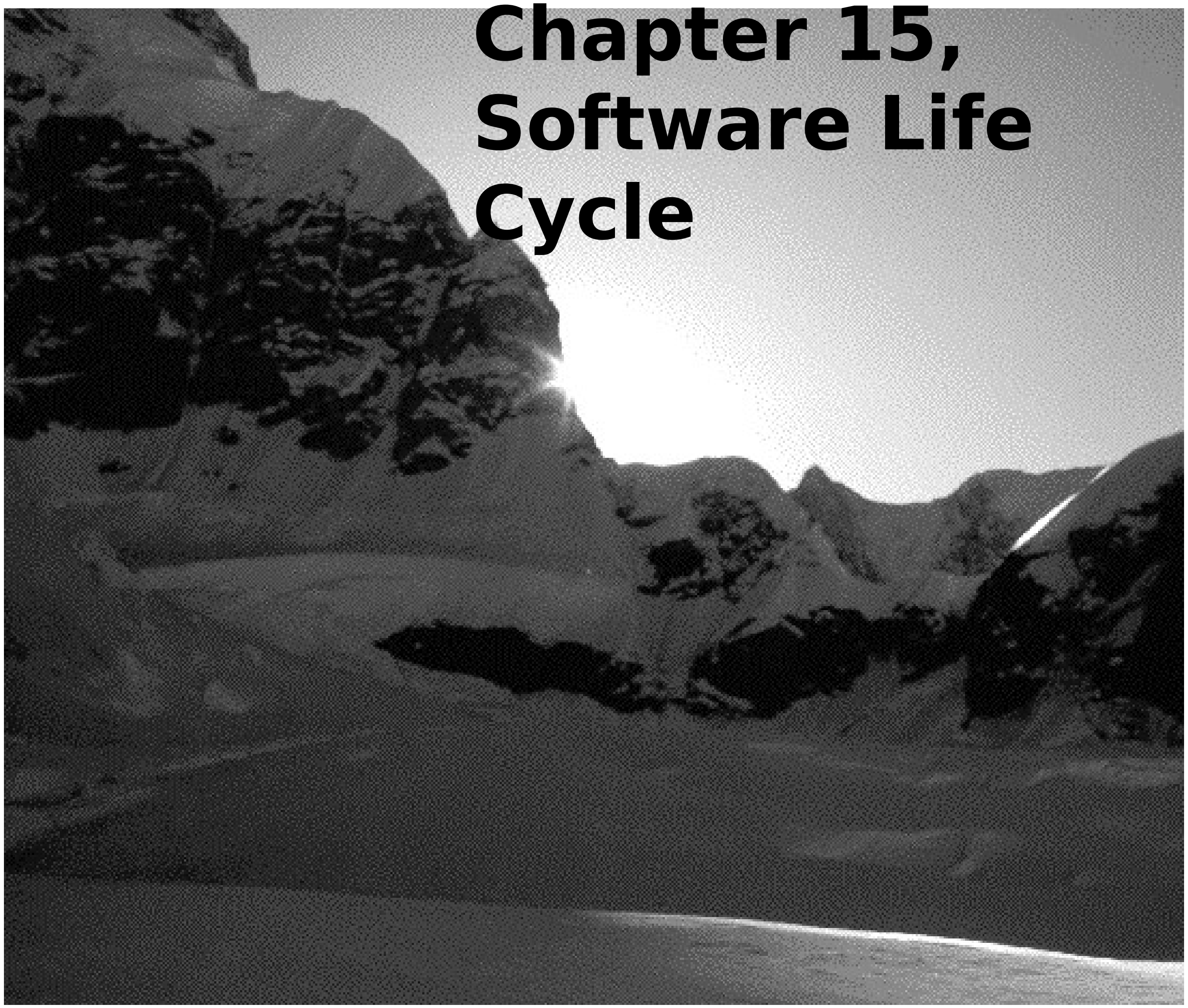
Object-Oriented Software Engineering
Using UML, Patterns, and Java



Chapter 15, Software Life Cycle

Lecture Road Map

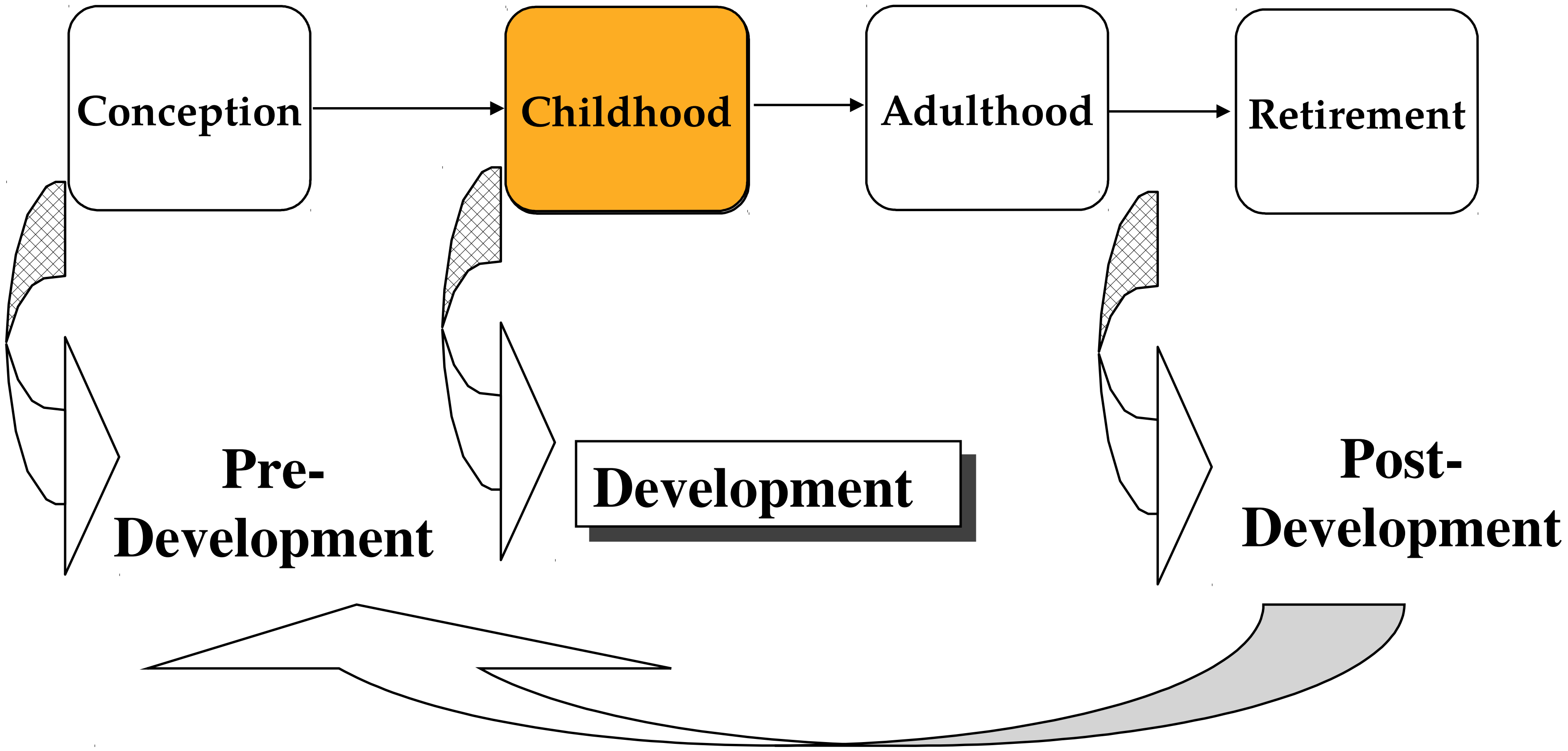
- Software Development as Application Domain
  - Modeling the software lifecycle
- IEEE Standard 1074 for Software Lifecycles
- Modeling the software life cycle
  - Sequential models
    - Pure waterfall model
    - V-model
  - Iterative models
    - Boehm’s spiral model
    - Unified Process (in the next lecture)
  - Entity-oriented models
    - Issue-based model
- Capability Maturity Model

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”)

Software Life Cycle

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



Typical Software Life Cycle Questions

- *Which activities* should we select for the software project?
- What are the *dependencies between activities*?
- How should we *schedule the activities*?
- To find these activities and dependencies we can use the same modeling techniques we use for software development:
  - Functional Modeling of a Software Lifecycle
    - Scenarios
    - Use case model
  - Structural modeling of a Software Lifecycle
    - Object identification
    - Class diagrams
  - Dynamic Modeling of a Software Lifecycle
    - Sequence diagrams, statechart and activity diagrams

Identifying Software Development Activities

- Questions to ask:
  - What is the problem?
  - What is the solution?
  - What are the best mechanisms to implement the solution?
  - How is the solution constructed?
  - Is the problem solved?
  - Can the customer use the solution?
  - How do we deal with changes that occur during the development? Are enhancements needed?

Software Development Activities (Example 1)

Requirements Analysis	What is the problem?	Application Domain
System Design	What is the solution?	
Detailed Design	What are the best mechanisms to implement the solution?	Solution Domain
Program Implementation	How is the solution constructed?	
Testing	Is the problem solved?	
Delivery	Can the customer use the solution?	
Maintenance	Are enhancements needed?	

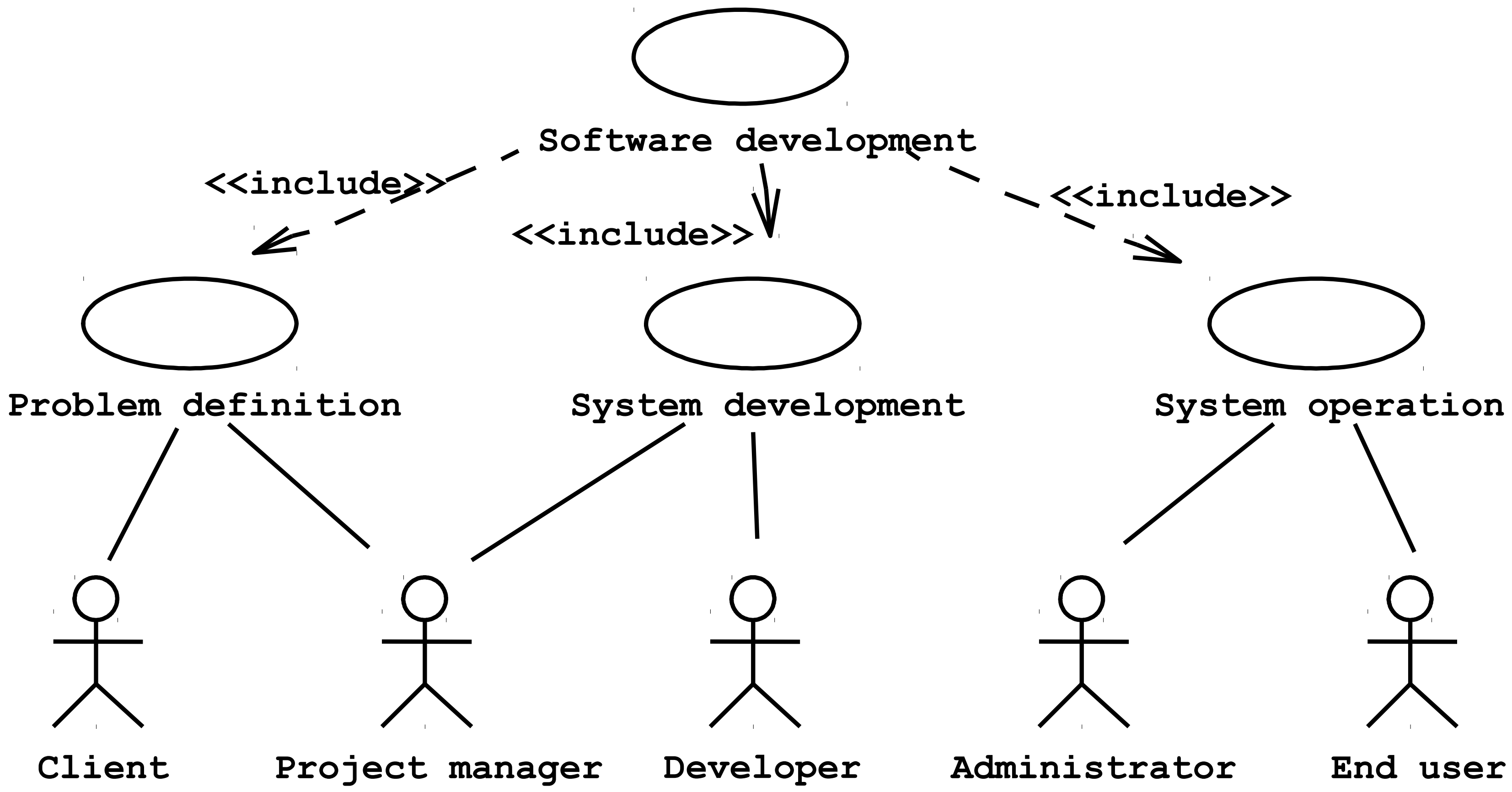
Software Development Activities (Example 2)

Requirements Analysis	What is the problem?	Application Domain
System Design	What is the solution?	
Object Design	What are the best mechanisms to implement the solution?	
Implementation	How is the solution constructed?	Solution Domain

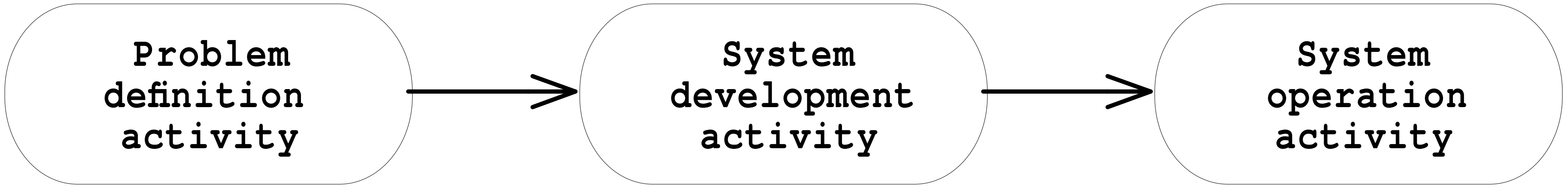
Definitions

- Software life cycle:
  - 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 life cycle

Functional Model of a simple life cycle model

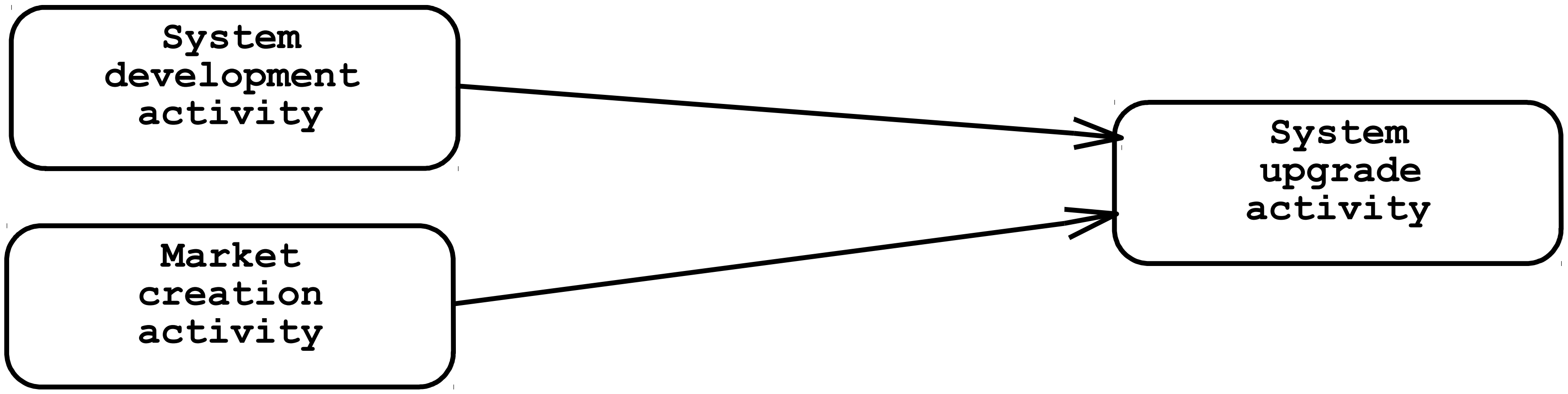


Activity Diagram for the same Life Cycle Model


Problem definition activity
System development activity
System operation activity

Software development goes through a linear progression of states called software development activities

Another simple Life Cycle Model

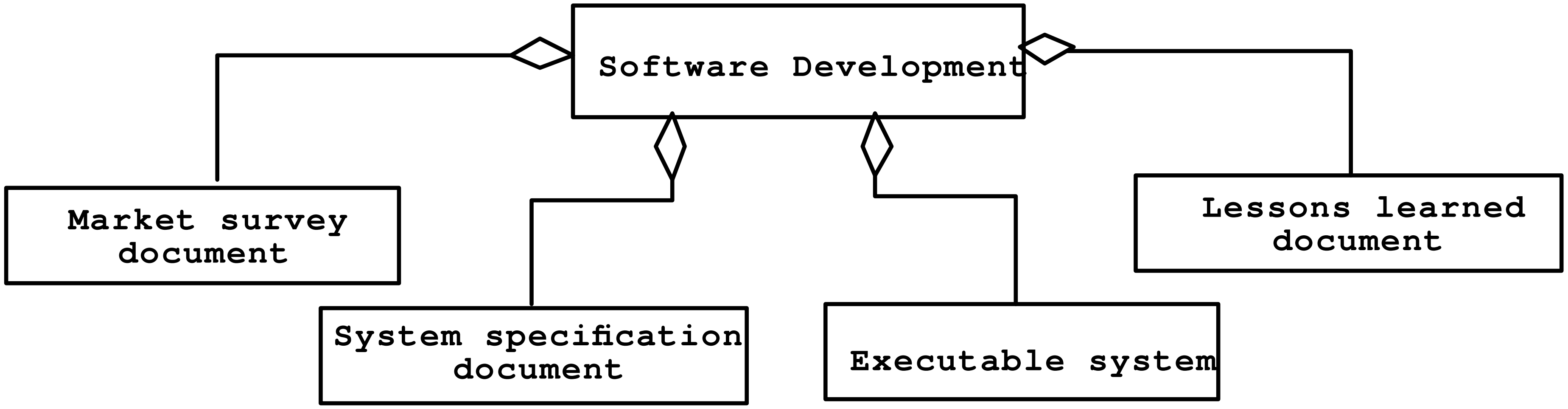


System Development and Market creation can be done in parallel. They must be done before the system upgrade activity

Two Major Views of the Software Life Cycle

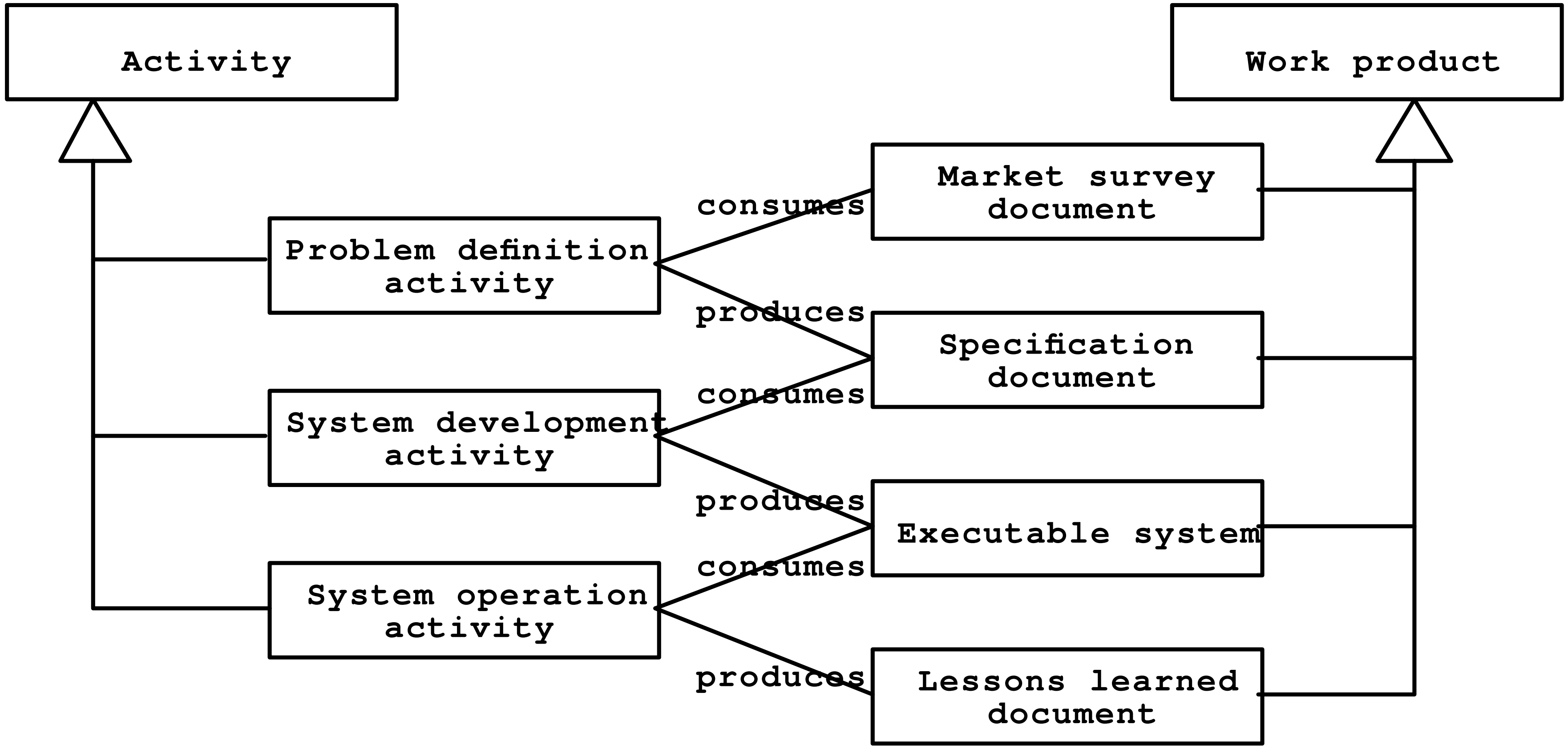
- Activity-oriented view of a software life cycle
    - Software development consists of a set of development activities
    - all the examples so far
- Entity-oriented view of a software life cycle
    - Software development consists of the creation of a set of deliverables.

Entity-centered view of Software Development

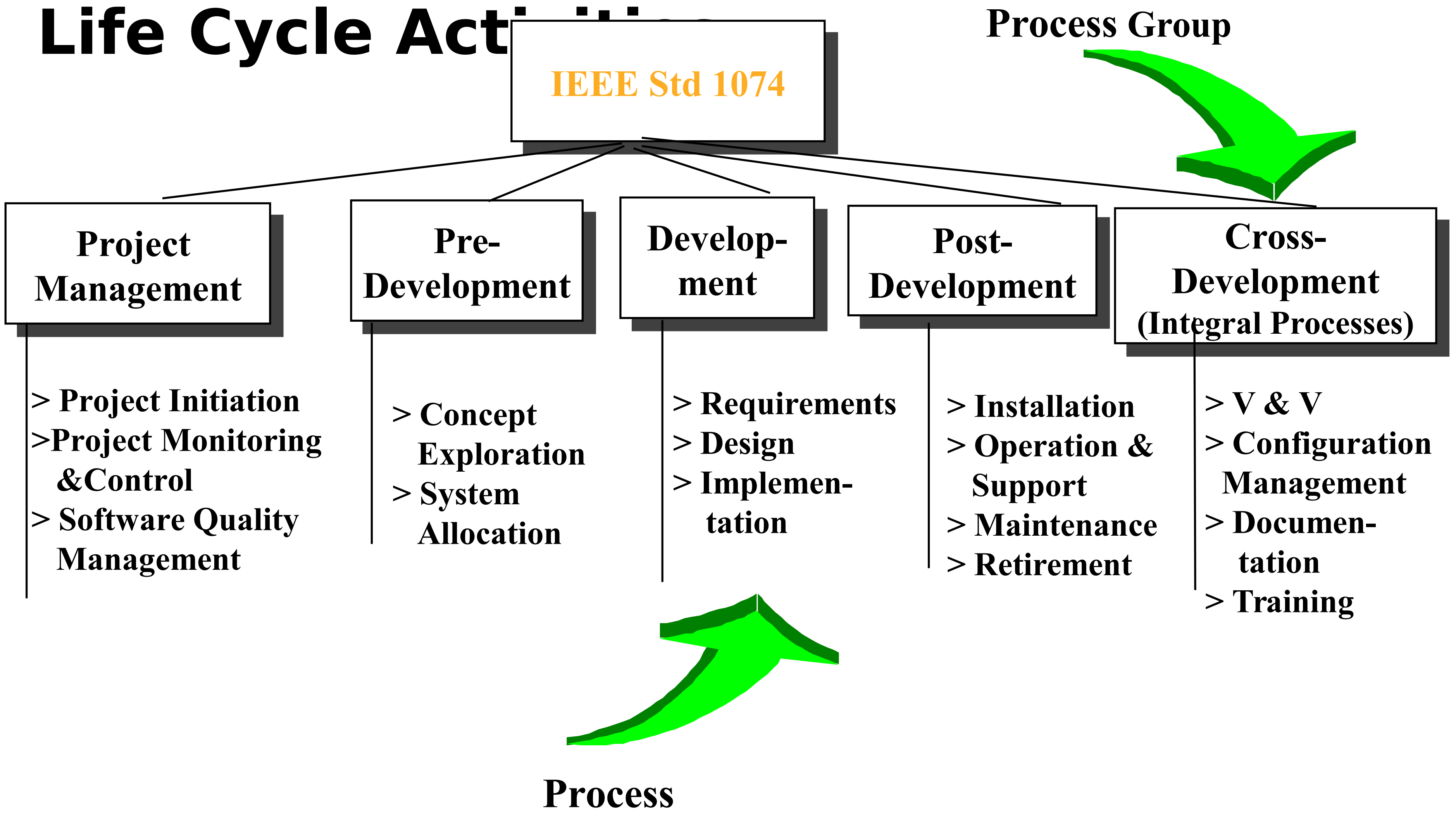


Software development consists of the creation of a set of deliverables

Combining Activities and Entities in One View

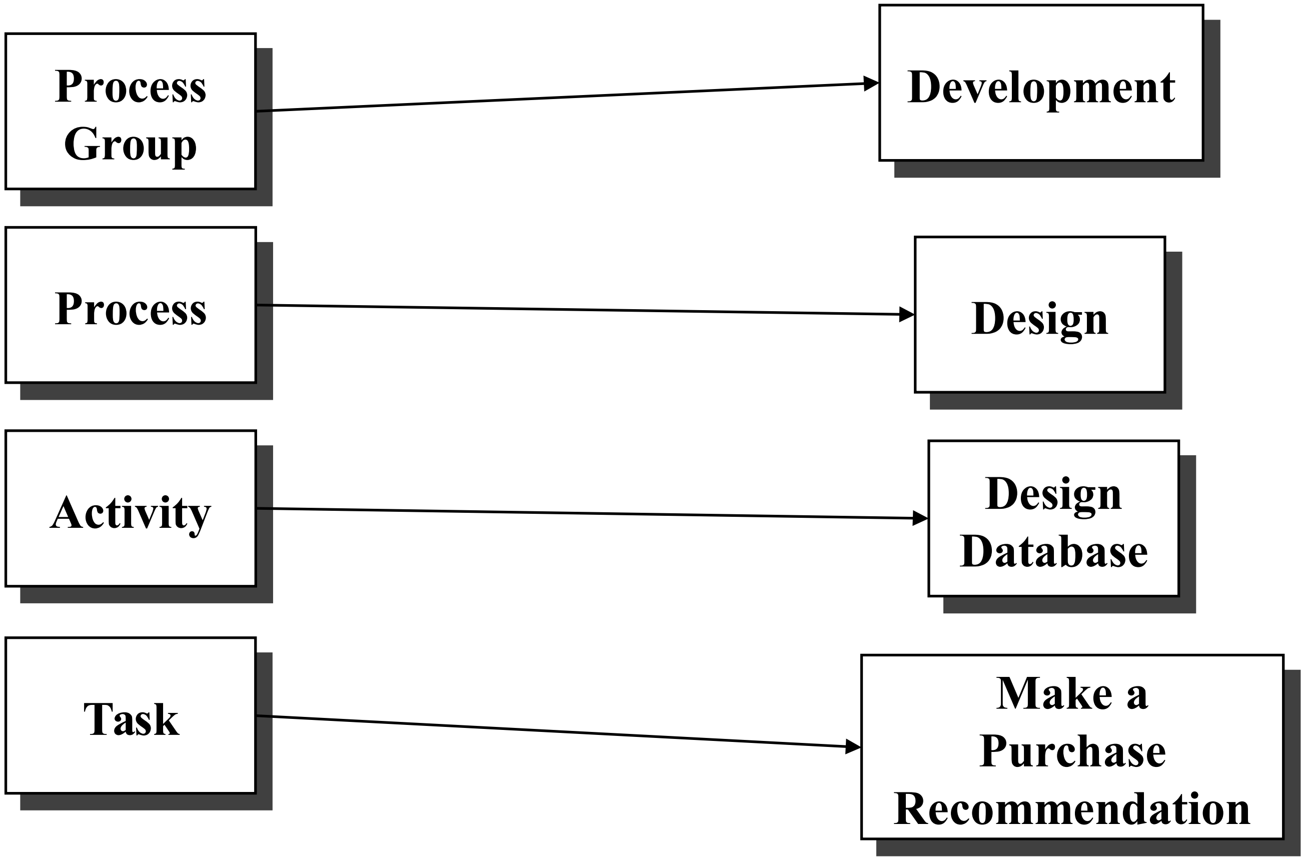


IEEE Std 1074: Standard for Software Life Cycle Activities

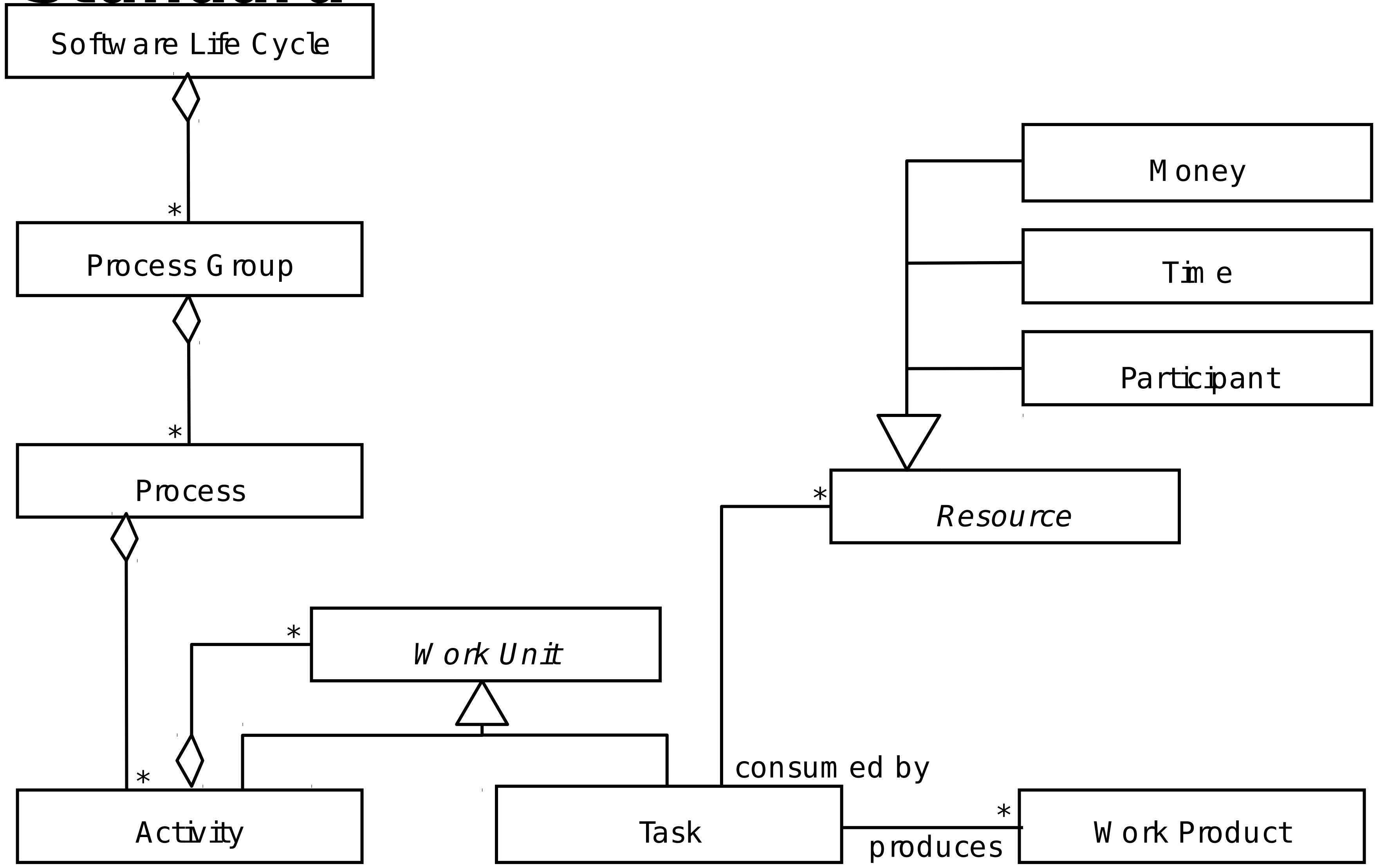


Processes, Activities and Tasks

- Process Group: Consists of a set of processes
- Process: Consists of activities
- Activity: Consists of sub activities and tasks



Object Model of the IEEE 1074 Standard



Process Maturity

- A software development process is mature
  - if the development activities are well defined and
  - if management has some control over the quality, budget and schedule 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
- CMM: Capability Maturity Model (SEI, Humphrey)

CMM levels

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

What does Process Maturity Measure?

- The real indicator of process maturity is the level of predictability of project performance (quality, cost, schedule).
- Level 1: Random, unpredictable performance
- Level 2: Repeatable performance from project to project
- Level 3: Better performance on each successive project
- Level 4: Substantial improvement (order of magnitude) in one dimension of project performance
- Level 5: Substantial improvements across all dimensions of project performance.

Key Process Areas

- To achieve a specific level of maturity, the organization must demonstrate that it addresses all the key process areas defined for that level.
- There are no key process areas for Level 1
- KPA Level 2: Basic software project management practice
- KPA Level 3: Infrastructure for single software life cycle model
- KPA Level 4: Quantitative understanding of process and deliverables
- KPA Level 5: Keep track of technology and process changes

Pros and Cons of Process Maturity

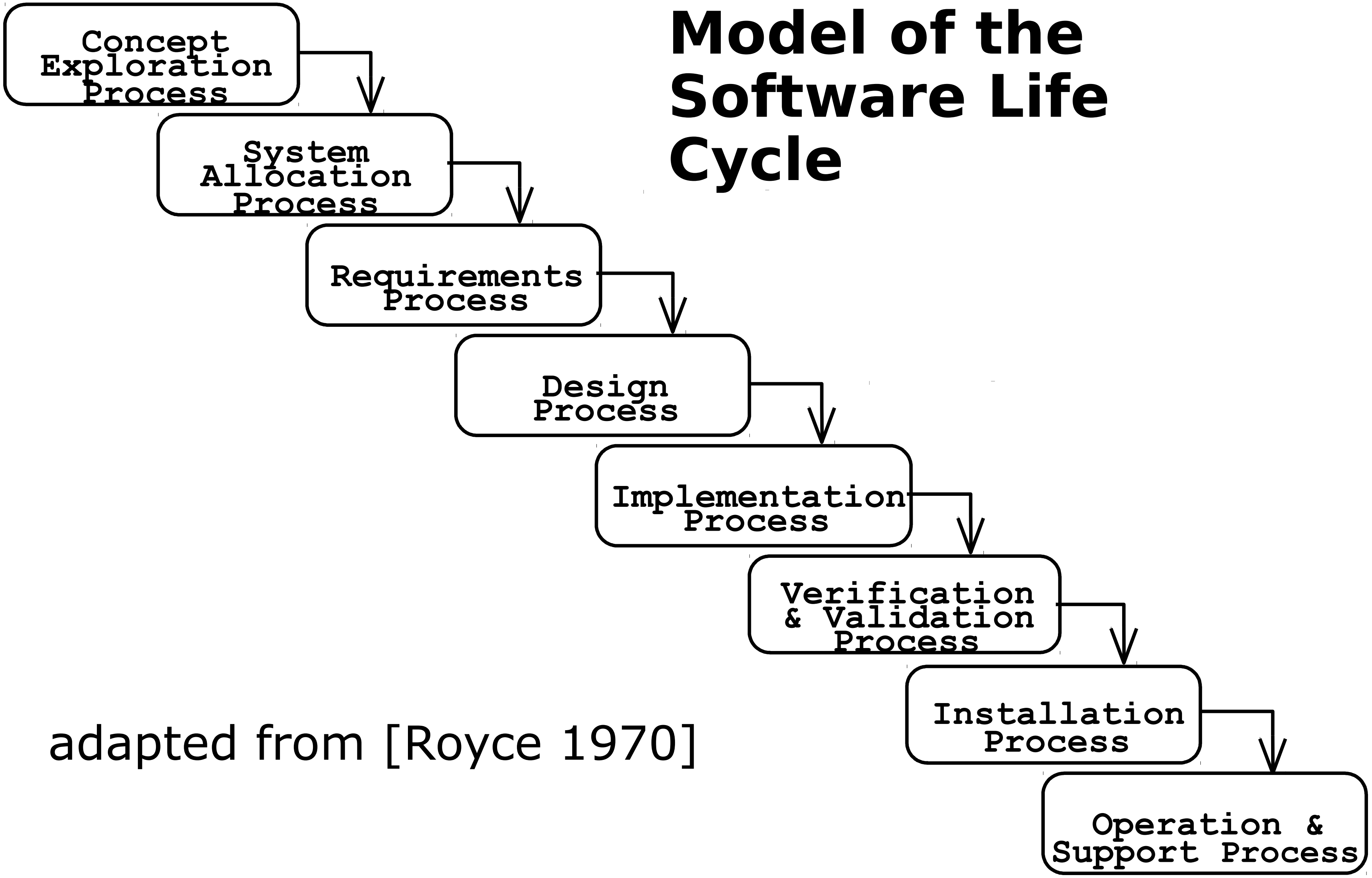
- Benefits:
  - Increased control of projects
  - Predictability of project cost and schedule
  - Objective evaluations of changes in techniques, tools and methodologies
  - Predictability of the effect of a change on project cost or schedule
- Problems:
  - Need to watch a lot (“Big brother“, „big sister“)
  - Overhead to capture, store and analyze the required information
- Agile Methodologies
  - Deemphasize the importance of process maturity

    => Lecture on Methodologies

Lecture Road Map

- 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
    - V-model
  - Iterative models
    - Boehm's spiral model (Unified Process in the next lecture)
  - Entity-oriented models
    - Issue-based model
- Capability Maturity Model

The Waterfall Model of the Software Life Cycle

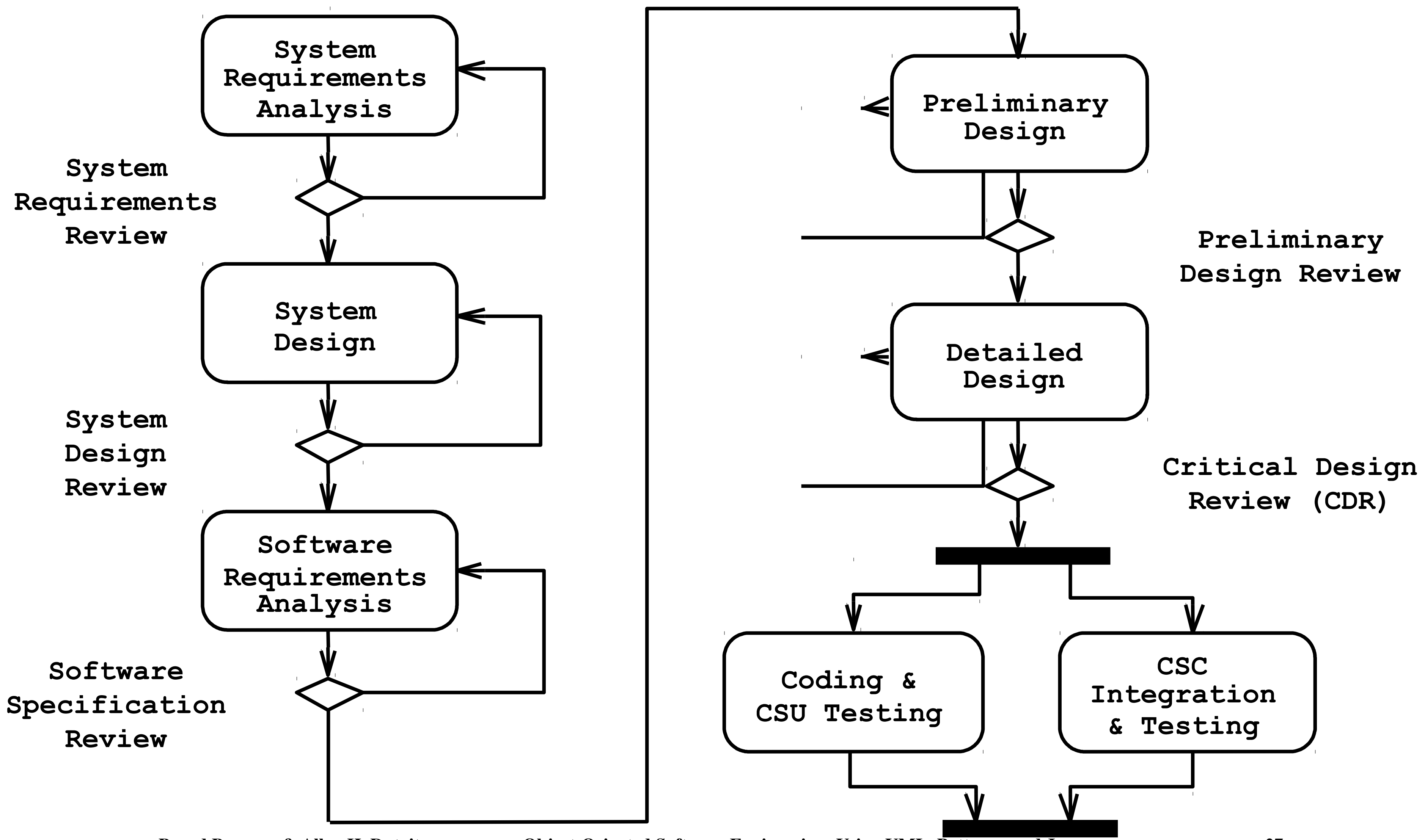

Concept Exploration Process
System Allocation Process
Requirements Process
Design Process
Implementation Process
Verification & Validation Process
Installation Process
Operation & Support Process

adapted from [Royce 1970]

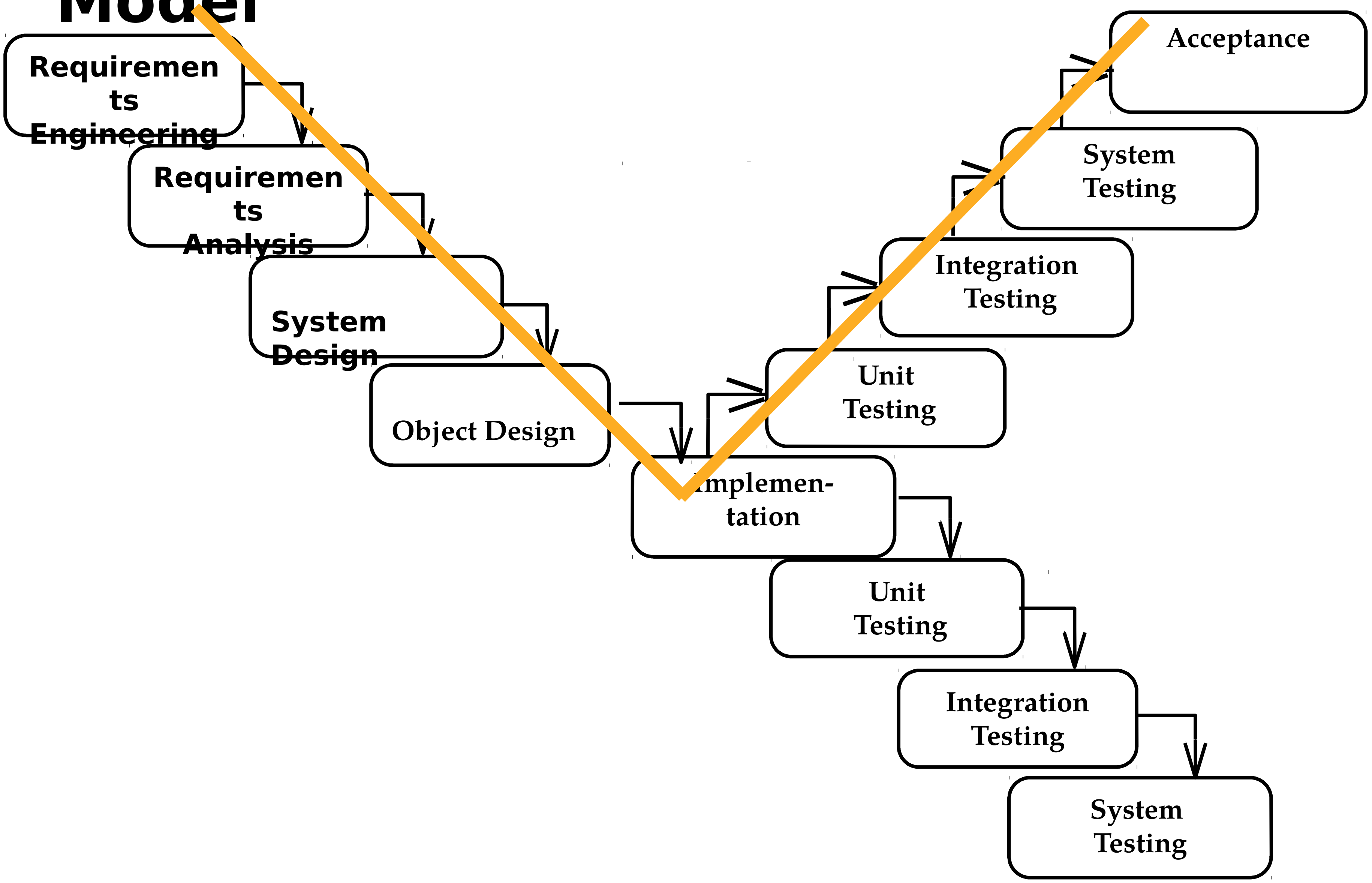
Example of a waterfall model : DOD Standard 2167A

- 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.

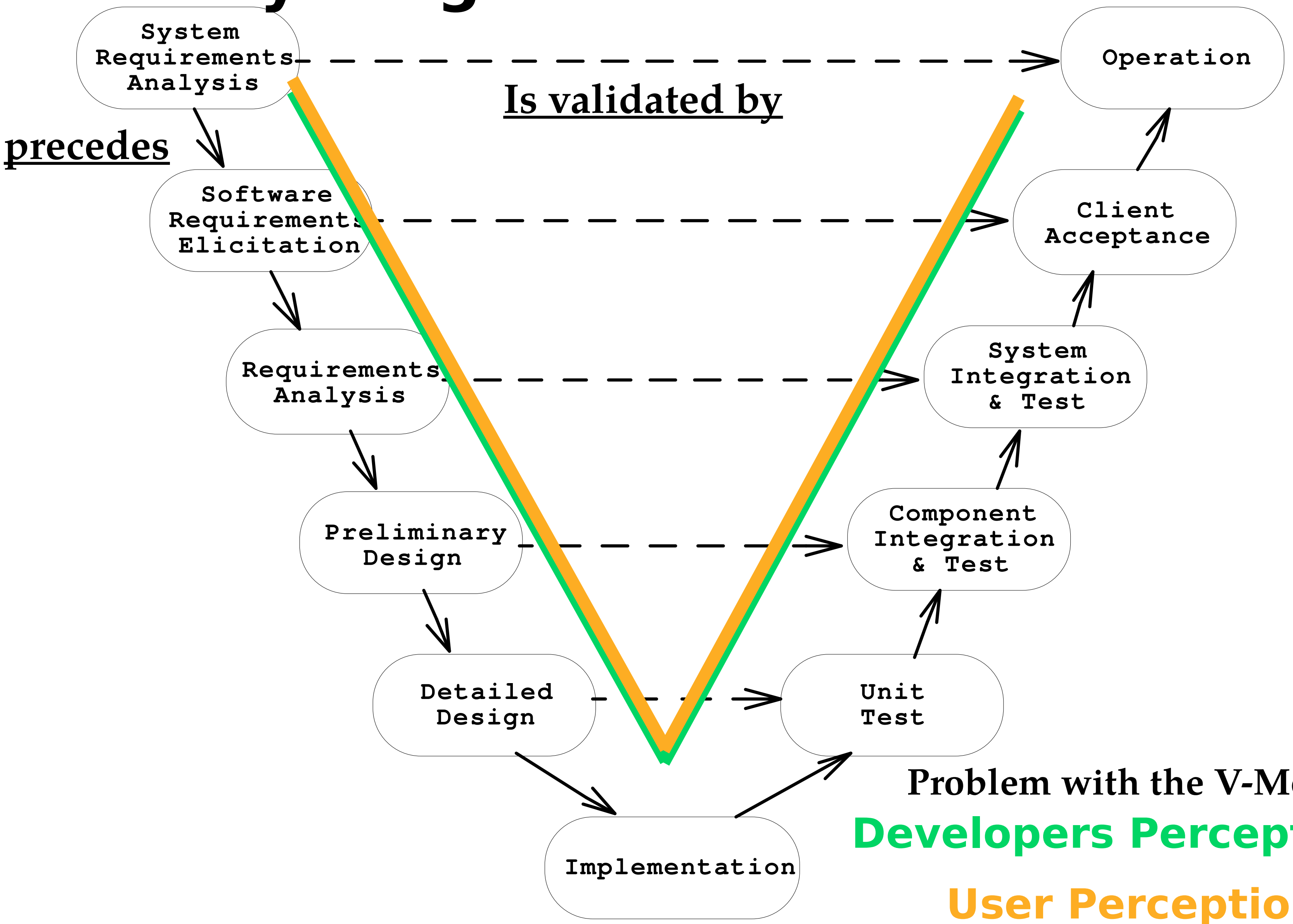
Activity Diagram of MIL DOD-STD-2167A


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



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.

The Alternative: Allow Iteration

http://en.wikipedia.org/wiki/File:Escher_Waterfall.jpg

Note: The image is copyrighted

Escher was the first:-)

Construction of Escher's Waterfall Model

http://www.cs.technion.ac.il/~gershon/EscherForReal/EscherWaterfall2Penrose.gif

Note: The image is copyrighted

Spiral Model

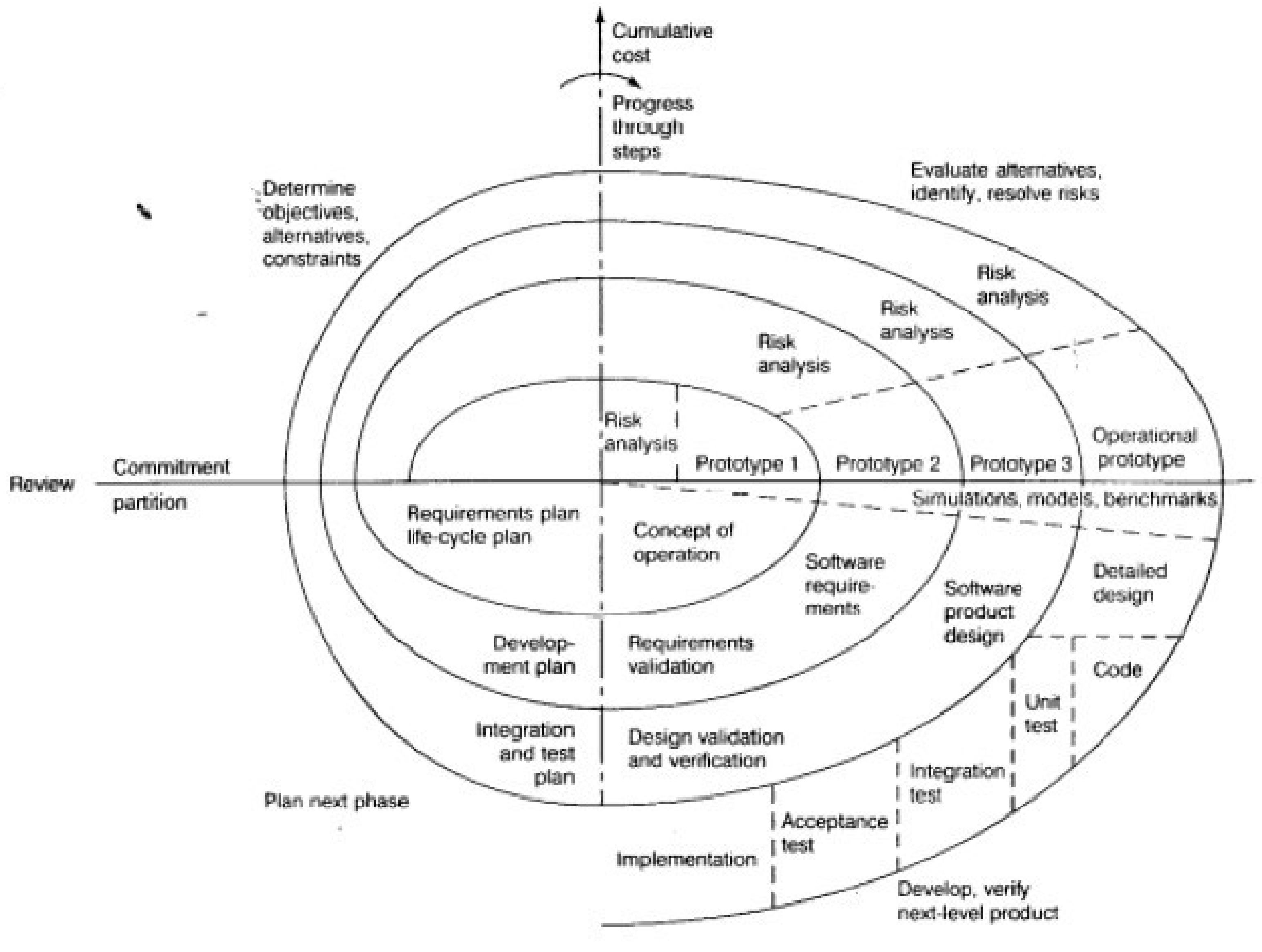
- The spiral model proposed by Boehm has the following set of activities
  - Determine objectives and constraints
  - Evaluate alternatives
  - Identify risks
  - Resolve risks by assigning priorities to risks
  - Develop a series of prototypes for the identified risks starting with the highest risk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

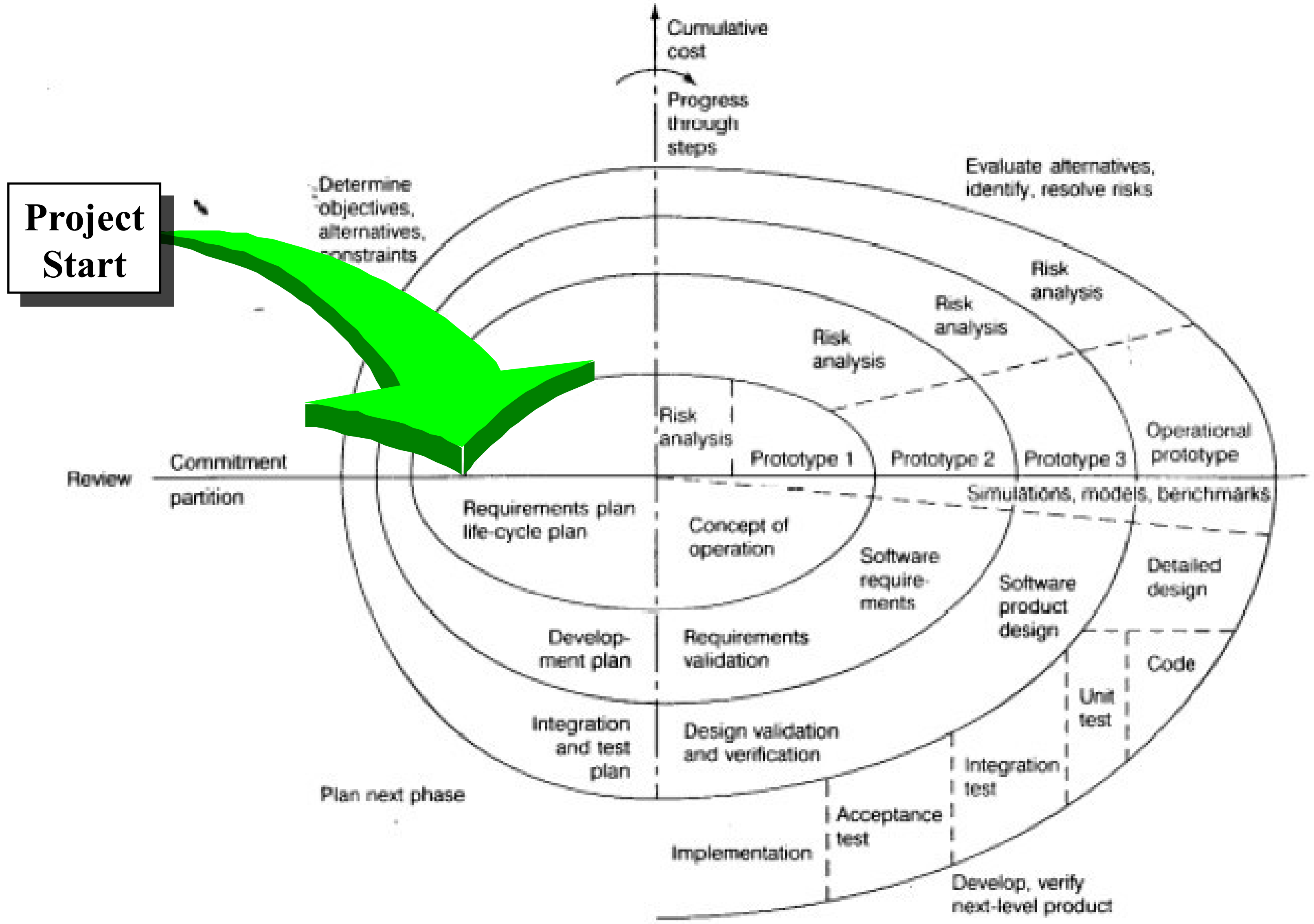
- Concept of Operations
- Software Requirements
- Software Product Design
- Detailed Design
- Code
- Unit Test
- Integration and Test
- Acceptance Test
- Implementation

- 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

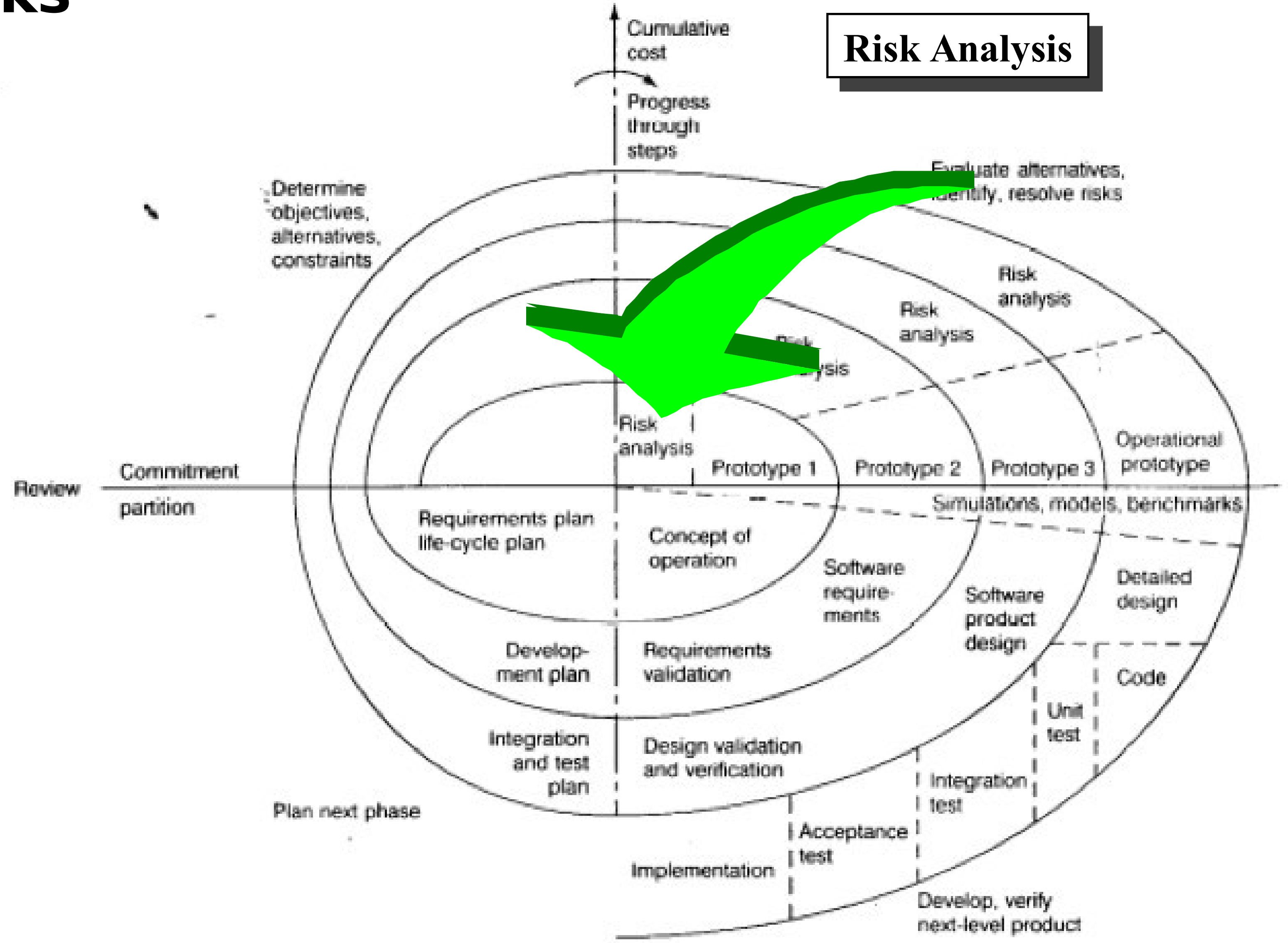


Round 1, Concept of Operations, Quadrant IV: Determine Objectives,Alternatives & Constraints

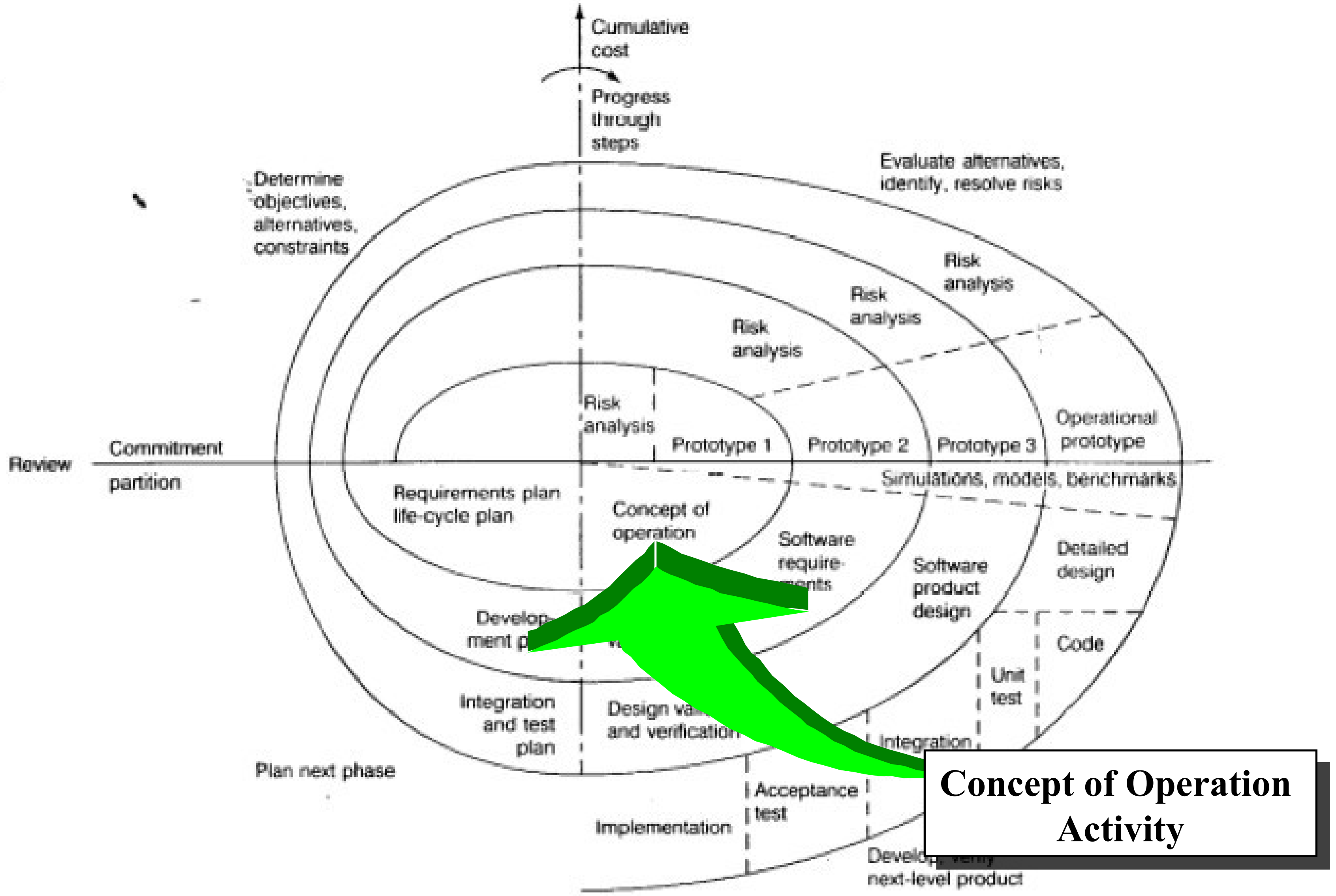


Round 1, Concept of Operations, Quadrant I:

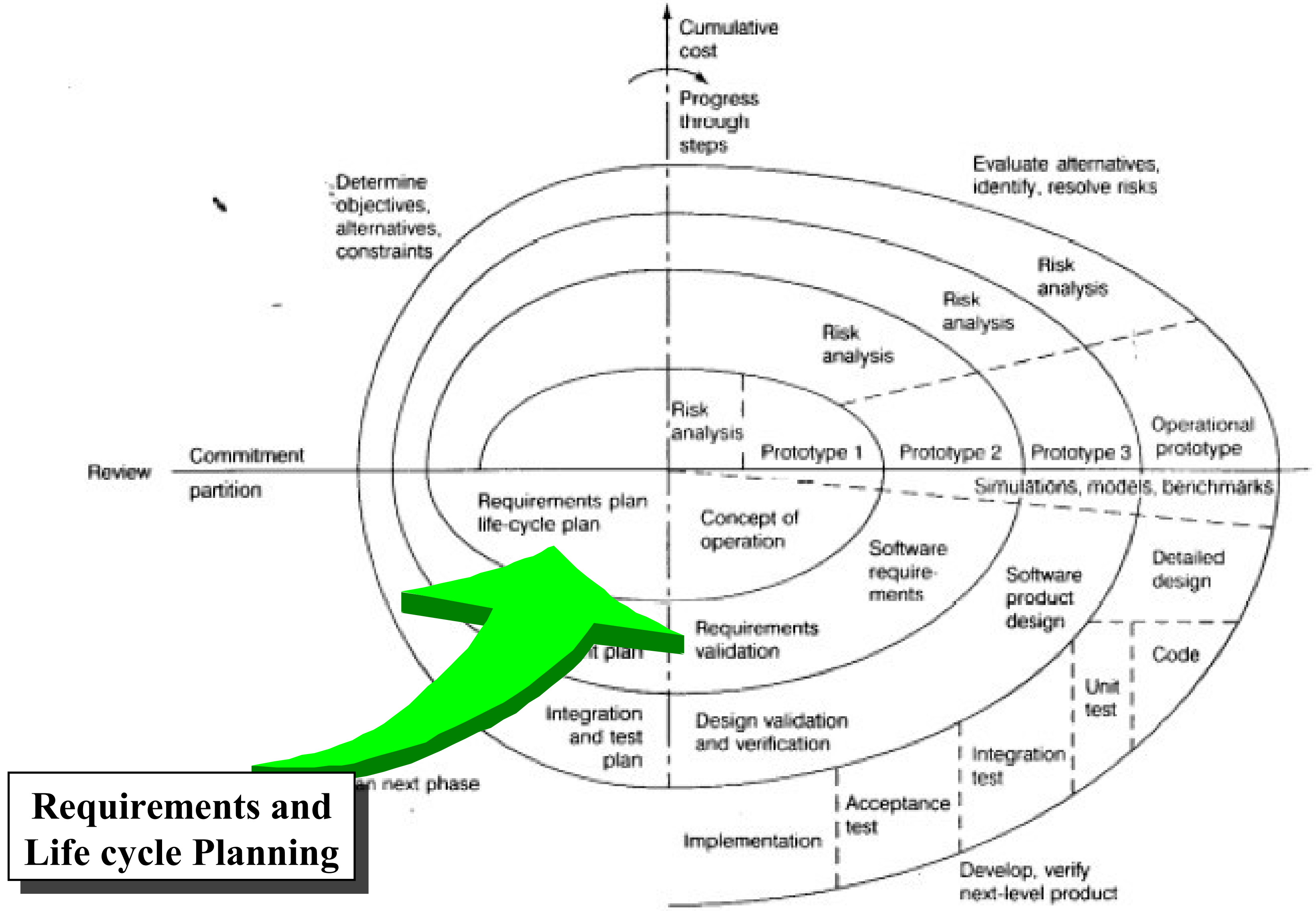
Evaluate Alternatives, identify & resolve Risks



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

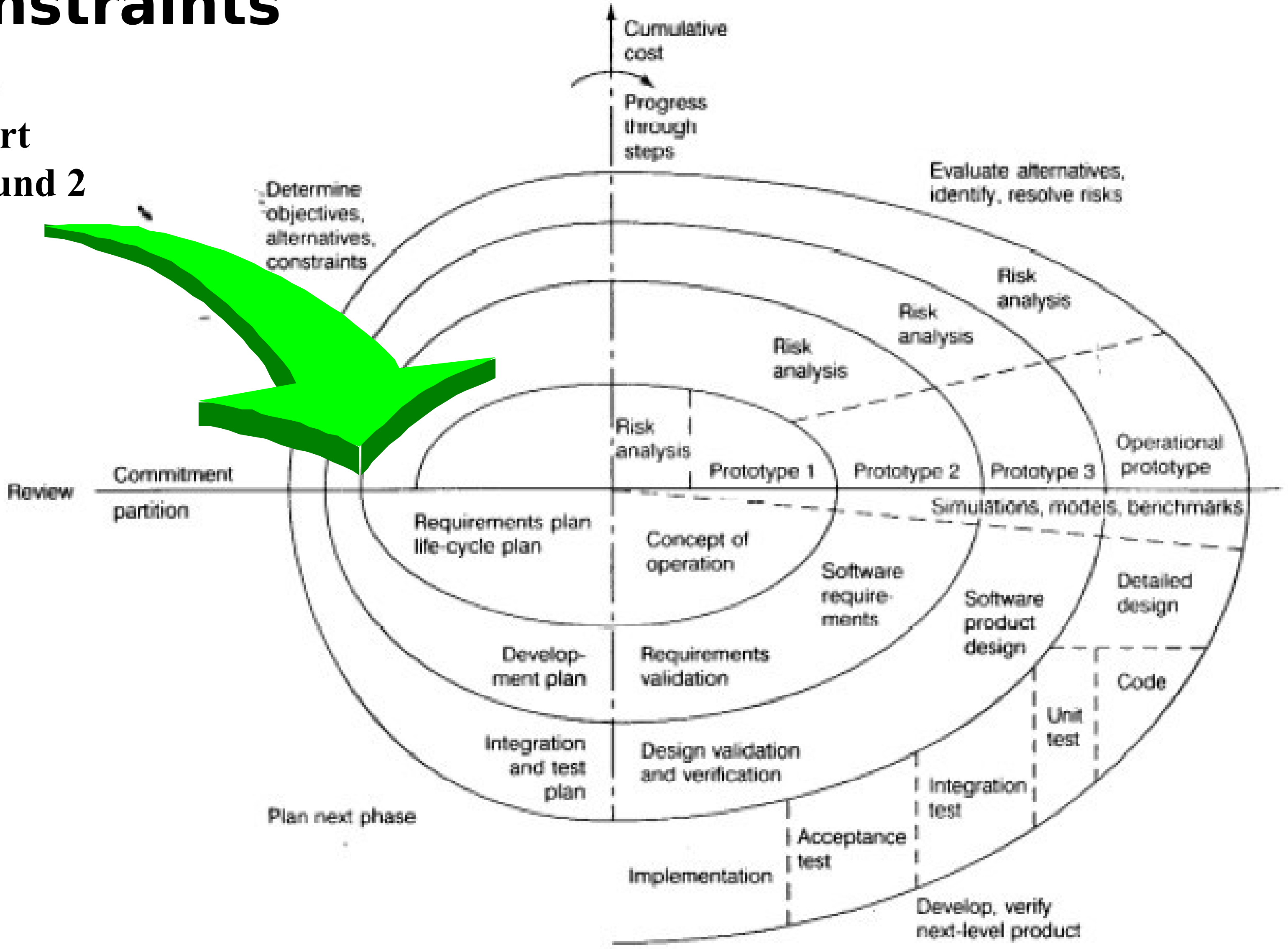


Round 1, Concept of Operations, Quadrant III:
Prepare for Next Activity

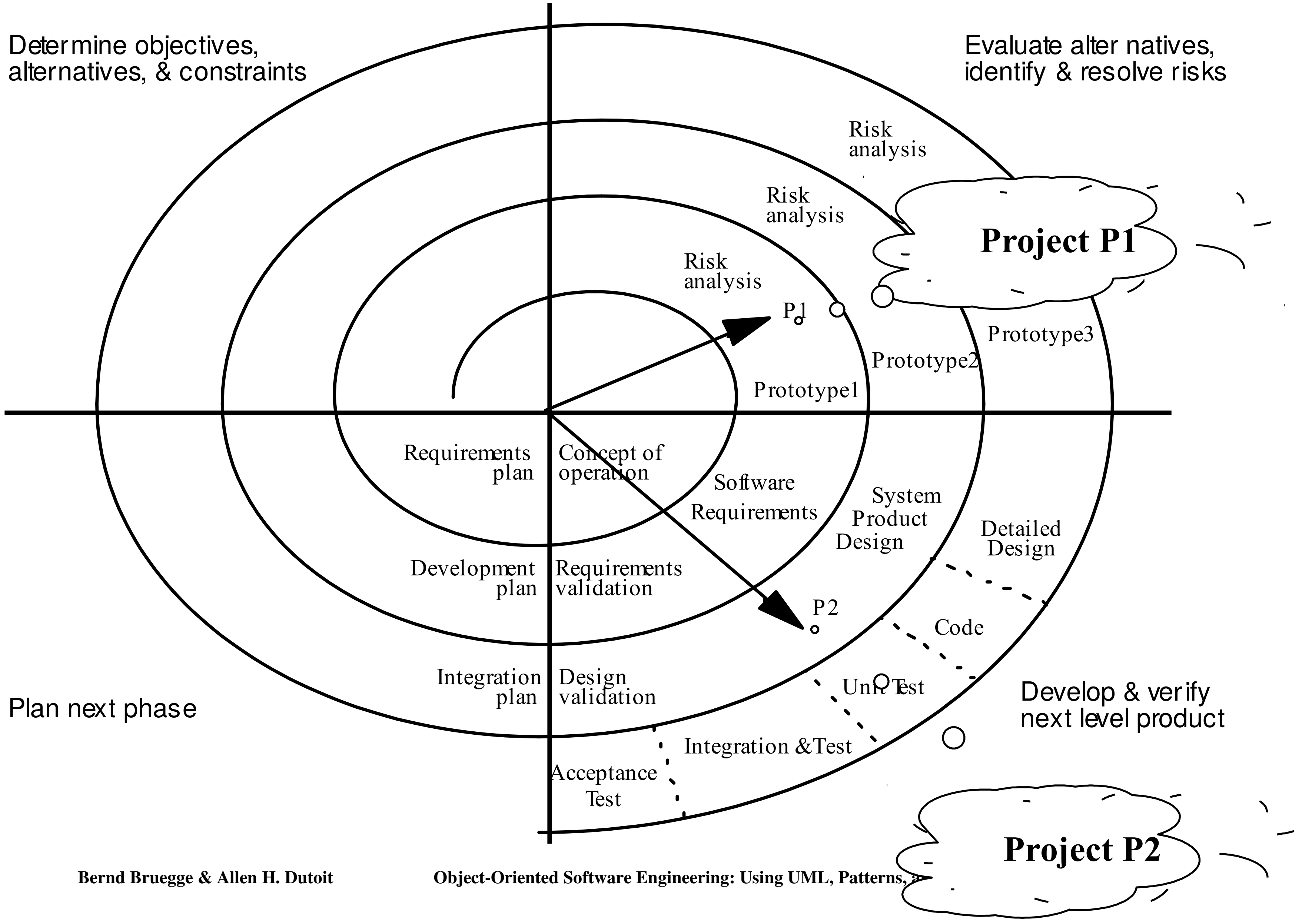


Round 2, Software Requirements, Quadrant IV:
Determine Objectives,Alternatives & Constraints

Start
of Round 2



Comparison of Projects

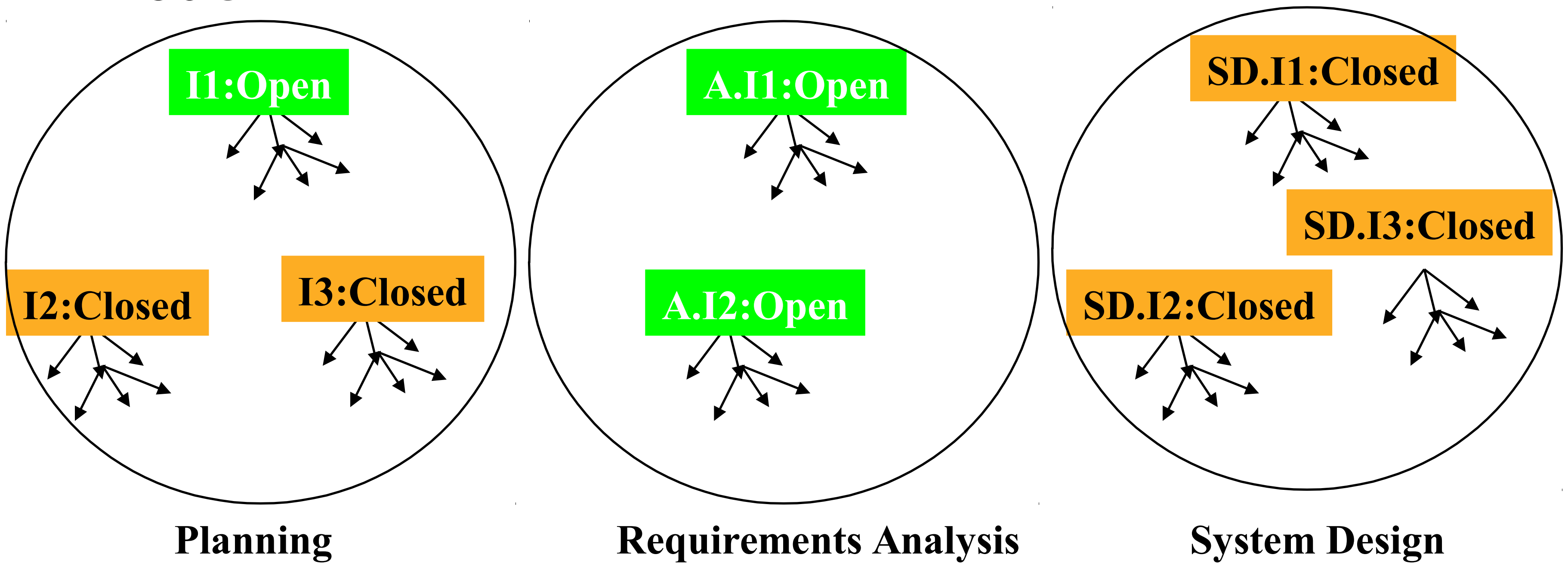


Limitations of Waterfall and Spiral Models

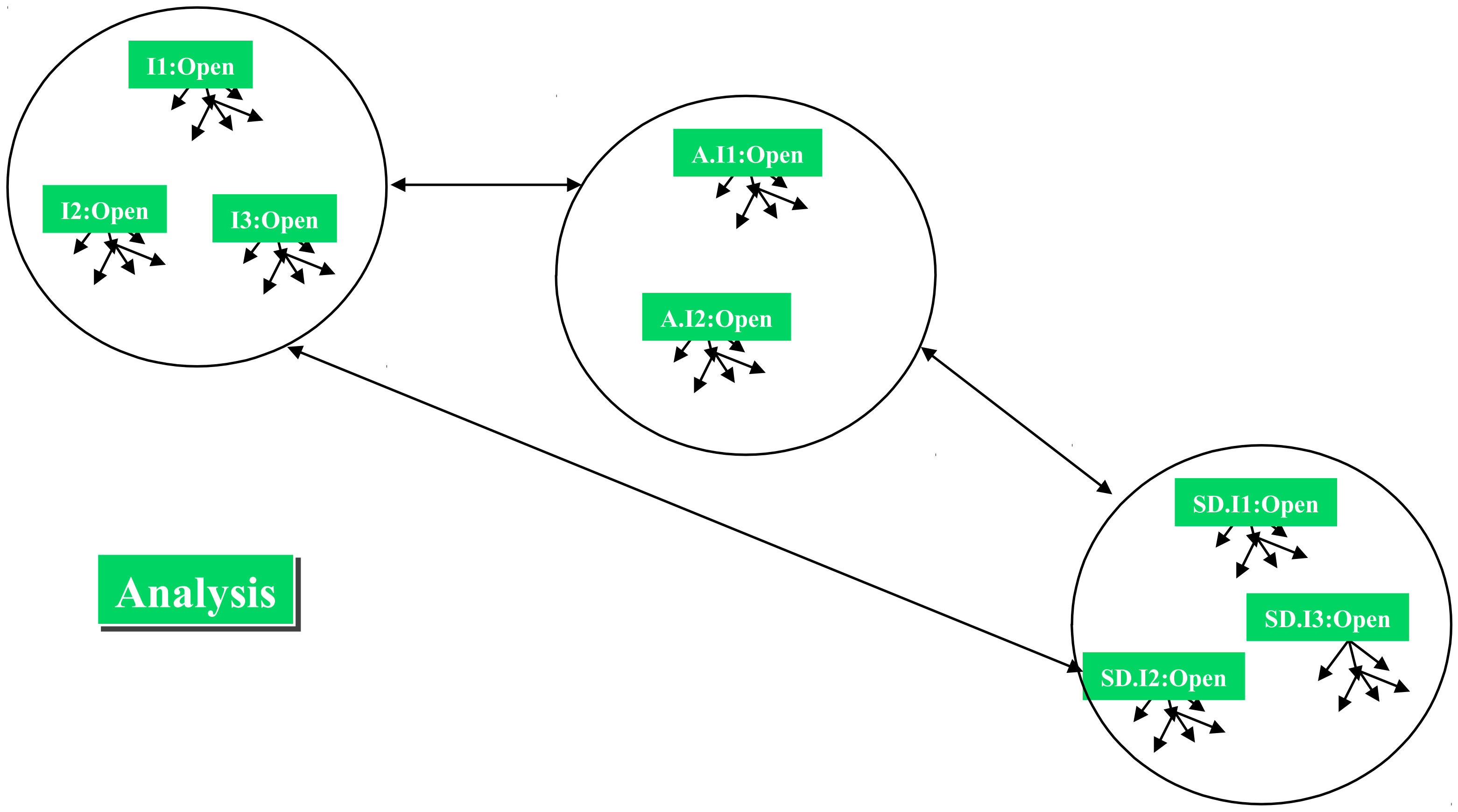
- Neither of these models deal well with frequent change
  - The Waterfall model assumes that once you are done with a phase, all issues covered in that phase are closed and cannot be reopened
  - The Spiral model can deal with change between phases, but does not allow change within a phase
- What do you do if change is happening more frequently?
  - “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

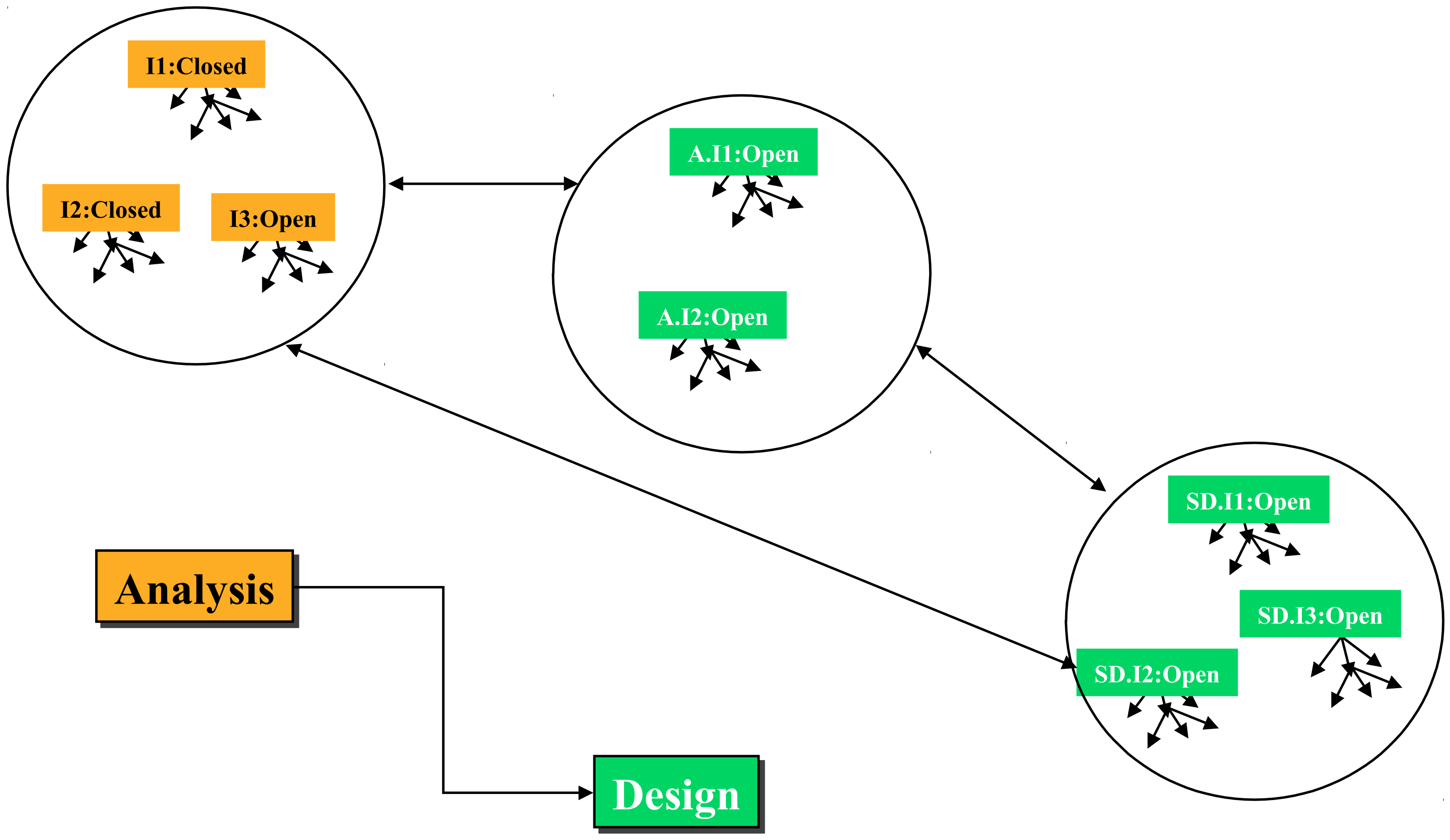


Waterfall Model: Analysis Phase

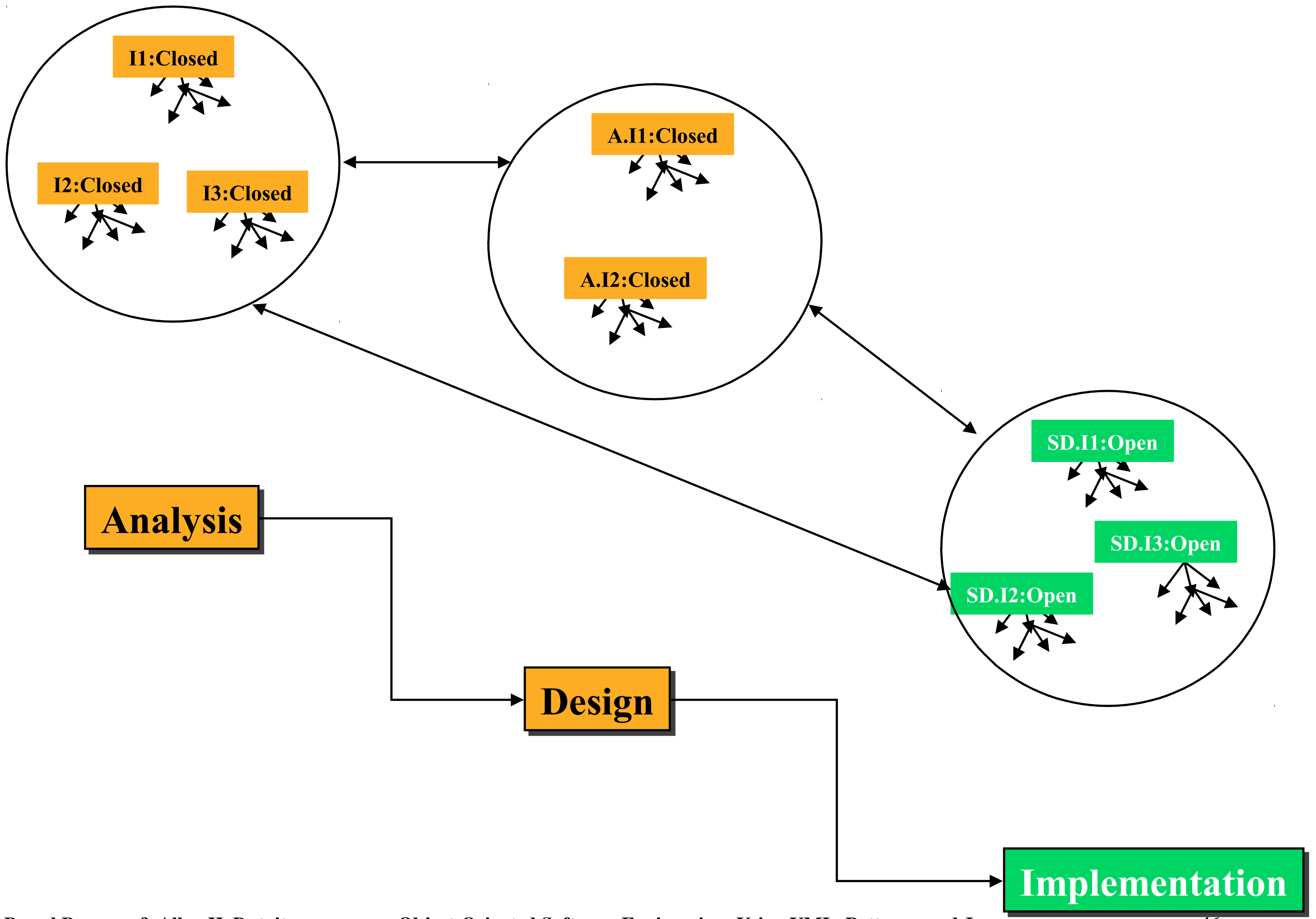


Analysis

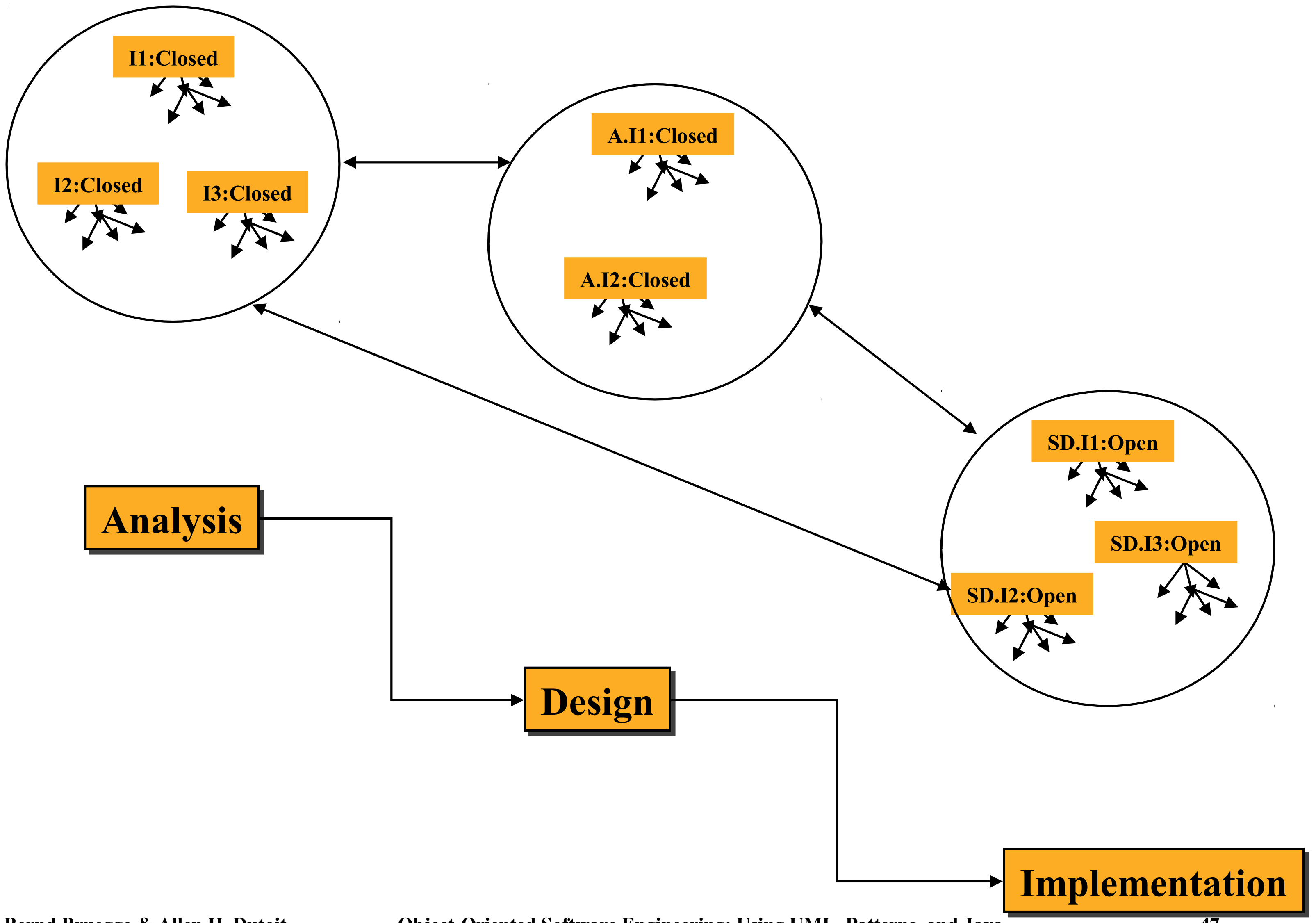
Waterfall Model: Design Phase



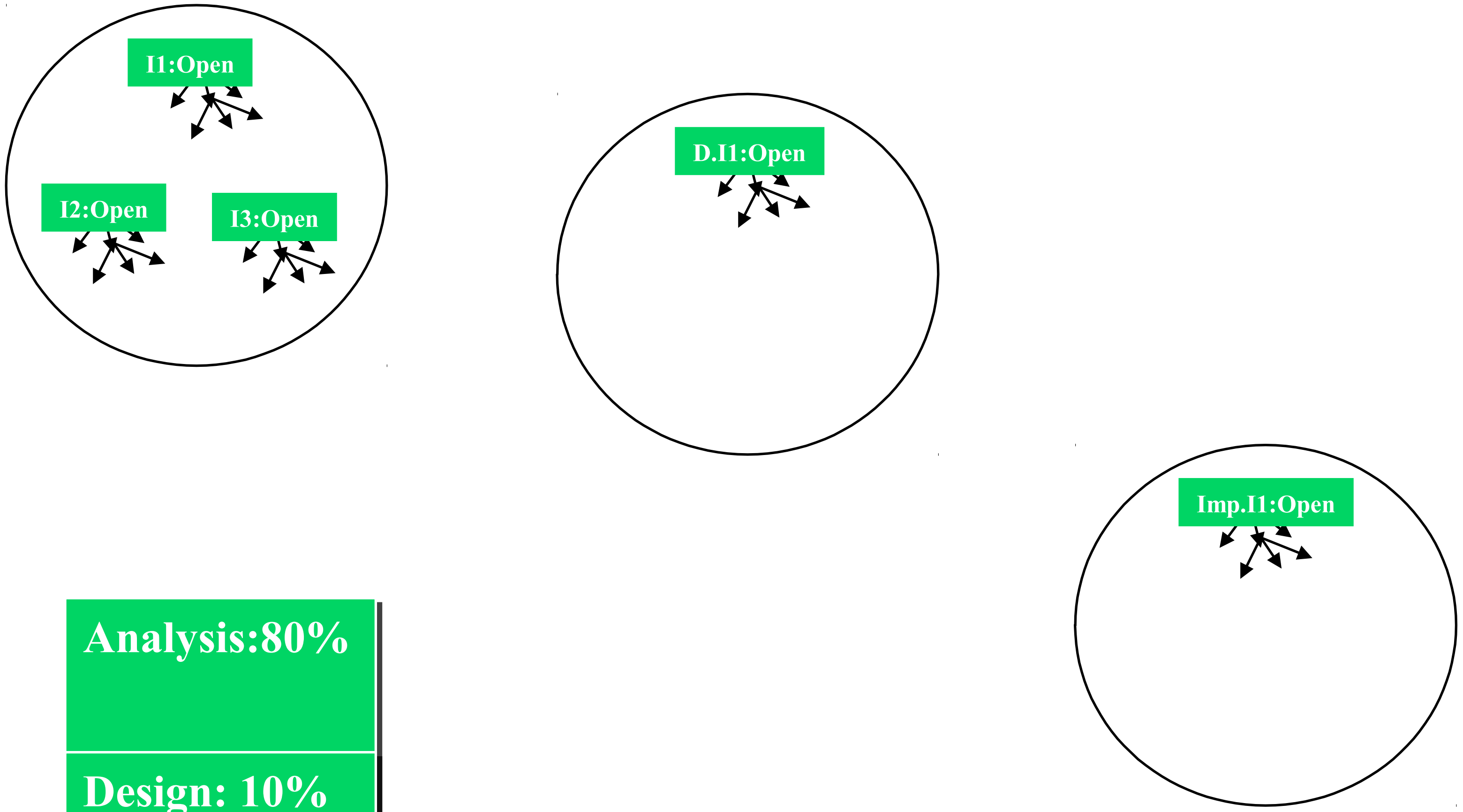
Waterfall Model: Implementation Phase



Waterfall Model: Project is Done



Issue-Based Model: Analysis Phase

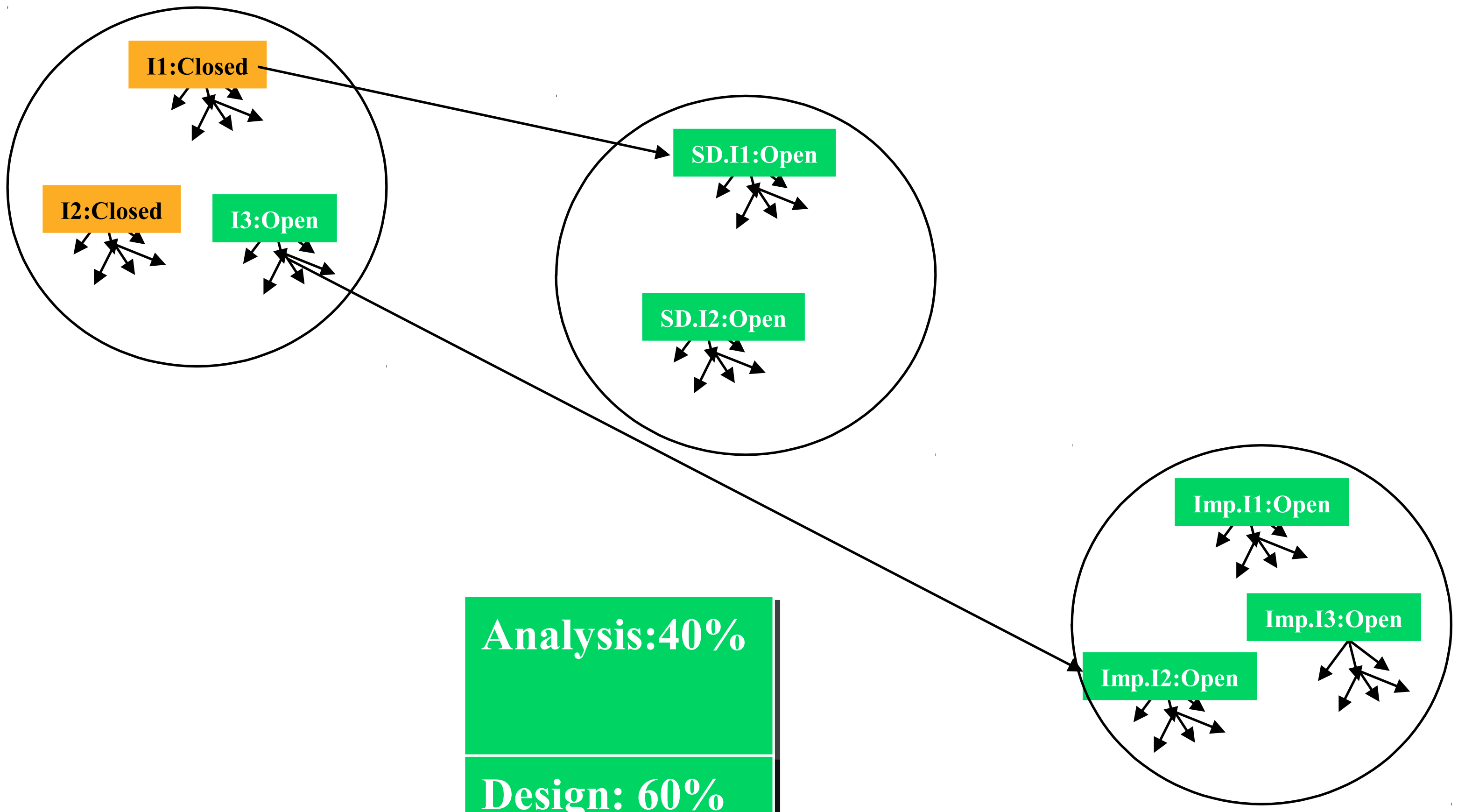


Analysis:80%

Design: 10%

Implemen-
tation: 10%

Issue-Based Model: Design Phase

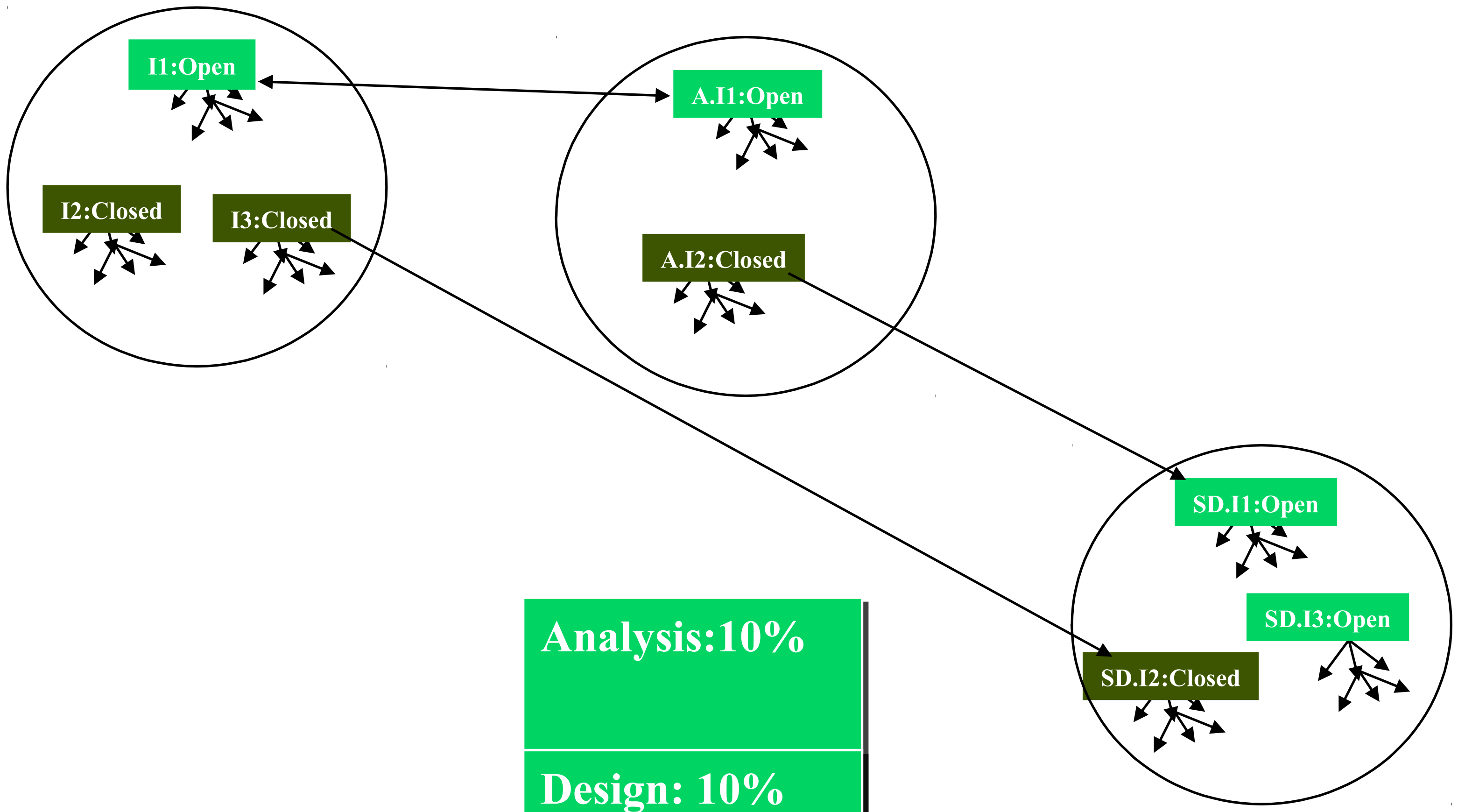


Analysis:40%

Design: 60%

Implemen-
tation: 0%

Issue-Based Model: Implementation Phase

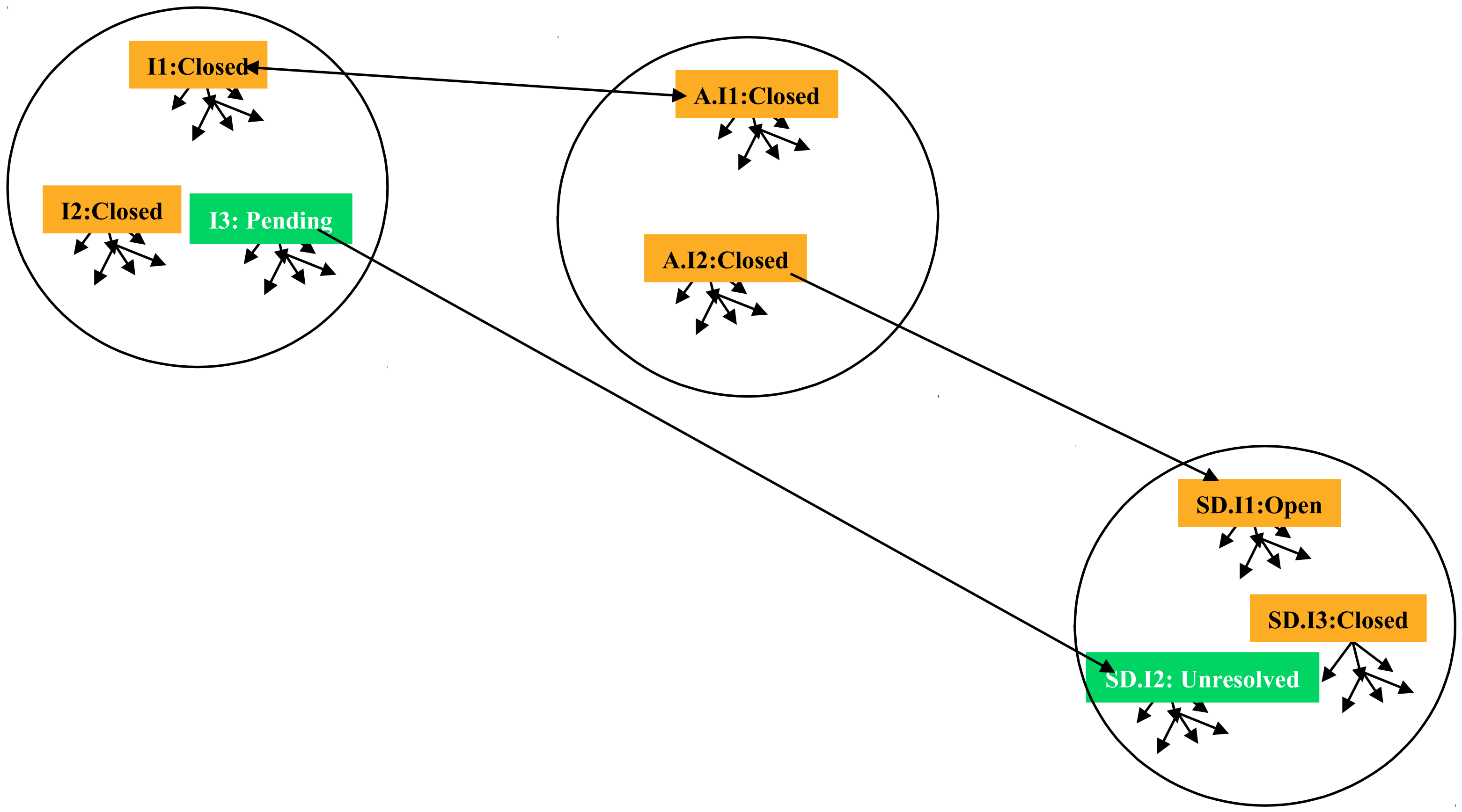


Analysis:10%

Design: 10%

Implemen-
tation: 60%

Issue-Based Model: Prototype is Done



Frequency of Change and Choice of Software Lifecycle Model

PT = Project Time, MTBC = Mean Time Between Change

- Change rarely occurs (MTBC » PT)
  - Linear Model (Waterfall, V-Model)
  - Open issues are closed before moving to next phase
- Change occurs sometimes (MTBC ≈ PT)
  - Iterative model (Spiral Model, Unified Process)
  - Change occurring during phase may lead to iteration of a previous phase or cancellation of the project
- Change is frequent (MTBC « PT)
  - Issue-based Model (Concurrent Development, Scrum)
  - Phases are never finished, they all run in parallel.

Summary

- Software life cycle models
  - Sequential models
    - Pure waterfall model and V-model
    - Sawtooth model
  - Iterative model
    - Boehm's spiral model
      - Rounds
      - Comparison of projects
  - Prototyping
    - Revolutionary and evolutionary prototyping
    - Time-boxed prototyping instead of rapid prototyping
  - Entity-oriented models
    - Issue-based model
    - Sequential models can be modeled as special cases of the issue-based model.

Additional and Backup Slides

Questions

- Boehm‘s spiral model is usually shown in a polar coordinate system.
- Why did Boehm use such a notation?
- What are the problems with such a notation?
- What happens if you attempt to remodel the spiral model in UML?

Industry Distribution across Maturity Levels (State of the Software Industry in 1995)

Maturity Level	Frequency
1 Initial	70%
2 Repeatable	15%
3 Defined	< 10%
4 Managed	< 5%
5 Optimizing	< 1%

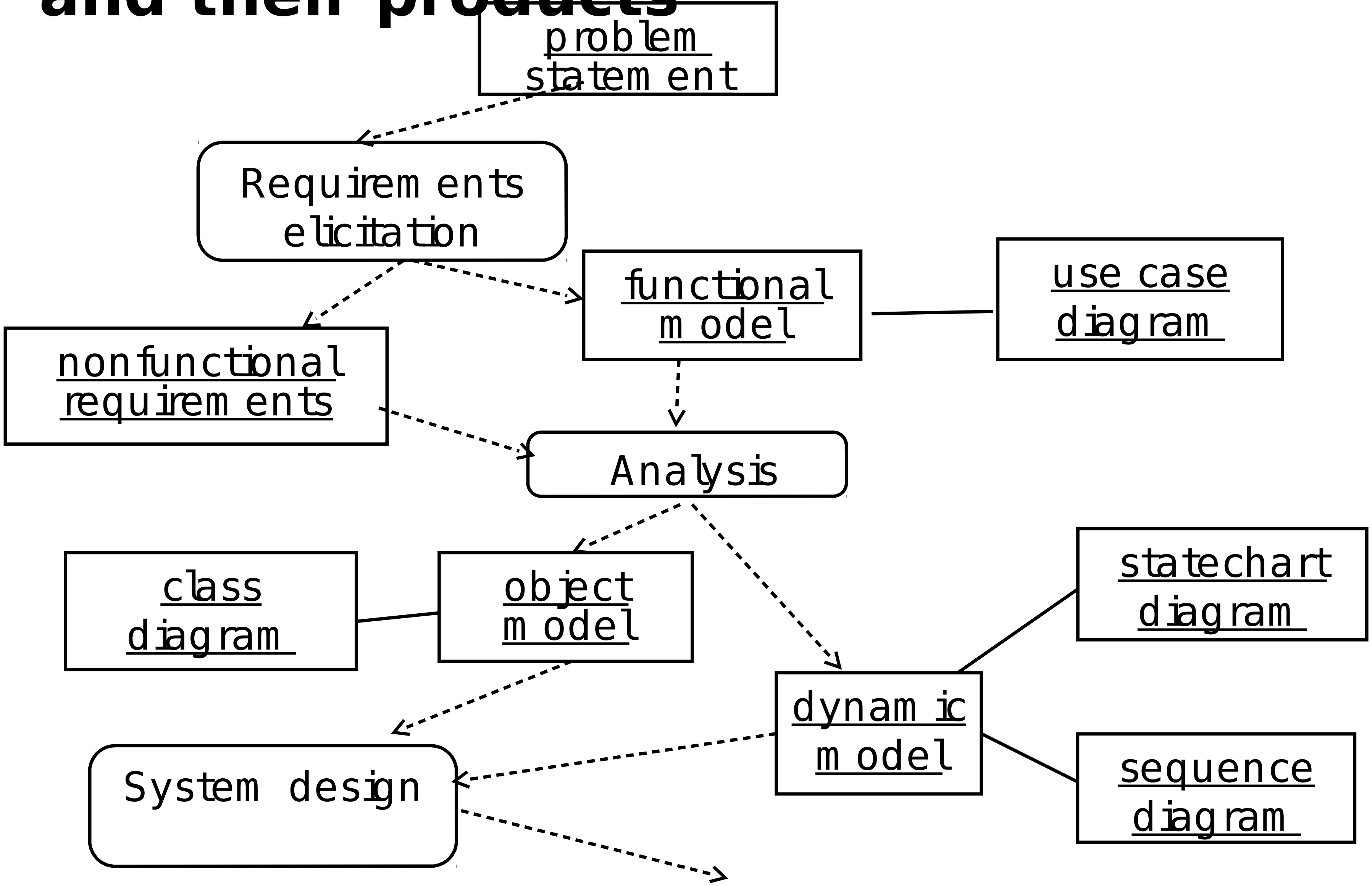
Source: Royce, Project Management, page 364
Citation needs to be updated

Movie of Escher’s Waterfall Model

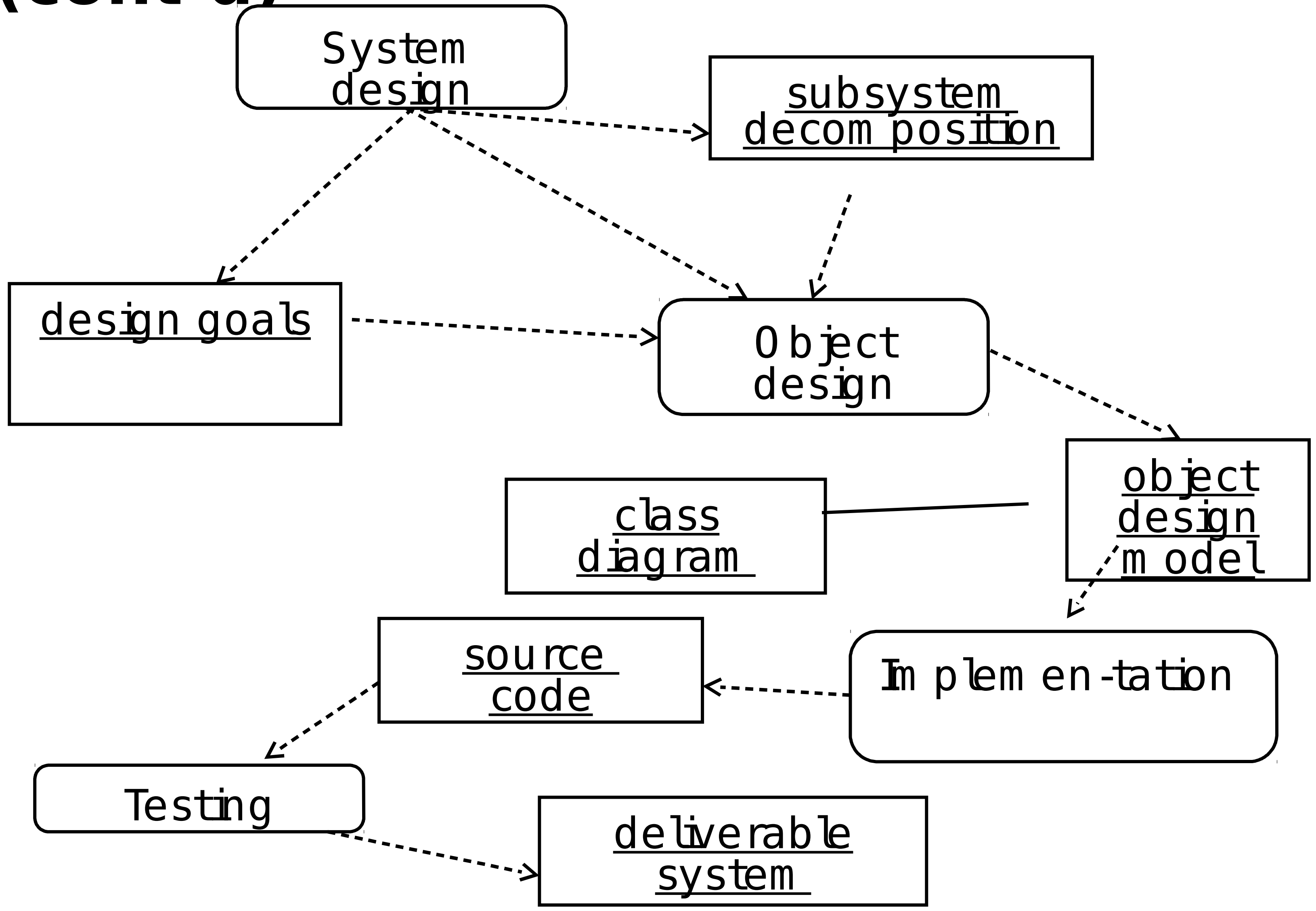
Escher for Real

http://www.cs.technion.ac.il/~gershon/EscherForRealWaterfallFull.avi

OOSE-Book: Development activities and their products



OOSE- Development activities (cont'd)



Insert: Types of Prototypes

- 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 ("Hack")
  - Implement part of the system to learn more about the requirements.
  - Good for paradigm breaks

Types of Prototyping

- Revolutionary Prototyping
  - Also called specification prototyping
  - Get user experience with a throwaway version to get the requirements right, then build the whole system
    - Advantage: Can be developed in a short amount of time.
    - Disadvantage: Users may have to accept that features in the prototype are expensive to implement
- 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 limits the duration of the prototype development**

References

- Readings used for this lecture
  - [Bruegge-Dutoit] Chapter 12
  - [Humphrey 1989] Watts Humphrey, Managing the Software Process, SEI Series in Software Engineering, Addison Wesley, ISBN 0-201-18095-2
- Additional References
  - [Royce 1970] Winston Royce, Managing the Development of Large Software Systems, Proceedings of the IEEE WESCON, August 1970, pp. 1-9
  - SEI Maturity Questionaire, Appendix E.3 in [Royce 1998], Walker Royce, **Software Project Management, Addison-Wesley, ISBN0-201-30958-0**

Additional References

- Overview of Capability Maturity Models
  - http://www.sei.cmu.edu/cmm/cmms/cmms.html
- Personal Process
  - http://www.sei.cmu.edu/tsp/psp.html
- Team Process:
  - http://www.sei.cmu.edu/tsp/tsp.html

Summary

- Software life cycle
  - The development process is broken into individual pieces called software development activities
- Software development standards
  - IEEE 1074
  - The standard allows the lifecycle to be tailored
- Capability Maturity Model
  - An attempt to characterize the maturity of a software development organization following a software lifecycle model.

Maturity Level 1: Chaotic Process

- Ad hoc approach to software development activities
- No problem statement or requirements specification
- Output is expected
  - but nobody knows how to get there in a deterministic fashion
- Similar projects may vary widely in productivity
  - "when we did it last year we got it done"

- ***Level 1 Metrics:*** Rate of Productivity (Baseline comparisons, Collection of data is difficult)
  - Product size (LOC, number of functions, etc)
  - Staff effort ("Man-years", person-months)
- ***Recommendation:*** Level 1 managers & developers should not concentrate on metrics and their meanings,
  - They should first attempt to adopt a process model (waterfall, spiral model, saw-tooth, macro/micro process lifecycle, unified process)

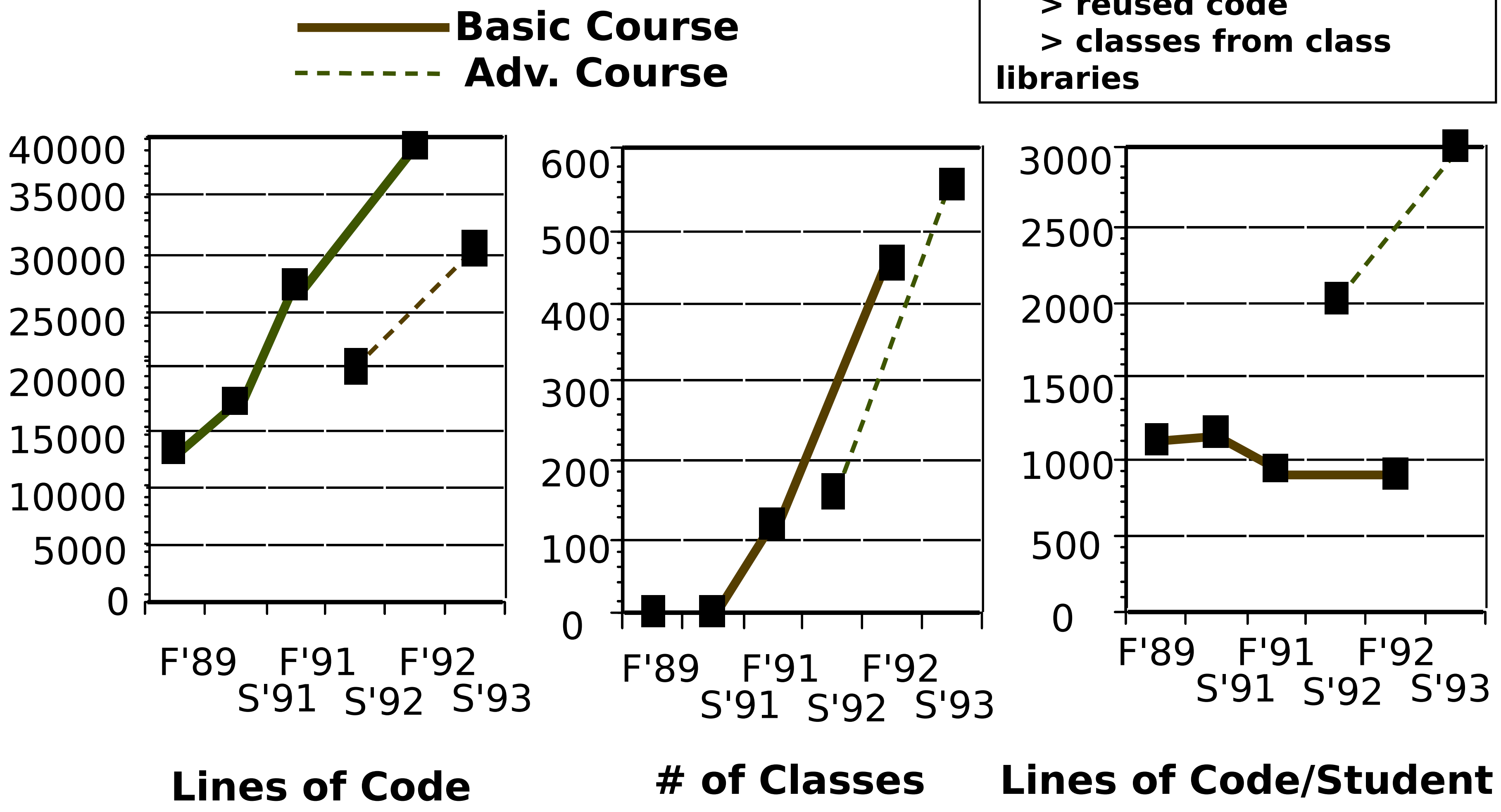
Maturity Level 2: Repeatable Process

- Inputs and outputs are defined
  - Input: Problem statement or requirements specification
  - Output: Source code
- Process itself is a black box (activities within process are not known)
  - No intermediate products are visible
  - No intermediate deliverables
- Process is repeatable due to some individuals who know how to do it
  - "Champion"

- ***Level 2 Metrics:***
  - Software size: Lines of code, Function points, classes or method counts
  - Personnel efforts: person-months
  - Technical expertise
    - Experience with application domain
    - Design experience
    - Tools & Method experience
  - Employee turnover within project

Example: LOC (Lines of Code) Metrics

Numbers do not include:
> reused code
> classes from class libraries



Maturity Level 3: Defined Process

- Activities of software development process are well defined with clear entry and exit conditions.
- Intermediate products of development are well defined and visible
- ***Level 3 Metrics*** (in addition to metrics from lower maturity levels):
  - Requirements complexity: Number of classes, methods, interfaces
  - Design complexity: Number of subsystems, concurrency, platforms
  - Implementation complexity: Number of code modules, code complexity
  - Testing complexity: Number of paths to test, number of class interfaces to test
  - Thoroughness of Testing:
    - Requirements defects discovered
    - Design defects discovered
    - Code defects discovered
    - Failure density per unit (subsystem, code module, class

Maturity Level 4: Managed Process

- Uses information from early project activities to set priorities for later project activities (intra-project feedback)
  - The feedback determines how and in what order resources are deployed
- Effects of changes in one activity can be tracked in the others
- ***Level 4 Metrics***:
  - Number of iterations per activity
  - Code reuse: Amount of producer reuse (time designated for reuse for future projects?)
  - Amount of component reuse (reuse of components from other projects and components)
- Defect identification:
  - How and when (which review) are defects discovered?
- Defect density:
  - When is testing complete?
- Configuration management:
  - Is it used during the development process? (Has impact on tracability of changes).
- Module completion time:
  - Rate at which modules are completed (Slow rate indicates that the process needs to be improved).

Maturity Level 5: Optimizing Process

- Measures from software development activities are used to change and improve the current process
- This change can affect both the organization and the project:
  - The organization might change its management scheme
  - A project may change its process model before completion

Determining the Maturity of a Project

- Level 1 questions:
  - Has a process model been adopted for the Project?
- Level 2 questions:
  - Software size: How big is the system?
  - Personnel effort: How many person-months have been invested?
  - Technical expertise of the personnel:
    - What is the application domain experience
    - What is their design experience
    - Do they use tools?
    - Do they have experience with a design method?
  - What is the employee turnover?

Maturity Level 3 Questions

- What are the software development activities?
- Requirements complexity:
  - How many requirements are in the requirements specification?
- Design complexity:
  - Does the project use a software architecture? How many subsystems are defined? Are the subsystems tightly coupled?
- Code complexity: How many classes are identified?
- Test complexity:
  - How many unit tests, subsystem tests need to be done?
- Documentation complexity: Number of documents & pages
- Quality of testing:
  - Can defects be discovered during analysis, design, implementation? How is it determined that testing is complete?
  - What was the failure density? (Failures discovered per unit size)

Maturity Level 4 and 5 Questions

- Level 4 questions:
  - Has intra-project feedback been used?
  - Is inter-project feedback used? Does the project have a post-mortem phase?
  - How much code has been reused?
  - Was the configuration management scheme followed?
  - Were defect identification metrics used?
  - Module completion rate: How many modules were completed in time?
  - How many iterations were done per activity?
- Level 5 questions:
  - Did we use measures obtained during development to influence our design or implementation activities?

Key Process Areas for Level 2 (Repeatable Process)

- Requirements Management: Requirements are baselined in a project agreement and maintained during the project
- Project Planning and Tracking: A software project management plan is established at the beginning of the project and is tracked during the project.
- Subcontractor Management: The organization selects and effectively manages qualified software subcontractors.
- Quality Assurance Management: All deliverables and process activities are reviewed and audited against standards and guidelines adopted by the organization.
- Configuration Management: Controlled items are defined and maintained throughout the entire project.

Key Process Areas for Level 3 (Defined Process)

- Organization process:
  - Permanent team maintains software life cycle.
  - A standard software life cycle model is used for all projects.
- Training program: The organization identifies training needs and develops training programs.
- Integrated Software management: Each project can tailor their specific process from the standard process.
- Software product engineering: Software is built according to the software life cycle, methods and tools.
- Inter-group coordination: The project teams interact with other teams to address requirements and issues.
- Peer reviews: Deliverables are reviewed on a peer level to identify defects and areas where changes are needed.

Key Process Areas for Level 4 (Managed Process)

- Quantitative process management:
  - Productivity and quality metrics are defined and constantly measured across the project.
  - These data are not immediately used during the project, but are stored in a database to allow for comparison with other projects.
- Quality management:
  - The organization has defined a set of quality goals for software products. It monitors and adjusts the goals and products to deliver high-quality products to the user.

Key Process Areas for Level 5 (Optimized Process

- Defect prevention: Failures in past projects are analyzed, using data from the metrics database.
- Technology change management: Technology enablers and innovations are constantly investigated.
- Process change management: The software process is constantly changed to deal with problems identified by the software process metrics.

Steps to Take in Using Metrics

Metrics are useful only when implemented in a careful sequence of process-related activities.

1. Assess your current process maturity level
2. Determine what metrics to collect
3. Recommend metrics, tools and techniques
   - whenever possible implement automated support for metrics collection
4. Estimate project cost and schedule and monitor actual cost and schedule during development
5. Construct a project data base:
   - Design, develop and populate a project data base of metrics data.
   - Use this database for the analysis of past projects and for prediction of future projects.
6. Evaluate cost and schedule for accuracy after the project is complete.
7. Evaluate productivity and quality
   - Make an overall assessment of project productivity and product quality based on the metrics available.