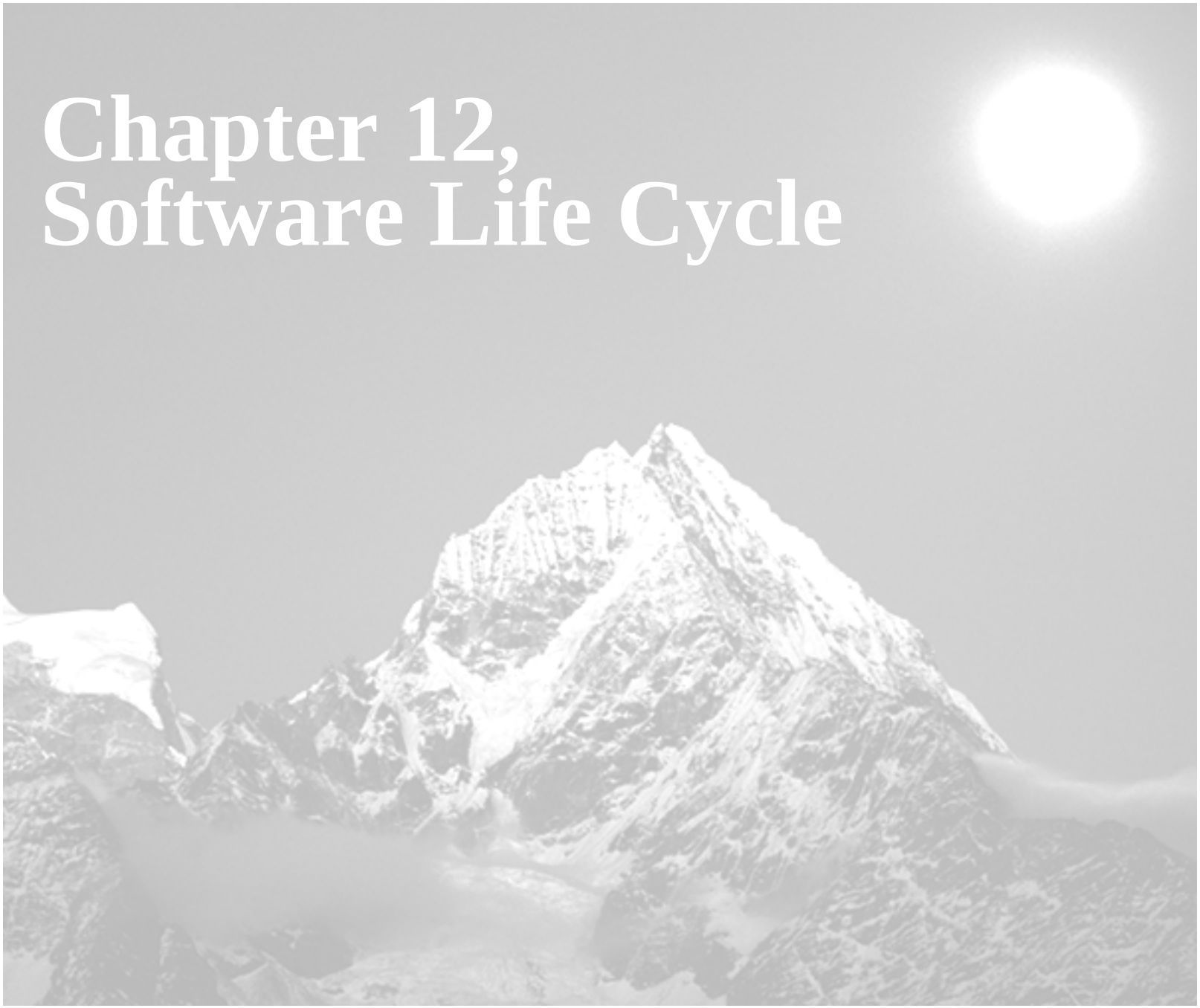
Object-Oriented Software Engineering
Conquering Complex and Changing Systems

Chapter 12, Software Life Cycle



Outline

- 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)**
- Process Maturity

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

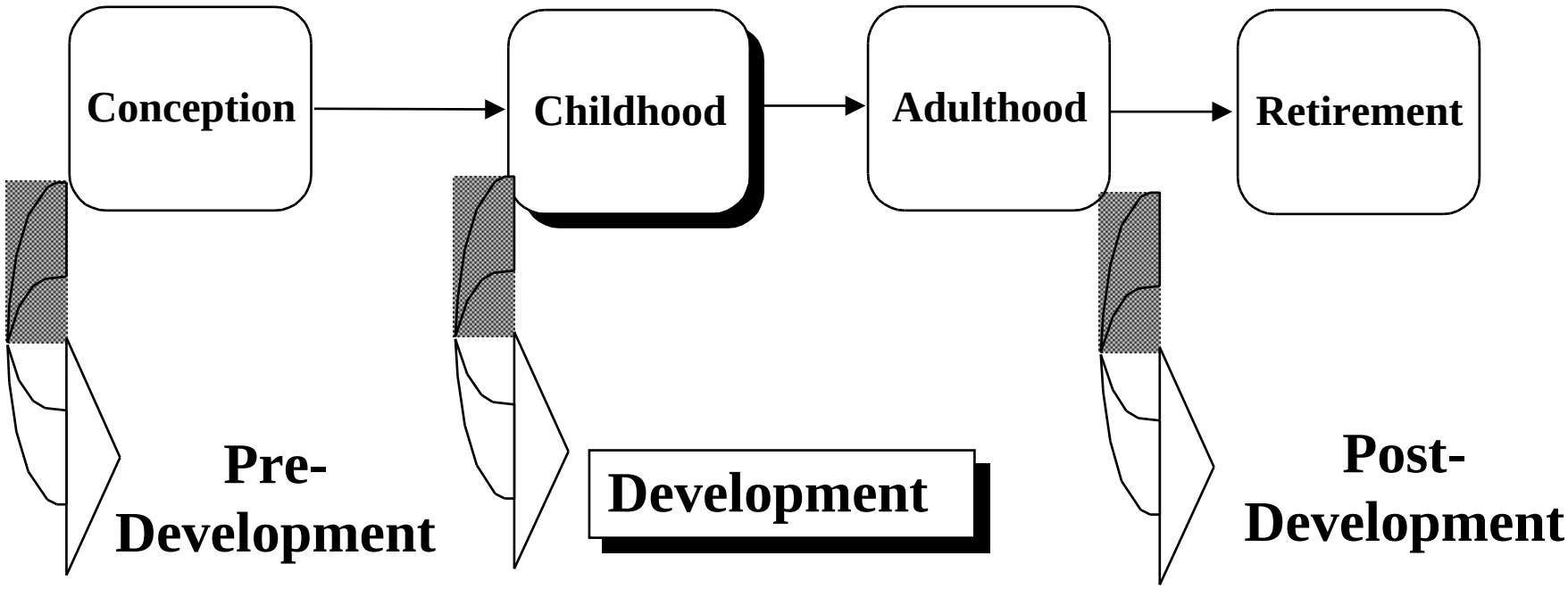
- Software lifecycle modeling: Attempt to deal with complexity and change

- 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?**
Program Design	**What are the mechanisms that best implement the solution?**
Program Implementation	**How is the solution constructed?**
Testing	**Is the problem solved?**
Delivery	**Can the customer use the solution?**
Maintenance	**Are enhancements needed?**

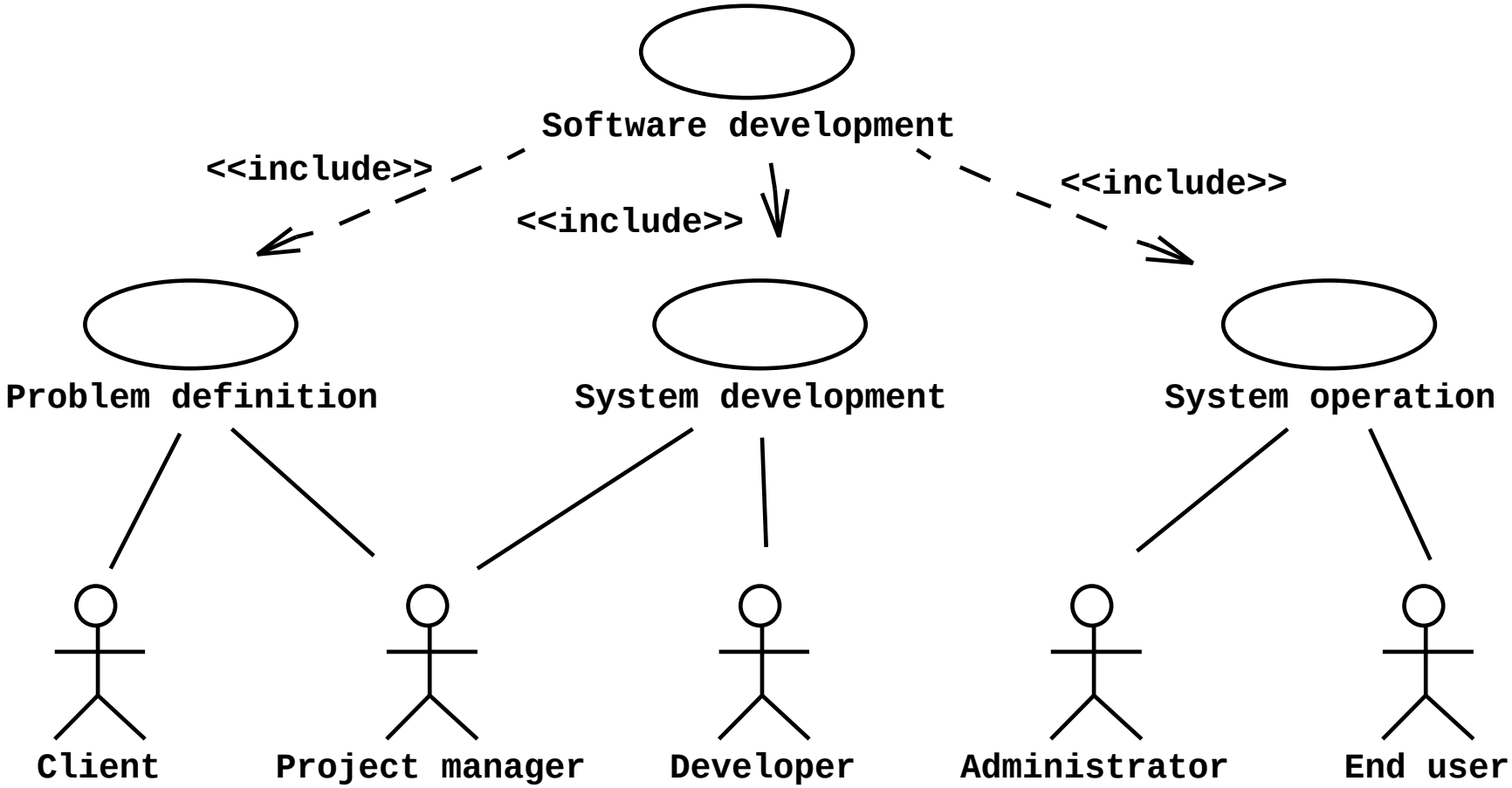
Problem Domain

Implementation Domain

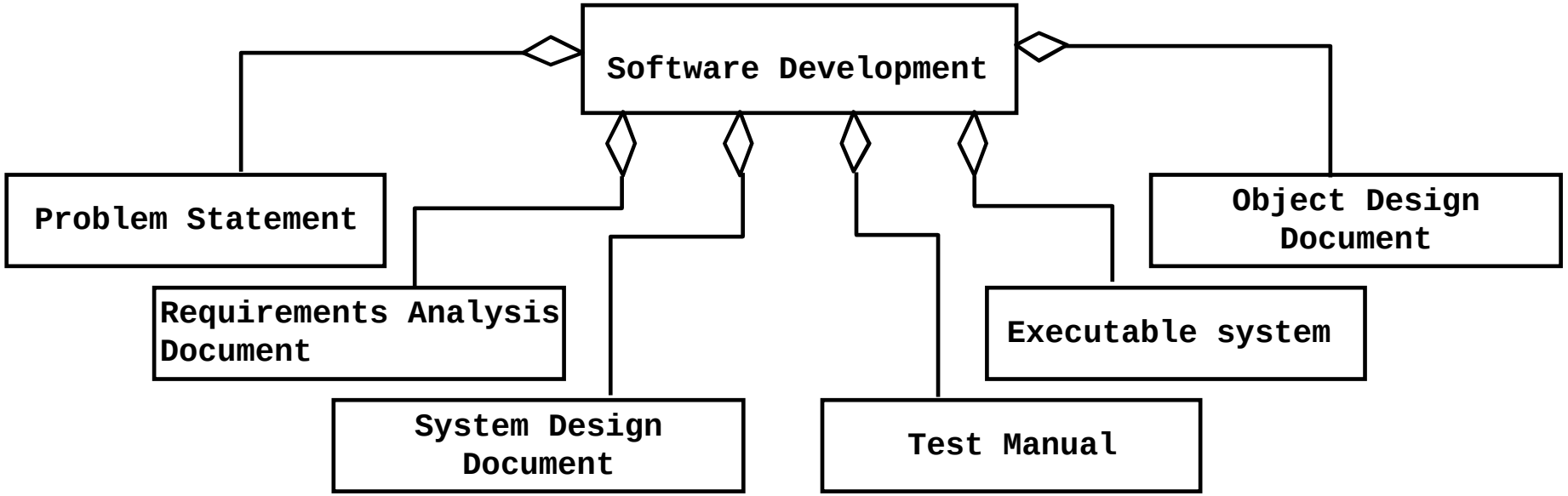
Alternative Identification of Software Development Activities

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?**

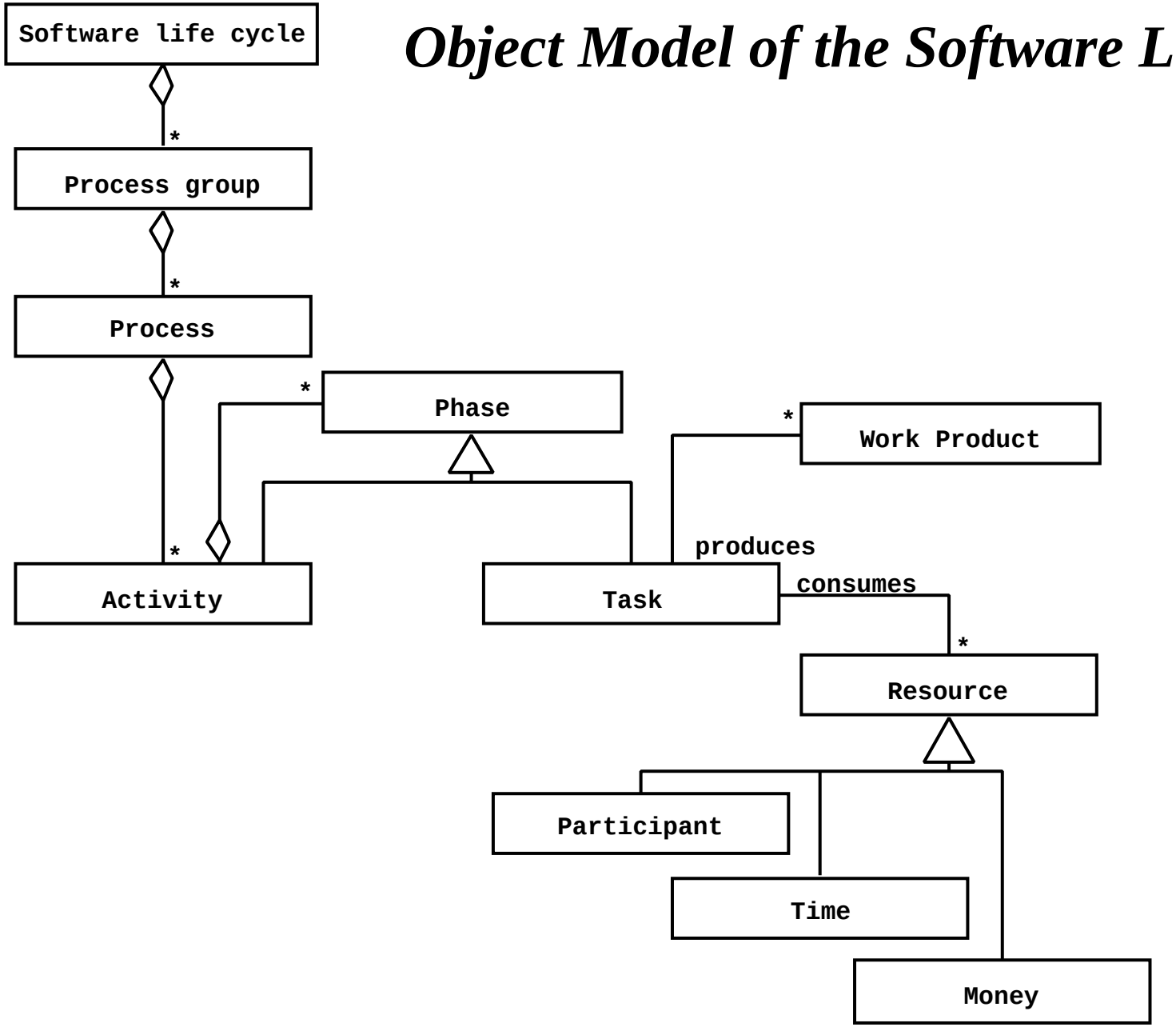
Software Development as Application Domain: A Use Case Model



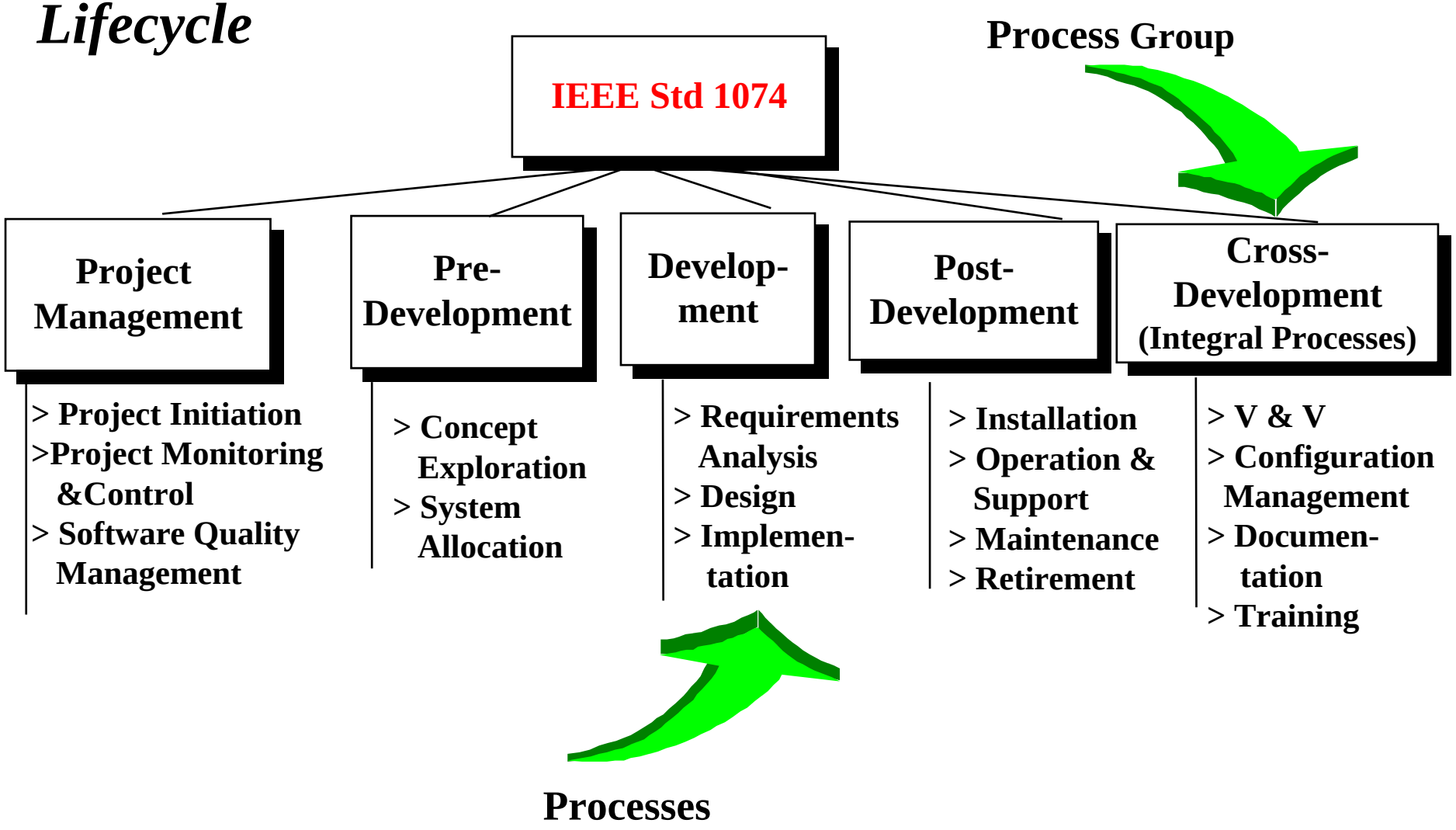
Software Development as Application Domain: Simple Object Model



Object Model of the Software Life Cycle

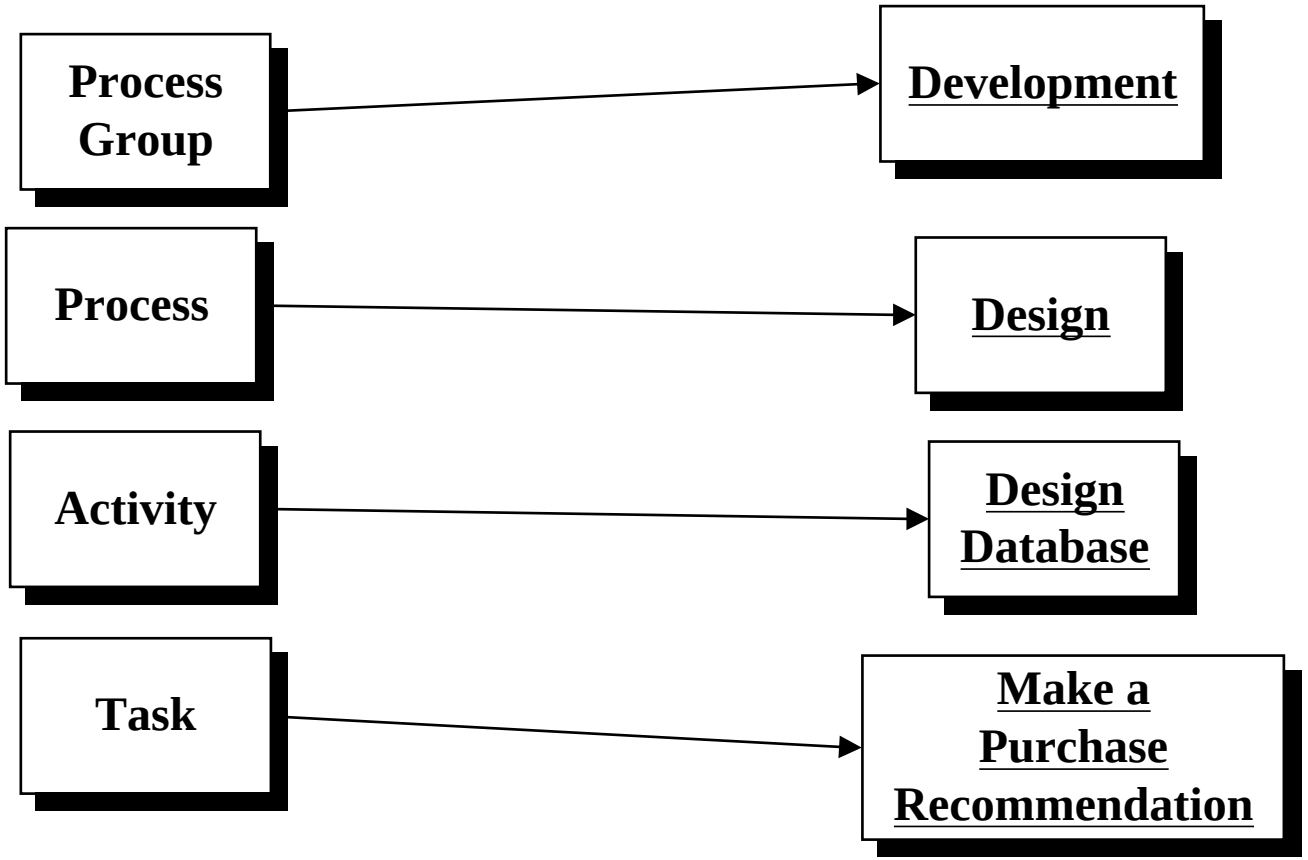


IEEE Std 1074: Standard for Software Lifecycle



Processes, Activities and Tasks

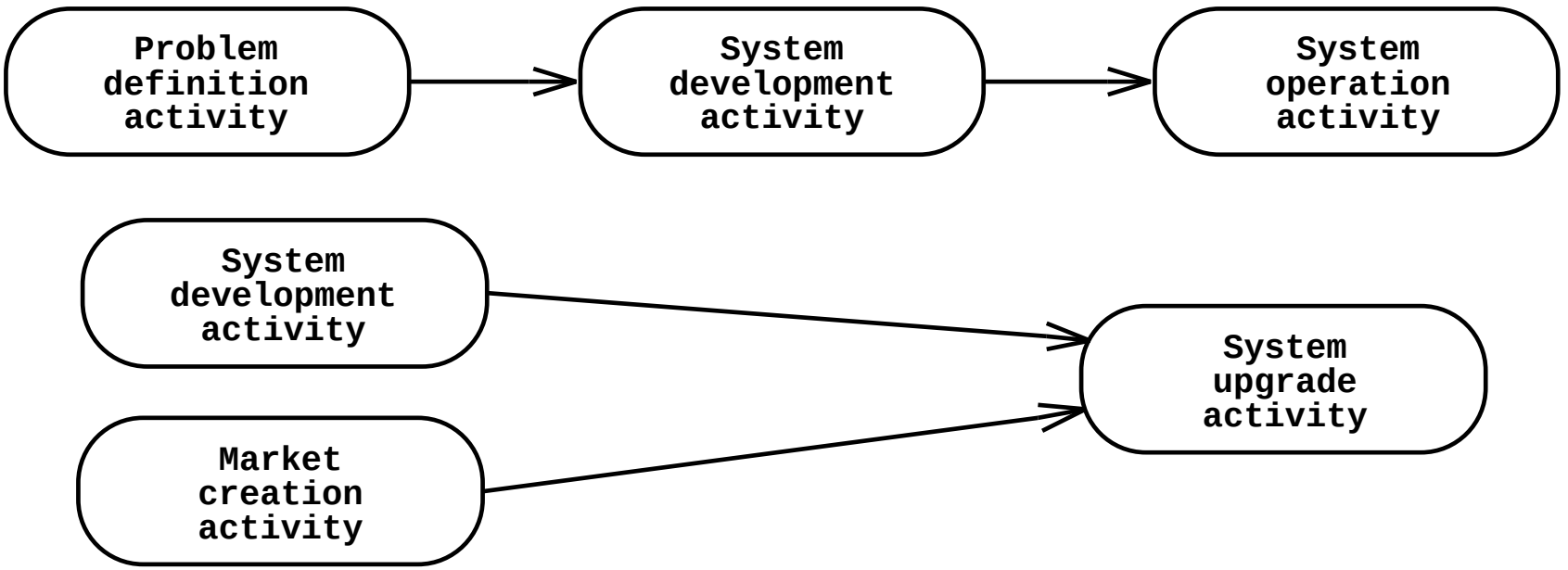
- Process Group: Consists of Set of Processes
- Process: Consists of Activities
- Activity: Consists of sub activities and tasks



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 Algorithms (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

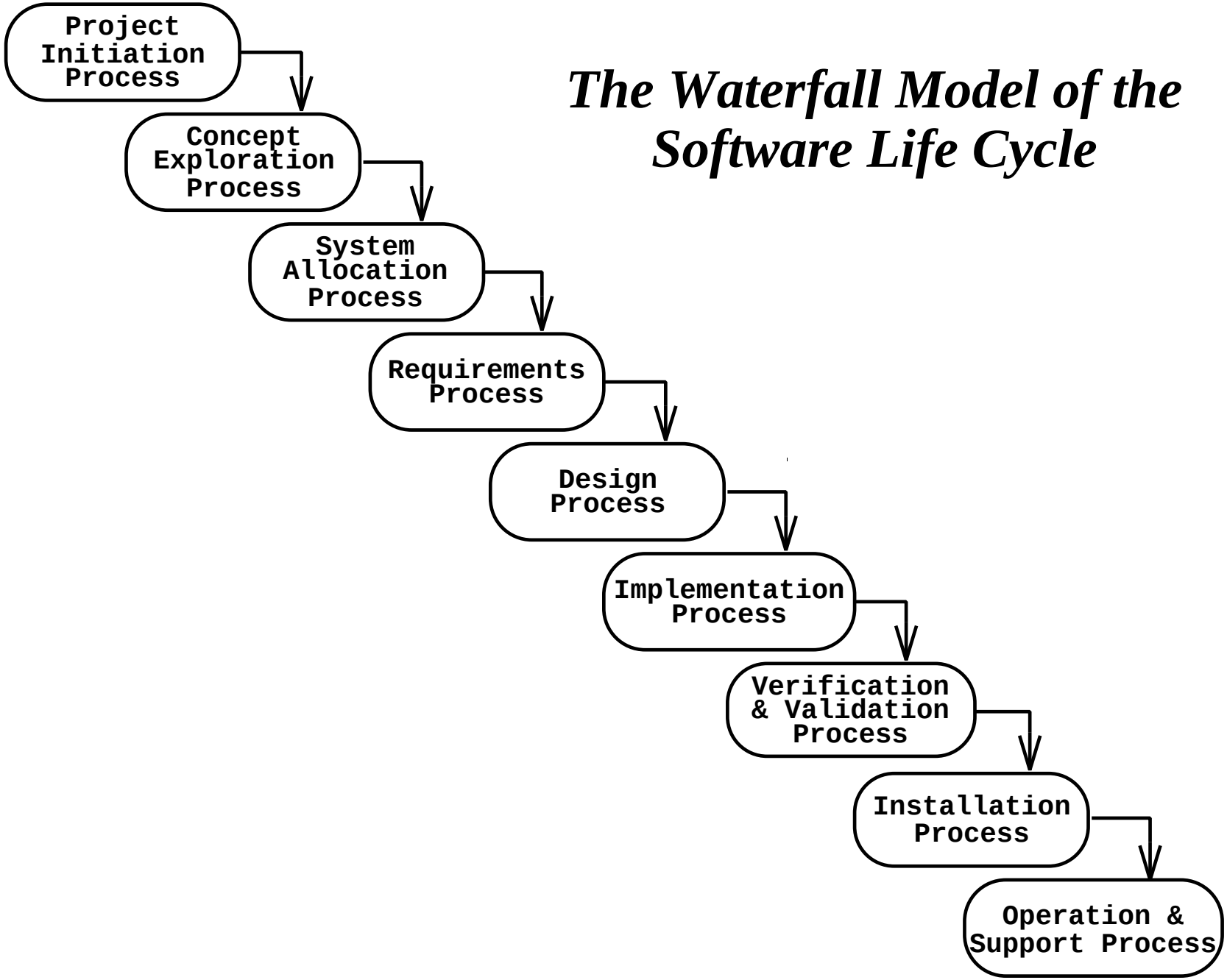


- **Note that the dependency association can mean one of two things:**
  - **Activity B depends on Activity A**
  - **Activity A must temporarily precede Activity B**
- **Which one is right?**

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

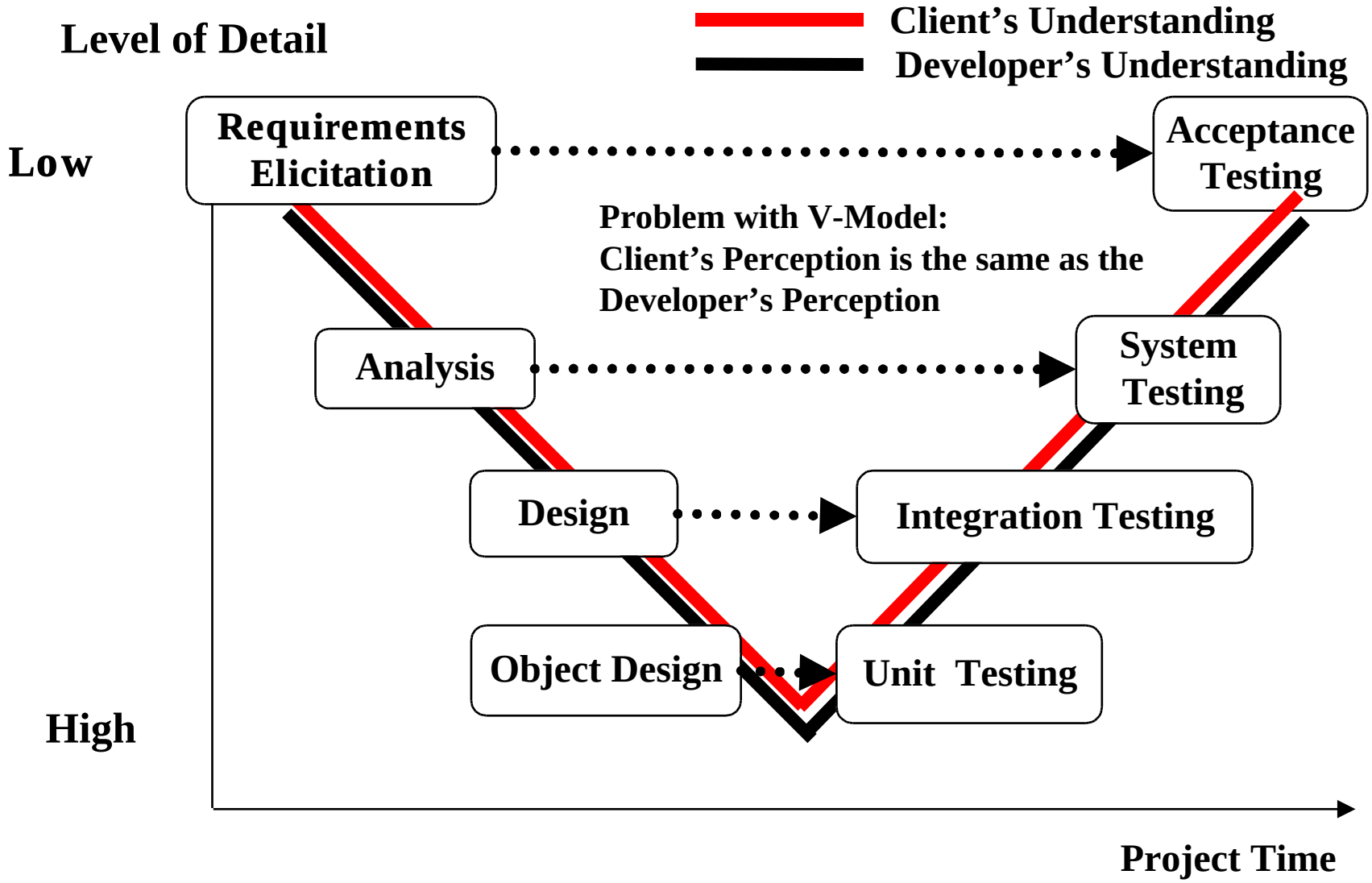
The Waterfall Model of the Software Life Cycle



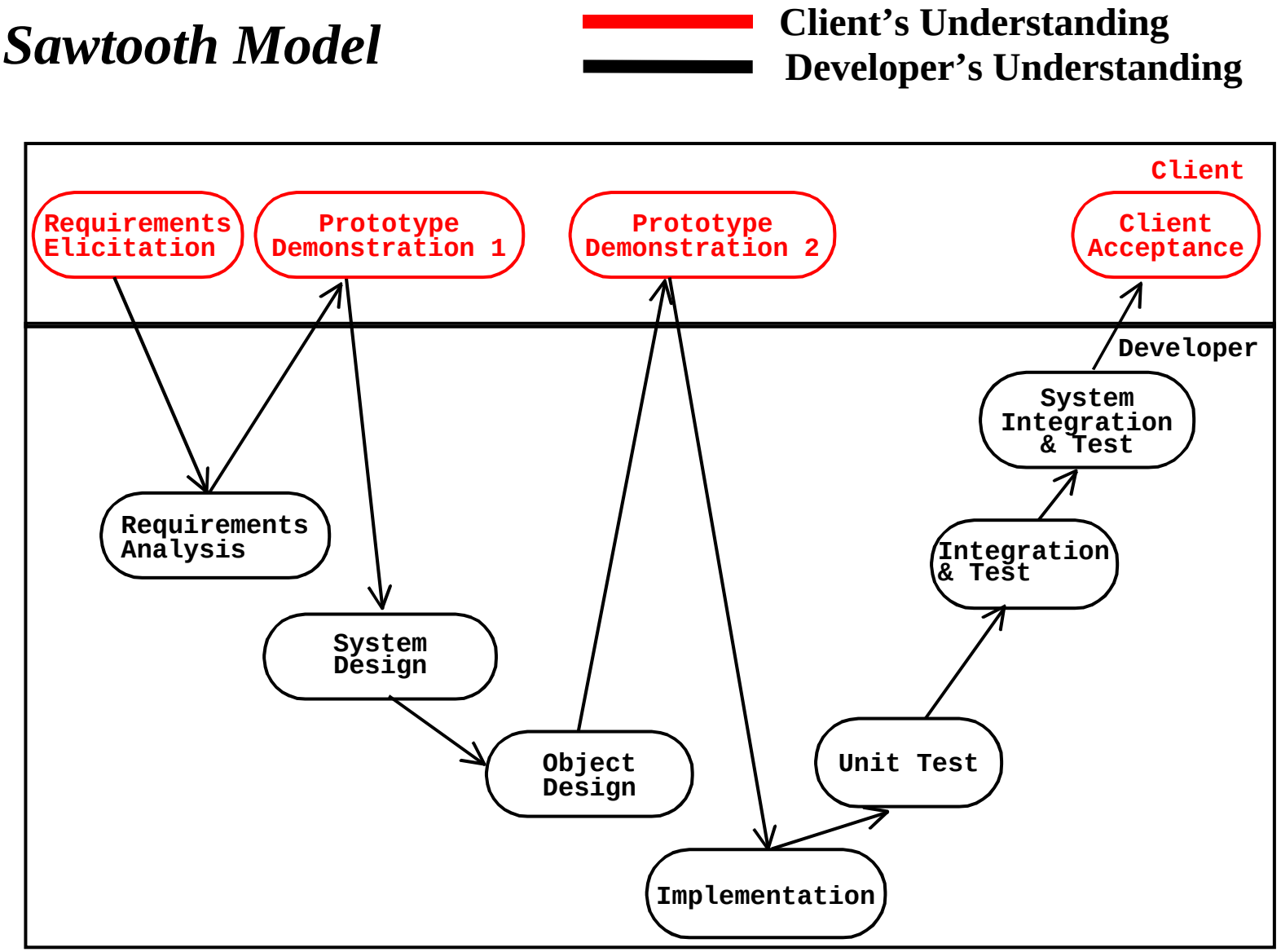
Problems with Waterfall Model

- 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**

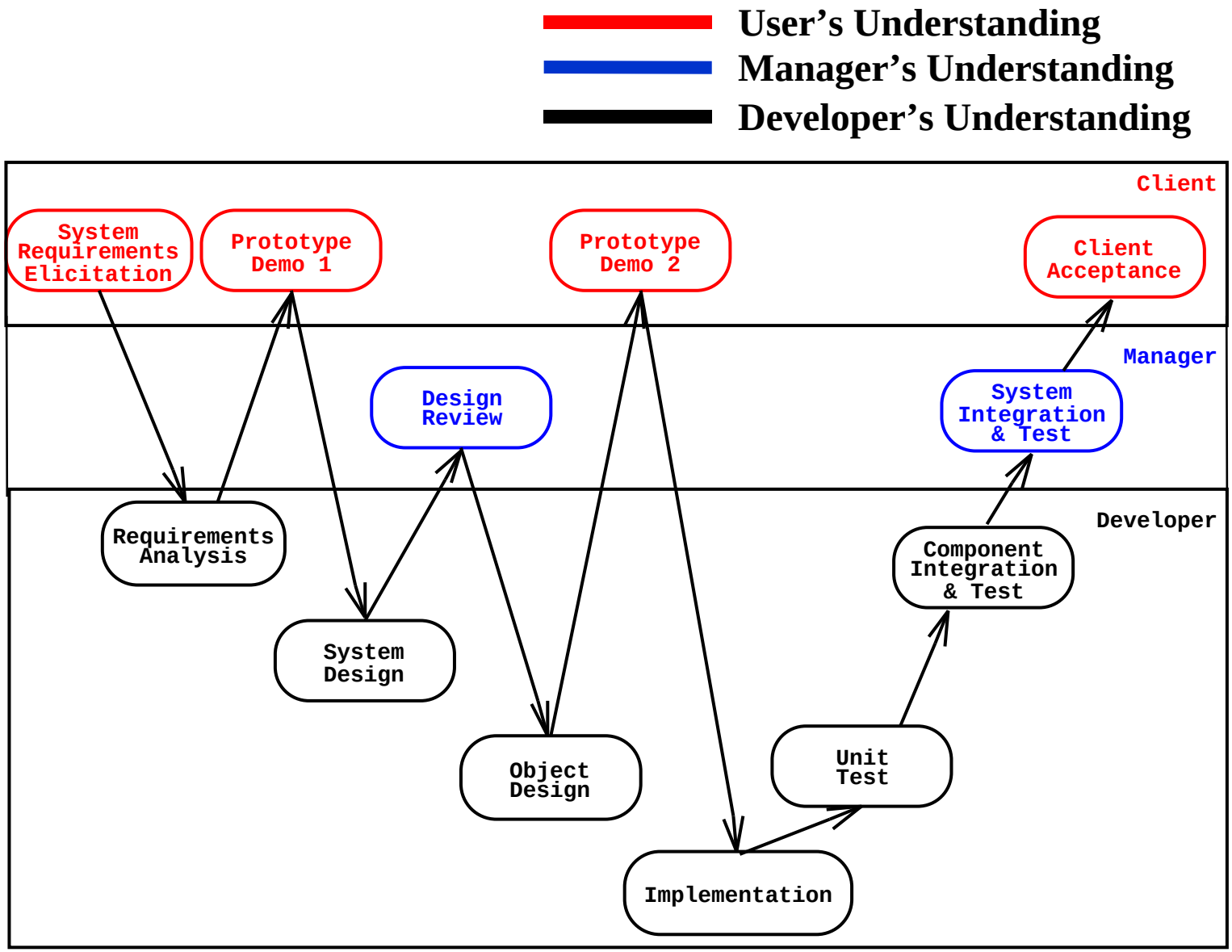
V Model: Distinguishes between Development and Verification Activities



Sawtooth Model



Sharktooth Model



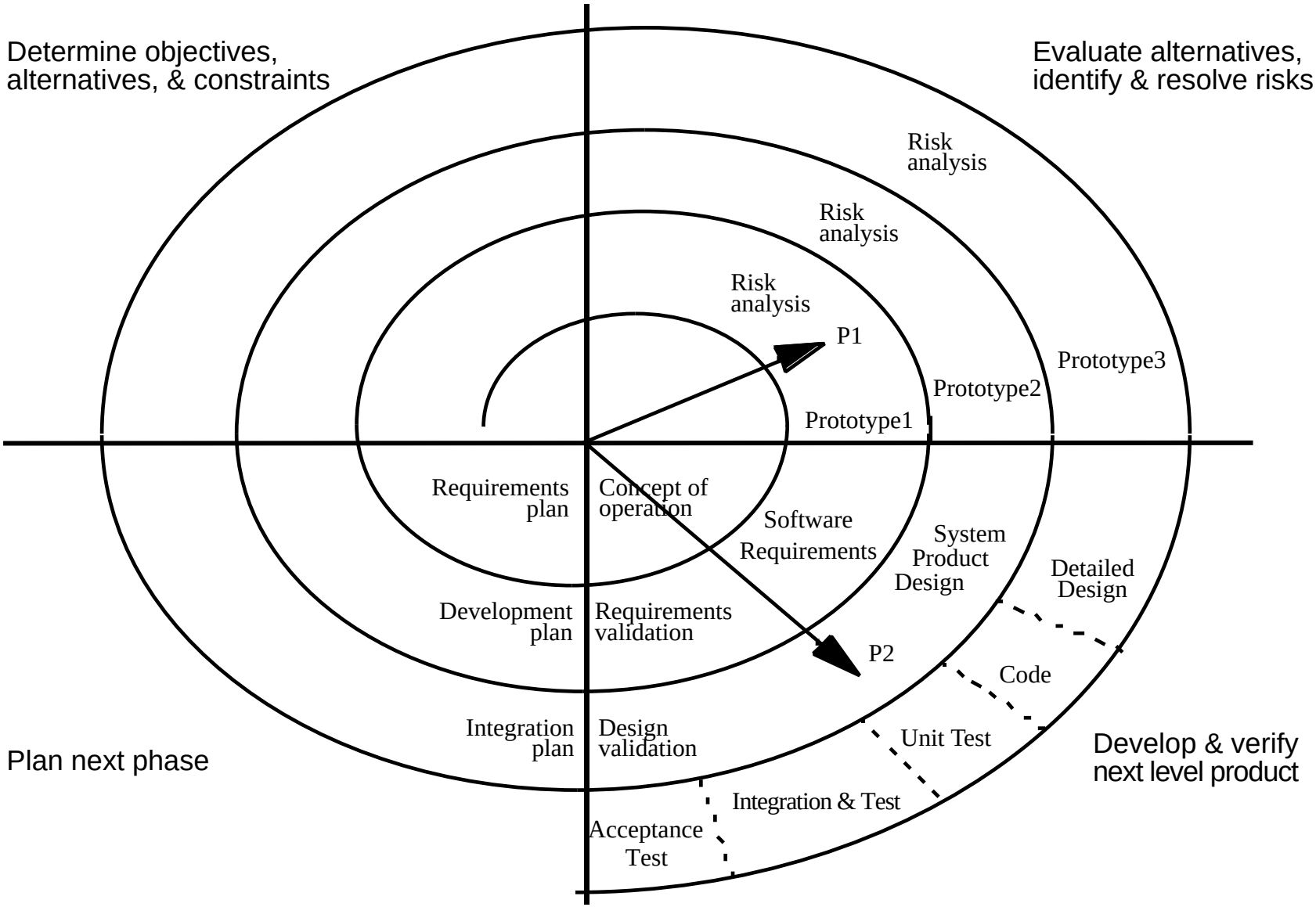
Problems with V Model

- 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

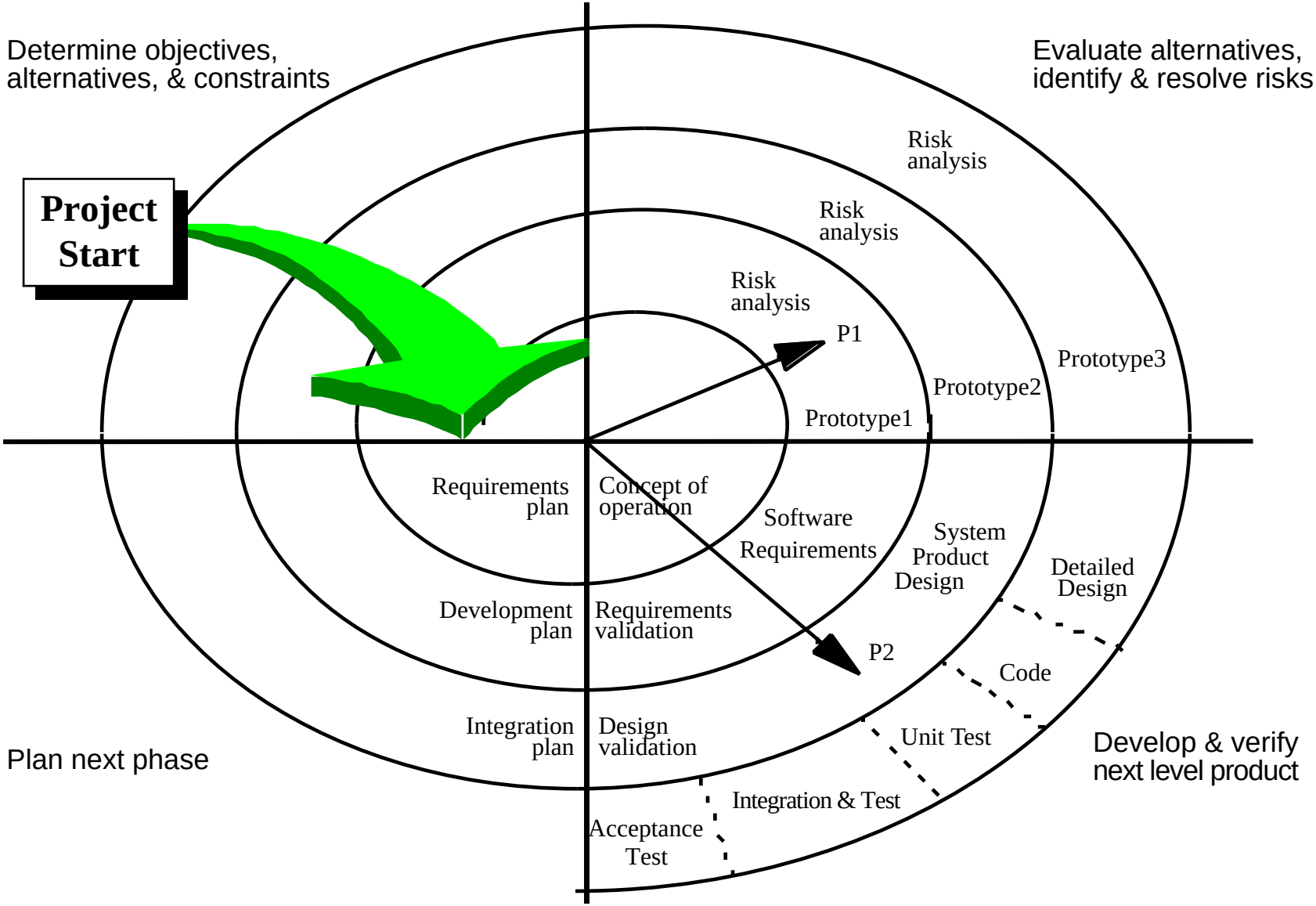


Activities ("Rounds") in Boehm's Spiral Model

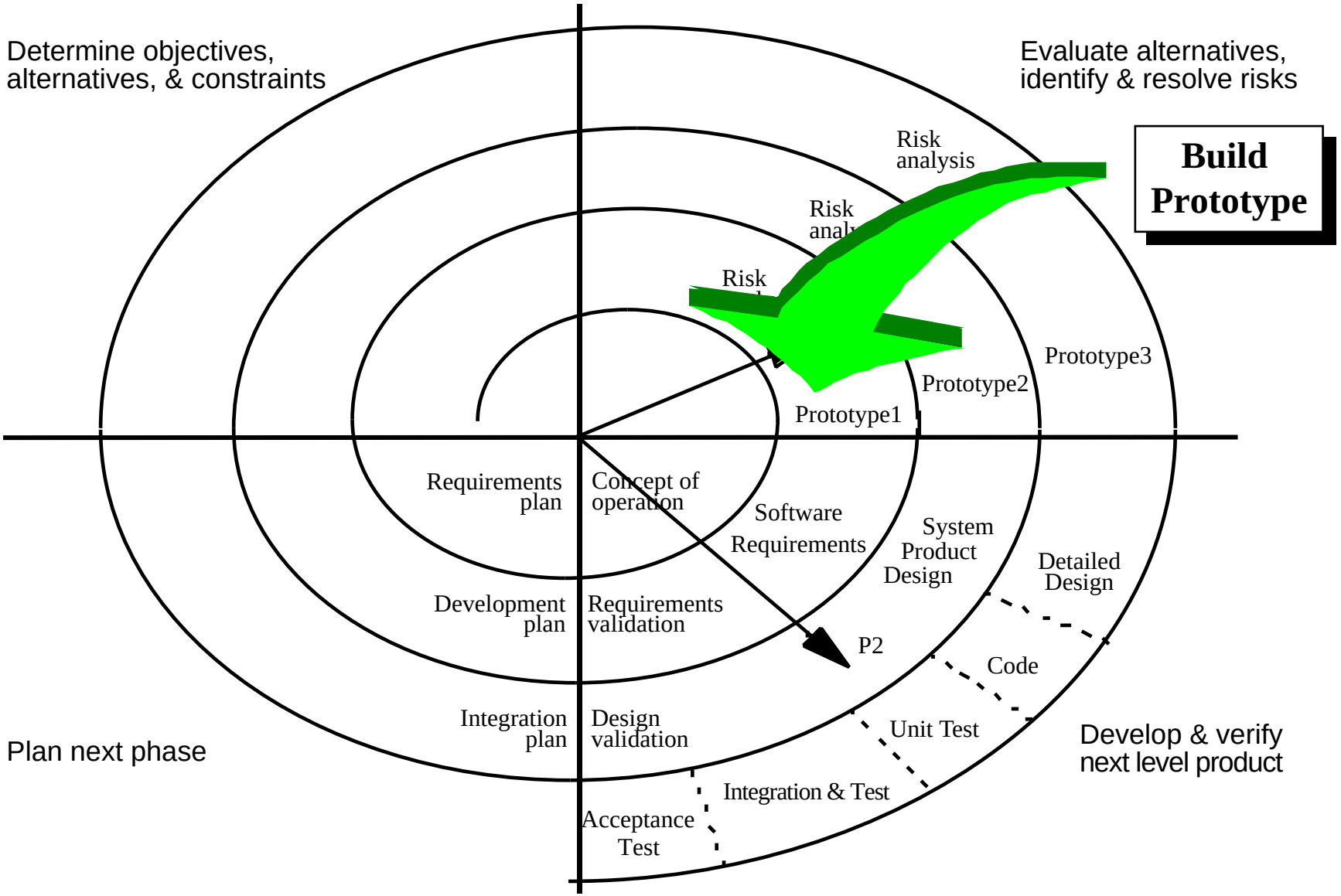
- Concept of Operations
- Software Requirements
- Software Product Design
- Detailed Design
- Code
- Unit Test
- Integration and Test
- Acceptance Test
- Implementation

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

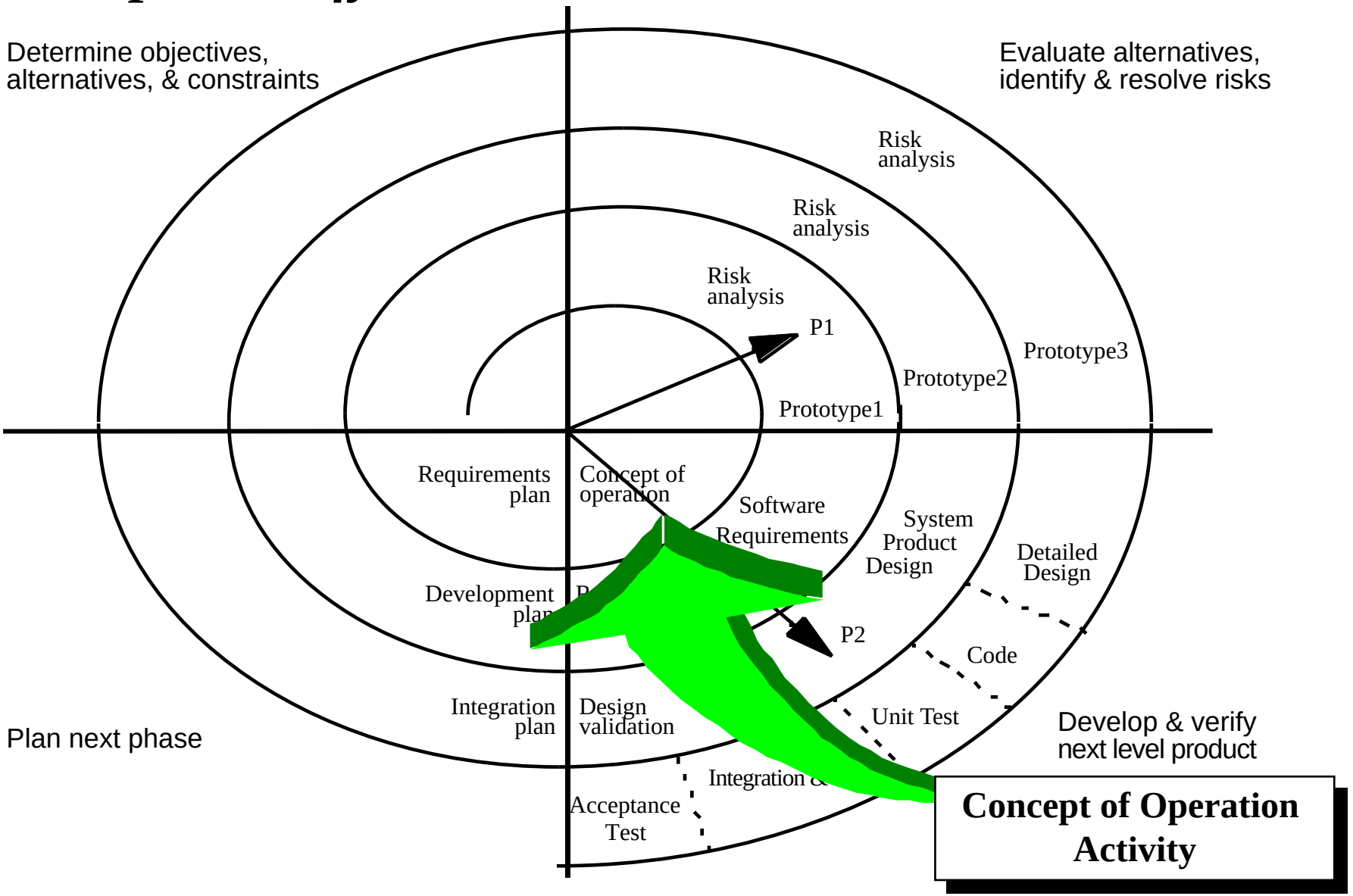
Determine Objectives, Alternatives and Constraints



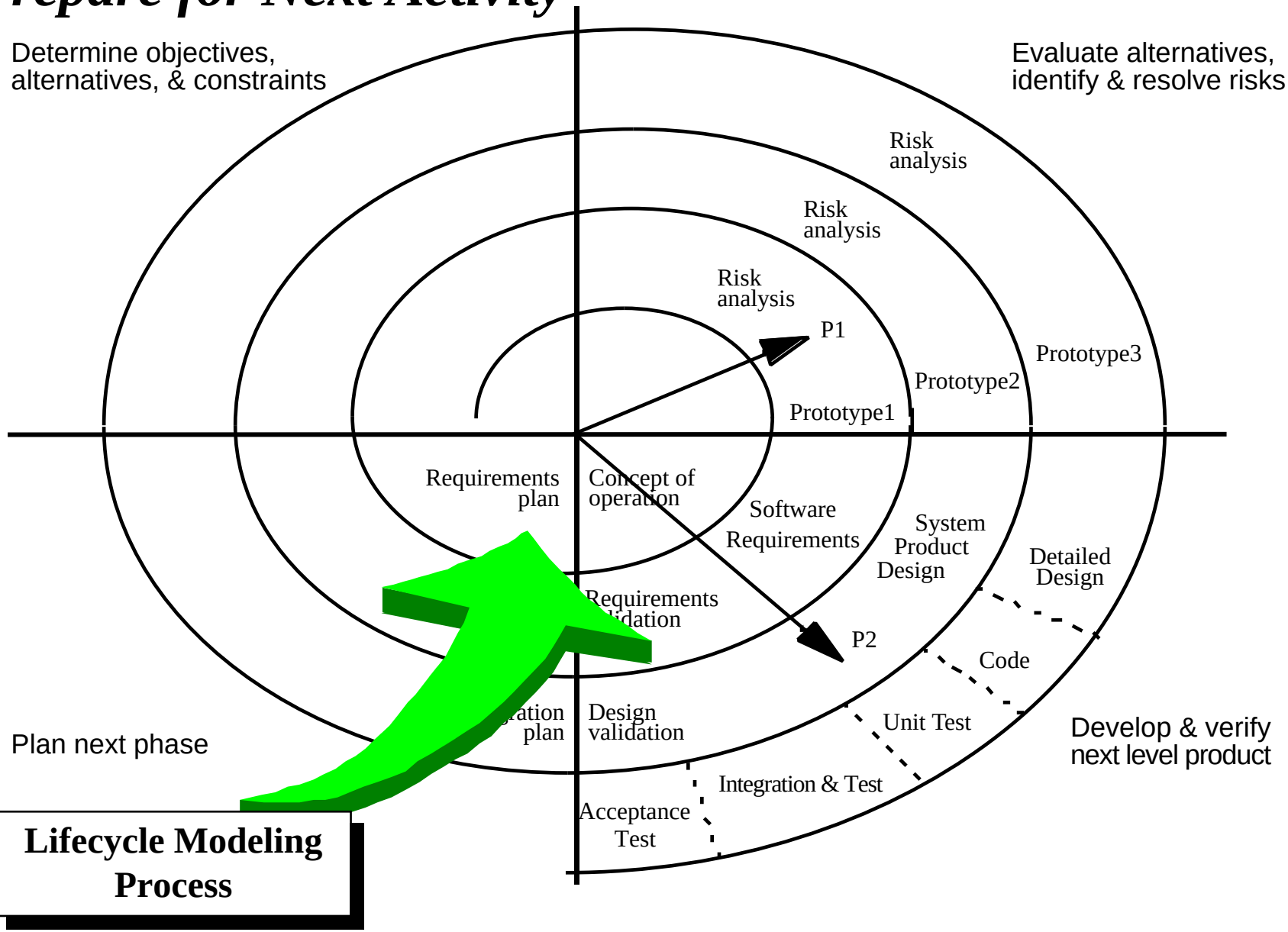
Evaluate Alternatives, Identify, resolve risks



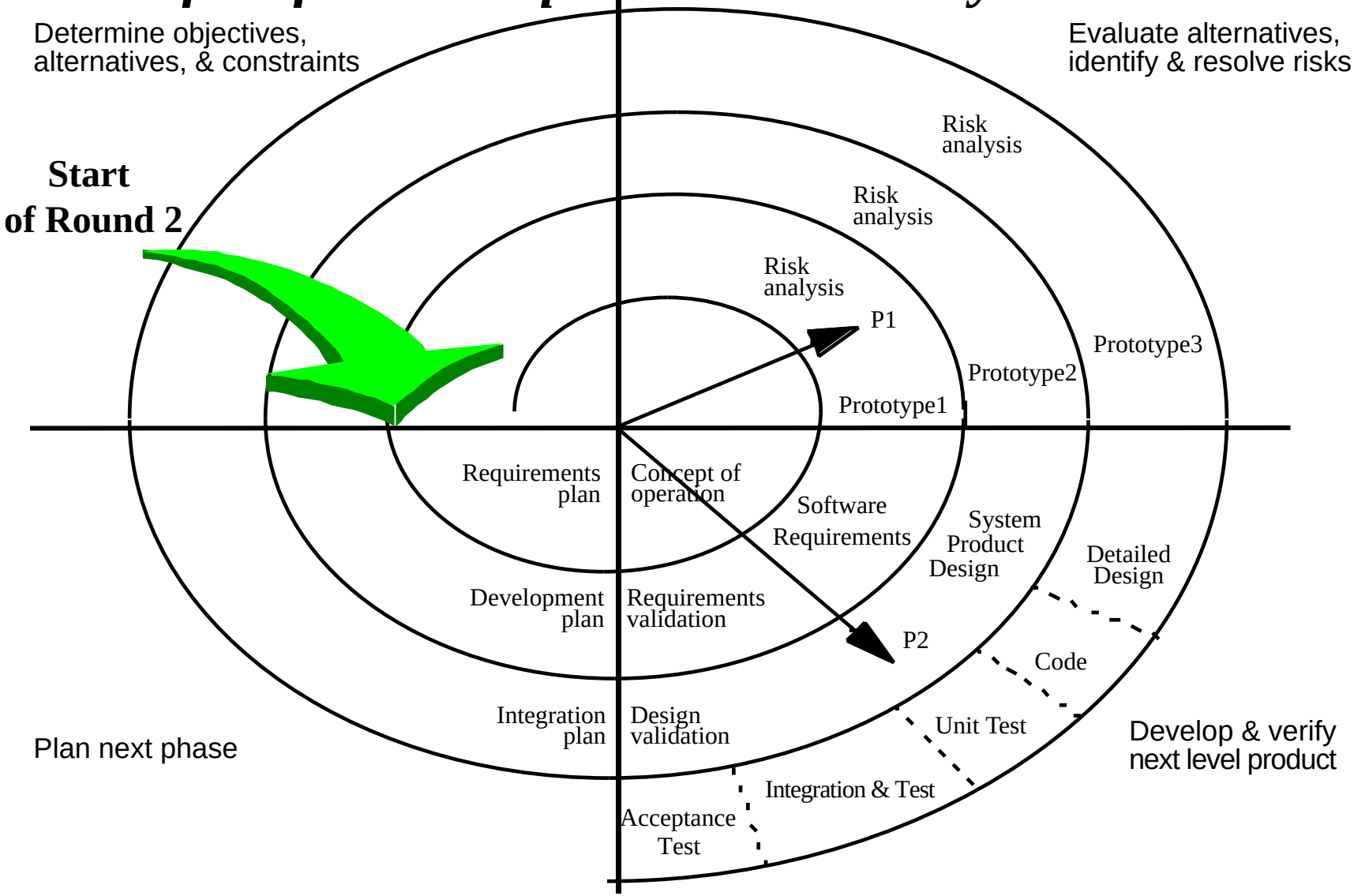
Develop & Verify Product



Prepare for Next Activity



Start of Software Requirements Activity



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 (Continued)

- 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

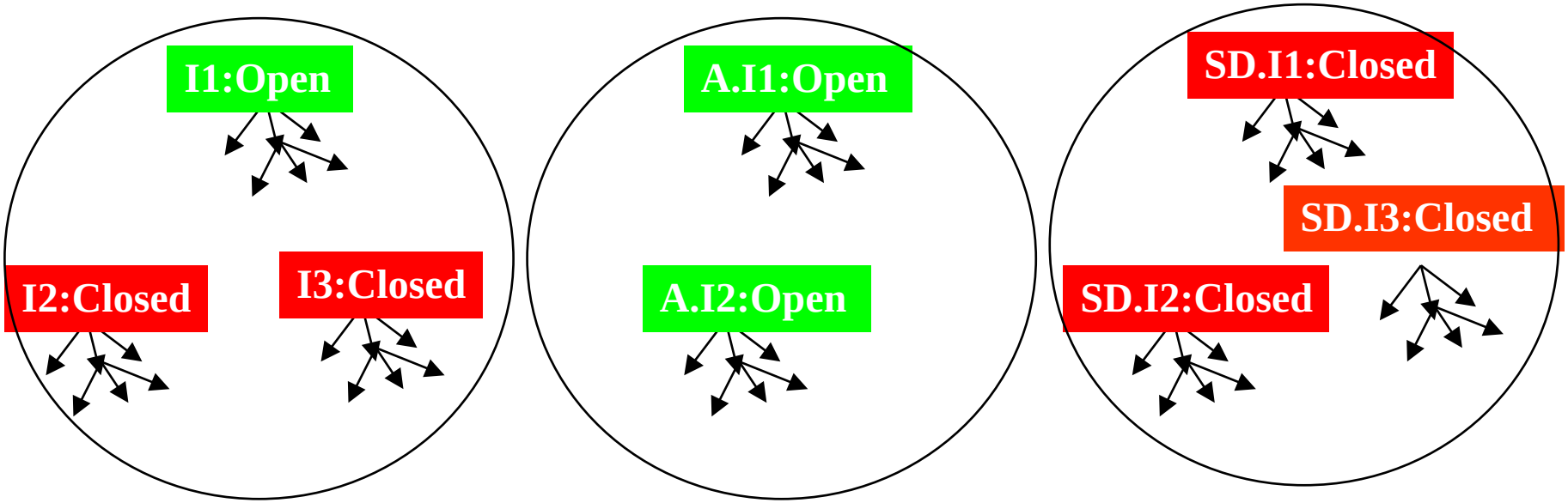
- Revolutionary prototyping is sometimes called rapid prototyping
- Rapid Prototyping is not a good term because it confuses *prototyping* with *rapid development*
  - Prototyping is a technical issue: It is **a particular model in the life cycle process**
  - Rapid development is a management issue. 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

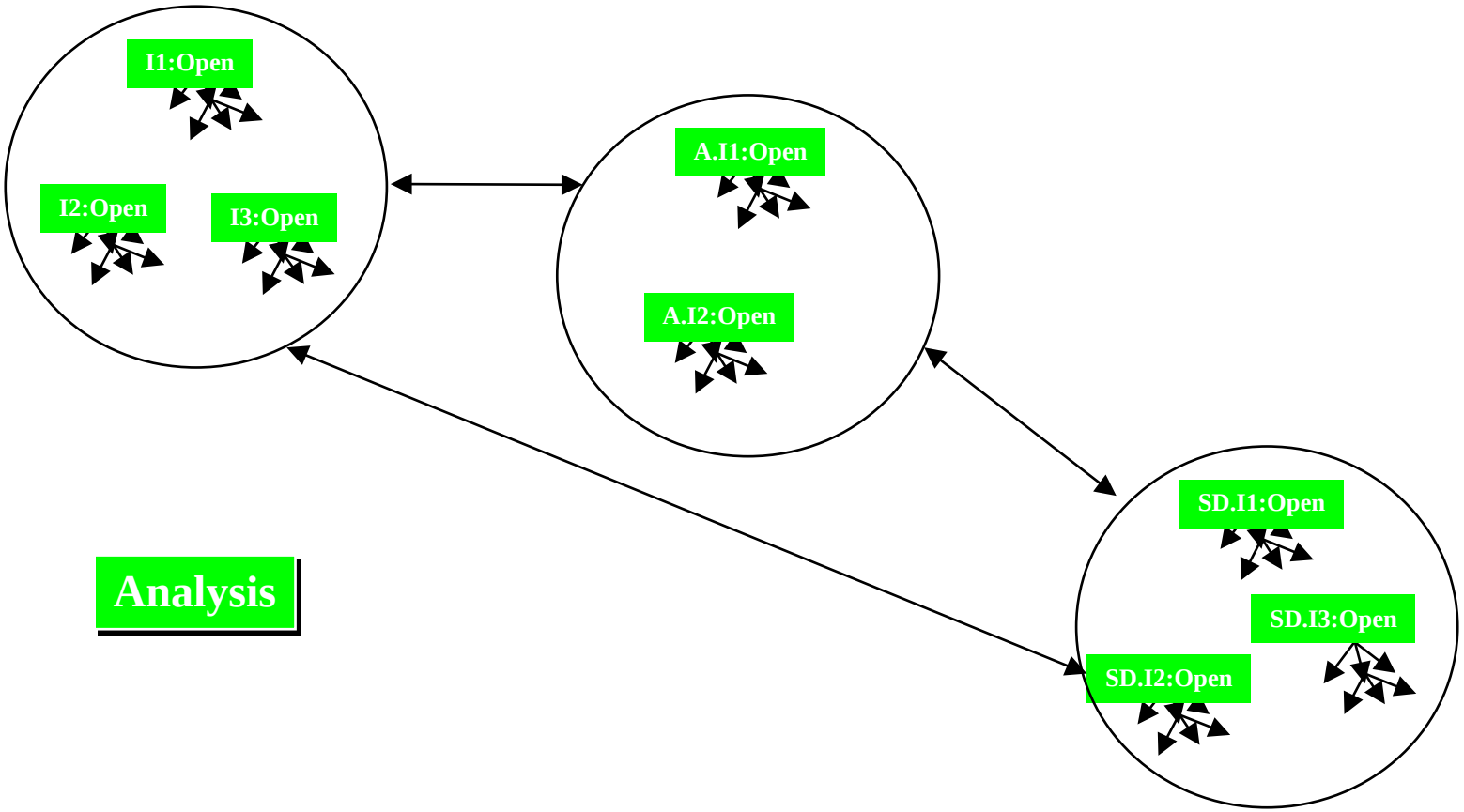
- A system is described as a collection of issues
  - **Issues are either closed or open**
  - **Closed issues have a resolution**
  - **Closed issues can be reopened (Iteration!)**
- The set of closed issues is the basis of the system model



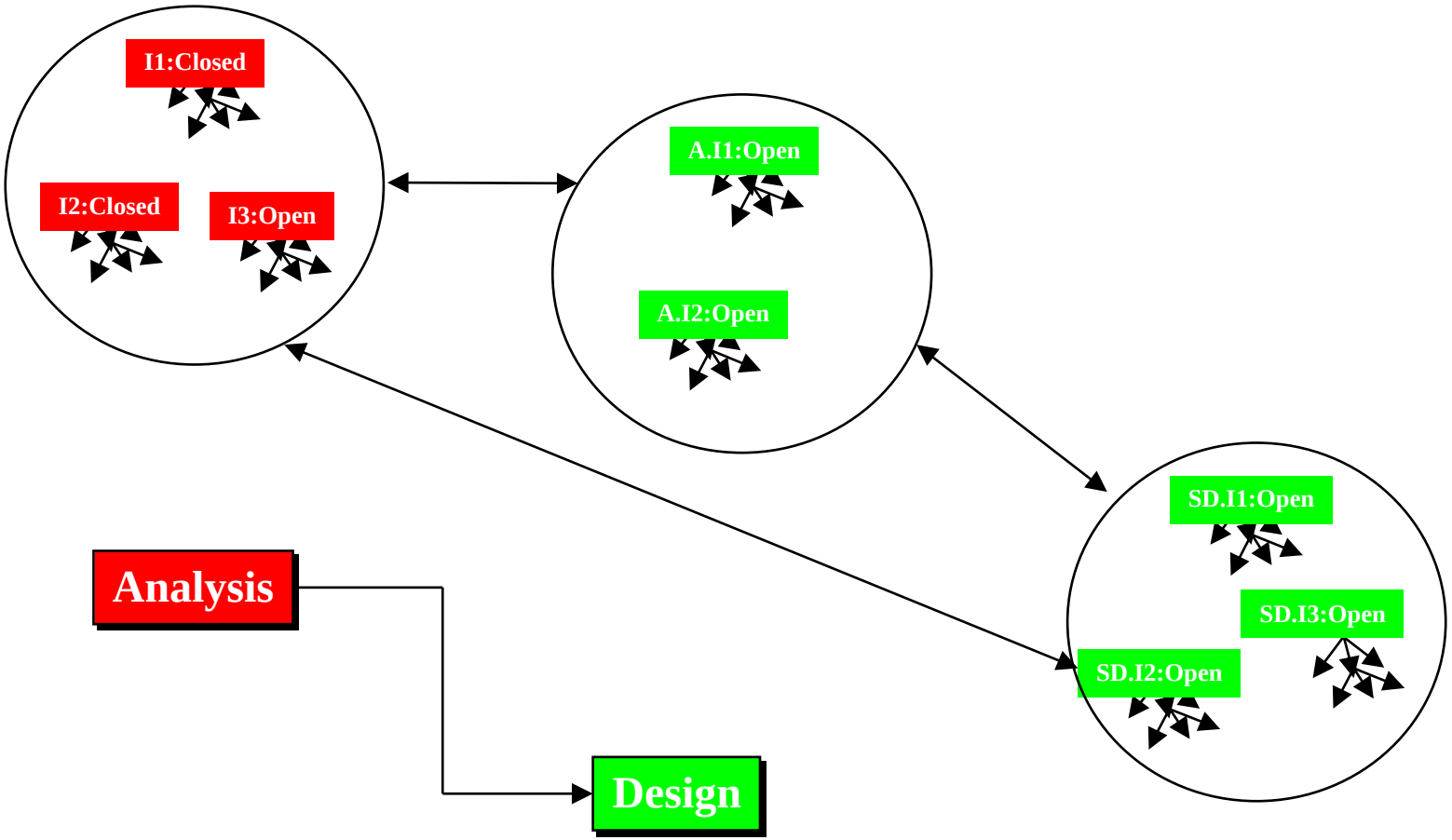
Frequency Change and Software Lifeycle

- **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**

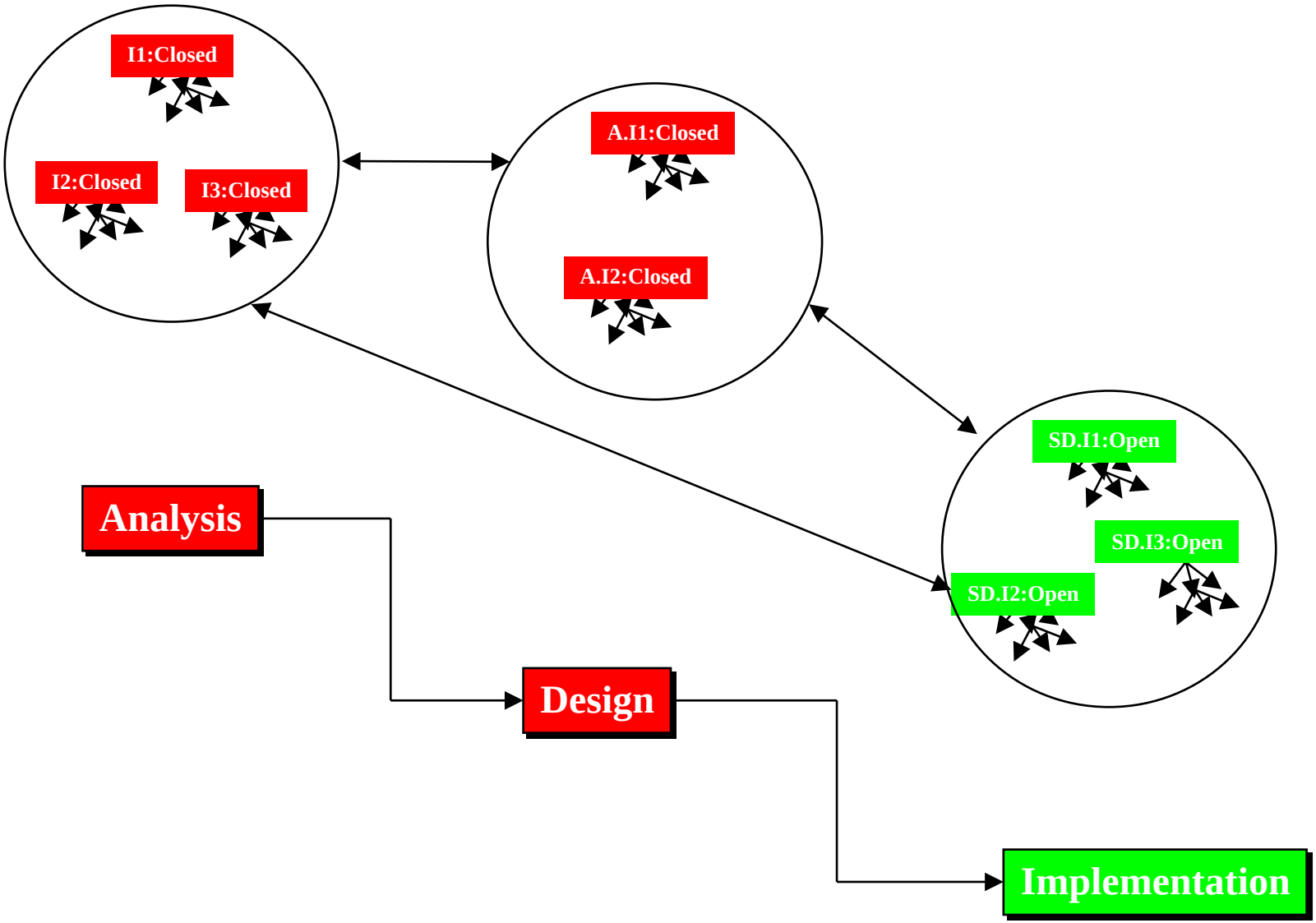
Waterfall Model: Analysis Phase



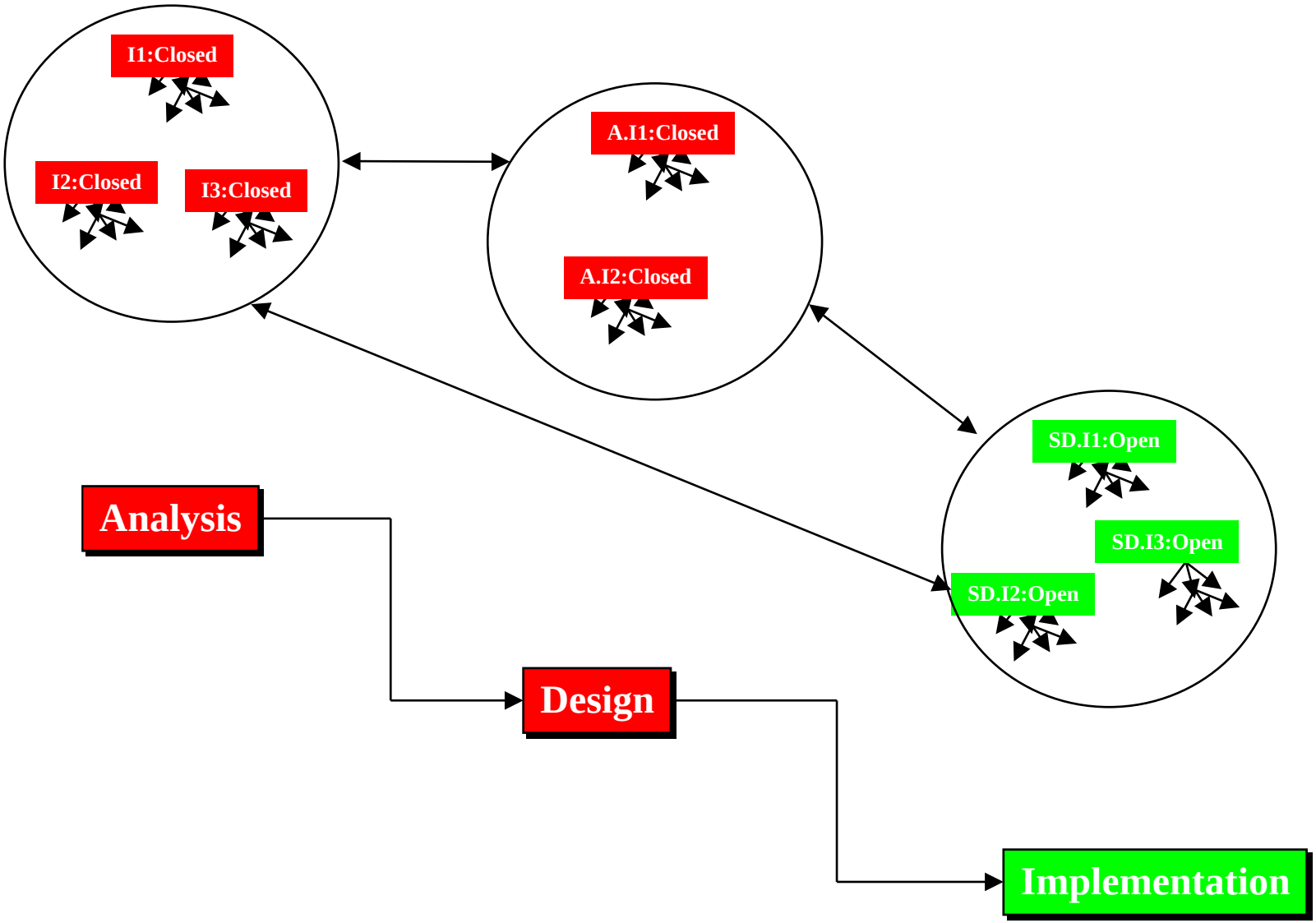
Waterfall Model: Design Phase



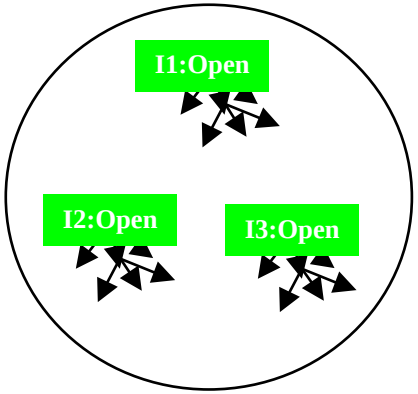
Waterfall Model: Implementation Phase

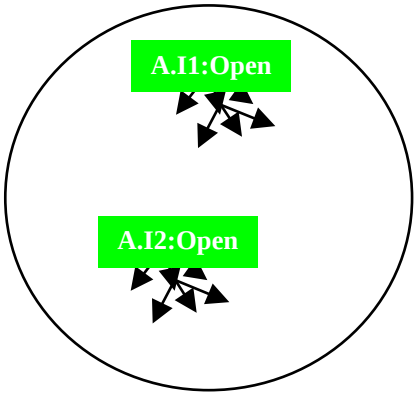


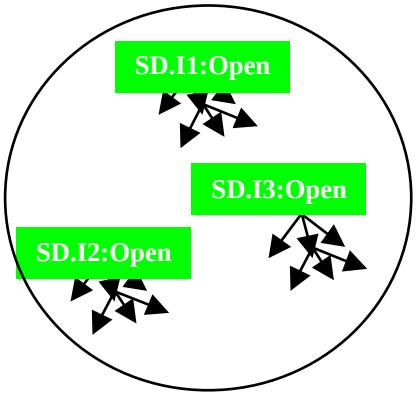
Waterfall Model: Project is Done

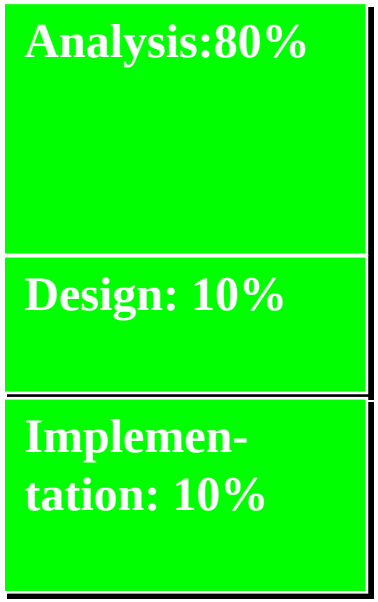


Issue-Based Model: Analysis Phase

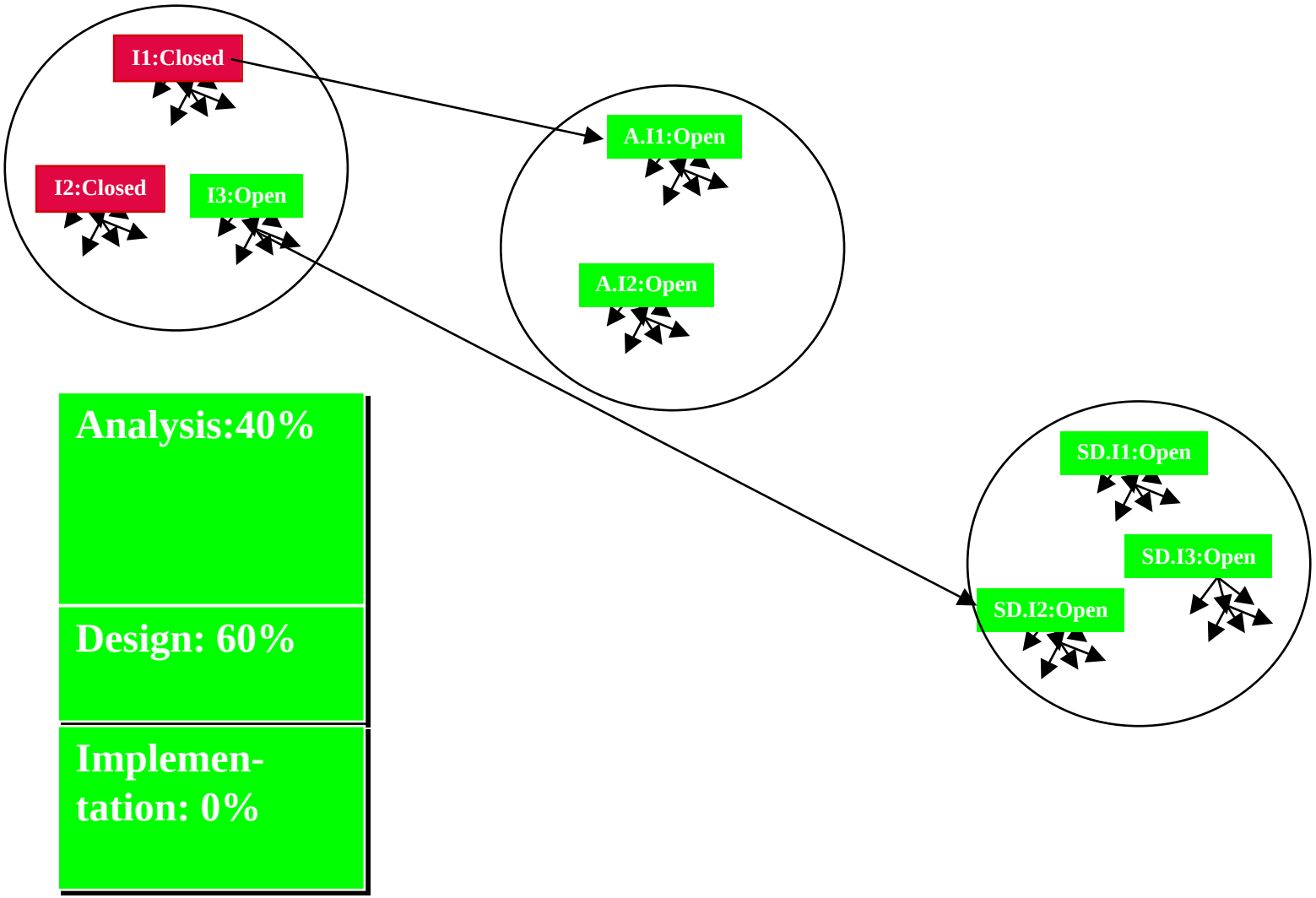




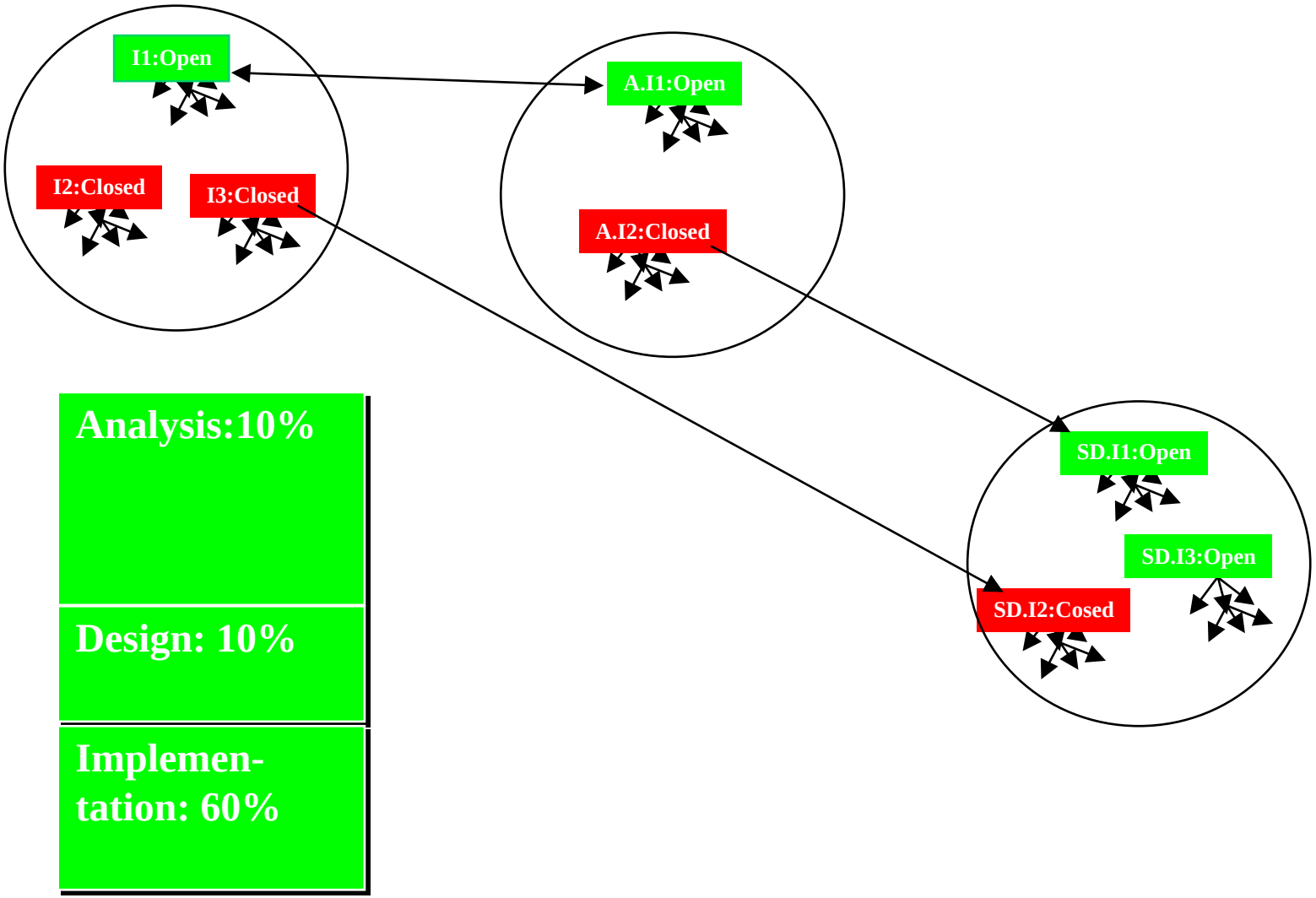




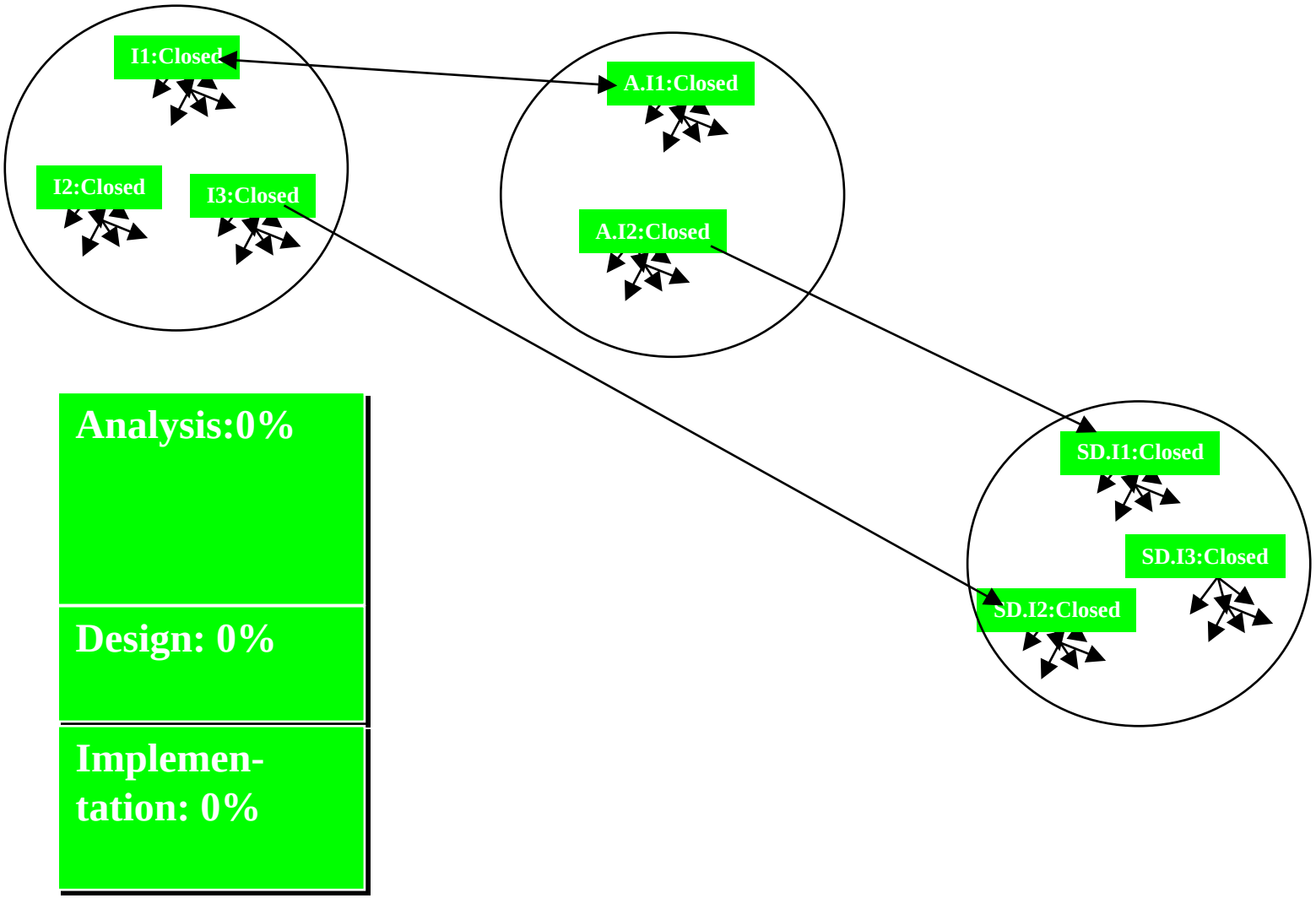
Issue-Based Model: Design Phase



Issue-Based Model: Implementation Phase



Issue-Based Model: Project is Done



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**

Summary

- A Software Life Cycle Model is a representation of the development process (as opposed to the system).
- Reviewed software life cycles
  - **Waterfall model**
  - **V-Model**
  - **Sawtooth Model**
  - **Boehm's Spiral Model**
  - **Issue-based Development Model (Concurrent Development)**
- The maturity of a development process can be assessed using a process maturity model, such as the SEI's CMM.