소프트웨어 공학 원리
(SEP521)

Software Development Process

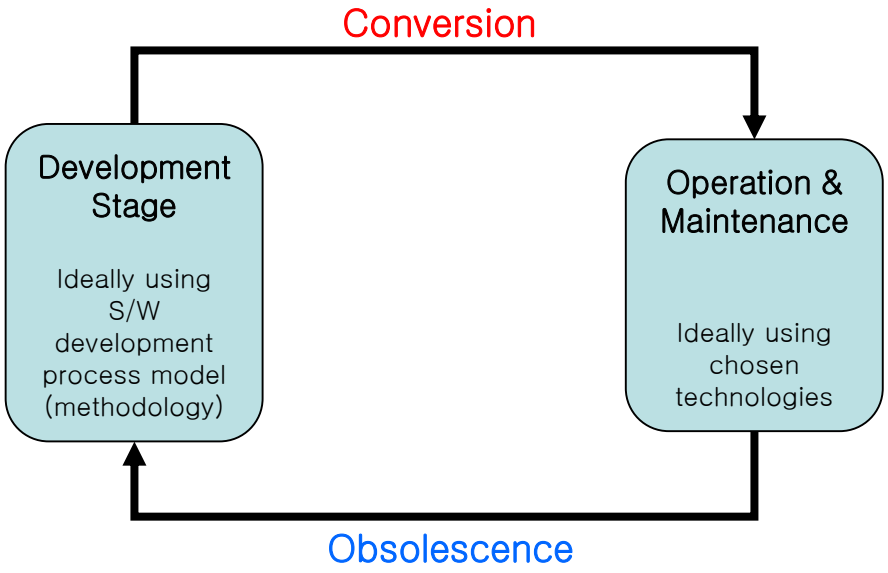
Jongmoon Baik

KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
KAIST
한국과학기술원
SINCE 1971

Software Development Processes
(Lifecycle Models)

What is a S/W Life Cycle?

- "The series of stages in form and functional activity through which an organism passes between successive recurrence of a specified primary stage" – Webster (1982)
- "Period of time that begins when a software product is conceived and ends when the product is retired from use" – Don Reifer (1997)



What is Software Process?

Software process – a set of activities, methods, best practices, deliverables, and automated tools that stakeholders use to develop and continuously improve software.

- Using a consistent process for software development:
  - Create efficiencies that allow management to shift resources between projects
  - Produces consistent documentation that reduces lifetime costs to maintain the systems
  - Promotes quality

What is S/W Process (Life Cycle) Model?

- Software Process Model
  - An abstract representation of a process
    - A description of a process from some particular perspective
  - Attempt to generalize the s/w development process into steps with associated activities and/or artifacts

- Proliferation of S/W Process Models
  - Provides flexibility for organizations to deal with the wide variety of software project situations, cultures, and environments
  - Weakens the defenses against some common sources of project failure

Process ≠ Lifecycle

- Software process is not the same as life cycle models.
  - process refers to the specific steps used in a specific organization to build systems
  - indicates the specific activities that must be undertaken and artifacts that must be produced
  - *process definitions* include more detail than provided lifecycle models

- Software processes are sometimes defined in the context of a lifecycle model.

Generic S/W Process Models

- Build-And Fix Model
- Waterfall Model
  - Process phases are distinct and separate
- Evolutionary Development
  - Process phases are interleaved
- Formal Systems Development
  - A mathematical system model is formally transformed to an implementation
- Component-Based Development
  - The system is assembled from existing components

Need to look at with respect

- Scope, time, resources, quality
- Stakeholders
- Environments
  - Business / Market
  - Cultures
- Moral, legal constraints

So, when looking at projects

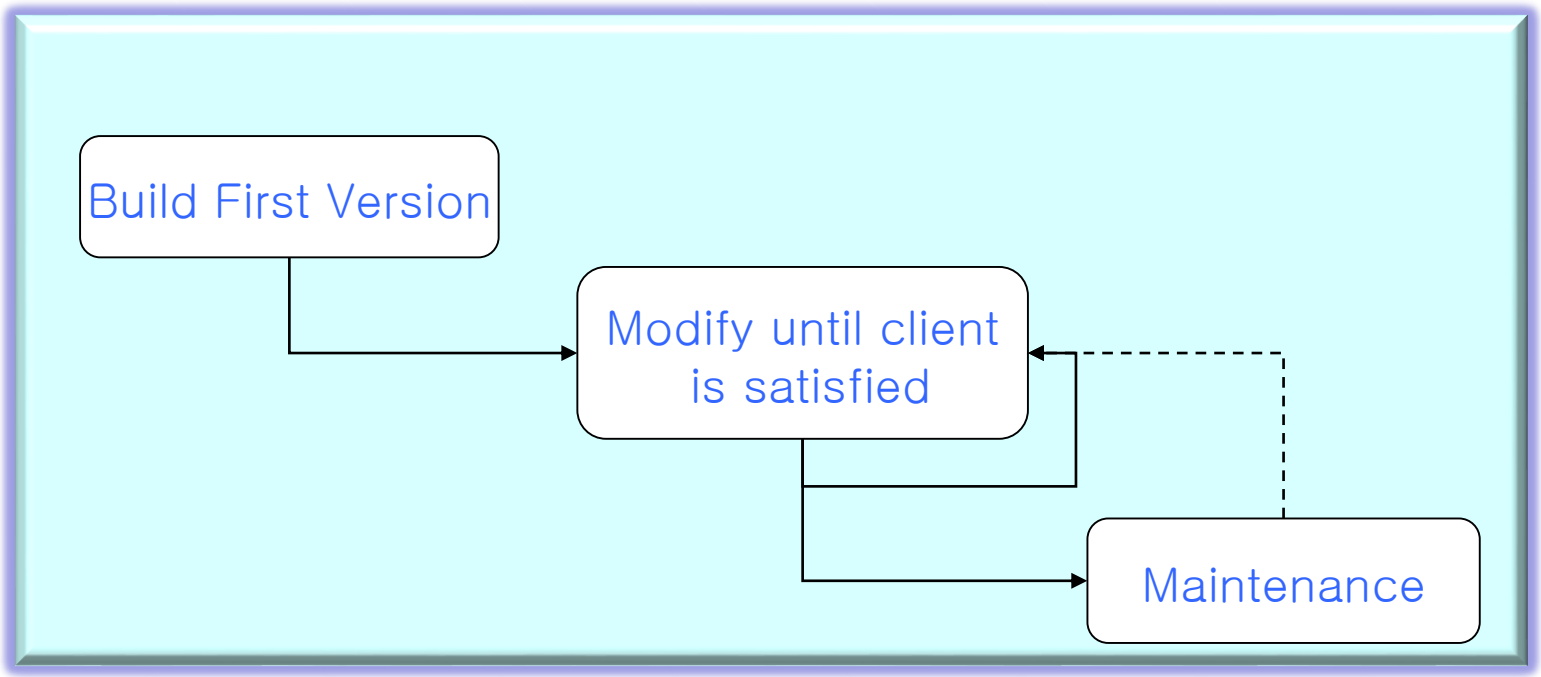
Need to ask:

"What SDLC would fit my project best?"

(Is this better than what type of project would fit each SDLC?)

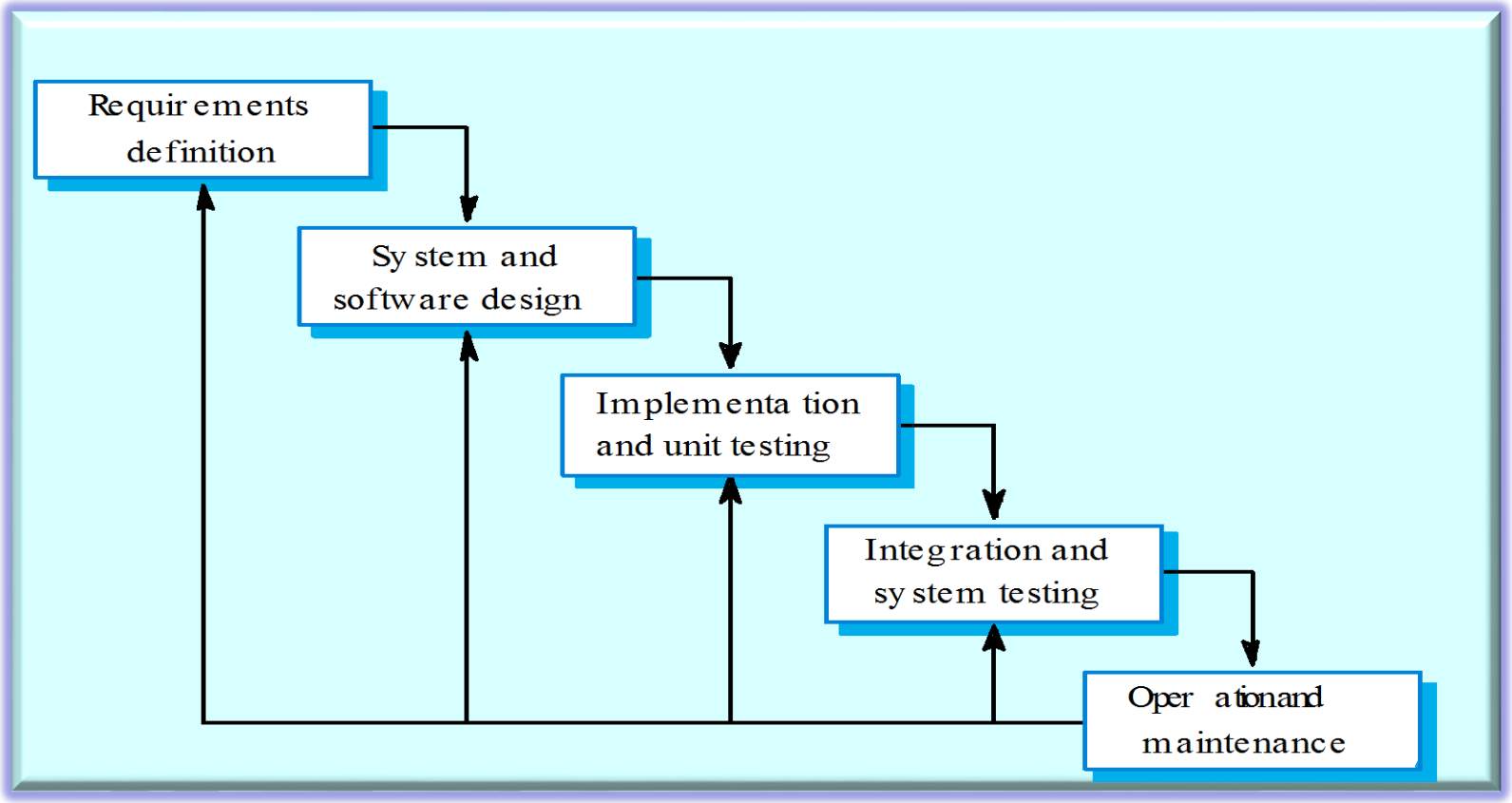
Build-And-Fix Model

- The worst (Ad Hoc) model for a software project
  - Become a mess, chuck it, start over
- No proper specifications and design steps
- Discouraged from using this model



Waterfall Model

- Formal "Sign-off" at the end of each phase
  - Documentation as product of each phase – Document-driven approach
- Referred to as the linear sequential model or the software life cycle



Waterfall Model:

- Clear progress state
  - Good for project Management
  - Engineers know their tasks
- Development is (relatively) slow and deliberate
  - Results in solid, well-constructed systems
  - All sub-goals within each phase must be met
  - Testing is inherent to every phase

Waterfall Model :

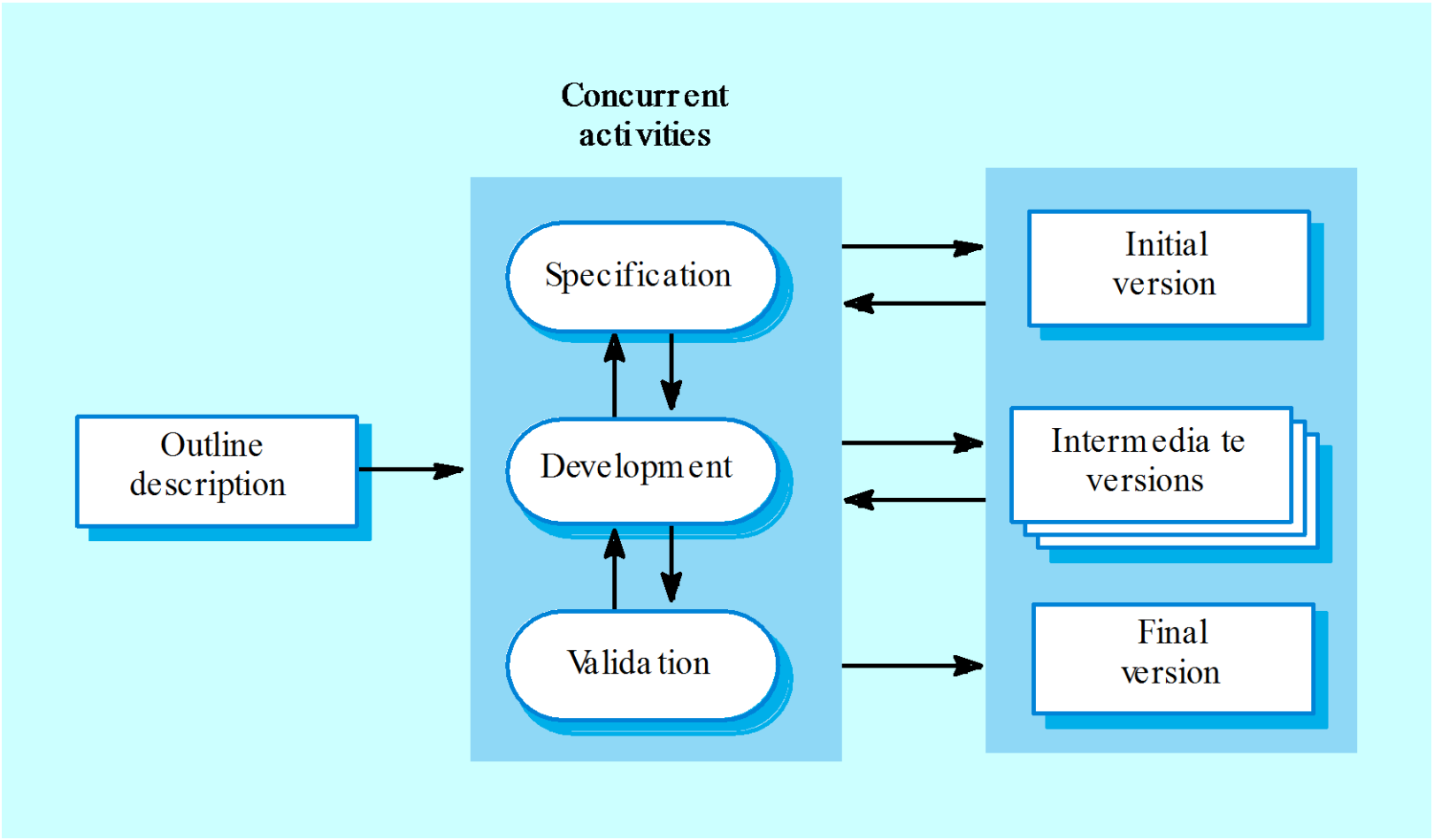
- Difficult (Expensive) to accommodate change after process is underway
  - One phase has to be complete before moving to the next phase
- Inflexible partitioning of the project into distinct stages
  - Difficult to respond to changing customer requirements
  - Few business systems have stable requirements

Applicability of Waterfall

- Therefore, Waterfall model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process

- Mostly used for large software system projects where a system is developed at several sites

Evolutionary Development -



Evolutionary Development -

- Basic Idea
  - Build prototype version of system, seek use comments, and keep refining until done
- Other side of formality spectrum from Waterfall Model

Two Flavors of Evolutionary Dev.

- Exploratory development:
  - Objective is to work with customers and to evolve a final system from an initial outline specification (prototype)
  - Should start with well-understood requirement and add new features as proposed by the customer

- Throw-prototyping
  - Objective is to understand the system requirements
  - Should start with poorly understood requirements to clarify what is really needed

Evolutionary Development: Advantages

- Faster system development than with waterfall model
- Useful when requirements are less-well understood
- Customer inputs throughout development yields system closer to their needs
- Steady visible signs of progress

Evolutionary Dev.: Drawbacks

- Lack of process visibility
  - Not known how long it will take to create an acceptable product
  - Not known how many iteration will be necessary
- Systems are often poorly structured
  - Continuous reflection of customer needs
- Special skills (e.g. in languages for rapid prototyping) may be required.

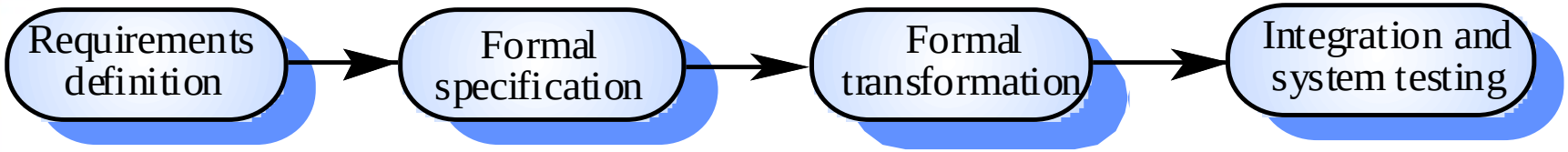
Applicability of Evolutionary Dev.

- For small or medium-size interactive systems
- For parts of large systems (e.g. the user interface)
- For short-lifetime systems

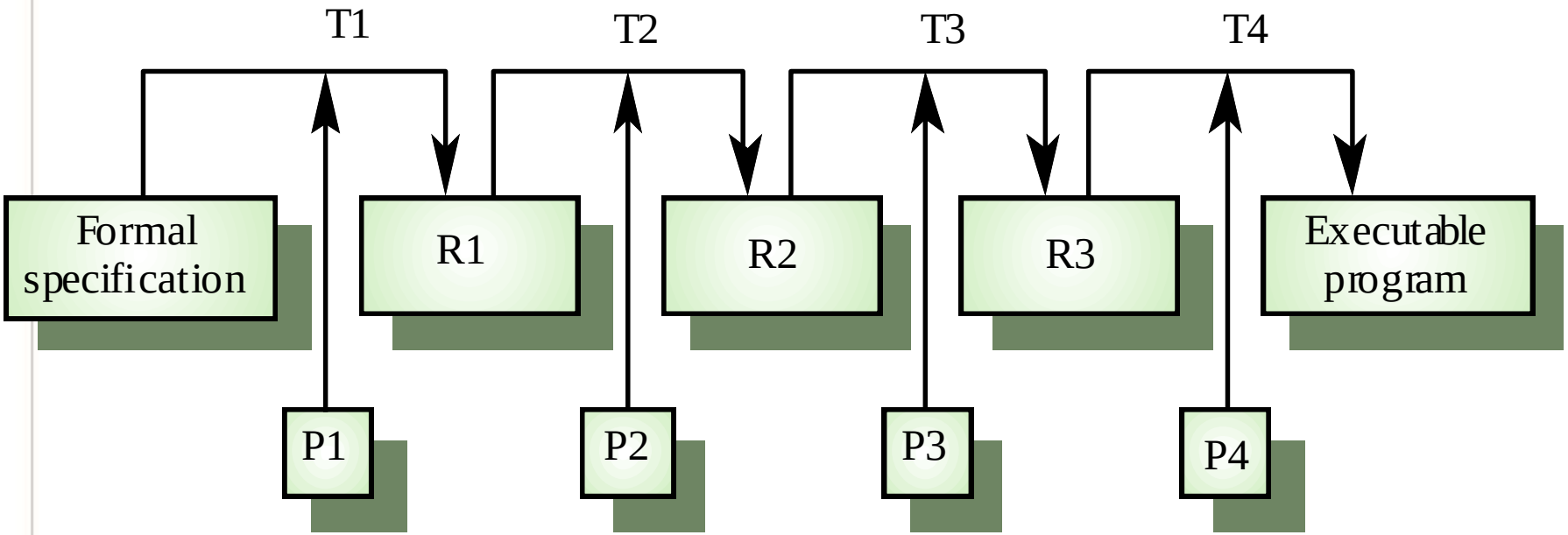
Formal Systems

- Based on the transformation of a mathematical specification through different representations to an executable program
- Transformations are ‘correctness-preserving’ so it is straightforward to show that the program conforms to its specification
- Embodied in the ‘*Cleanroom*’ approach to software development
- Each transformation should be sufficiently close to avoid excessive verification efforts and reduce the possibility of transformation errors

Formal Systems Development - II



Formal transformations



Proofs of transformation correctness

Formal Systems Dev.: Advantages

- Transformations are small, making verification tractable

- Resulting implementations are proven to meet specifications, so testing (of components) is unnecessary
  - Although, overall system characteristics (e.g. performance, reliability) still need verification

Formal Systems Dev.: Drawbacks

- Need for specialised skills and training
  - Expertise for the mathematical notations used for formal specifications
- Development time high (usually not worth the time/cost)
  - No significant quality or cost advantages over other approaches
- Difficult to formally specify some aspects of the system, e.g. user interfaces

Applicability of Formal Systems Dev.

- Critical systems
  - Systems that require strict safety, reliability, and security requirements
- Critical components within large systems

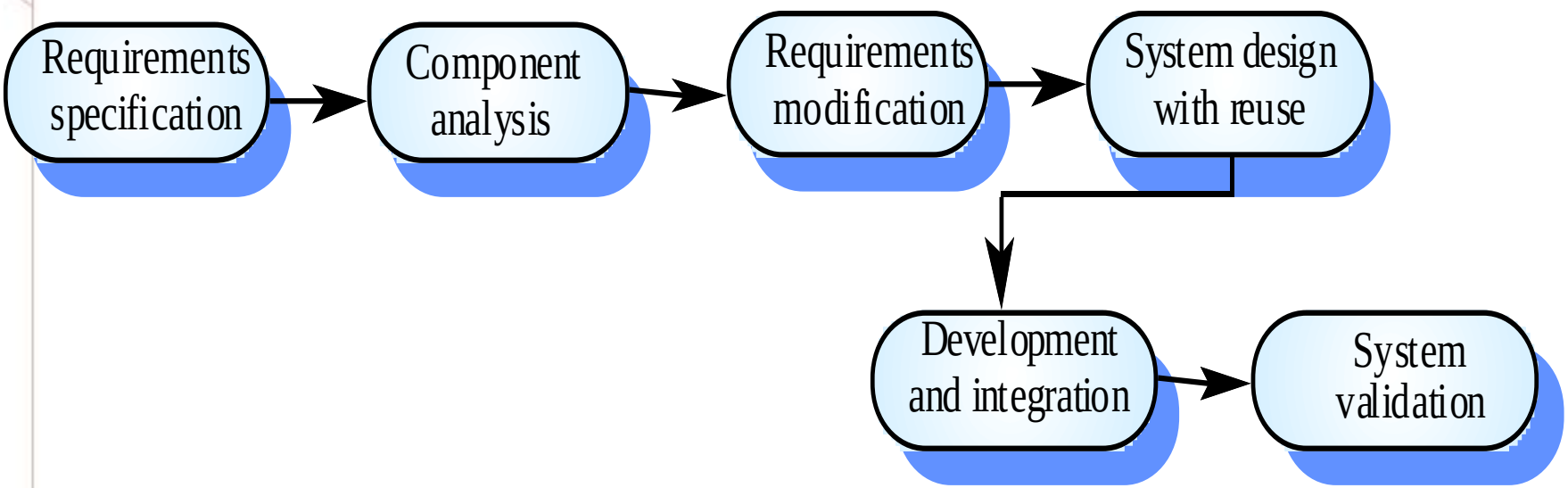
Component-Based

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems

- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration

- This approach is becoming more important but still limited experience with it

Component-Based Development - II



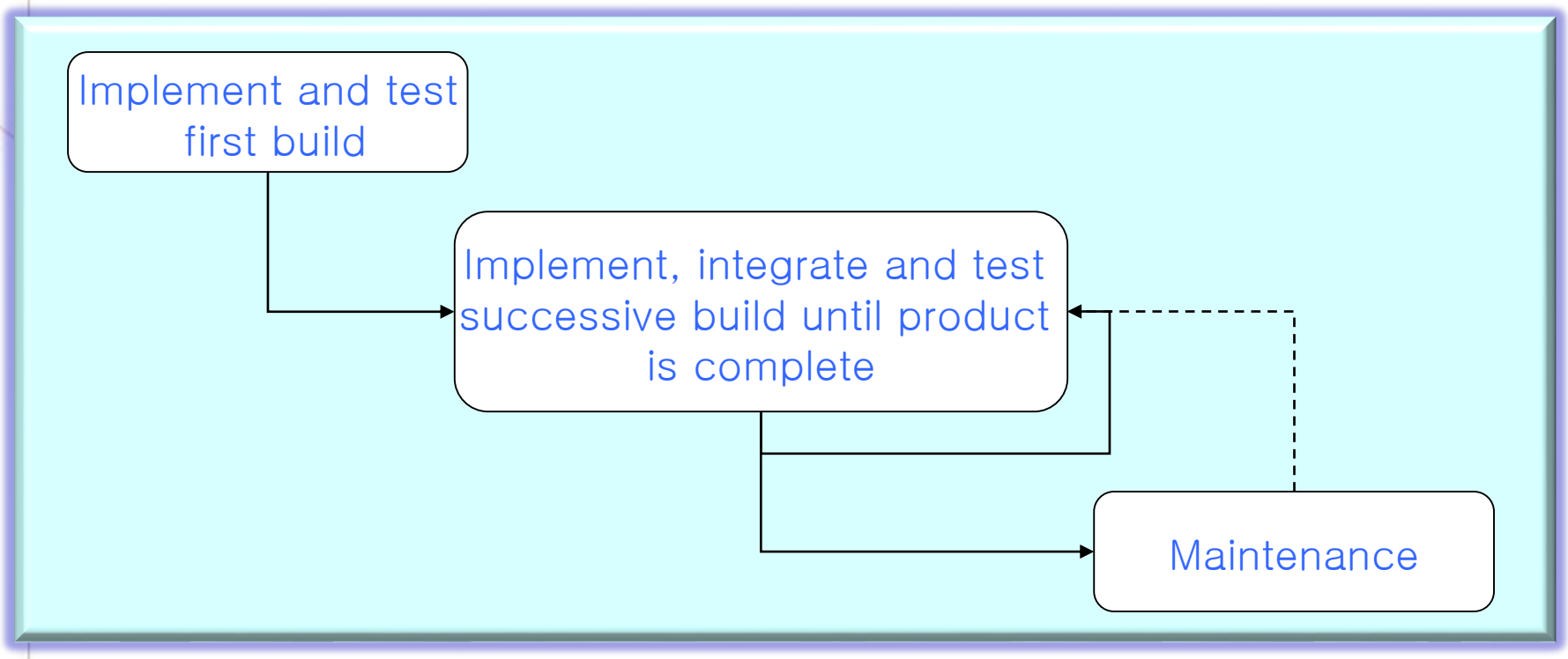
Process iteration

- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
- Iteration can be applied to any of the generic process models.
- Two (related) approaches
  - Incremental Model;
  - Spiral Model

Incremental Model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments (Builds) with each increment delivering part of the required functionality.
  - Each increments provides more functionality than the previous increment
- User requirements are prioritized and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development



- Series of incremental builds until the product is finished
- Value assignment to each build not yet implemented
- Cost estimation of developing each build
- Value-to-Cost ratio is the criterion used for next build selection

Incremental Model:

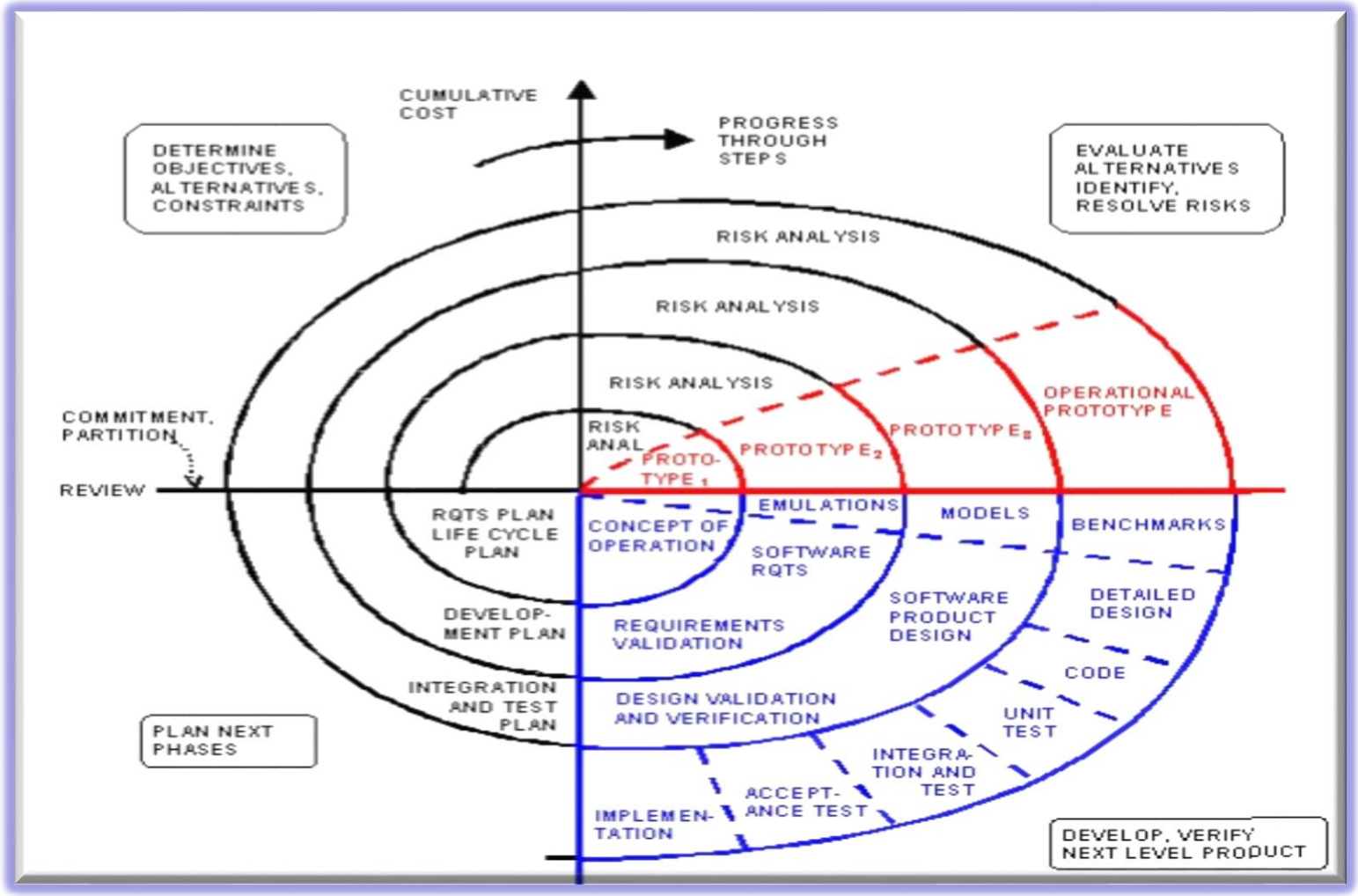
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Spiral Model - I

- Represented as a spiral rather than as a sequence of activities with backtracking.
  - Combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model
- Provides the potential for rapid development of incremental versions of software
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

[Boehm 1988]: “A Spiral Model of Software Development and Enhancement”

Spiral Model - II



Spiral Model Sectors - I

- Determine objectives, alternatives, & constraints
  - Specific objectives for the phase (performance, functionality, ability to accommodate changes etc.)
  - Alternative means of implementing this portion of the product (design A, design B, reuse, buy, etc.)
  - Constraints imposed on the application of the alternatives (cost, schedule, interface, etc.)
- Evaluate Alternatives, Identify, resolve risks
  - Evaluate alternatives relative to the objectives and constraints
  - Identify areas of uncertainty that significant sources of project risk
  - Formulate a cost effective strategy for resolving sources of risk
    - Prototyping, simulation, benchmarking, reference checking, administering user questionnaires, analytic modeling, or combinations of these and other resolution techniques

Spiral Model Sectors - II

- Develop, Verify next-level product
  - A development model for the system is chosen which can be any of the generic models.
    - Evolutionary model, waterfall model, incremental development, combinations, etc.

- Plan Next Phases
  - The project is reviewed and the next phase of the spiral is planned
    - The review covers all products developed during the previous cycle, plans for the next cycle, resource required
    - Ensure that all concerned parties are mutually committed to the approach for the next phase

Six Spiral Model Essentials

1. Concurrent determination of artifacts in each cycle
2. Each cycle addresses objectives, constraints, alternatives, risks, artifact elaboration, stakeholders' commitment
3. Risk-driven activity level of effort
4. Risk-driven artifact degree of detail
5. Managing stakeholder commitments via anchor-point milestones
6. Emphasis on system and life-cycle issues
   - vs. software and development issues

Spiral Model : Advantages

- The spiral model is a realistic approach to the development of large-scale software products because the software evolves as the process progresses. In addition, the developer and the client better understand and react to risks at each evolutionary level.

- The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.

- It maintains a systematic stepwise approach, like the classic life cycle model, but incorporates it into an iterative framework that more reflect the real world.

- If employed correctly, this model should reduce risks before they become problematic, as consideration of technical risks are considered at all stages.

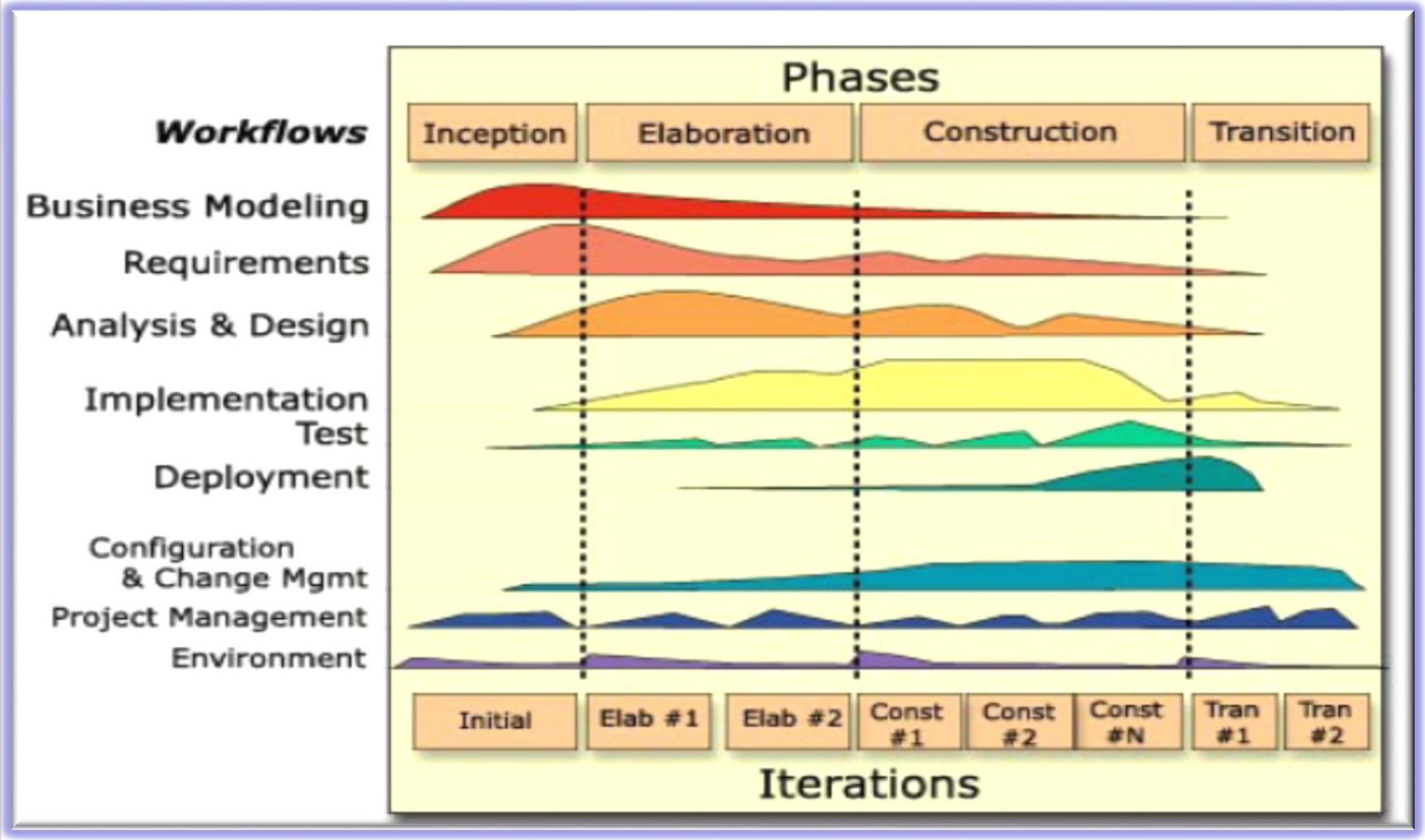
IBM-UP Model - I

IBM-UP (Unified Process)

- A modern process model derived from the work on the UML and associated process.

- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time
  - A static perspective that shows process activities
  - A proactive perspective that suggests good practice

IBM-UP Model - II



Best practices of UP

- Develop software iteratively
- Manage requirements
- Use component based architecture
- Visually model software
- Verify software quality
- Control changes to software

Limitations of UP

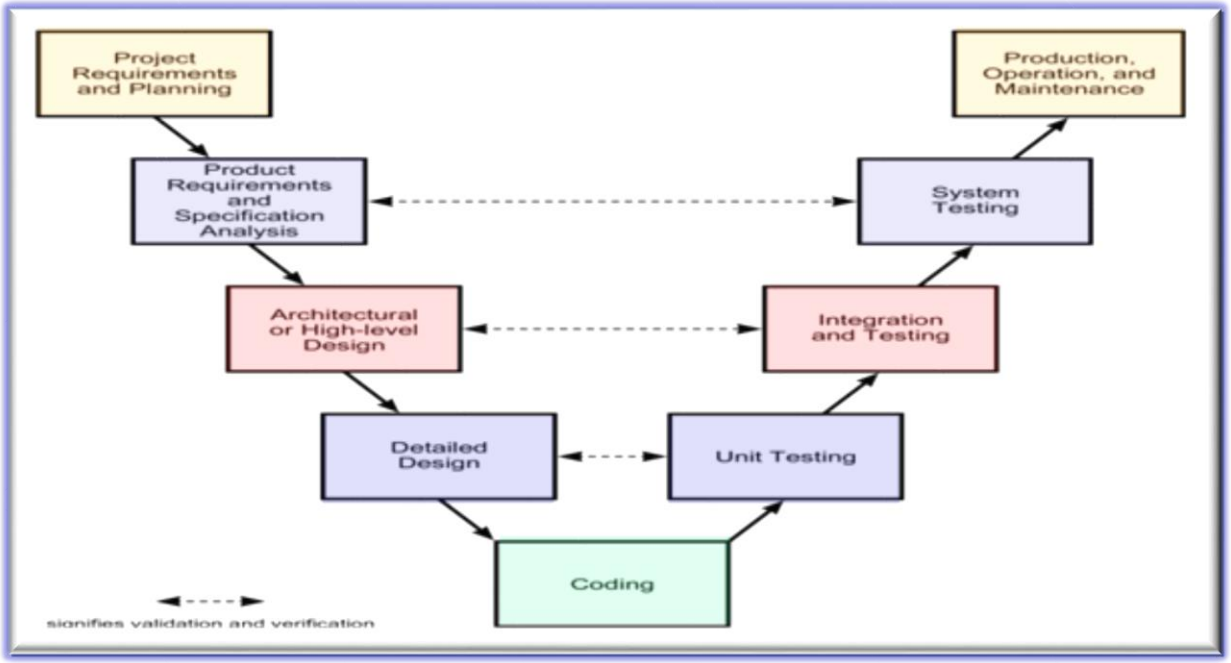
- High Adopting UP is often thought to be expensive.
- Does not cover any non-software aspects of development
  - e.g., system engineering. product-line engineering, safety engineering
- Considered too practical, as the practicality has been based on current status of the IBM tool suite

V Model - I

- Often used in system engineering environments to represent the system development lifecycle.
  - summarizes the main steps taken to build *systems not specifically software*
  - describes appropriate deliverables corresponding with each step in the model.

V Model - II

- The left side of the V: the specification stream where the system specifications are defined.
- The right side of the V: the testing stream where the systems is being tested against the specifications defined on the left side.
- The bottom of the V where the tails meet, represents the development stream.



Current State of the Art

- Iterative, cyclic development
- Agile Processes? – covered later
- Software is grown rather than birthed whole
- Short cycles, Small teams
- Component development (Reuse, COT, Product Line, etc)
- More integration vice new development (SoS)?

When looking at a new project,

DO NOT make your project fit a SDLC!!!

- INSTEAD, find the right SDLC and tailor it to your project (if it can be).
- Your organization may drive this
  - But any lifecycle, process should be seen as a tool to assist development, not an end in and of it self.

Process Model Decision

Objectives, Constraints			Alternatives		Model	Example
Growth Envelope	Understanding of Requirements	Robustness	Available Technology	Architecture Understanding		
Limited			COTS		Buy COTS	Simple Inventory Control
Limited			4GL, Transform		Transform or Evolutionary Development	Small Business - DP Application
Limited	Low	Low		Low	Evolutionary Prototype	Advanced Pattern Recognition
Limited to Large	High	High		High	Waterfall	Rebuild of old system
	Low	High			Risk Reduction followed by Waterfall	Complex Situation Assessment
		High		Low		High-performance Avionics
Limited to Medium	Low	Low-Medium		High	Evolutionary Development	Data Exploitation
Limited to Large			Large Reusable Components	Medium to High	Capabilities-to-Requirements	Electronic Publishing
Very Large		High			Risk Reduction &Waterfall	Air Traffic Control
Medium to Large	Low	Medium	Partial COTS	Low to Medium	Spiral	Software Support Environment

45

Process Assessment and Improvement

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting and learning.
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**— provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01]
- **SPICE—The SPICE (ISO/IEC15504)** standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process. [ISO08]
- **ISO 9001:2000 for Software—**a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies. [Ant06]

Personal Software Process (PSP)

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **High-level design.** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- **High-level design review.** Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

Team Software Process (TSP)

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
  - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 30.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

Standard Processes

- IEEE and ISO Software Engineering Standards
  - http://standards.ieee.org/catalog/olis/se.html
  - 1074-1997IEEE Standard for Developing Software Life Cycle Processes
  - 1517-1999 (R2004)IEEE Standard for Information Technology - Software Life Cycle Processes - Reuse Processes
  - 12207.0-1996IEEE/EIA Standard: Industry Implementation of International Standard ISO/IEC 12207:1995 Standard for Information Technology--Software Life Cycle Processes.
  - 15288-2004IEEE Std 15288-2004 (Adoption of ISO/IEC 15288:2002, IDT), Systems Engineering---System Life Cycle Processes
  - ……

Q & A