CSCI 5828: Foundations of Software Engineering

Lecture 5 and 6: Modeling the Process and Life Cycle

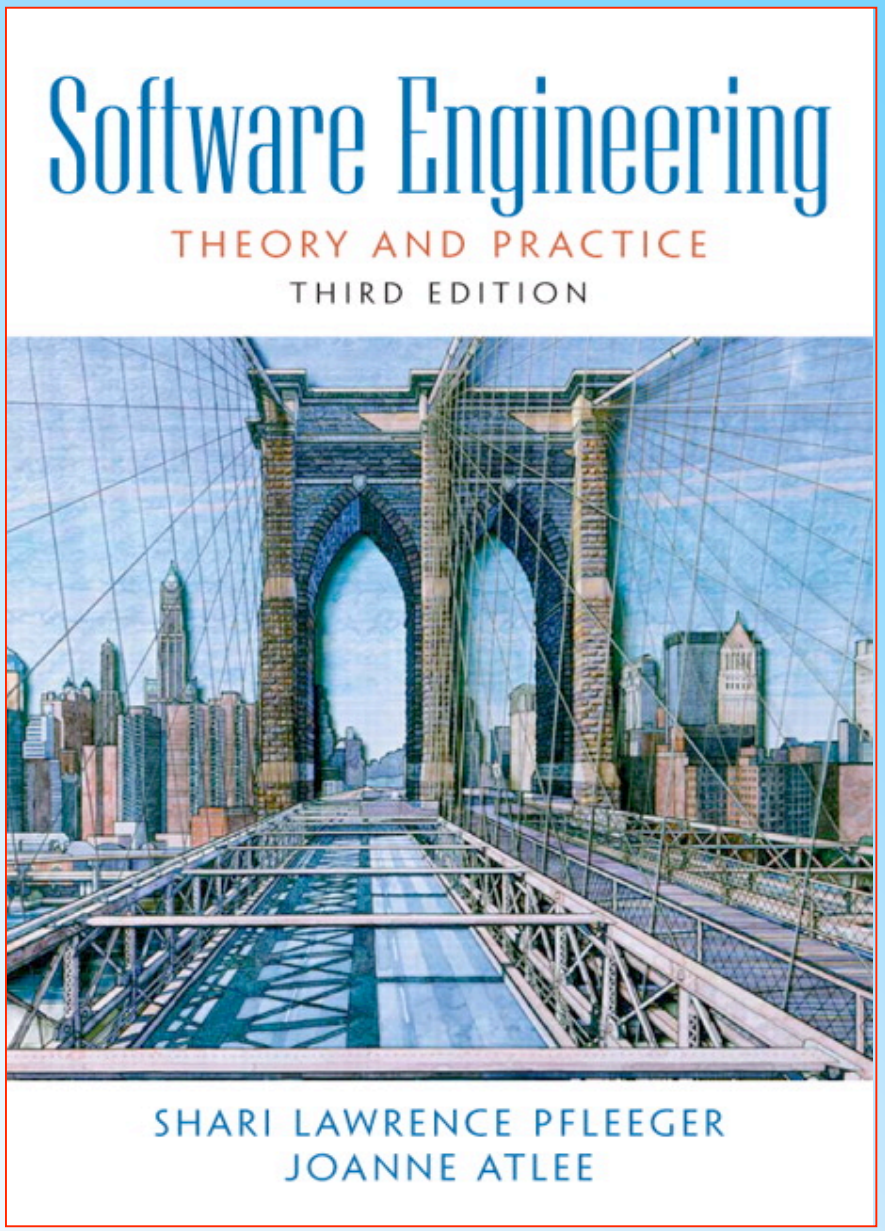
Slides created by Pfleeger and Atlee for the SE textbook

Some modifications to the original slides have been made by Ken Anderson for clarity of presentation

01/29/2008 — 01/31/2008

Chapter 2

Modeling the Process and Life Cycle



ISBN 0-13-146913-4
Prentice-Hall, 2006

Contents

Chapter 2 Objectives

- Discuss the definition of “process” or “life cycle”
- Discuss standard terminology:
  - Software dev. products, processes, and resources
- Present several software life cycles
- Cover tools and techniques for process modeling

2.1 The Meaning of Process

- **process**:
  - a series of steps involving activities, constrains, and resources that produce an intended ouput of some kind
- A process involves a set of tools and techniques

2.1 The Meaning of Process
Process Characteristics

- Prescribes all major process activities
- Uses resources, subject to set of constraints
  - (such as a schedule or a budget)
  - Constraints may apply to an activity, resource or product
- Produces intermediate and final products
- May be composed of subprocesses with hierarchy or links
- Each process activity has entry and exit criteria
- Activities are organized in sequence, so timing is clear

- Each process has guiding principles, including the goals of each activity

2.1 The Meaning of Process
The Importance of Processes

- Impose consistency and structure on a set of activities
  - especially across projects in a single organization
  - or two or more projects performed by the same team
- Aids engineers in understanding, controlling, and improving the activities within the process
- Allows engineers to capture/measure our experiences and use them to improve future performance

2.2 Software Process Models
Reasons for Modeling a Process

- To form a common understanding across different stakeholders
- To find inconsistencies, redundancies, omissions
- To find and evaluate appropriate activities for reaching process goals
- To tailor a general process for a particular situation in which it will be used

2.2 Software Process Models
Software Life Cycle

- When a process involves building software, the process may be referred to as software life cycle
  - Requirements analysis and definition
  - System (architecture) design
  - Program (detailed/procedural) design
  - Writing programs (coding/implementation)
  - Testing: unit, integration, system
  - System delivery (deployment)

  -

  - Maintenance

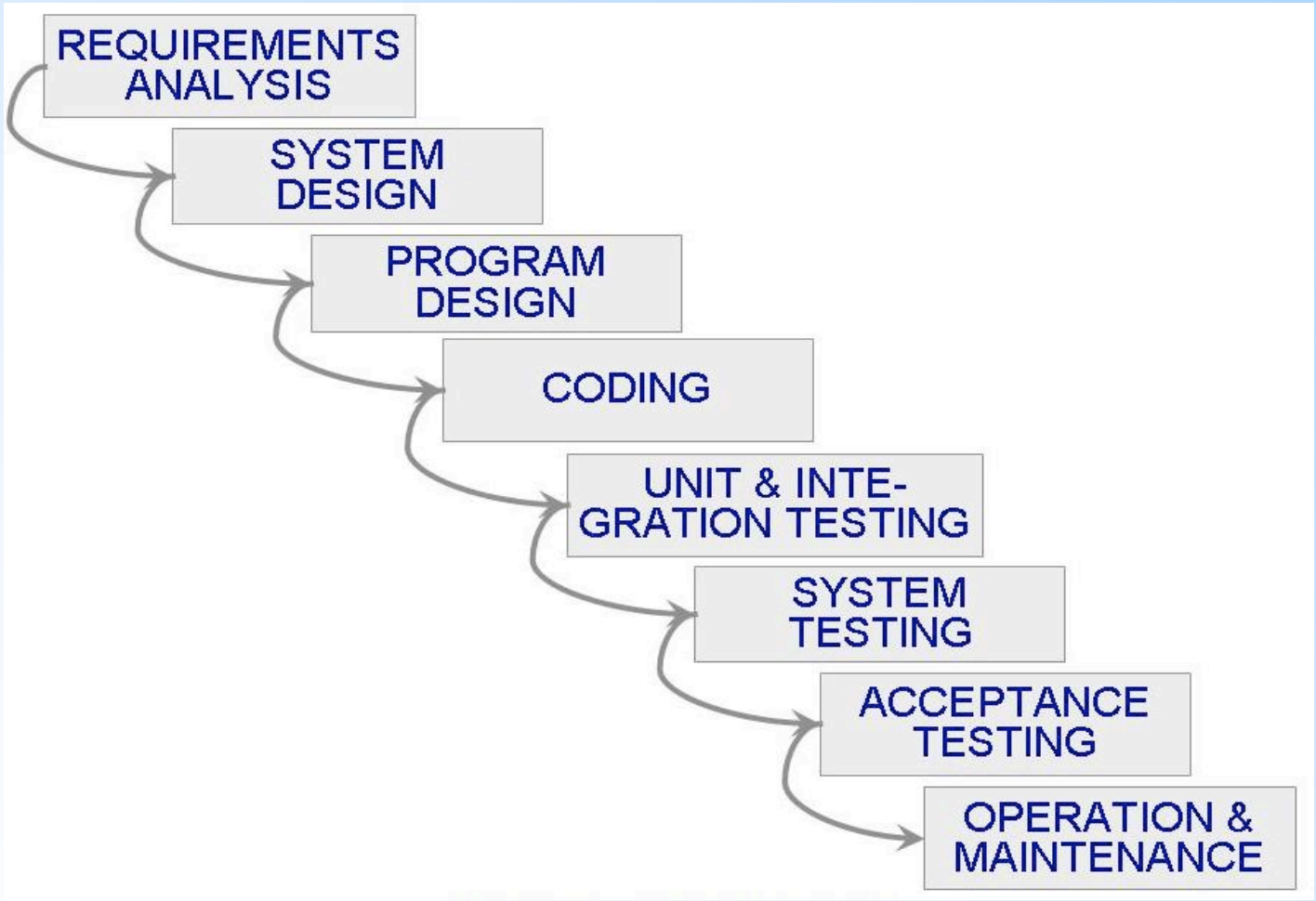
2.2 Software Process Models
Software Development Process Models

- Waterfall model
- V model
- Prototyping model
- Operational specification
- Transformational model
- Phased development: increments and iterations
- Spiral model
- Agile methods

2.2 Software Process Models
Waterfall Model

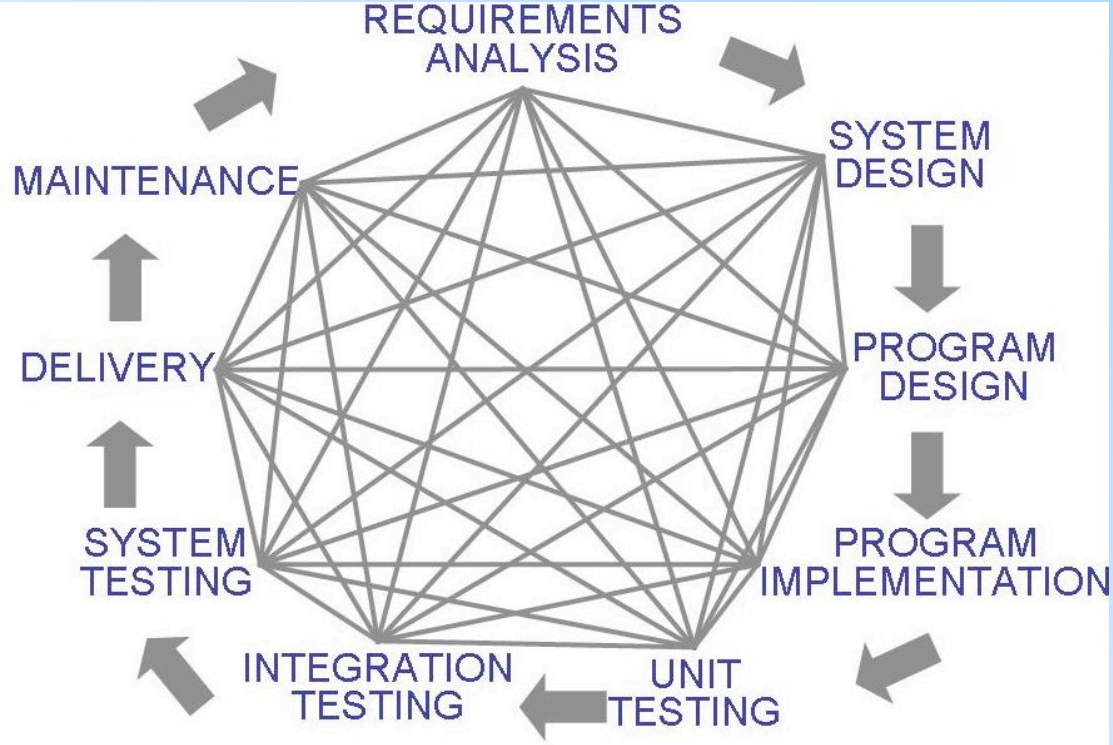
- One of the first process development models proposed (circa 1970)
- Works for well understood problems with minimal or no changes in the requirements
- Simple and easy to explain to customers
- It presents
  - a very high-level view of the development process
  - a sequence of process activities
- Each major phase is marked by milestones and deliverables (artifacts)

2.2 Software Process Models
Waterfall Model (continued)



2.2 Software Process Models
Waterfall Model (continued)

- There is no iteration in the original waterfall model
- Most software projects apply a great many iterations



2.2 Software Process Models
Sidebar 2.1 Drawbacks of The Waterfall Model

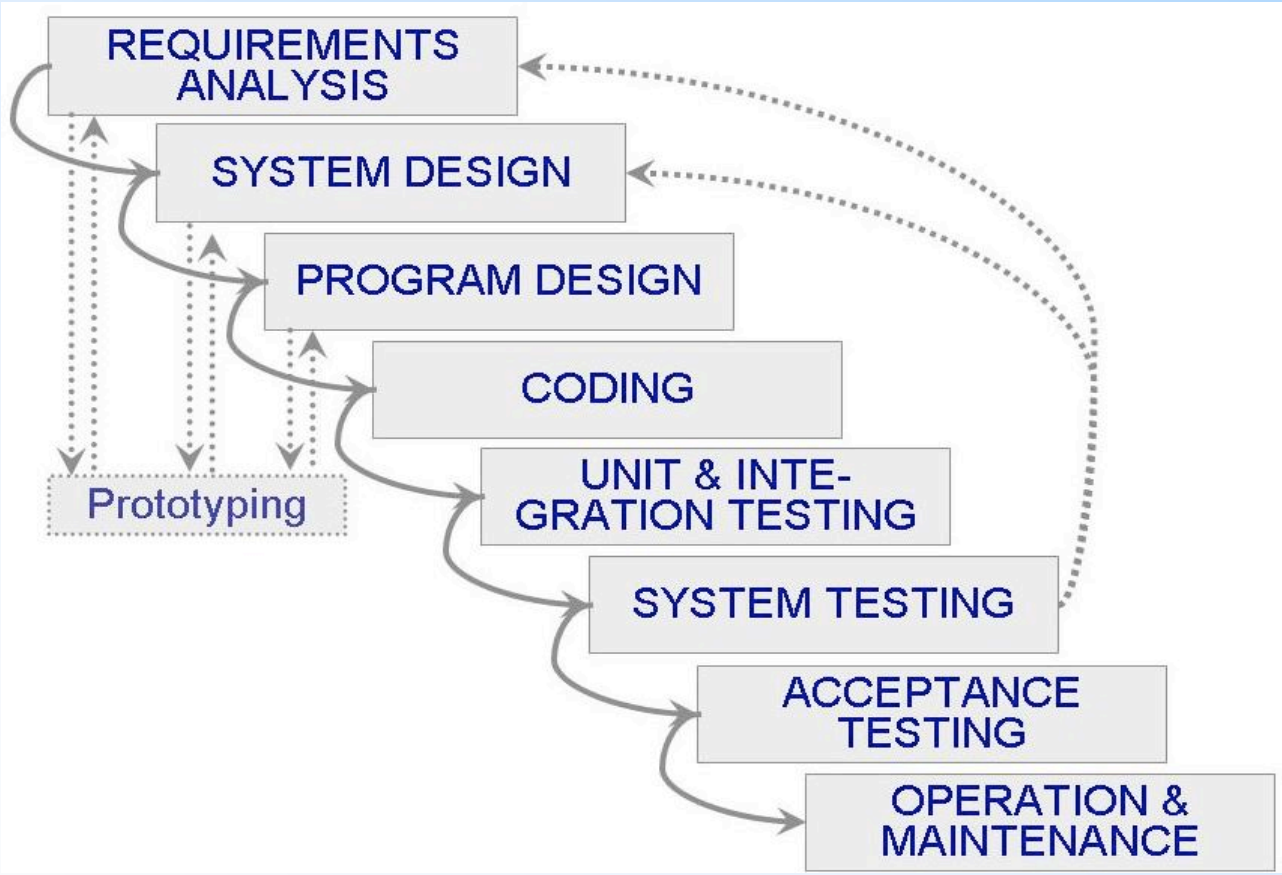
- Provides no guidance on how to handle changes to products and activities during development (assumes requirements can be frozen)
- Views software development as a manufacturing process rather than as a creative process
- There is no iterative activities that lead to creating a final product
- From customer perspective, there can be a long wait before a final product is delivered

2.2 Software Process Models
Waterfall Model with Prototype

- A prototype is a partially developed product
- Prototyping helps
  - developers assess alternative design strategies (design prototype)
  - users understand what the system will be like (user interface prototype)
- Protopyping is useful for verification and validation
  - validation: have all requirements been implemented?
  - verification: have all requirements been implemented correctly and with high quality?

2.2 Software Process Models
Waterfall Model with Prototype (continued)

- Waterfall model with prototyping

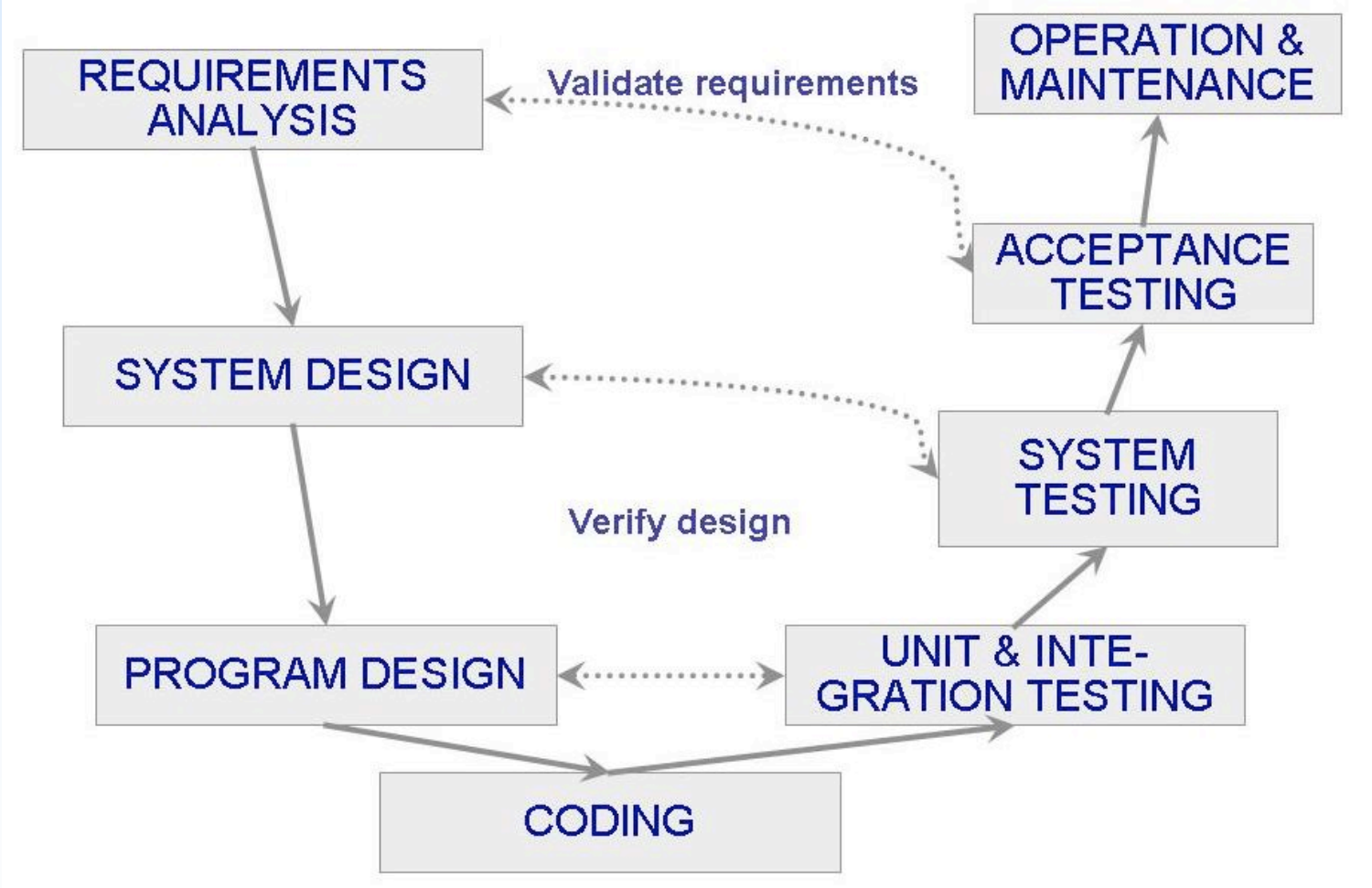


2.2 Software Process Models
V Model

- A (slight) variation of the waterfall model
- Uses unit testing to verify program design
- Uses integration testing to verify architectural design
- Uses acceptance testing to validate the requirements

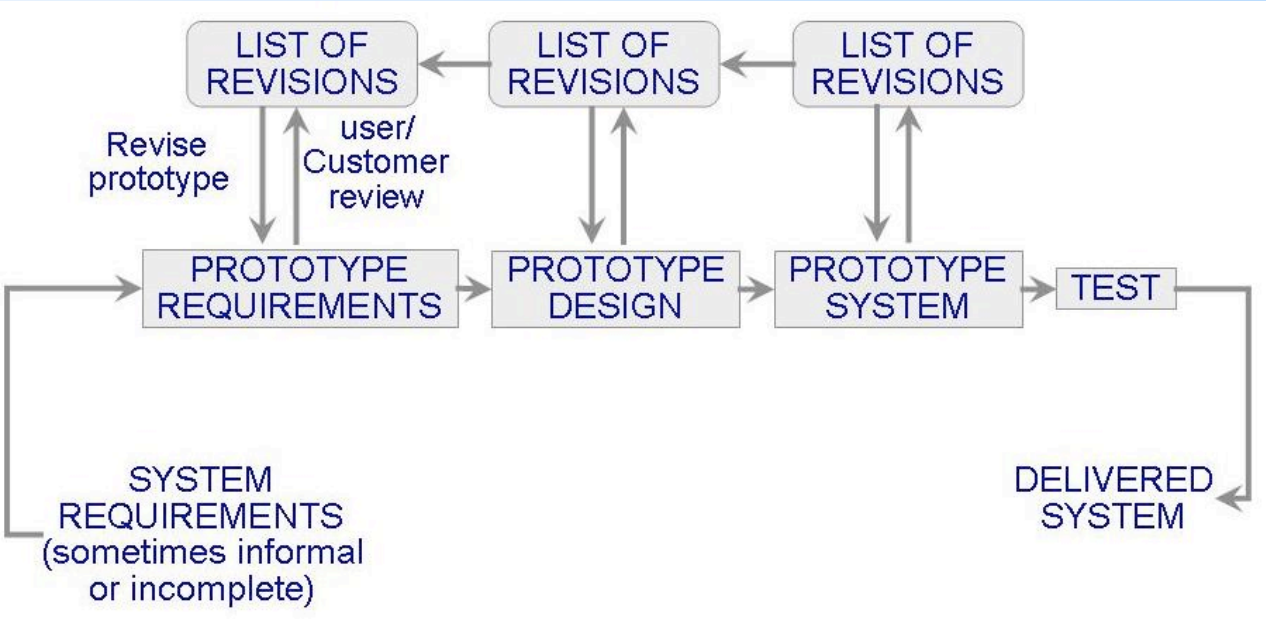
- If problems are found during verification and validation, the “left side” of the V can be re-executed before testing on the “right side” is re-enacted

2.2 Software Process Models
V Model (continued)



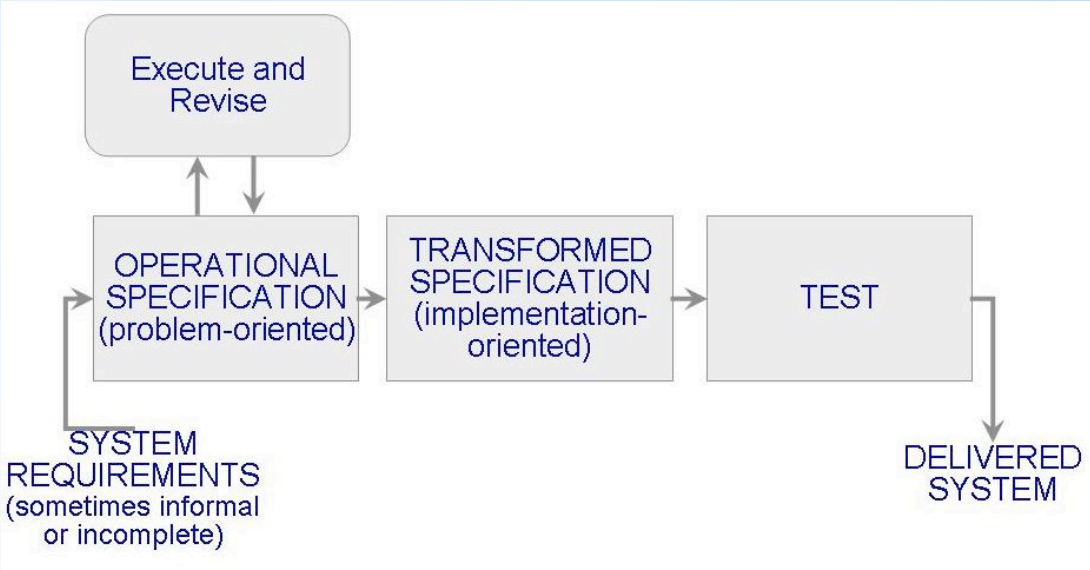
2.2 Software Process Models
Prototyping Model

- Allows repeated investigation of the requirements or design
- Reduces risk and uncertainty in the development



2.2 Software Process Models
Operational Specificiation Model

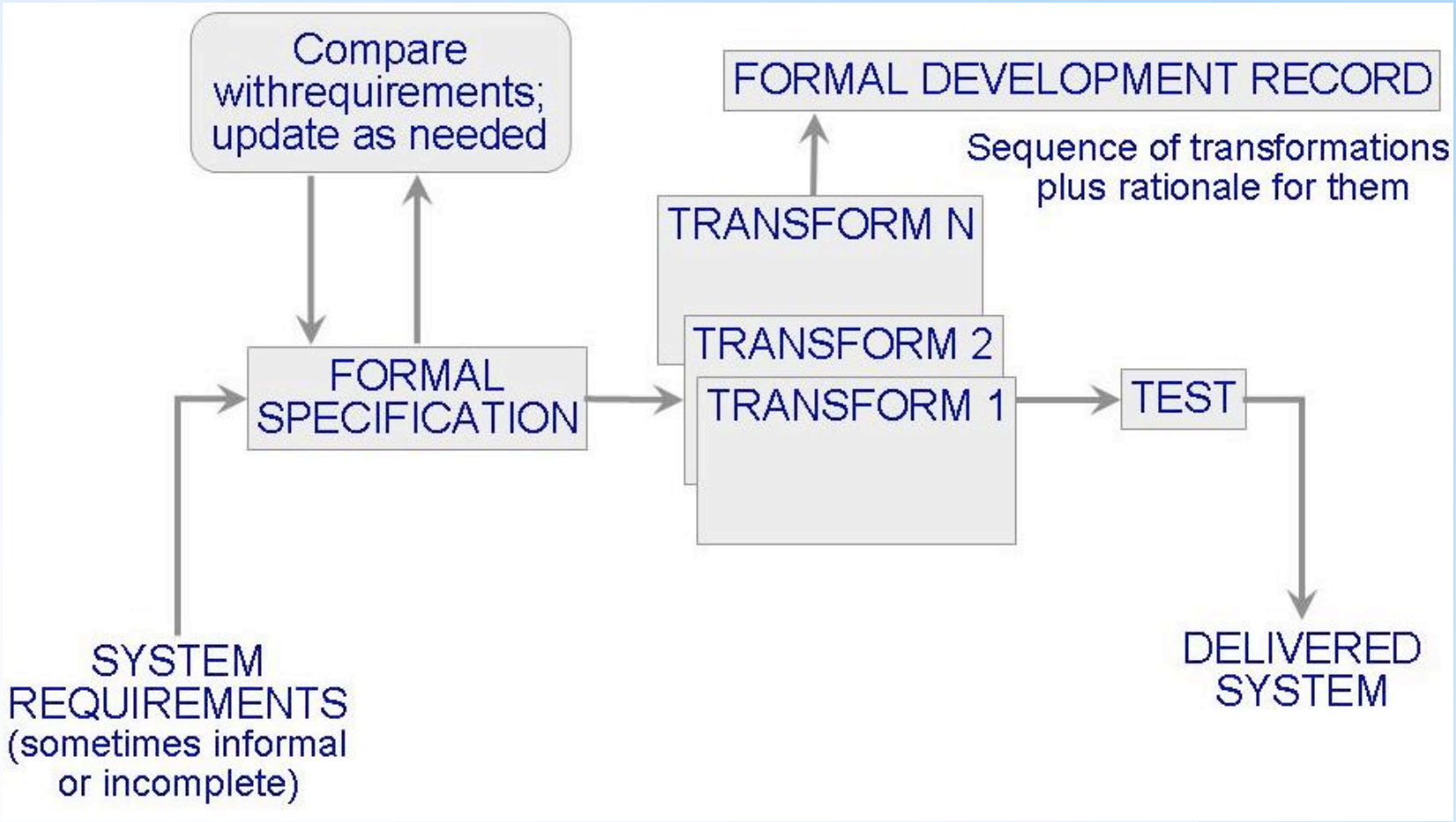
- Requirements are executed (examined) and their implication evaluated early in the development process
- Functionality and the design are allowed to be merged



2.2 Software Process Models
Transformational Model

- Fewer major development steps
- Applies a series of transformations to change a specification into a deliverable system
  - Change data representation
  - Select algorithms
  - Optimize
  - Compile
- Relies on formalism by requiring an initial, formal specification (to allow transformations)

2.2 Software Process Models
Transformational Model (continued)

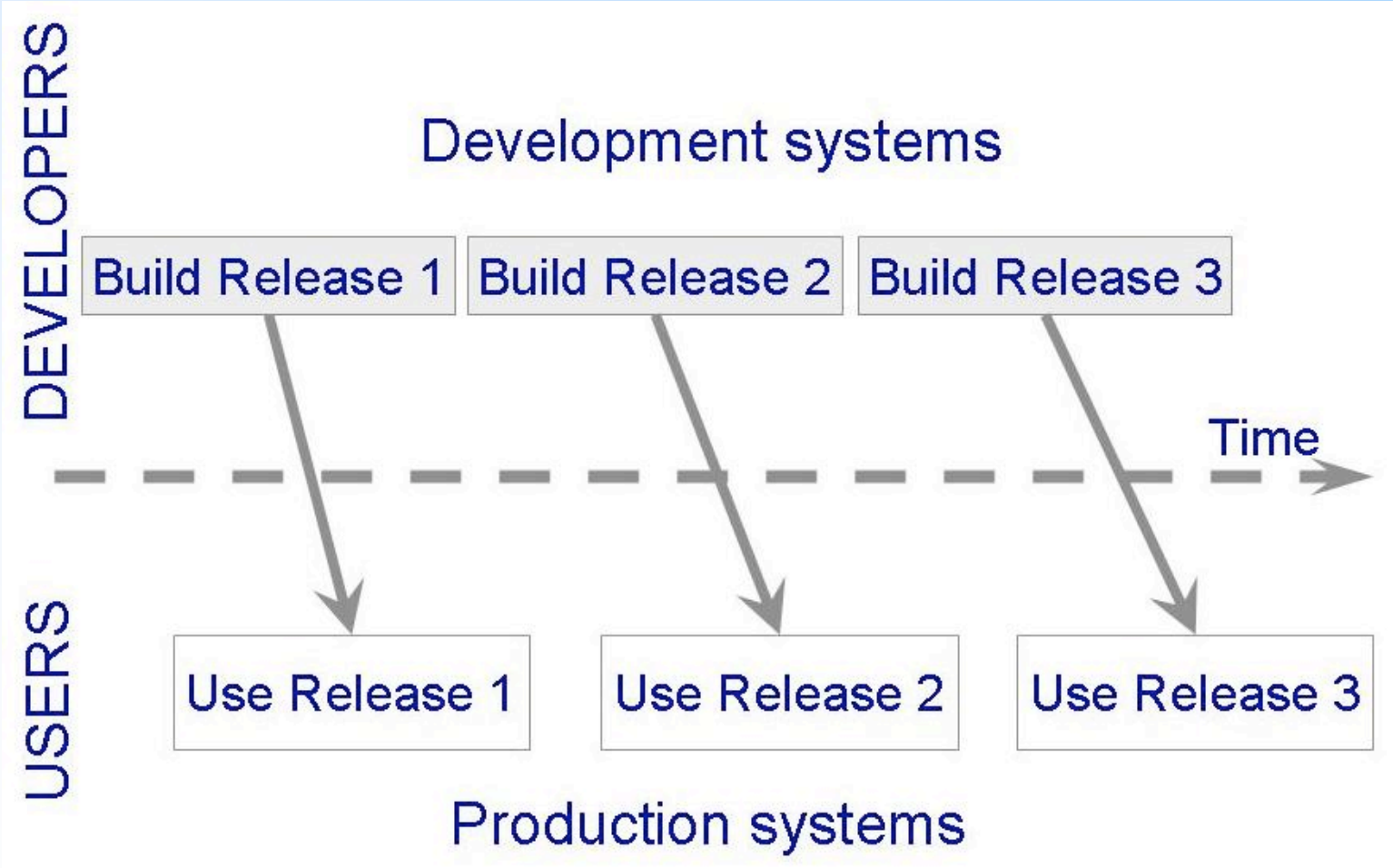


2.2 Software Process Models
Phased Development: Increments and Iterations

- Shorter cycle time
- System delivered in pieces
  - enables customers to have some functionality while the rest is being developed
- Allows two systems functioning in parallel
  - the production system (release *n*): currently being used
  - the development system (release *n+1*): the next version

2.2 Software Process Models

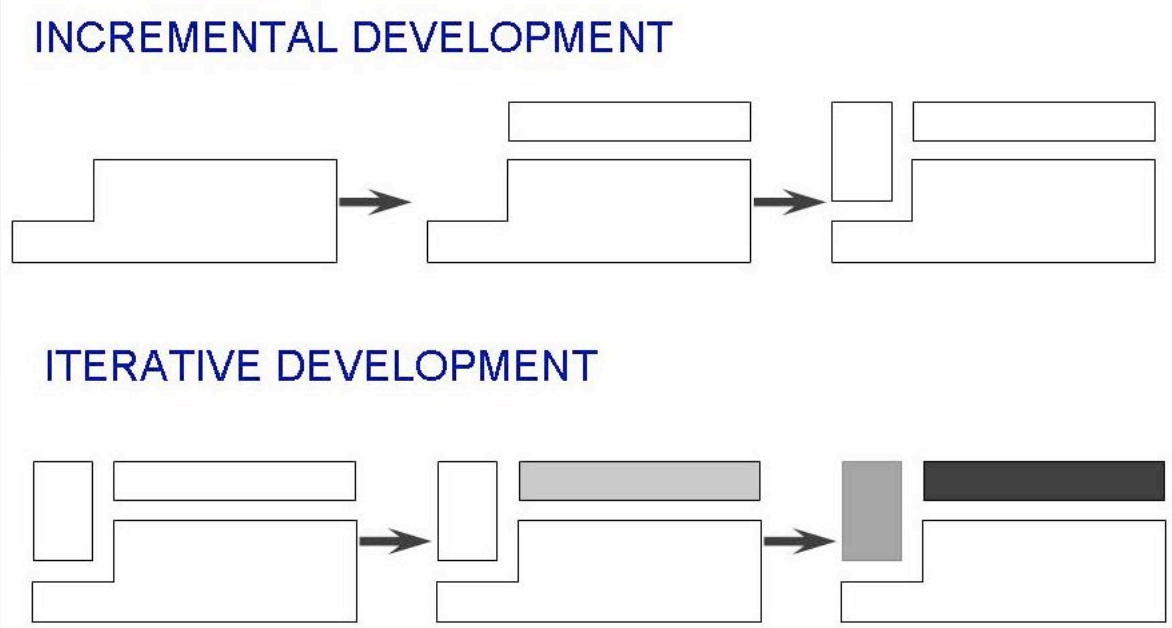
Phased Development: Increments and Iterations (continued)



2.2 Software Process Models

Phased Development: Increments and Iterations (continued)

- **Incremental development:** starts with small functional subsystem and adds functionality with each new release
- **Iterative development**: starts with full (skeleton) system, then changes functionality of each subsystem with each new release



2.2 Software Process Models

Phased Development: Increments and Iterations (continued)

- Phased development is desirable for several reasons
  - Training can begin early, even though some functions are missing
  - Frequent releases allow developers to fix unanticipated problems globally and quickly
  - The development team can focus on different areas of expertise with different releases

2.2 Software Process Models
Spiral Model

- Suggested by Boehm (1988)
- Combines development activities with risk management to minimize and control risks
- The model is presented as a spiral in which each iteration is represented by a circuit around four major activities
  - Plan
  - Determine goals, alternatives, and constraints
  - Evaluate alternatives and risks
  - Develop and test

2.2 Software Process Models
Spiral Model (continued)

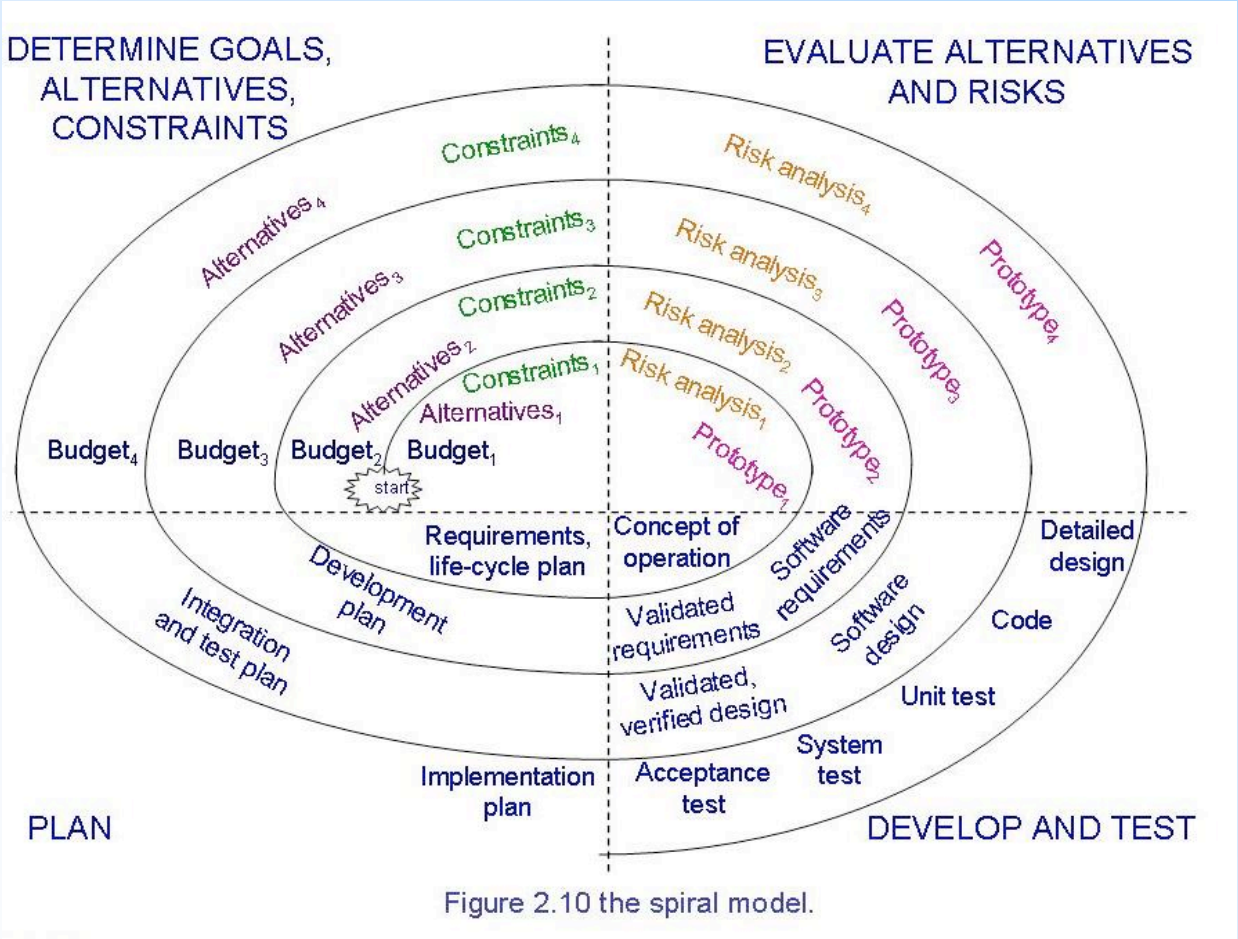


Figure 2.10 the spiral model.

2.2 Software Process Models
Agile Methods

- Emphasis on flexibility in producing software quickly
- Agile manifesto
  - Value individuals and interactions over process and tools
  - Prefer to invest time in producing working software rather than in producing comprehensive documentation
  - Focus on customer collaboration rather than contract negotiation
  - Concentrate on responding to change rather than on creating a plan and then following it

2.2 Software Process Models
Agile Methods: Examples of Agile Process

- Extreme programming (XP)
- Crystal: a collection of approaches based on the notion that every project needs a unique set of policies and conventions
- Scrum: 30-day iterations; multiple self-organizing teams; daily “scrum” coordination
- Adaptive software development (ASD)

2.2 Software Process Models
Agile Methods: Extreme Programming

- Emphasis on four characteristics of agility
  - *Communication*: continual interchange between customers and developers
  - *Simplicity*: select the simplest design or implementation
  - *Courage*: commitment to delivering functionality early and often
  - *Feedback*: loops built into the various activitites during the development process

2.2 Software Process Models
Agile Methods: Twelve Facets of XP

- The planning game *(customer defines value)*
- Small releases
- Metaphor *(common vision, common names)*
- Simple design
- Writing tests first
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration *(small increments)*
- Sustainable pace *(40 hours/week)*
- On-site customer
- Coding standards

2.2 Software Process Models
Sidebar 2.2 When is Extreme Too Extreme?

- Extreme programming's practices are interdependent
  - A vulnerability if one of them is modified
  - Can hinder adoption
- Requirements expressed as a set of test cases must be passed by the software
  - System may pass the tests but is not what the customer is paying for
- Refactoring issue
  - Difficult to rework a system without degrading its architecture

2.2 Software Process Models
Sidebar 2.3 Collections of Process Models

- Software development is a problem-solving activity
  - But process models rarely include (model) problem solving activities
- Curtis, Krasner, and Iscoe (1988) performed a field study (of 17 large software projects) to determine which problem-solving factors should be captured in a process model
- The results identified five behavioural models that need to supplement a traditional process model description
  - business milieu, company, project, team, and individual
  - information from these models can be used to predict impact on the activities of the traditional process model
- A process model, therefore, should not only describe a series of tasks, but also should detail factors that contribute to a project's inherent uncertainty and risk
  - based on the dynamics of individuals, teams, and the organization

2.3 Process Modeling: Tools and Techniques

- The notation used to document a process model depends on what we want to capture in the model
  - text can be used to capture a process as functions
  - graphics can be used to capture a process as “boxes and arrows”
  - A combination can link diagrams to supplemental info
- The two major types of models
  - *Static models*: depict the process, showing how input is transformed into output
  - *Dynamic models*: enact the process, enabling analysis

2.3 Tools and Techniques for Process Modeling
Static Modeling: Lai Notation

- Lai Notation: Not a lot of detail in the textbook
  - So, I went hunting for the paper that it was based on
  - I found it at the following URL (!)
    - http://stinet.dtic.mil/cgi-bin/GetTRDoc?AD=ADA262314&Location=U2&doc=GetTRDoc.pdf
  - The original paper is 105 pages long and goes into lots of detail that I can't cover here
  - But, this should give you an idea for the amount of work it takes to create a process modeling notation

2.3 Tools and Techniques for Process Modeling
Static Modeling: Lai Notation

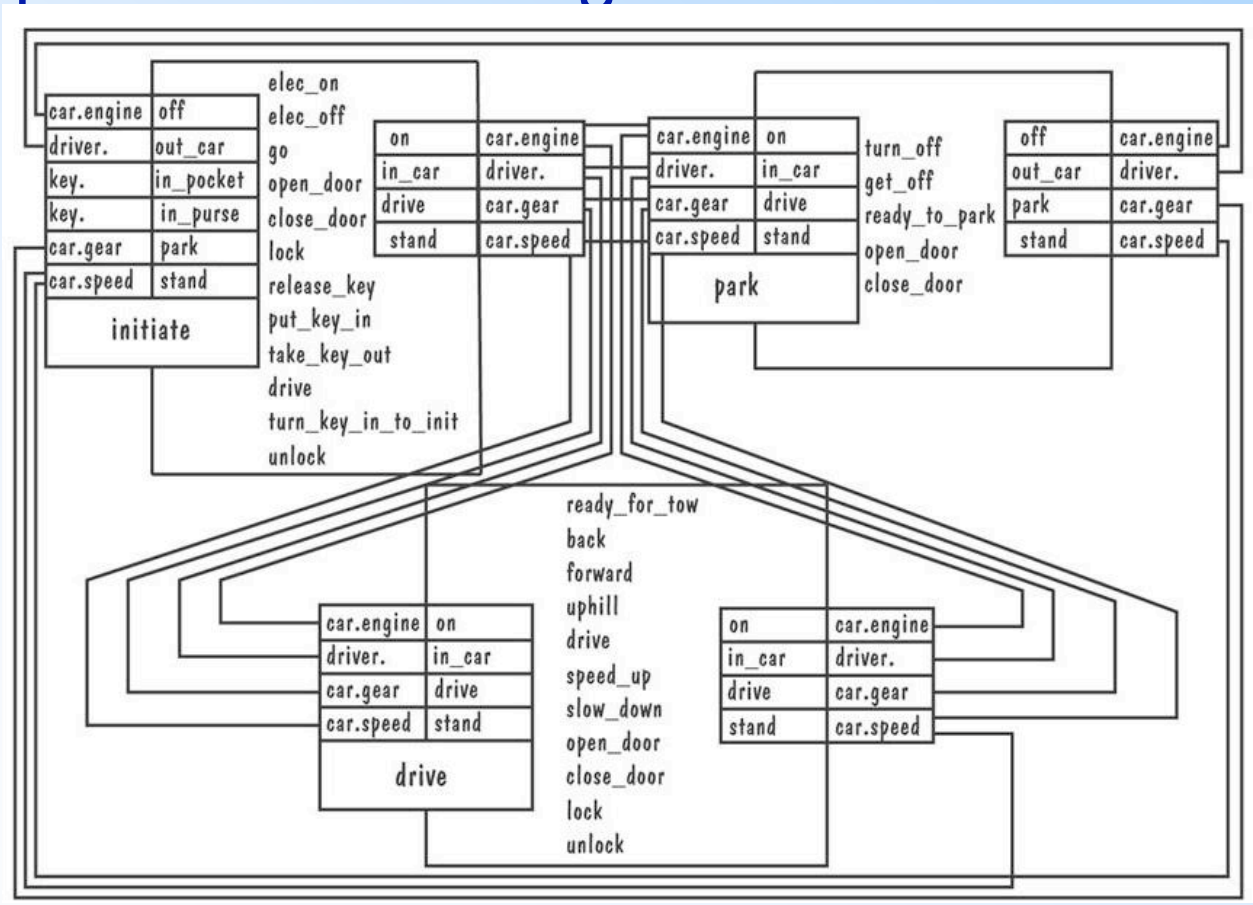
- Seven types of process elements
  - Activity: a task in the process, augmented with meta information
  - Sequence: the order of activities
  - Process model: a particular view of the activities
  - Resource: A necessary item, tool, or person
  - Control: An external influence over process enactment
  - Policy: A high-level constraint
  - Organization: A mapping of logical roles to physical groups
- Several supporting templates, e.g., an Artifact Definition Template

2.3 Tools and Techniques for Process Modeling
Static Modeling: Lai Notation

Name	*Car*	
Synopsis	*This is the artifact that represents a class of cars.*	
Complexity type	*Composite*	
Data type	*(car_c, user -defined)*	
Artifact -state list		
parked	*((state_of(car.engine) = off) (state_of(car.gear) = park) (state_of(car.speed) = stand))*	*Car is not moving, and engine is not running.*
initiated	*((state_of(car.engine) = on) (state_of(car.key_hole) = has -key) (state_of(car -driver(car.)) = in -car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))*	*Car is not moving, but the engine is run ning*
moving	*((state_of(car.engine) = on) (state_of(car.keyhole) = has -key) (state_of(car -driver(car.)) = driving) ((state_of(car.gear) = drive) or (state_of(car.gear) = reverse)) ((state_of(car.speed) = stand) or (state_of(car.speed) = slow) or (state_of(car.speed) = medium) or (state_of(car.speed) = high))*	*Car is moving forward or backward.*
Sub -artifact list		
	doors	*The four doors of a car.*
	engine	*The engine of a car.*
	keyhole	*The ignition keyhole of a car.*
	gear	*The gear of a car.*
	speed	*The speed of a car.*
Relations list		
car-key	*This is the relation between a car and a key.*	
car-driver	*This is the relation between a car and a driver.*	

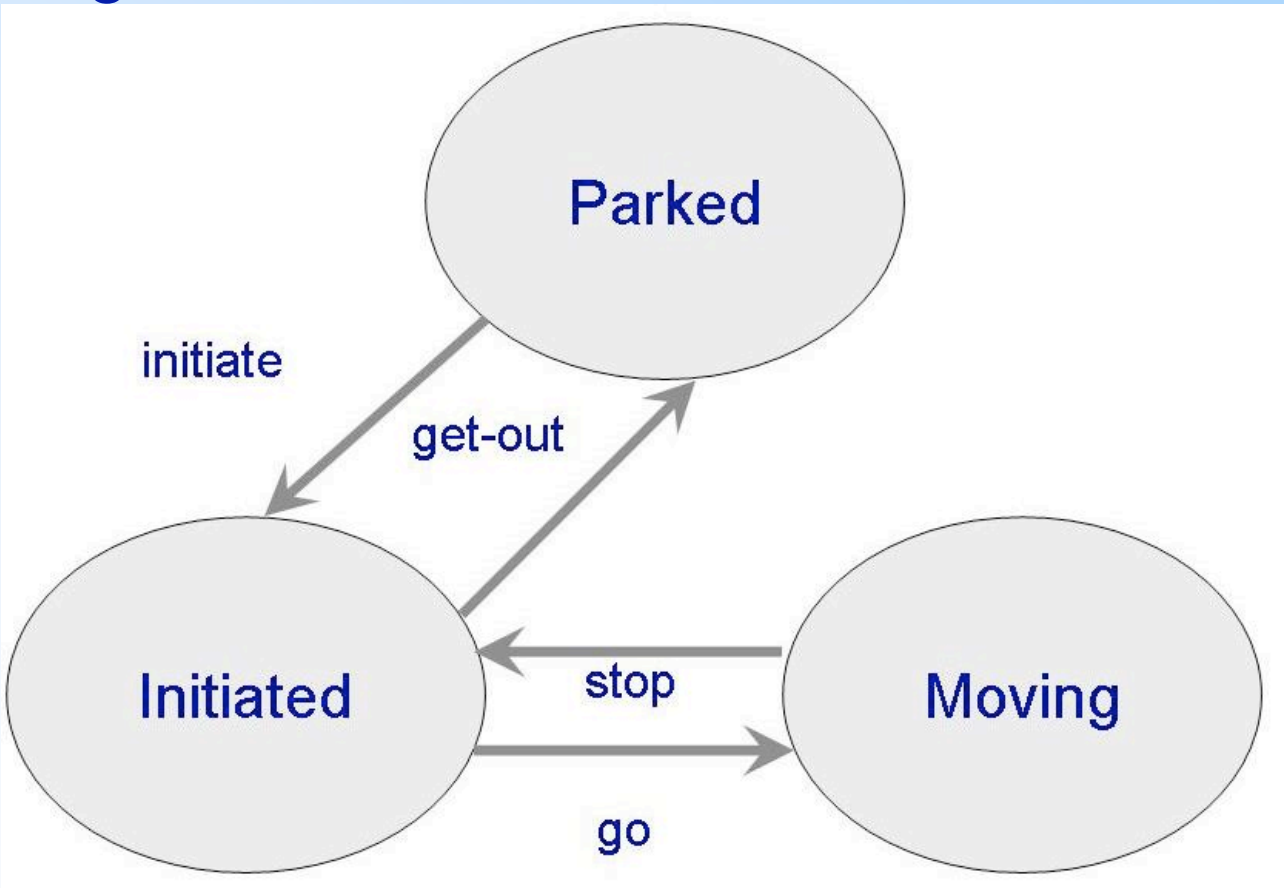
2.3 Tools and Techniques for Process Modeling
Static Modeling: Lai Notation (continued)

- The process of starting a car



2.3 Tools and Techniques for Process Modeling
Static Modeling: Lai Notation (continued)

- Transition diagram illustrates the transition for a car



2.3 Tools and Techniques for Process Modeling Dynamic Modeling

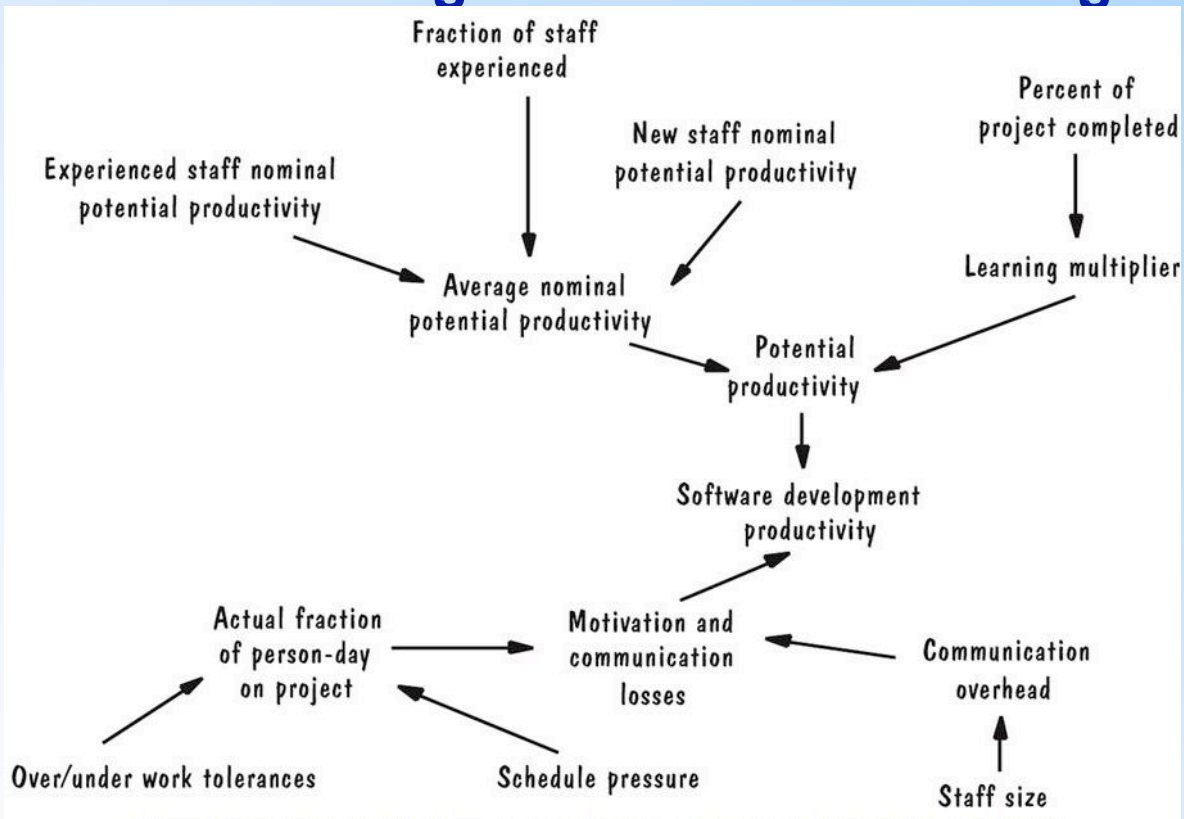
- Enables enaction of process to see what happens to resources and artifacts as activities occur
- Simulate alternatives and make changes to improve the process
- Example: systems dynamics model

2.3 Tools and Techniques for Process Modeling Dynamic Modeling: System Dynamics

- Introduced by Forrester in the 1950's
- Abdel-Hamid and Madnick applied it to software development
- One way to understand system dynamics is by exploring how software development process affects productivity

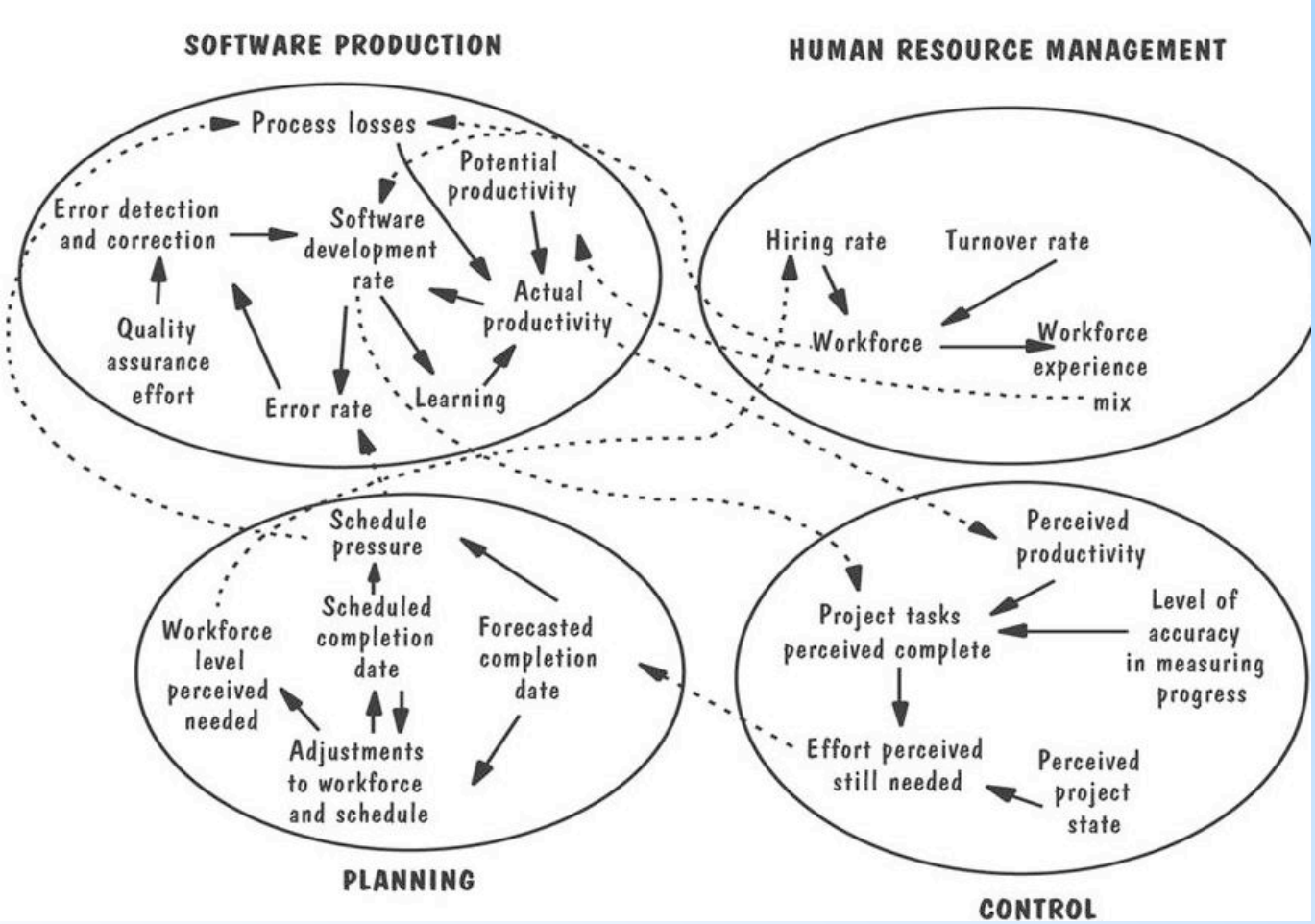
2.3 Tools and Techniques for Process Modeling Dynamic Modeling: System Dynamics (continued)

- **Pictorial presentation of factors affecting productivity**
- **Arrows indicate how changes in one factor change another**



2.3 Tools and Techniques for Process Modeling Dynamic Modeling: System Dynamics (continued)

- **A system dynamic model containing four major areas affecting productivity**



2.3 Tools and Techniques for Process Modeling Sidebar 2.4 Process Programming

- A program to describe and enact the process
  - Eliminate uncertainty
  - Basis of an automated environment to produce software
- Does not capture inherent variability of underlying development process
  - Implementation environment, skill, experience, understanding the customer needs
- Provides only sequence of tasks
- Gives no warning of impending problems

2.4 Practical Process Modeling
Marvel Case Studies

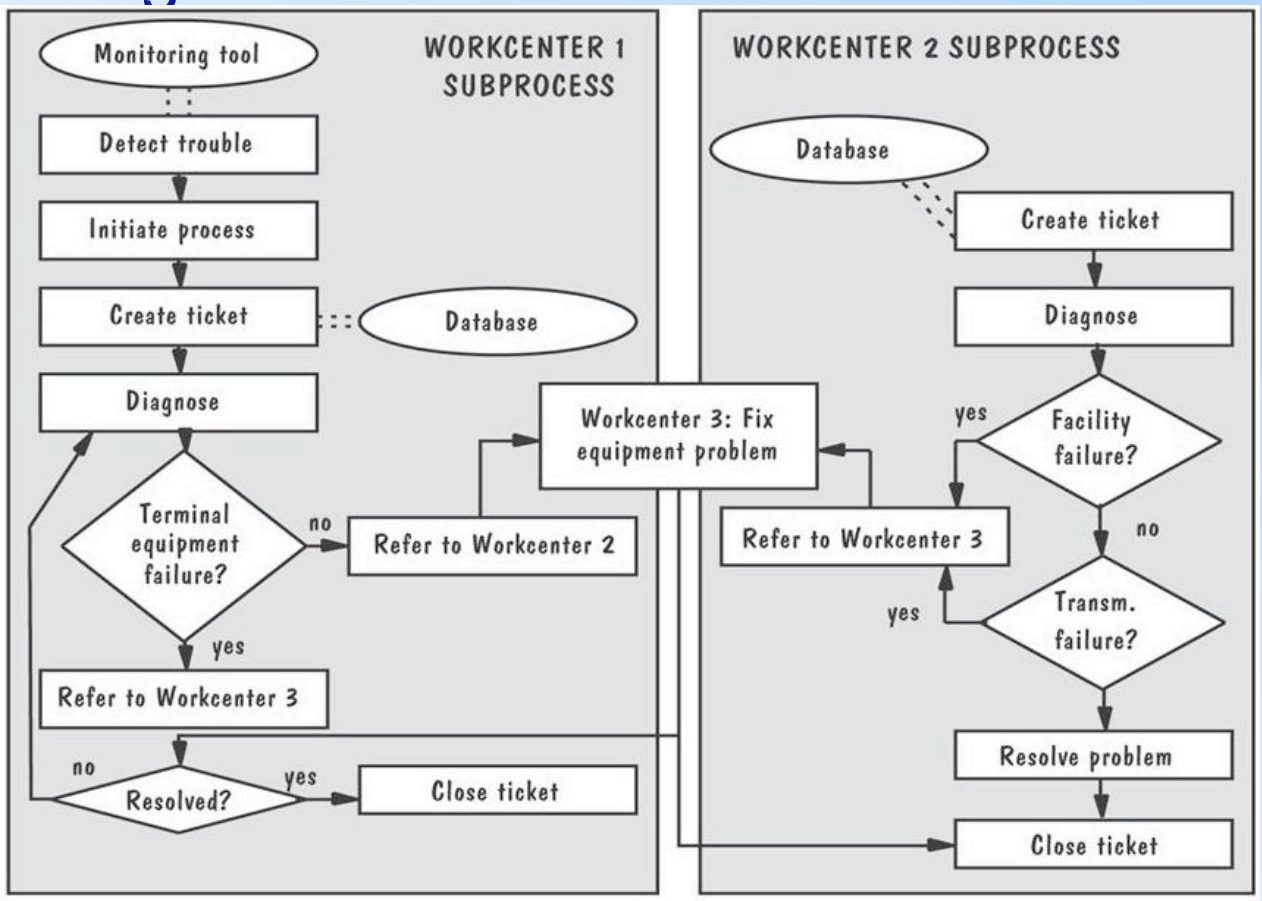
- Uses Marvel process language (MPL)
- Three constructs: classes, rules, tool envelopes
- Three-part process description
  - rule-based specification of process behavior
  - object-oriented definition of model's information process
  - set of envelopes to interface between Marvel and external software tools

2.4 Practical Process Modeling
Marvel Case Studies (continued)

- Involved two AT&T networks
  - network carried phone calls
  - signaling network responsible for routing calls and balancing the network load
- Marvel was used to describe the signaling fault resolution

2.4 Practical Process Modeling
Marvel Case Studies (continued)

- Signaling Fault Resolution Process



2.4 Practical Process Modeling
Example of Marvel Commands

```
TICKET:: superclass ENTITY
  status : (initial, open, referred_out, referral_done,
                  closed, fixed) = initial;
  diagnostics     : (terminal, non_terminal, none) = none;
  level           : integer;
  description     : text;
  referred_to     : link WORKCENTER;
  referrals       : set_of link TICKET;
  process         : link PROC_INST;
end

diagnose [?t: TICKET]:
 (exists PROC_INST ?p suchthat (linkto [?t.process ?p]))
 :
 (and (?t.status = open}(?t.diagnostics = none))
 {TICKET_UTIL diagnose ?t.Name}
 (and (?t.diagnostics = terminal)
        (?p.last_task = diagnose)
        (?p.next_task = refer_to_WC3));
 (and (?t.diagnostics = non_terminal)
        (?p.last_task = diagnose)
        (?p.next_task = refer_to_WC2));
```

Class definition for trouble tickets

Rule for diagnosing ticket

2.4 Practical Process Modeling

Desirable Properties of Process Modeling Tools and Techniques

- Facilitates human understanding and communication
- Supports process improvement
- Supports process management
- Provides automated guidance in performing the process
- Supports automated process execution

2.5 Information System Example
Piccadilly Television Advertising System

- Needs a system that is easily maintained and changed
- Requirements may change
  - Waterfall model is not applicable
- User interface prototyping is an advantage
- There is uncertainty in regulation and business constraints
  - Need to manage risks
- Spiral model is the most appropriate

2.5 Information System Example
Piccadilly System (continued)

- Risk can be viewed in terms of two facets
  - *Probability*: the likelyhood a particular problem may occur
  - *Severity*: the impact it will have on the system
- To manage risk, it needs to include characterization of risks in the process model
  - Risk is an artifact that needs to be described

2.5 Information System Example
Lai Artifact Table for Piccadilly System

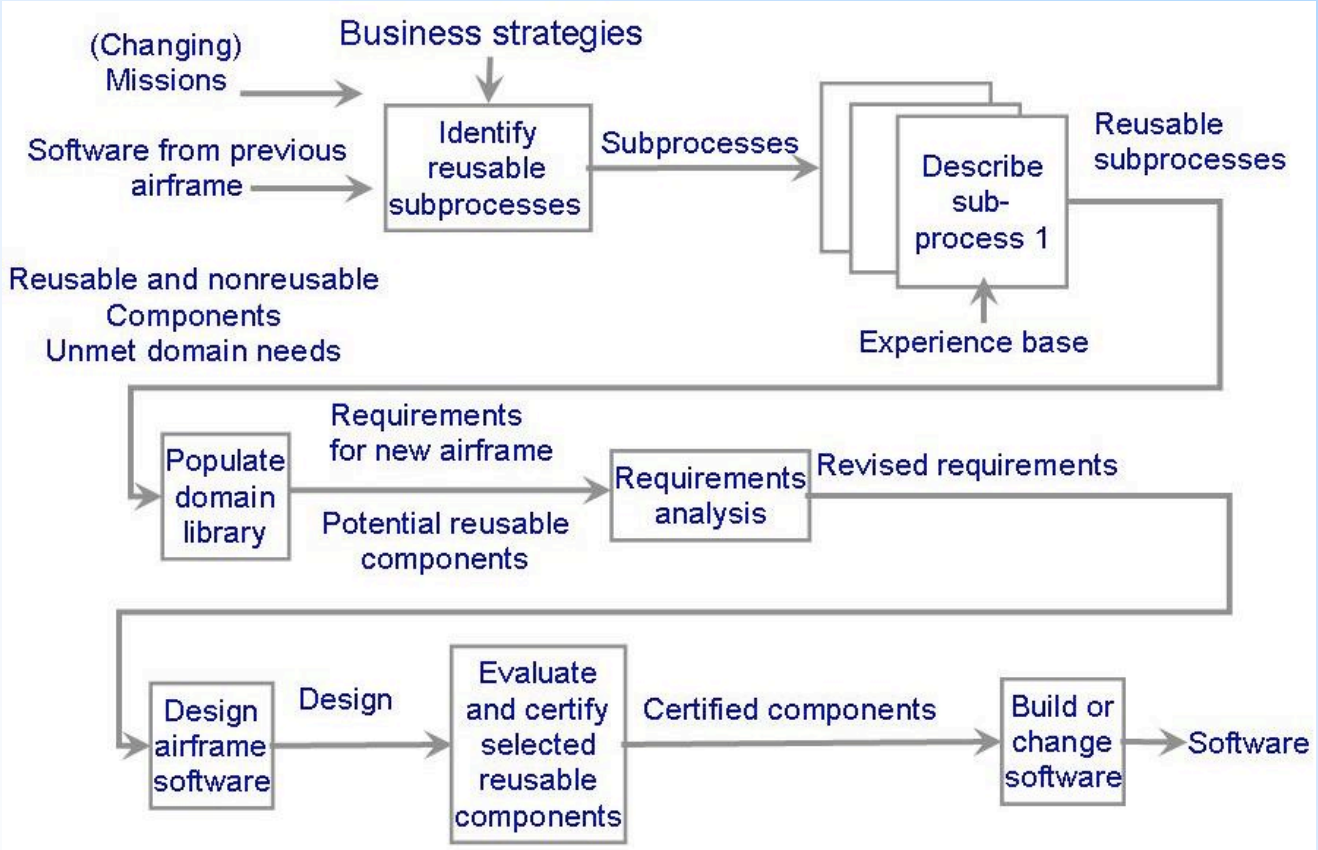
Name	*Risk (problemX)*	
Synopsis	*This is the artifact that represents the risk that problem X will occur and have a negative affect on some aspect of the development process.*	
Complexity type	*Composite*	
Data type	*(risk_s, user_defined)*	
Artifact-state list		
low	*((state_of(probability.x) = low) (state_of(severity.x) = small))*	*Probability of problem is low, severity problem impact is small.*
high-medium	*((state_of(probability.x) = low) (state_of(severity.x) = large))*	*Probability of problem is low, severity pr oblem impact is large.*
low-medium	*((state_of(probability.x) = high) (state_of(severity.x) = small))*	*Probability of problem is high, severity problem impact is small.*
high	*((state_of(probability.x) = high) (state_of(severity.x) = large))*	*Probability of pr oblem is high, severity problem impact is large.*
Sub-artifact list		
	probability.x	*The probability that problem X will occur.*
	severity.x	*The severity of the impact should problem X occur on the project.*

2.6 Real Time Example
Ariane-5 Software

- Involved reuse of software from Ariane-4
- Proposed reuse process model
  - (note: not actually used!)
  - Identify resuable subprocesses, describe them and place them in a library
  - Examine the requirements for the new software and the reusable components from library and produce revised set of requirements
  - Use the revised requirements to design the software
  - Evaluate all reused design components to certify the correctness and consistency
  - Build or change the software

2.6 Real Time Example
Ariane-5 Software (continued)

- Reuse process model presentation



2.7 What this Chapter Means for You

- Process development involves activities, resources, and product
- Process model includes organizational, functional, behavioral, and other perspectives
- A process model is useful for guiding team behavior, coordination, and collaboration

Coming Up Next…

- Week 4
  - Lecture 7
    - Chapter 3 of Concurrency Textbook
  - Lecture 8
    - Chapter 4 of Concurrency Textbook