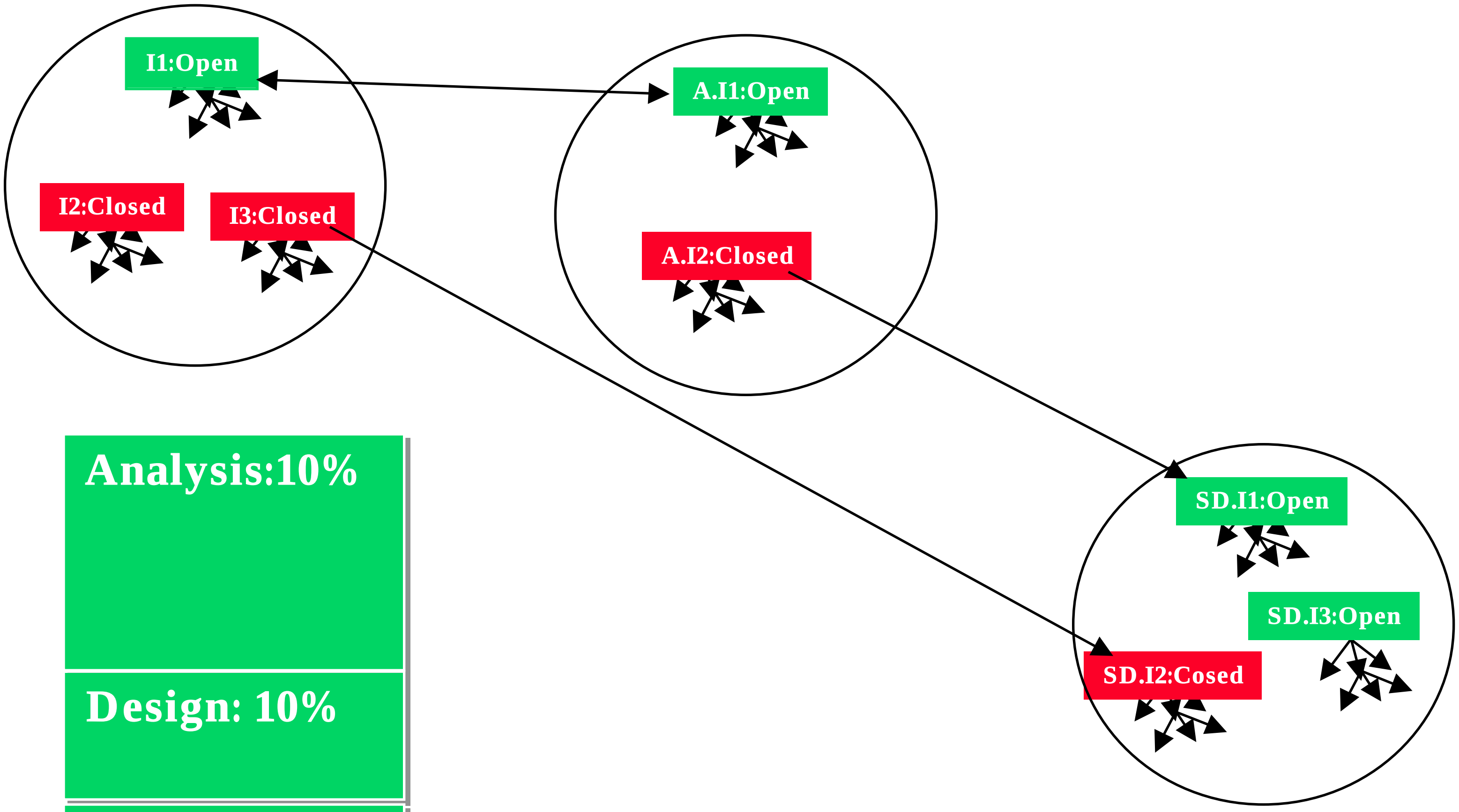


Analysis:40%

Design: 60%

Implementation: 0%

Issue-Based Model: Implementation Phase



I1:Open

I2:Closed

I3:Closed

A.I1:Open

A.I2:Closed

SD.I1:Open

SD.I3:Open

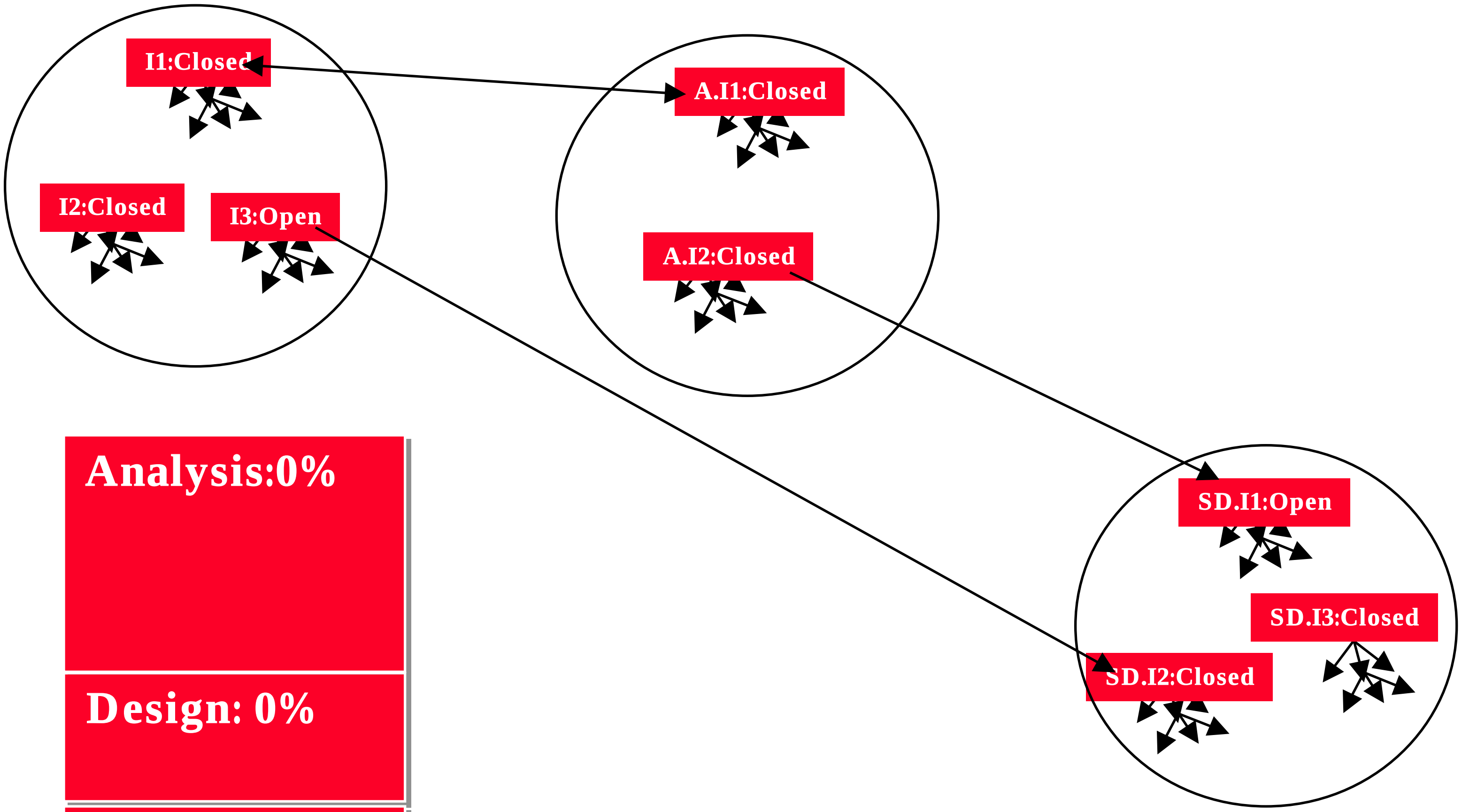
SD.I2:Cosed

Analysis:10%

Design: 10%

Implementation: 60%

Issue-Based Model: Project is Done



I1:Closed

I2:Closed

I3:Open

A.I1:Closed

A.I2:Closed

SD.I1:Open

SD.I3:Closed

SD.I2:Closed

Analysis:0%

Design: 0%

Implementation: 0%

Process Maturity

- **A software development process is mature if the development activities are well defined and if management has some control over the management of the project**
- **Process maturity is described with a set of maturity levels and the associated measurements (metrics) to manage the process**
- **Assumption: With increasing maturity the risk of project failure decreases.**

Capability maturity levels

1. Initial Level
   - **also called ad hoc or chaotic**
2. Repeatable Level
   - **Process depends on individuals ("champions")**
3. Defined Level
   - **Process is institutionalized (sanctioned by management)**
4. Managed Level
   - **Activities are measured and provide feedback for resource allocation (process itself does not change)**
5. Optimizing Level
   - **Process allows feedback of information to change process itself**