Lecture Slides for Managing and Leading Software Projects

Chapter 2: Process Models for Software Development

developed by

Richard E. (Dick) Fairley, Ph.D.

to accompany the text

Managing and Leading Software Projects

published by Wiley, 2009

Chapter 2 Topics

- Introduction to Process Models
- Objectives of This Chapter
- A Development-Process Framework
- Tailoring the System Engineering Framework for
- Software-Only Projects
- Traditional Software Development Process Models
- Iterative-Development Process Models
- Designing an Iterative-Development Process
- The Role of Prototyping in Software Development

Additional Information (1)

- Appendix 2A to Chapter 2 provides an introduction to elements of the following frameworks, standards, and guidelines that are concerned with software development processes:
  - the SEI Capability Maturity Model® Integration CMMI-DEV-v1.2,
  - ISO/IEC and IEEE/EIA Standards 12207,
  - IEEE/EIA Standard 1058, and
  - the Project Management Body of Knowledge (PMBOK®).

Additional Information (2)

- Terms used in Chapter 2 and throughout the text are defined in Appendix A
- These presentation slides and other supporting material are available at the URL listed in the Preface to the textbook

Objectives for Chapter 2

- After reading this chapter and completing the exercises you should understand:
  - elements of the development process framework in Figure 2.1b of the text
  - distinctions among users, customers, and acquirers
  - tailoring of the framework for software-only systems
  - several commonly used process models for software development
  - ways in which the various development process models influence management of software projects
  - an example of process design

Key Factors in Software Engineering

- Three key factors in software engineering
  - people: numbers, skills, morale
  - processes: procedures for doing the work
  - technology: platform and domain
- Good *processes* help *people* apply *technology*
  - efficiently: without wasting time, effort, or resources
  - and effectively: while obtaining the desired result

What is a Process?

- A process is a systematic way of doing something
  - systematic = teachable & repeatable
- A process model specifies
  - the work activities to be accomplished
  - interconnections among the work activities
  - the input work products on which work activities are based
  - the output work products produced by work activities

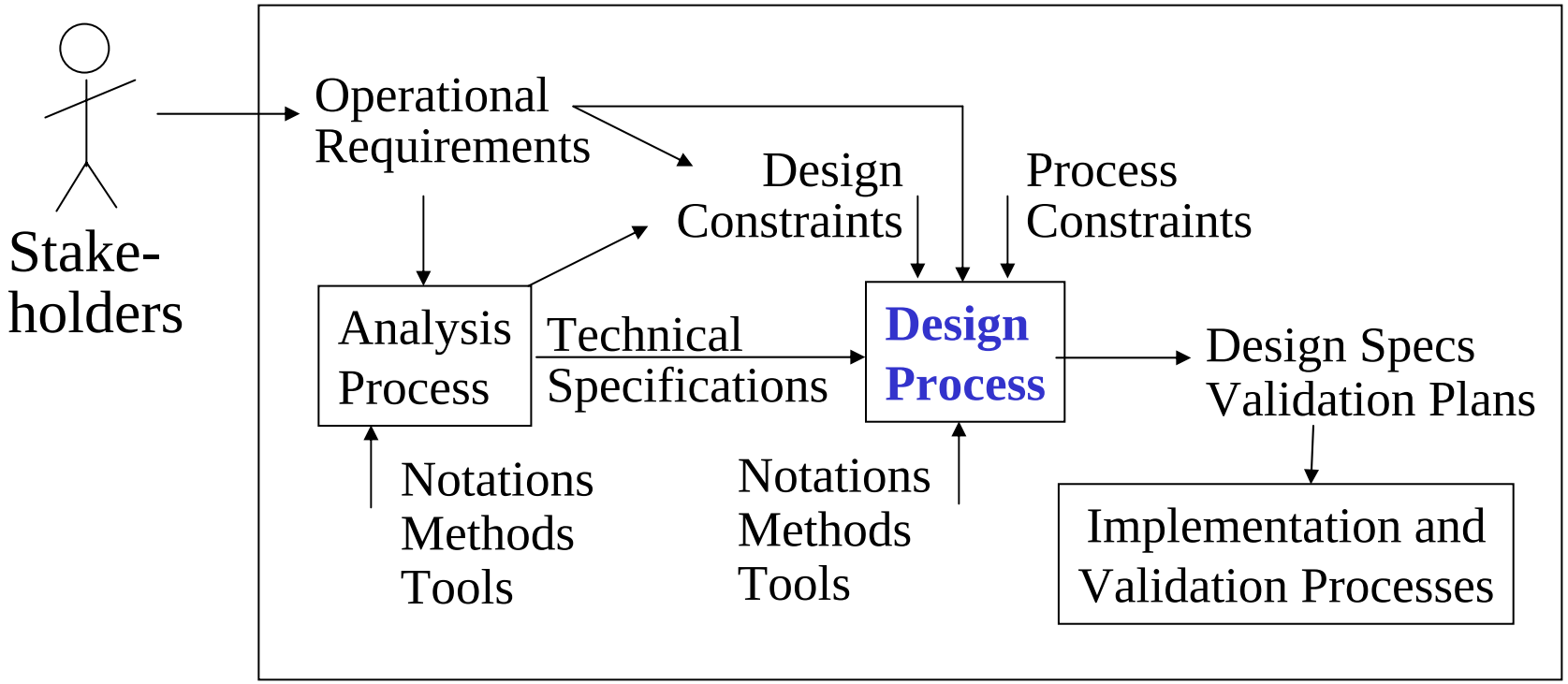
Each work process transforms one or more input work products into one or more output work products

Process Models

- Each element of a process model has
  - required inputs
  - procedures for accomplishing the process
  - work products to be produced
  - acceptance criteria for the output work products

Each element of a process occurs within a context

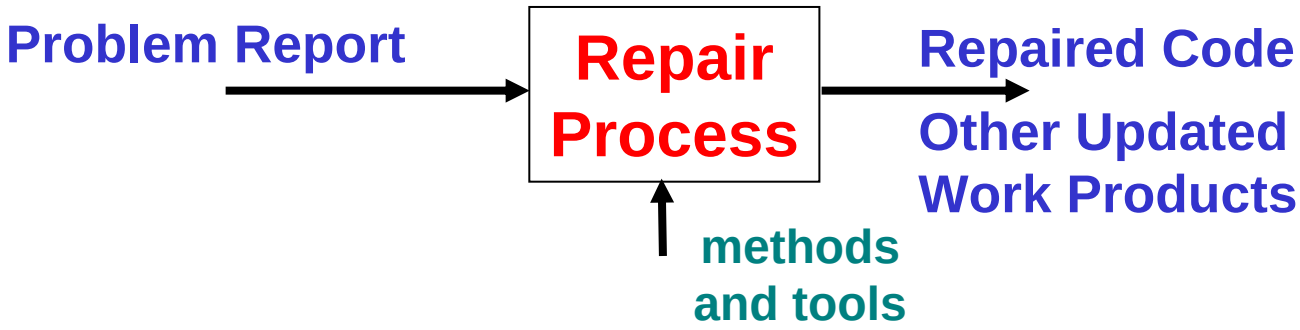
An Example: The Context of Software Design



Software engineering processes are best accomplished in an iterative manner

Process Models (2)

- A process includes procedures for conducting the work activities; for example:



An Example
-- Defect Repair Procedures --

Fixing a customer-reported defect involves the following procedures:

1. reproducing the failure
2. finding the defect
3. fixing the mistake
4. modifying the test suite
5. performing regression testing
6. documenting the fix
7. updating other work products as necessary
8. checking-in the modified code and documents
9. distributing the modified code
10. closing the problem report

note: fixing a user-reported defect involves a Change Control process

Some Terminology

- A process is a description of how to accomplish a work activity
- A procedure is a set of steps for accomplishing the work tasks of a process
- A technique is the way an individual accomplishes a procedure

processes and procedures are prescriptive
techniques are idiosyncratic to each individual who performs the procedures of a process

Five Basic Tenets of Process Engineering (1)

The basis tenets of process engineering are:

1. Better work processes result in better work products, where "better work products" means:
   - enhanced product features
   - improved product quality
   - less rework
   - easier modifications
2. Work processes must be designed with the same care used to design work products; work processes must be designed to:
   - satisfy process requirements and process constraints
   - fit the needs of individual projects
   - make the work processes efficient and effective

Five Basic Tenets of Process Engineering (2)

3. Work processes for each project should be derived from a process framework
   - a process framework is a generic process model that can be tailored to meet the needs of a variety of situations
   - tailoring of a framework involves adding, deleting, and modifying elements to adapt the framework to the needs of particular projects
4. Process design and process improvement result in:
   - shorter schedules
   - higher quality
   - lower costs
   - happier users and customers
   - happier workers and managers

Five Basic Tenets of Process Engineering (3)

5. Process improvement seldom happens spontaneously
   - investment in process engineering saves more time, effort, and cost than is invested;
   - however, a positive ROI (Return on Investment) requires an on-going investment of time, effort, and resources.

The SEI Process Models

- The SEI CMMIs are the best-known process models for software engineering
- SEI: Software Engineering Institute
- CMMI: Capability Maturity Models Integrated
- See: http://www.sei.cmu.edu/cmmi/models/models.html

The SEI-CMMI models specify **what to do** but **not how to do it**
i.e., processes and procedures but not techniques

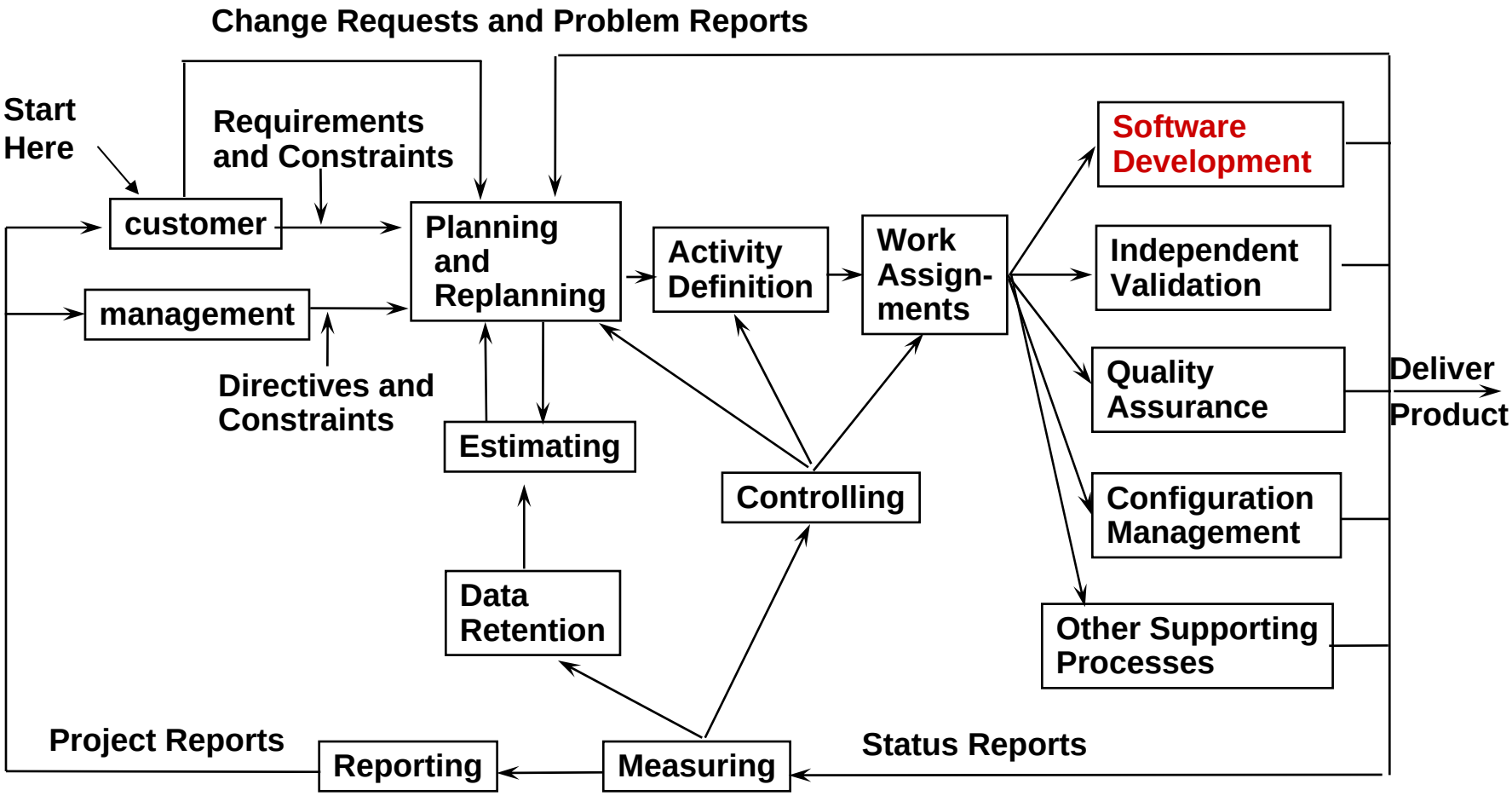
CMMI-DEV-v1.2 Project Management Process Areas

- Project Planning
- Project Monitoring and Control
- Supplier Agreement Management
- Quantitative Project Management
- Risk Management
- Integrated Project Management +IPPD*

* Integrated Process and Product Development

Each process area has goals, recommended practices, typical work products and examples

A Workflow Model for Software Projects



Software Development Process Models

- A process model for software development emphasizes:
  - the work activities to be performed in making a software product,
  - the order in which the work activities and tasks are to be performed,
  - the ways in which work activities and tasks can be overlapped and iterated, and
  - the work products that result from, and flow among, the various work activities.

Technical Issues in Software Projects

Product development includes:

- System engineering
- Software requirements engineering
- Software design
- Software implementation
- Software verification and validation
- System integration and validation

Some Terminology

- Lifecycle verification is the process of determining that a work product satisfies the conditions placed on it by other work products and work processes
  - i.e., is the work product complete, correct, and consistent with other work products and work processes
- Lifecycle validation is the process of determining that a work product satisfies its intended use when used by its intended users in its intended environment
  - i.e., is the work product suitable for use

note: work products include project plans, requirements, design specs, code, test plans, etc

Verification Techniques

- Verification techniques include:
  - traceability,
  - reviews,
  - prototyping,
  - analysis, and
  - functional testing.

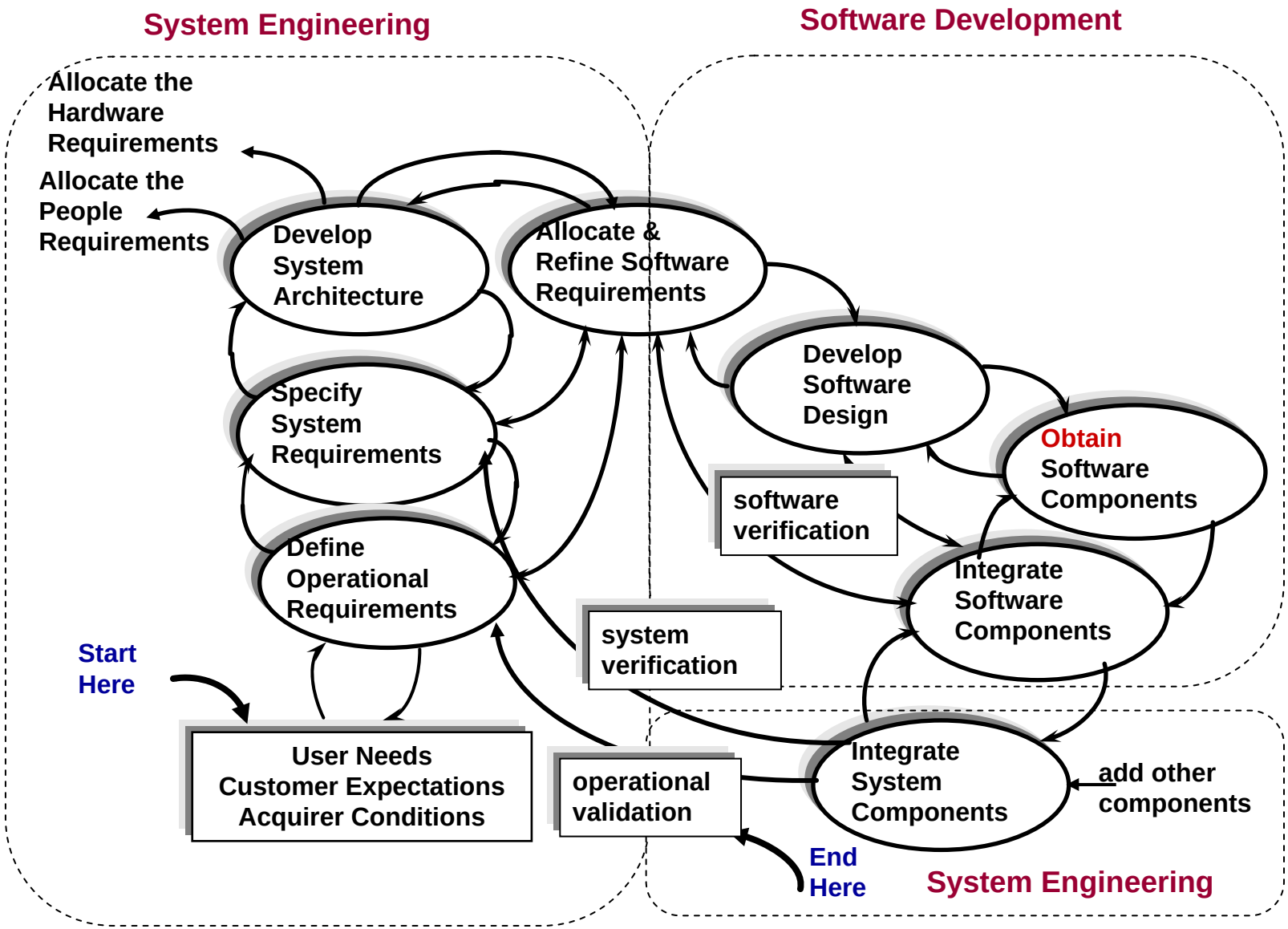
verification can, and should, be applied to all significant work products of a software project

Validation Techniques

- Validation techniques include:
  - reviews,
  - system testing,
  - operational testing, and
  - demonstrations.

validation can, and should, be applied to all significant work products of a software project

An Engineering Model for Developing Software-Intensive Systems



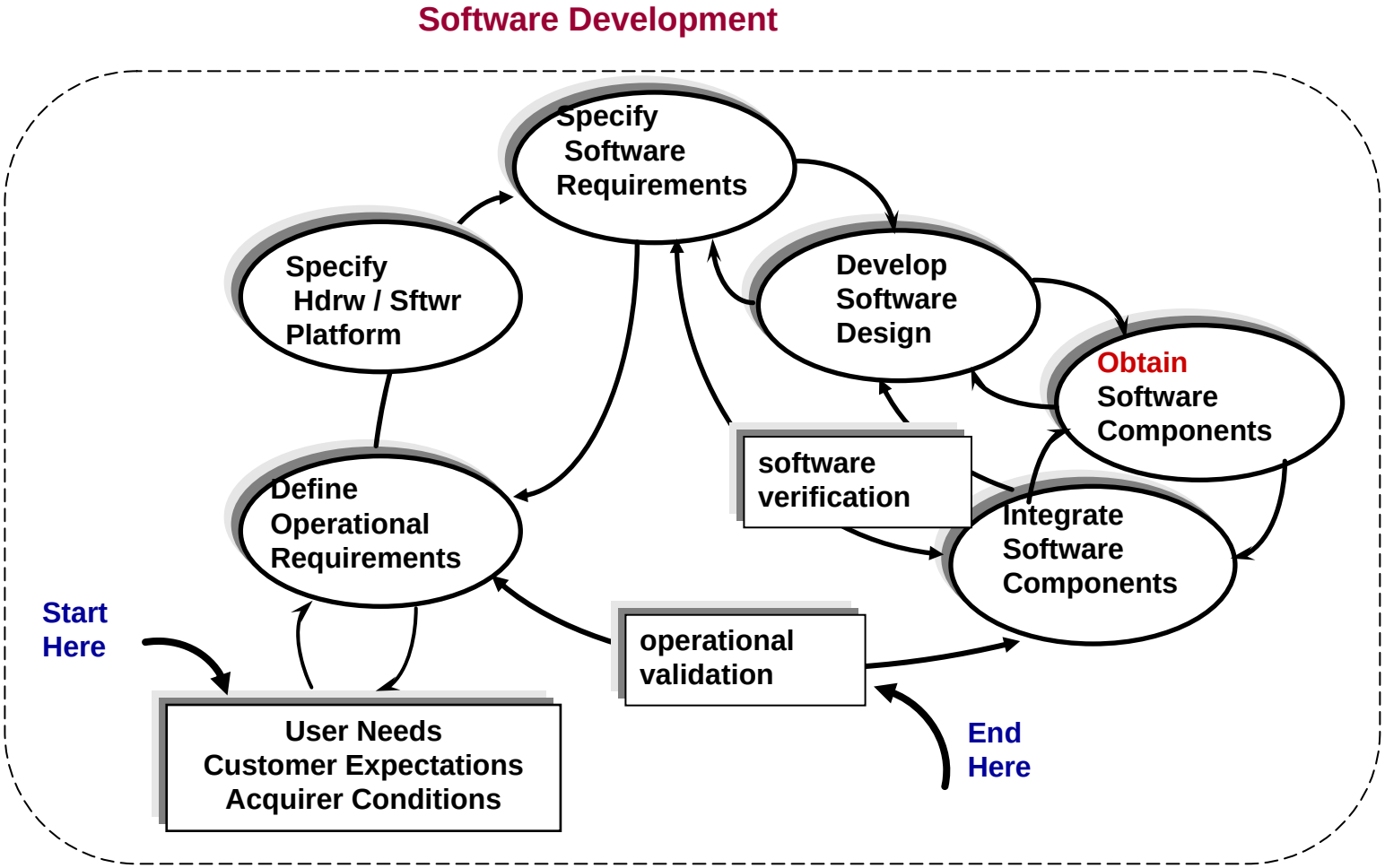
System Engineering of Software-Intensive Systems

- Software-intensive systems are composed of:
  - hardware
    - computers and other devices
  - software
    - new and existing
  - people
    - users, operators, maintainers

System Engineering

- The responsibilities of systems engineers include:
  - defining operational requirements,
  - specifying system requirements,
  - developing the system design,
  - allocating system requirements to system components,
  - integrating the system components as they become available,
  - verifying that the system to be delivered is correct, complete, and consistent with respect to its technical specifications, and
  - validating operation of the system with its intended users in its intended operational environment.

A Model for Developing Software-Only Systems



Ways to Obtain Software Components (1)

- Implement in-house (build them)
- Buy or license them from a vendor
- Procure from a subcontractor
- Reuse from another system
  - with or without modification
- Obtain from a library
- Obtain from open sources
- Other ways?

Q: How do you and your organization obtain your software components?

Obtaining the Software Components (2)

- A combination of techniques for obtaining components may be used
- *The ways in which the components are obtained influence the ways in which the project will be managed*
  - for example, subcontracting and COTS procurement require expertise in *acquisition management*

Some *Basic Elements* of a Software Development Environment

- coding style guidelines
- code documentation guidelines
- debugging tools
- peer reviews
- testing strategies, procedures, and tools
- periodic backups
- a version control system
- a defect tracking system
- a design documentation tool
- a requirements traceability tool
- a project planning and tracking system

Q: how does your development environment compare to this list?

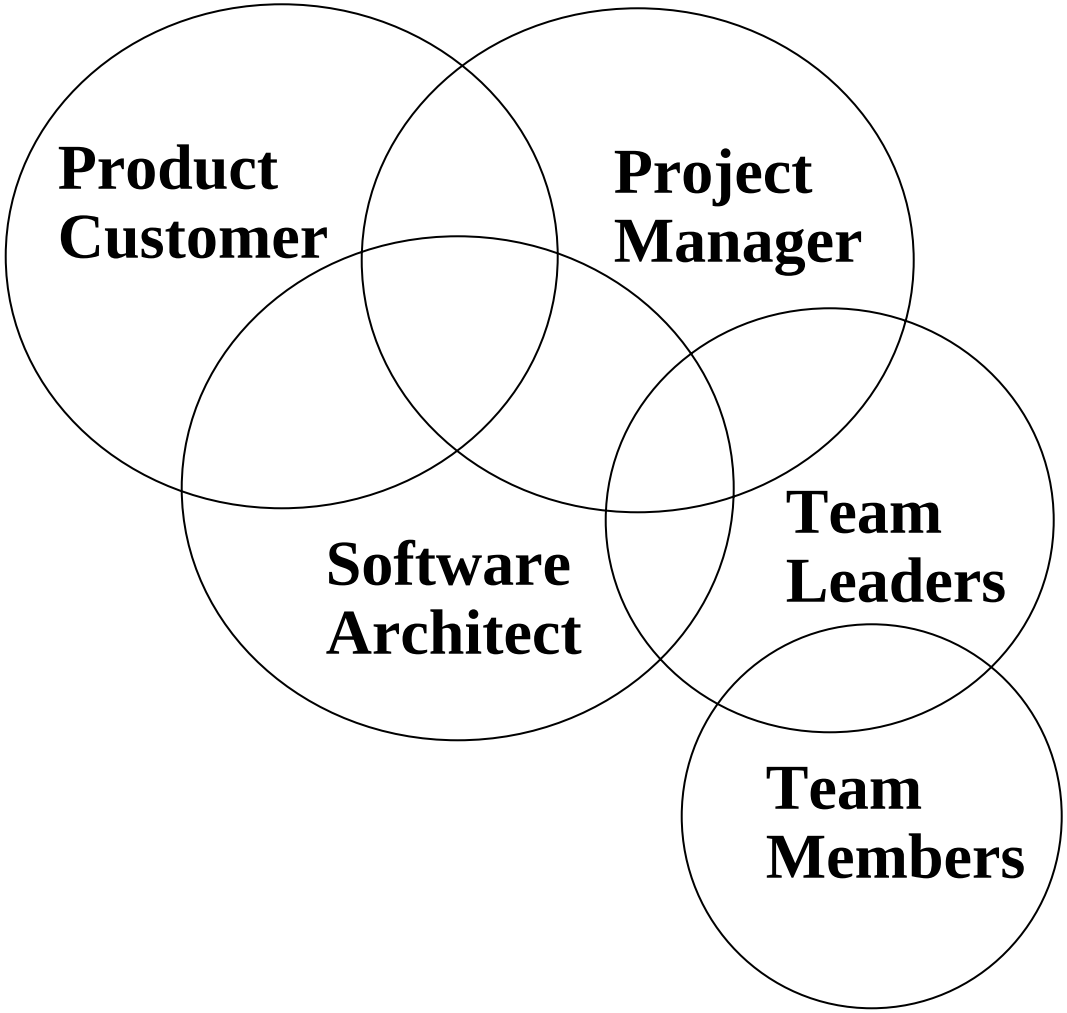
The System Engineering Framework

- Elements of the process framework depicted in Figure 2.1b are presented in the following sections of the text:
- Users, customers, and acquirers are covered in Section 2.2.1
- Operational requirements are covered in Chapter 3, as is the topic of requirements engineering for the software components of systems
- System requirements and system design are covered in Section 2.2.2
- Developing the software architecture and obtaining the software components are covered in Section 2.2.3
- Verification and validation are covered in Section 2.2.4

Some Terminology

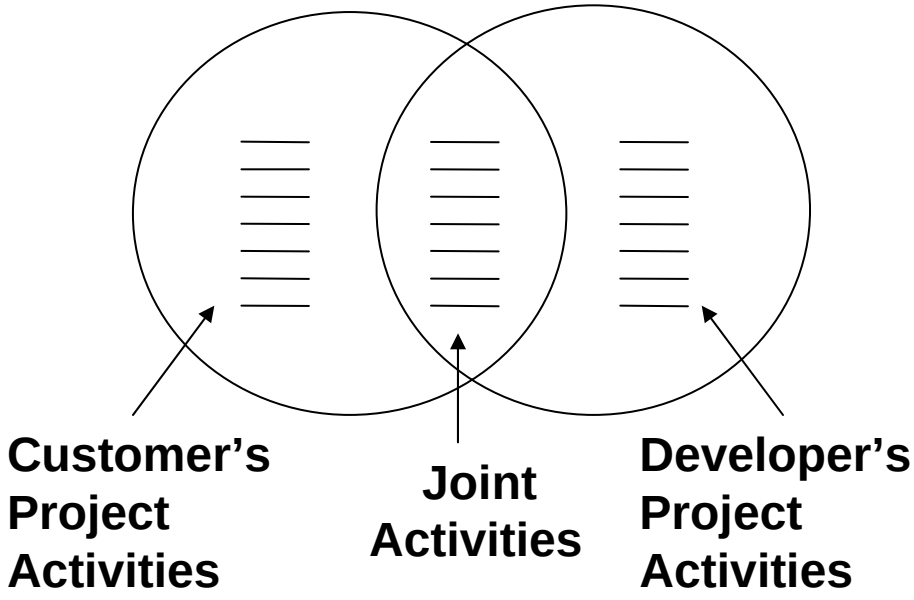
- Users are those individuals (or other systems, as in the case of embedded systems) who will utilize the delivered software to accomplish their work activities or pursue recreational pastimes
- Customers are those individuals and organizations who specify requirements and constraints, and accept the deliverable work products of a project
- The customer is called the acquirer in situations where the contractual agreement between customer and developer is a legally binding contract; in these cases the development organization is called the supplier.
  - in some cases, the acquirer may be a third-party agent who represents one or more customers or user communities and who provides the communication interface between the supplier and the customers/users.

Every Project Involves Multiple Stakeholders

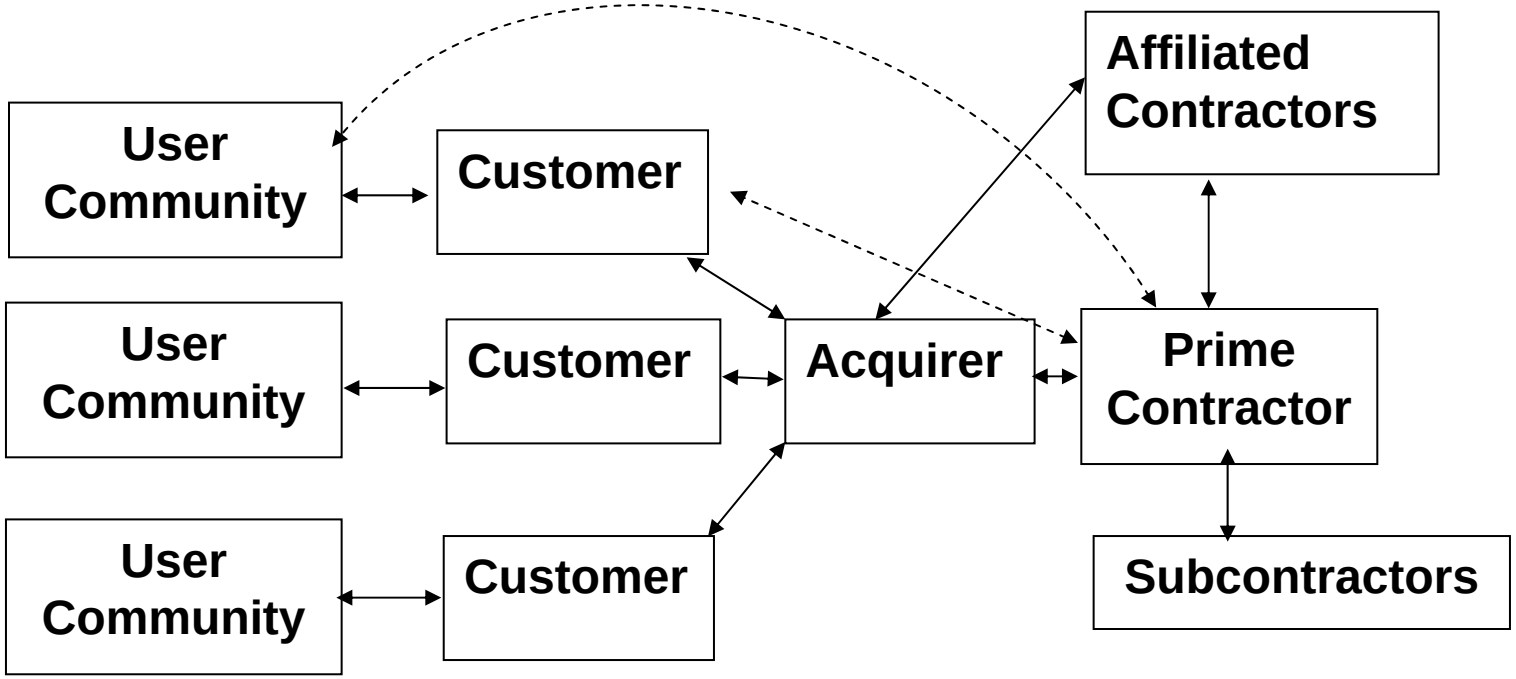


Every project involves a collection of stakeholders

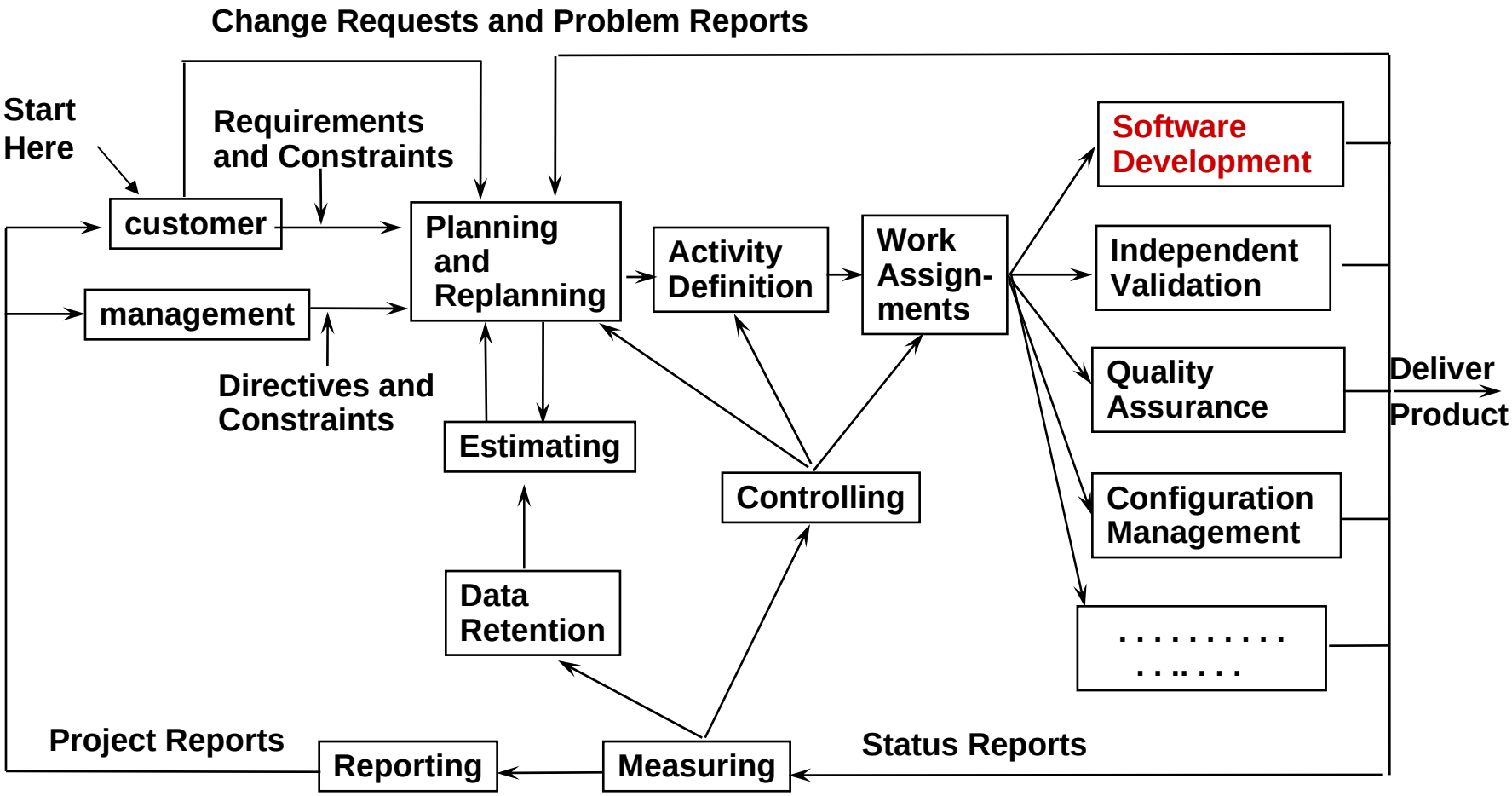
Every Project is Two or More Projects (1)



Every Project is Two or More Projects (2)



A Workflow Model for Software Projects



Designing and Tailoring Software Development Processes

- There are many different ways to develop software
  - because software is not subject to physical constraints
- *Therefore, we must design our work processes with the same care we use to design our work products*
  - the work processes and work products must be congruent
- Tailoring involves:
  - adding, modifying, and deleting elements of a generic process model to meet the needs of each particular project

Some Process Models for Software Development

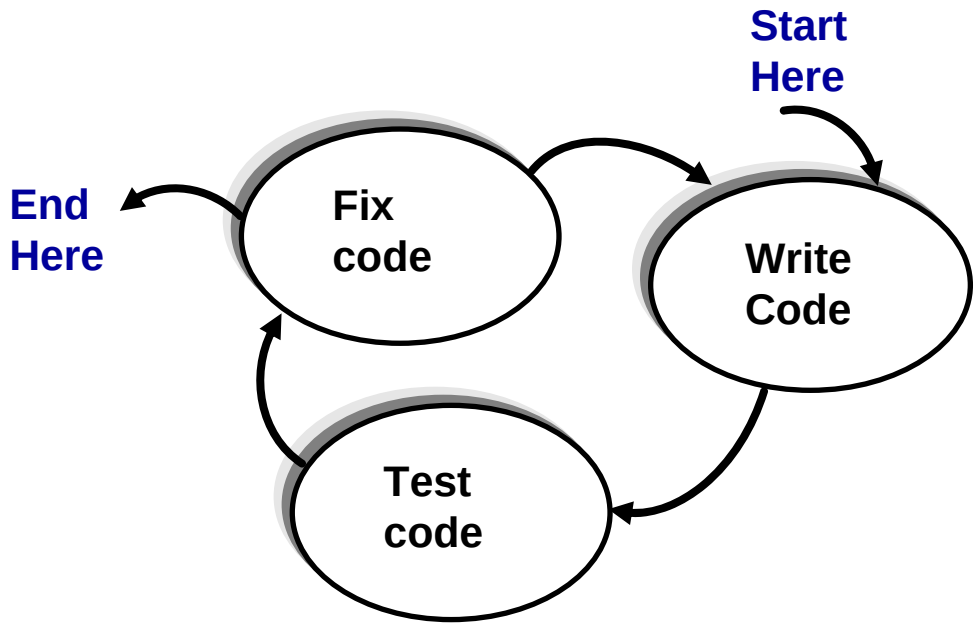
Conventional Models	Emphasis
Code and fix	Writing code without analysis or plan
Rqmts to code	Implementing code based on operational requirements
Waterfall	Sequential phases and milestone reviews
Iterative Models	**Emphasis**
Incremental-build	Iterative implementation, verification, and demonstration cycles
Agile	Customer Involvement
Evolutionary	Exploratory Development
Spiral	Risk Management

Code-and-Fix

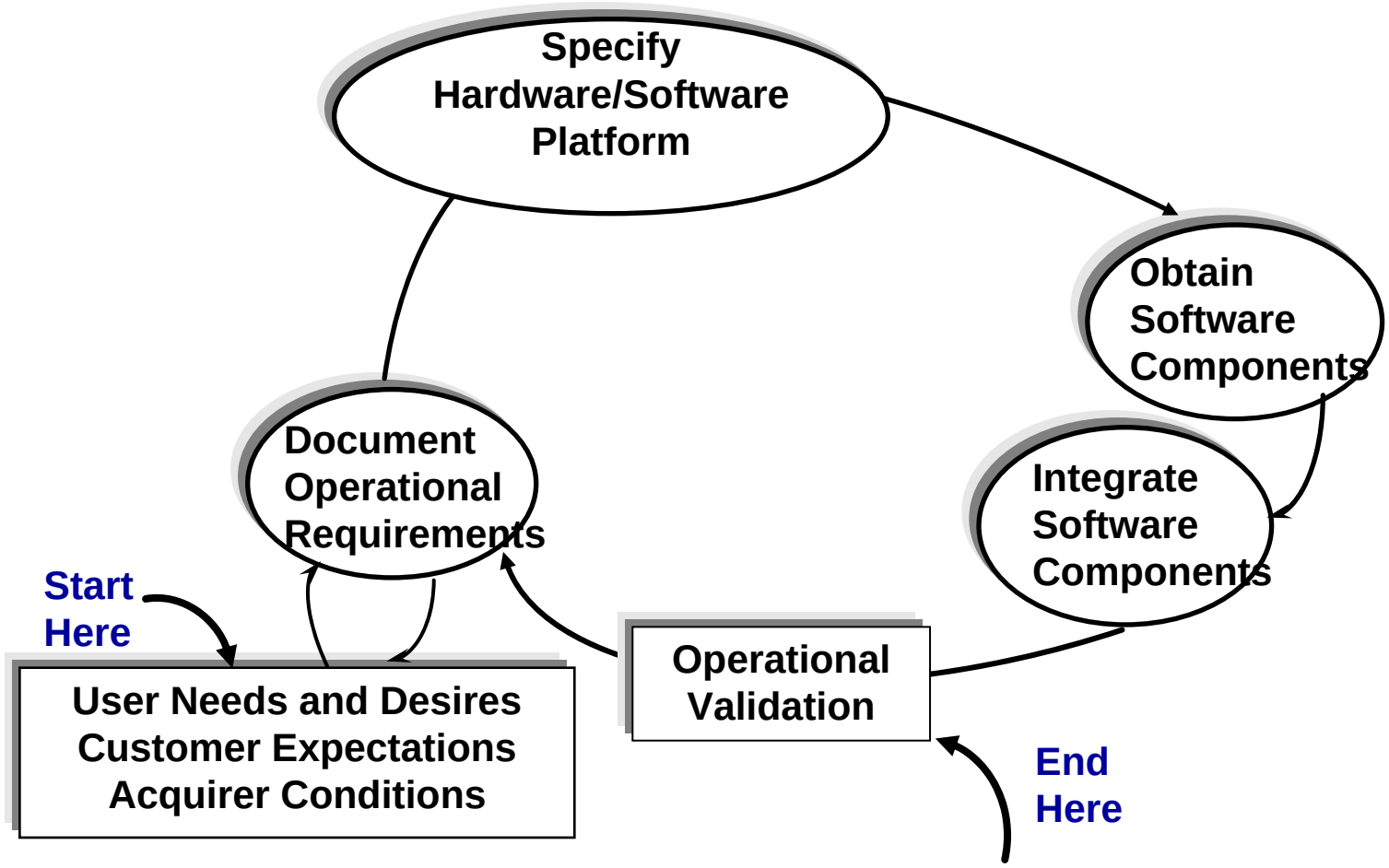
- Code-and-fix means to start writing the programs without any system engineering, requirements analysis, software design, or test planning
- Code-and-fix focuses attention on implementation details before
  - understanding the problem (analysis)
  - developing a solution (design)
  - determining success criteria (objectives)
  - developing plans for V&V

development of a software system requires various descriptions at various levels of abstraction

Tailoring of the Engineering Model for Code-and-Fix

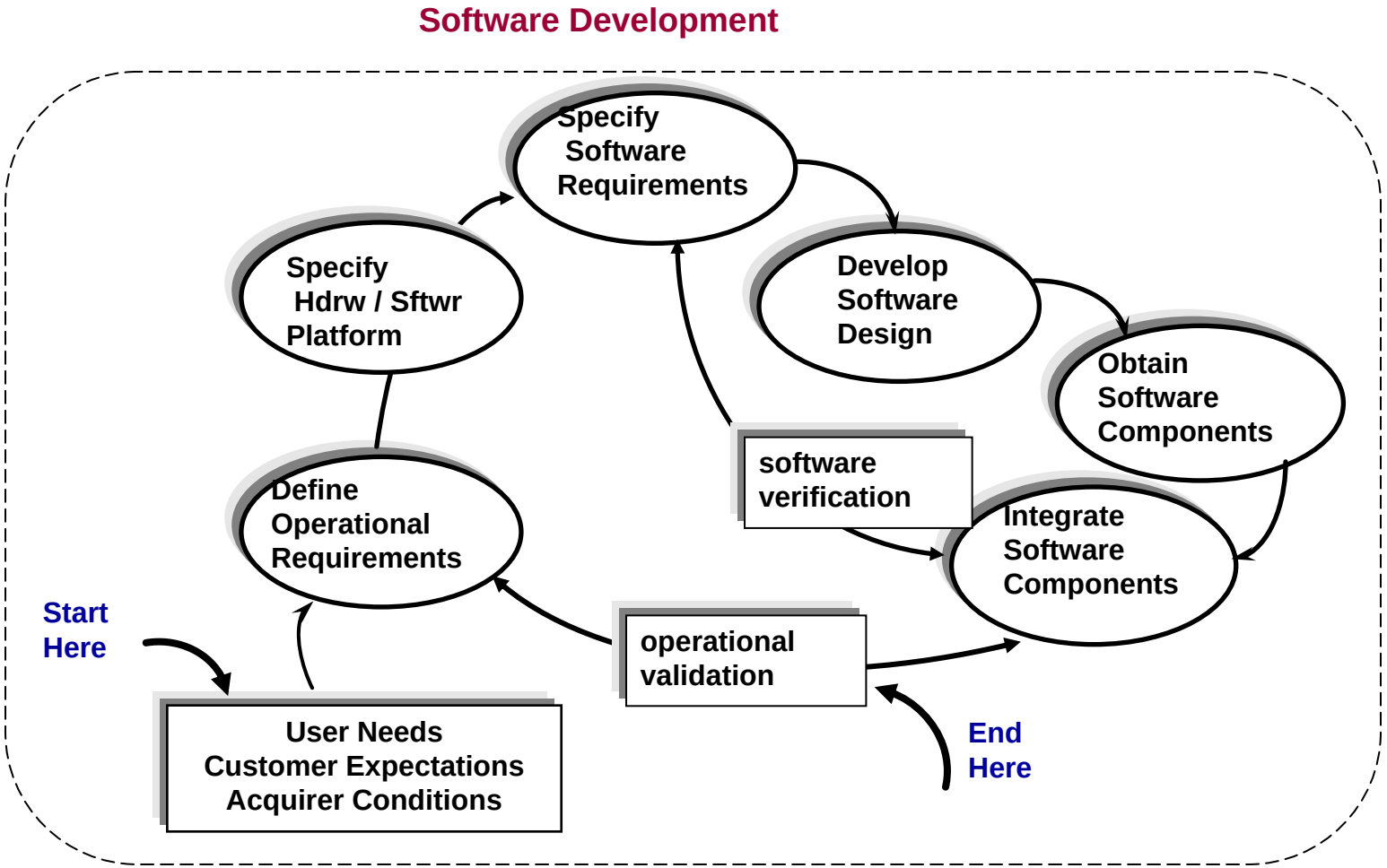


Tailoring of the Engineering Model for Requirements-to-Code



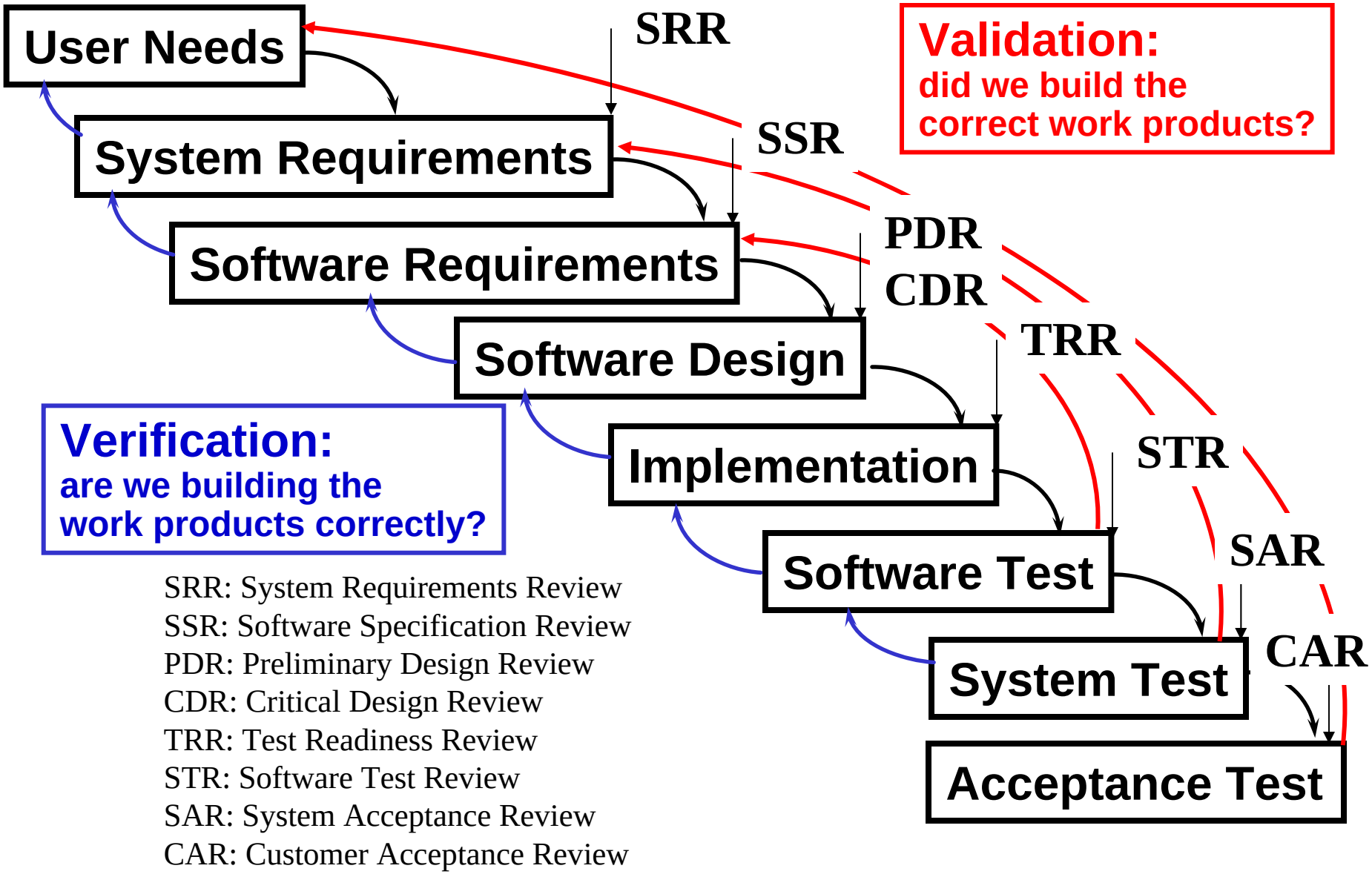
note the absence of iteration arrows other than documenting operational requirements

Tailoring the Engineering Model for Sequential Waterfall Development



note the absence of iteration arrows

The Linear Waterfall Model with Milestone Reviews



SRR: System Requirements Review
SSR: Software Specification Review
PDR: Preliminary Design Review
CDR: Critical Design Review
TRR: Test Readiness Review
STR: Software Test Review
SAR: System Acceptance Review
CAR: Customer Acceptance Review

Advantages of the Waterfall Approach

- Compared to Code-and-Fix and Requirements-to-Code, the Waterfall Model requires us to:
  - develop requirements before design
  - design before writing code
  - write code before integrating it
  - test programs after integrating them
  - have milestone reviews
  - establish and control work product *baselines*

Baselines and Version Control

- A *baseline* is a work product that has been accepted by the involved parties and placed under version control
  - acceptance requires application of some *acceptance criteria*
- A baseline cannot be changed without the agreement of the involved parties
- A baseline is changed because:
  1. the requirements for the work product changed, or
  2. a defect was found in the work product

baselines and version control are fundamental techniques of software engineering; they are elements of Configuration Management

Version Control of Work Product Baselines

- Version control of work product baselines ensures:
  - agreement that work product baselines are acceptable before baselining
  - protection from uncontrolled change
  - communication of changes
  - an audit trail of work product evolution
- Version control requires:
  - a plan for producing and accepting baselines
  - an automated version control system (tool)
  - a change request and change control system
  - a Change Control Board (CCB)

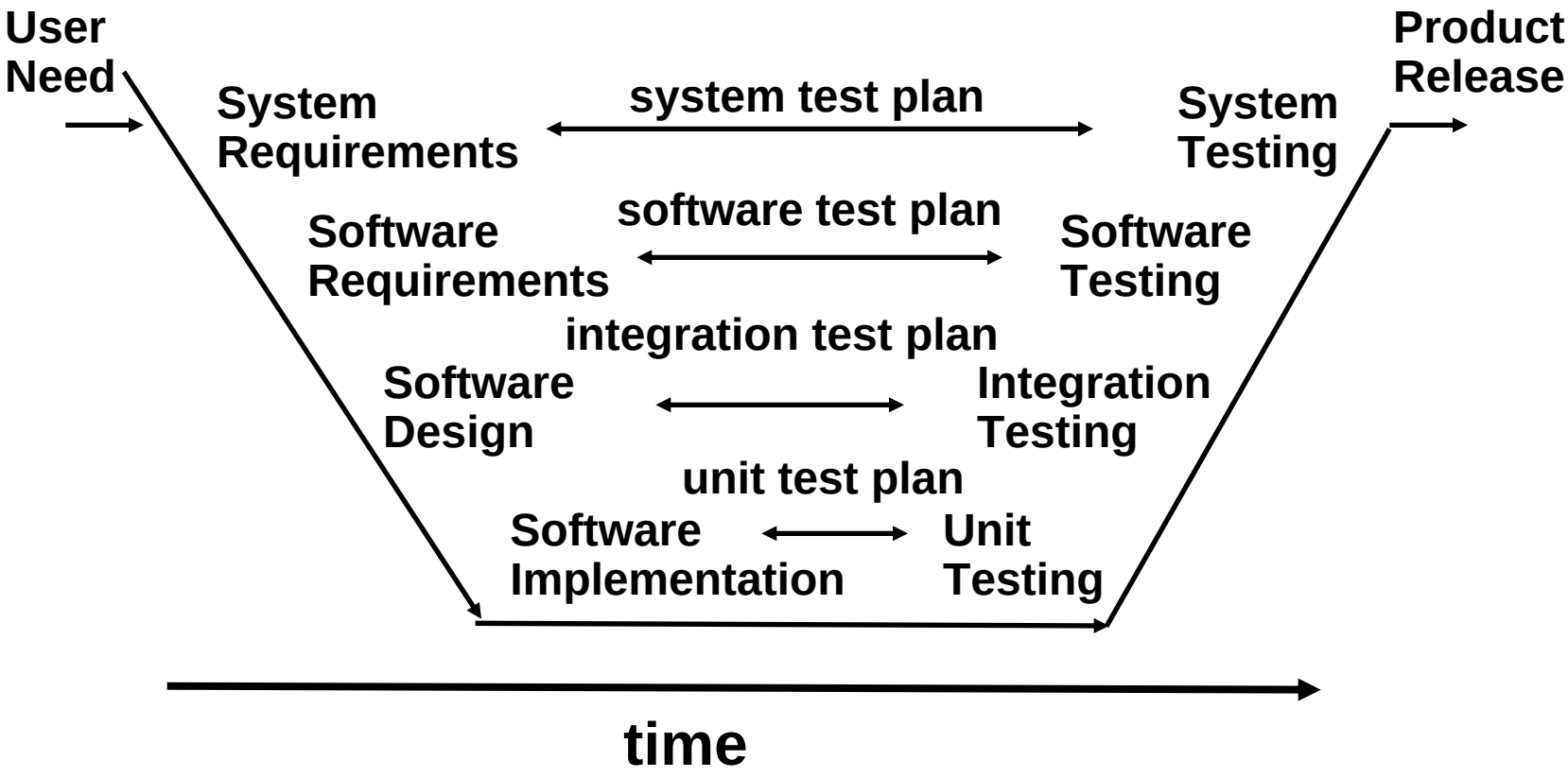
baselines and version control are important aspects of all process models; not just Waterfall

Managing Waterfall Projects

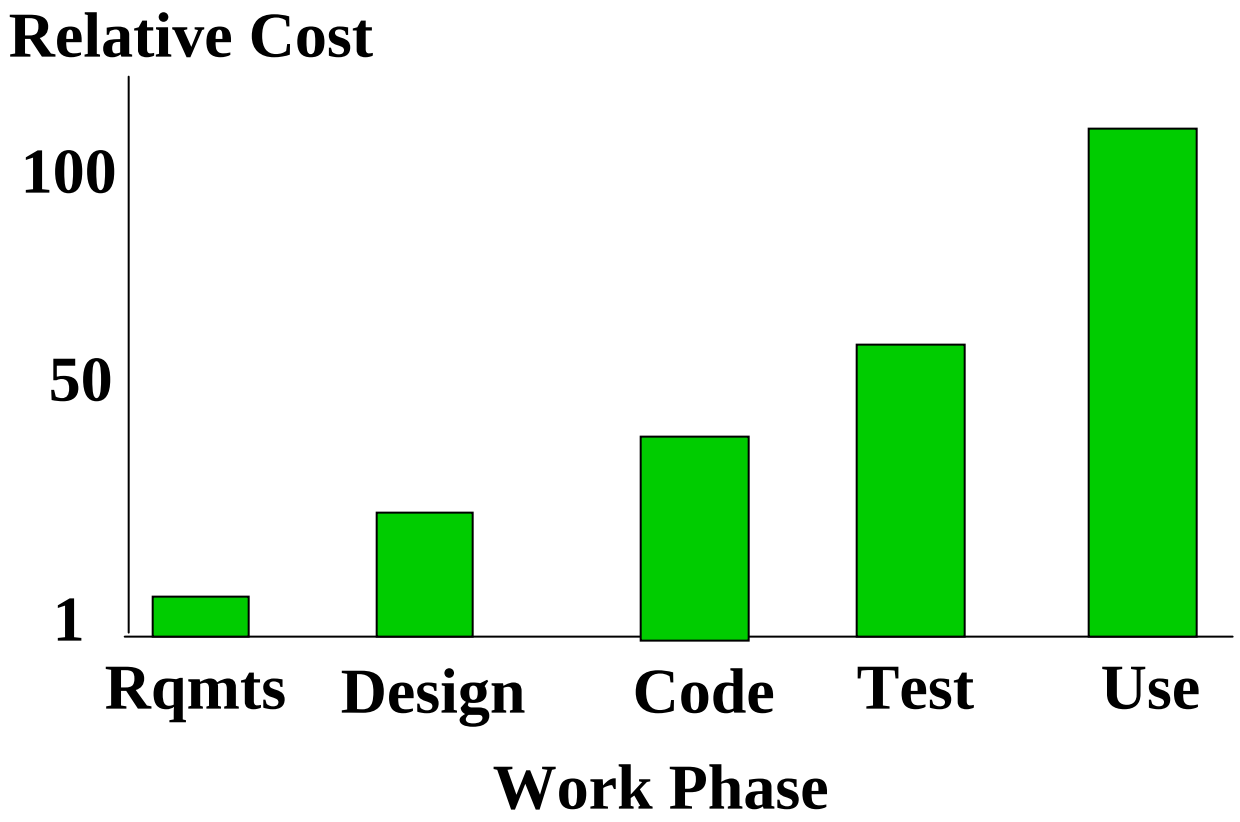
- The primary mechanisms for planning, measuring, and controlling Waterfall projects are:
  1. milestone reviews
  2. baselines
  3. version control

The V Model
(Illustrates A Problem of the Waterfall Model)

If we rely on waterfall testing alone the defects created first are detected last



Relative Cost to Fix a Software Defect



Q: Why is this true?

The Waterfall Approach

- The Waterfall Model requires that we (attempt to):
  - specify the requirements completely, consistently, correctly, and unambiguously on the first attempt
  - design the software completely and correctly on the first attempt
  - write all of the software interfaces and internal details correctly on the first attempt
  - integrate the components in one large step
  - do system testing and acceptance testing at the end

the linear waterfall model is a one-pass process

Some Realities of Software Development

1. Requirements always change because of:
   - changing customer desires and user needs
   - initial requirements analysis inadequate
   - understandings and insights gained through experience
   - changing technology
   - changing competitive situation
   - personnel turnover: engineering, management, marketing, customer
2. The design is never right the first time
   - design is a creative, problem solving process
3. Frequent demonstrations of progress and early warning of problems are desirable

Iterative development models are best

Iterative Development (1)

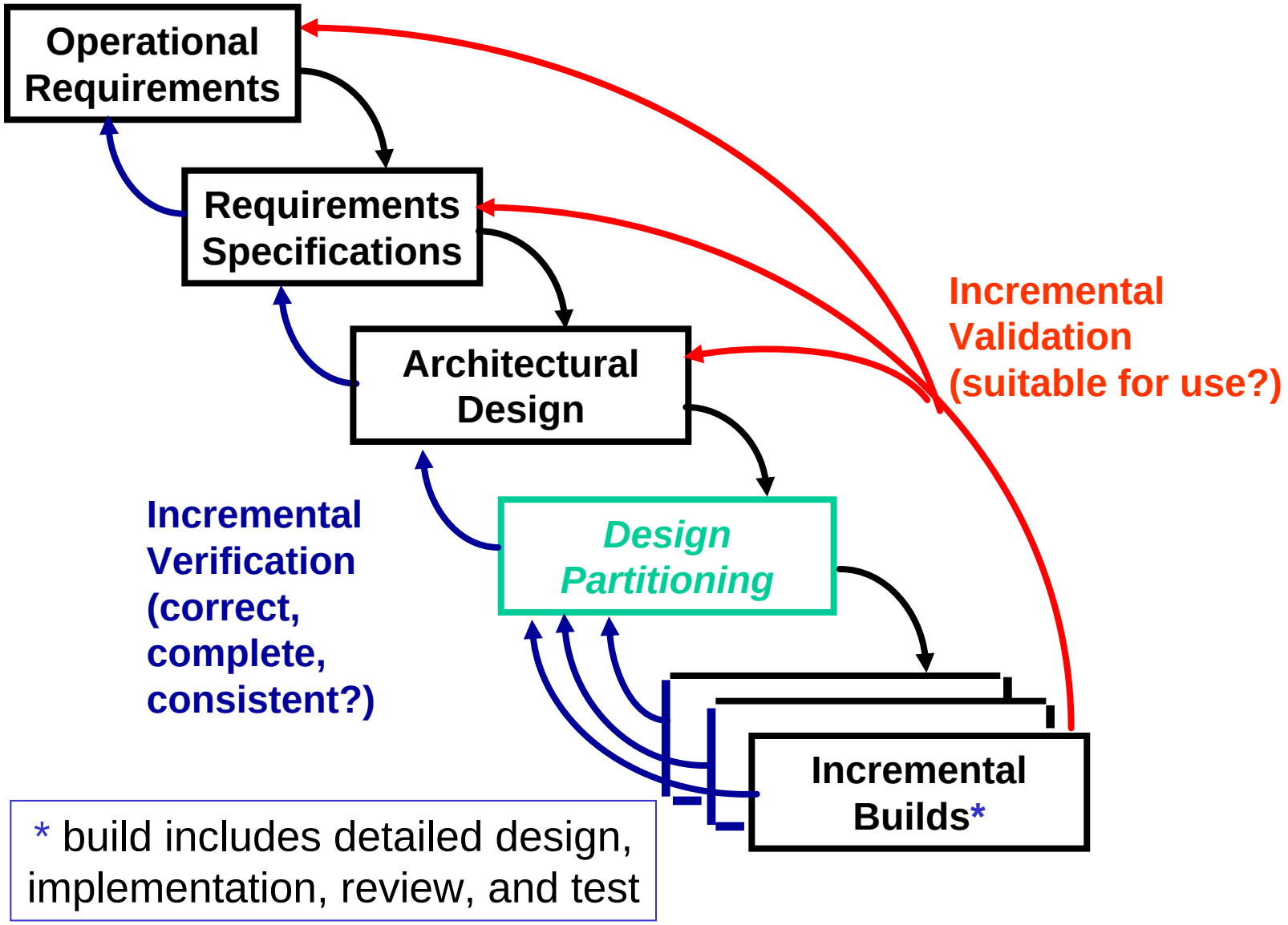
- *Iteration* is the process by which the desired result is developed through repeated cycles
- In software engineering, an iterative approach allows step-by-step revision of, and addition to, the work products
- Different types of iterative models support:
  - revision of & additions to requirements
  - revision of & additions to the design
  - revision of & additions to the code
  - testing a part of the system
  - and so forth

iteration should be planned;
not done as an afterthought

Iterative Development (2)

- The goals of iterative development are:
  - frequent demonstrations of progress
  - early warning of problems
  - ability to gracefully incorporate changes
- Four types of iterative development models
  1. incremental-build: iterative code-test-demo
  2. agile: satisfying operational requirements iteratively
  3. evolutionary: exploratory development
  4. spiral: managing risk

The Incremental-build Process



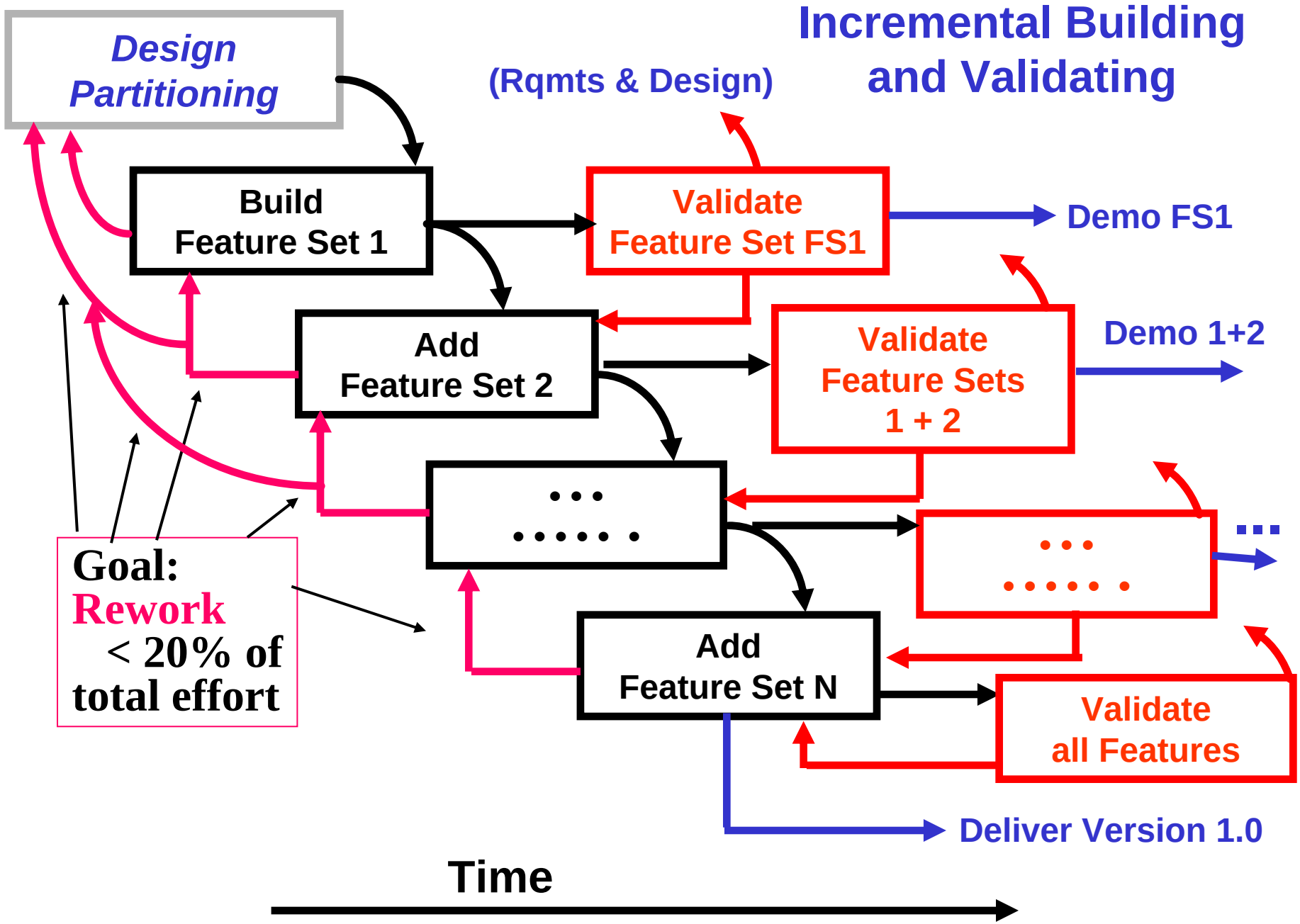
* build includes detailed design, implementation, review, and test

The Incremental Build Approach

- Design is partitioned into a series of *prioritized* builds
  - each build adds capabilities to the existing base in priority order
    - each build produces a demonstrable version
      - –usually on a weekly basis
  - most critical parts are built first
    - and tested/demonstrated most often

Partitioning Criteria

- Safety-critical: safety features first
- Security-critical: security layers first
- Data-intensive: data schema first
- User-intensive: user interface first
- Others?


Incremental Building and Validating
Design Partitioning
(Rqmts & Design)
Build Feature Set 1
Validate Feature Set FS1
Demo FS1
Add Feature Set 2
Validate Feature Sets 1 + 2
Demo 1+2
Goal: Rework < 20% of total effort
Add Feature Set N
Validate all Features
Deliver Version 1.0
Time

Rework*

- Rework
  - evolutionary
  - avoidable
    - retrospective
    - corrective
- Avoidable rework often accounts for 30% to 50% (or more) of total effort
  - avoidable rework is the bane of software development
  - and should be the first line of attack in process improvement

* Iterative Rework: "The Good, The Bad, and the Ugly: by R. Fairley and M. J. Willshire, *IEEE Computer*, September, 2005

Validation Techniques

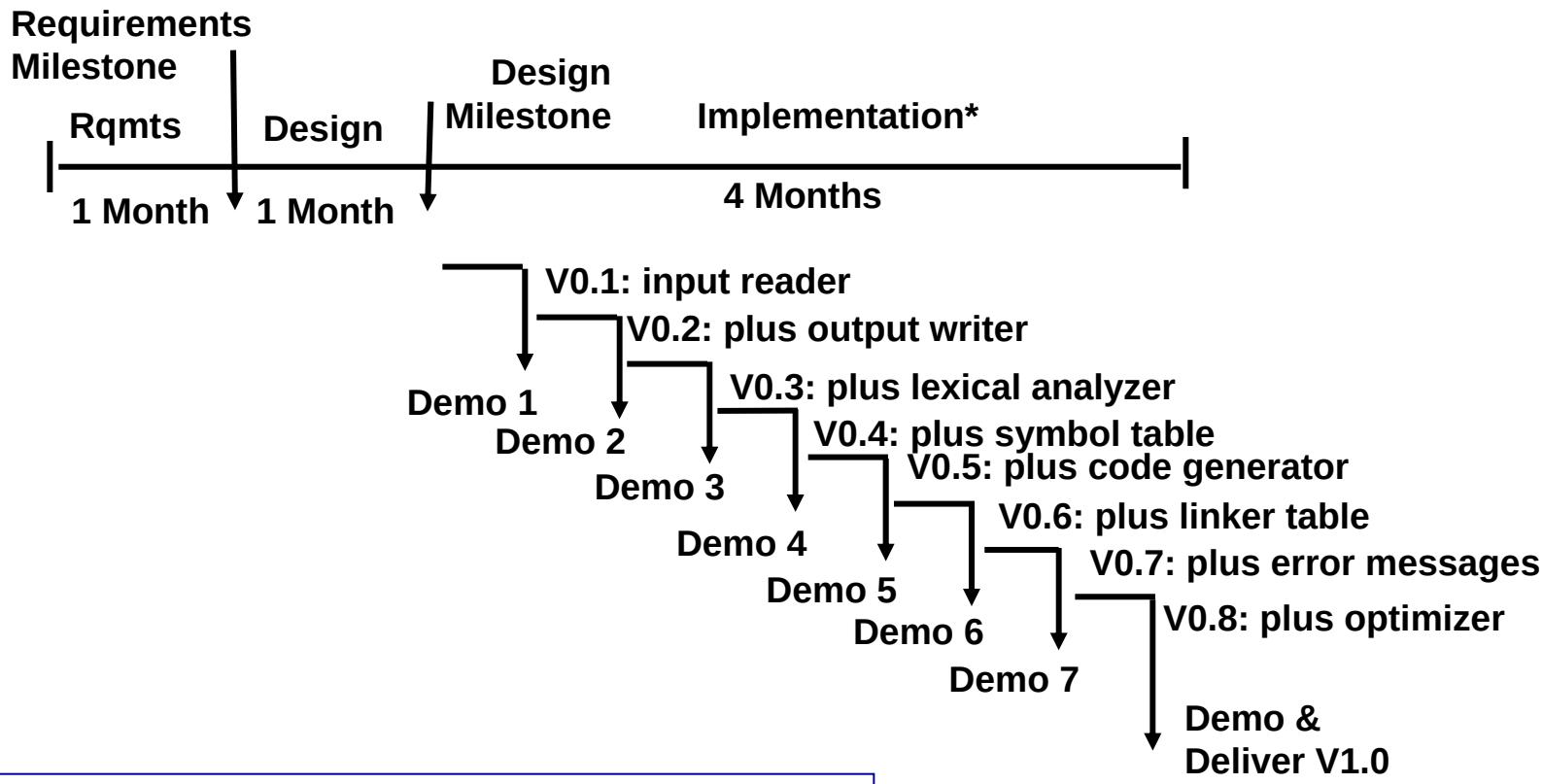
- Peer Review
- Testing
- Demonstration
- Analysis

validation: did we build the correct work products?
i.e., do the work products satisfy their intended use when used by their intended users in their intended environments?

Guidelines for Incremental-build Development

- Build an “official” demonstration version every week
  - developers may do more frequent “sandbox” builds
- Build product versions according to priority of requirements
  - build and test the most critical parts first
- Do independent review and testing of each version
- Incorporate requirements changes into subsequent builds
- Redesign if rework exceeds 20% of effort when building two successive product builds
  - excessive evolutionary rework: requirements changes
  - excessive retrospective rework: bad design partitioning
  - excessive corrective rework: bad coding

An Incremental-build Example
Implementing a Compiler



*implementation of each increment includes detailed design, coding, review, integration, testing, and demonstration

Advantages of Incremental-build Development

- Allows early and continuing demonstrations of progress
- Components built first are tested most
- Can incorporate some changes to requirements in later versions
- Provides early warning of problems
  - schedule, technical, . . .
- Can make trade-offs of features and schedule
  - during development
  - at delivery time
- Permits better allocation of staff resources than waterfall
- Can provide incremental deliveries to customers, if desired

Incremental-build Delivery

- Deliver of early versions provides early feedback from users
- Allows product delivery on schedule
  - with most important features working
  - with systematic planning for later versions
- At delivery time, it is better to be
  - 100% complete with 80% of the most important product features

  than to be
  - 80% complete with 100% of features
    - and nothing to deliver

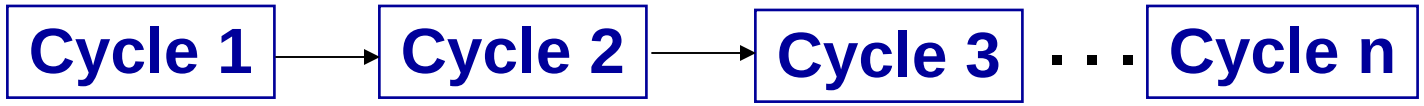
Managing Incremental-build Development

- Management is more complex than in the waterfall approach:
  - more “pieces and parts” in various stages of development
  - more communication and coordination
  - an automated version control system is essential
  - must decide how incremental validation will be handled

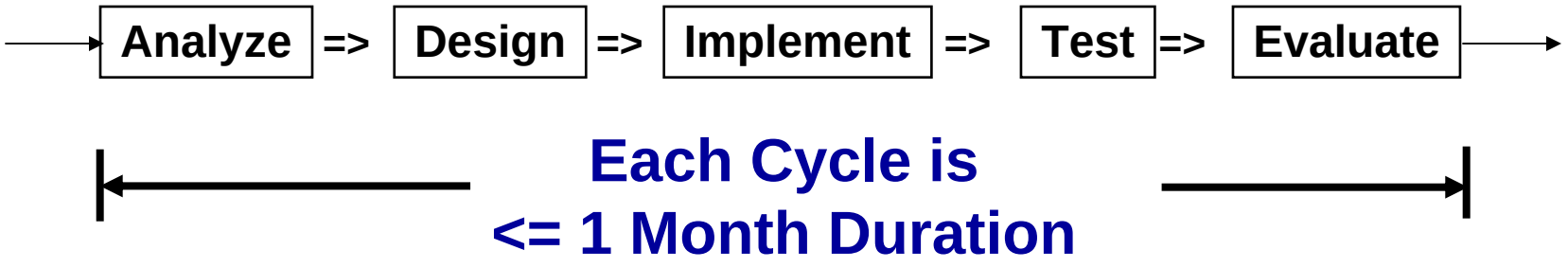
Frequent demonstrations of progress (or lack thereof) are the primary mechanism of controlling an incremental- build project

The Evolutionary Model

- Used when a stable first version of the requirements cannot be (mostly) specified *in advance:*



- Details of each cycle:



Evolutionary Development Guidelines

- Used when requirements cannot be mostly specified in advance
- Evolutionary cycles end when
  - project is converted to an incremental approach
  - or, project is cancelled because it is infeasible
  - or, product is delivered
- Using the evolutionary approach indicates a *high-risk project*

The Spiral Approach

- The Spiral Development Process is a meta-level model for iterative development models
  - earlier activities are revisited, revised, and refined on each pass of the spiral
- Each cycle of a spiral model involves four steps:

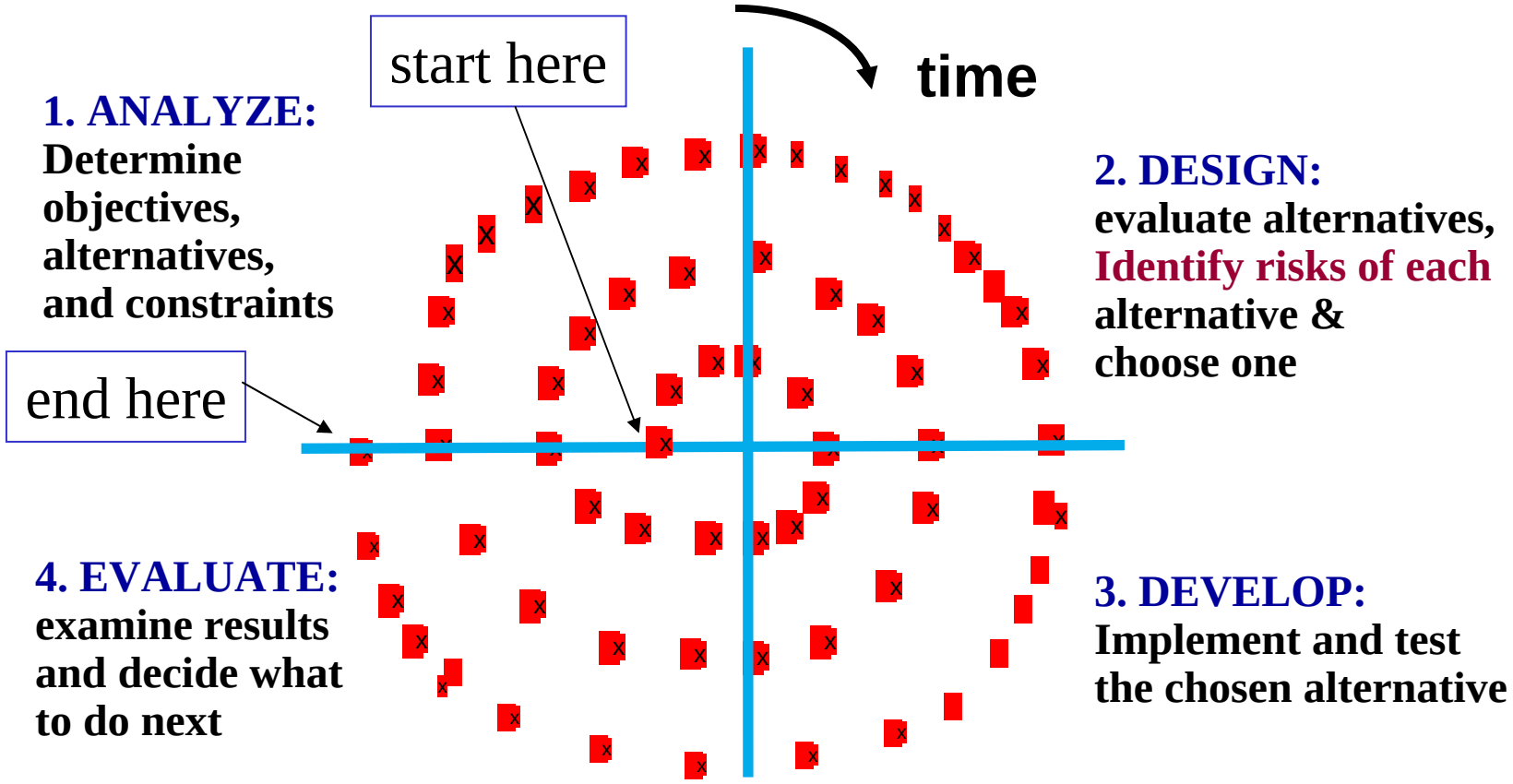
  Step 1 - determine objectives, alternatives, and constraints

  Step 2 - identify risks for each alternative and choose one of the alternatives

  Step 3 - implement the chosen alternative

  Step 4 - evaluate results and plan for the next cycle of the spiral
- The cycles continue until the desired objectives are achieved (or until time and resources are used up)

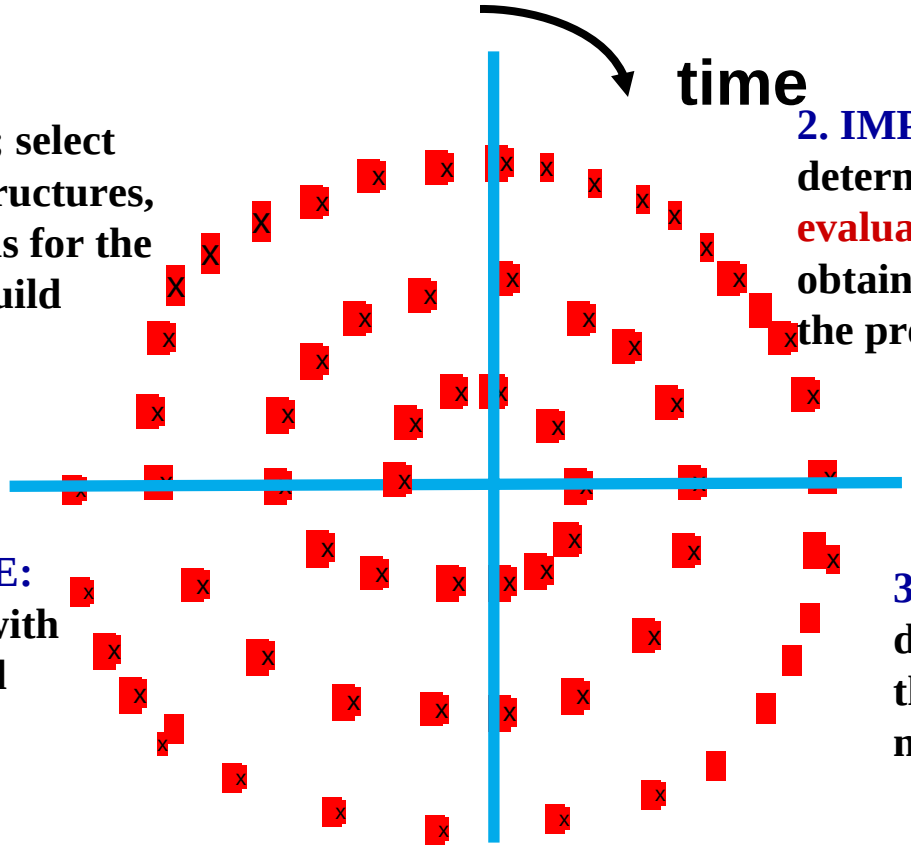
An *Evolutionary* Spiral Model



Each Cycle is One Month or Less

An *Incremental* Spiral Model

1. DESIGN:
revise as necessary; select algorithms, data structures, and interface details for the next incremental build



2. IMPLEMENT:
determine ways to obtain the code
evaluate the risk of each
obtain the code and integrate it into the present version

3. VERIFY & VALIDATE:
determine acceptability of the build; rework as necessary

4. DEMONSTRATE:
evaluate the build with customer, users and other stakeholders

Each Cycle is One Week or Less

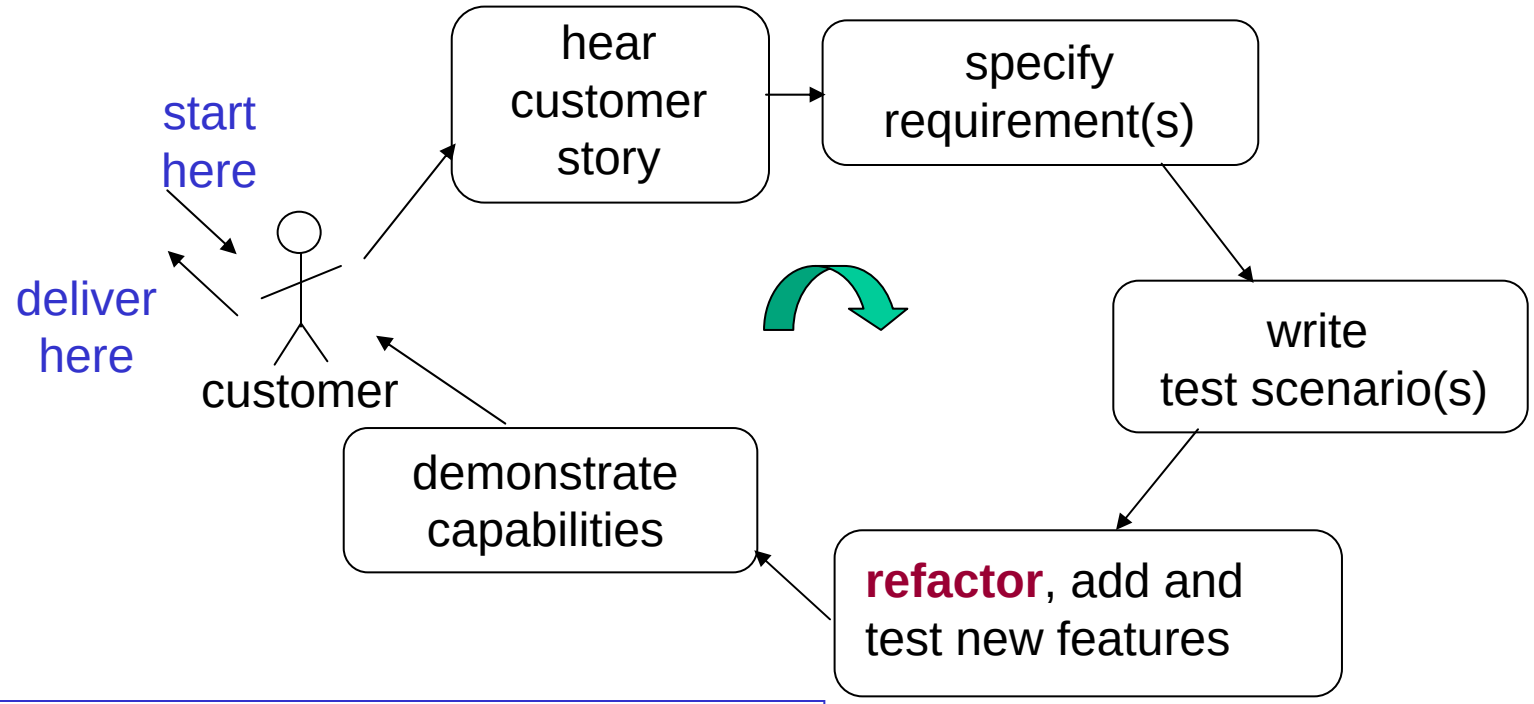
Agile Development: The Agile Manifesto*

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

* see http://agilemanifesto.org/

An Agile Development Model

- Emphasis is on satisfying operational requirements



- Iterations may occur on a daily basis
- Deliveries to users may be frequent

refactor: modify structure w/o changing behavior

Agile Principles (1)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

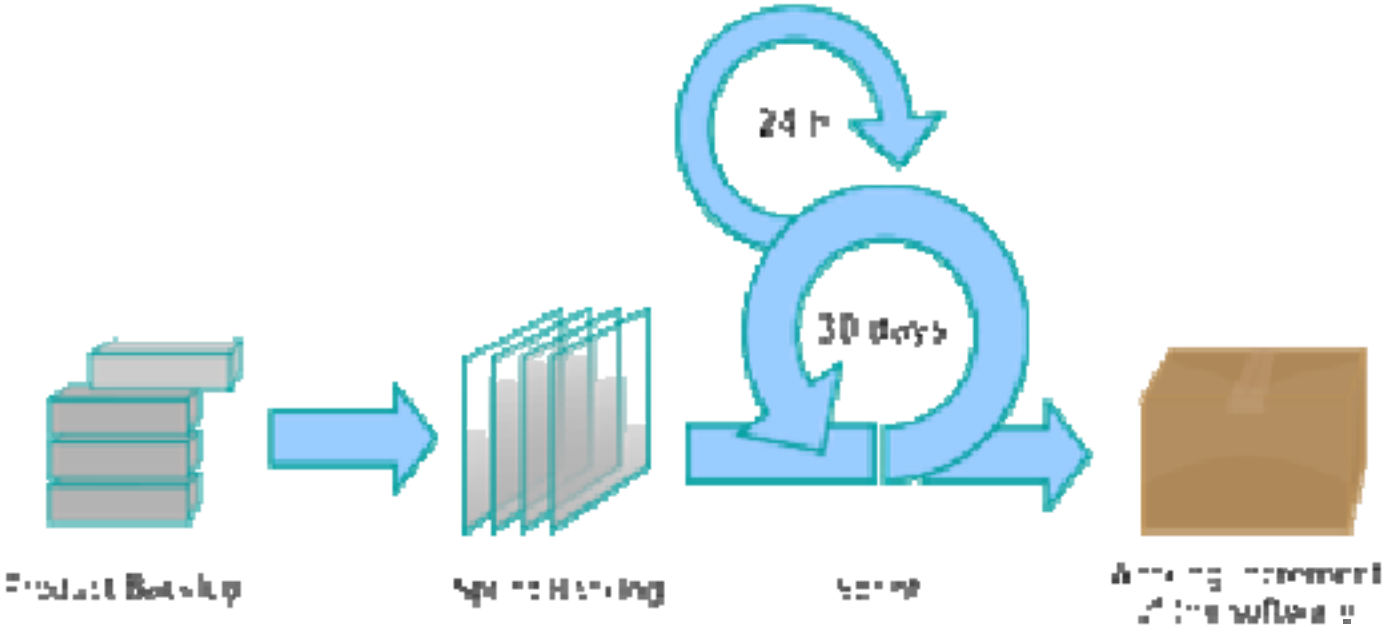
Agile Principles (2)

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.

Agile Principles (3)

- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The SCRUM Model*



* http://en.wikipedia.org/wiki/Scrum_(development)

Issues for Agile Development

- Scalability
  - scalability requires designing the product to partition it into coordinated “small projects”
  - but there is no design phase in agile development
- Skill of the software developers
  - architectural design is replaced with a “design metaphor”
- Ability of customer to express user needs
  - result is only as good as the requirements expressed by the customer
- Without good object-oriented practices, a functionally structured product may result
  - which can be difficult to modify
    - during development and during maintenance

The agile model is not for novices or amateurs

Best Kinds of Projects for Agile Development

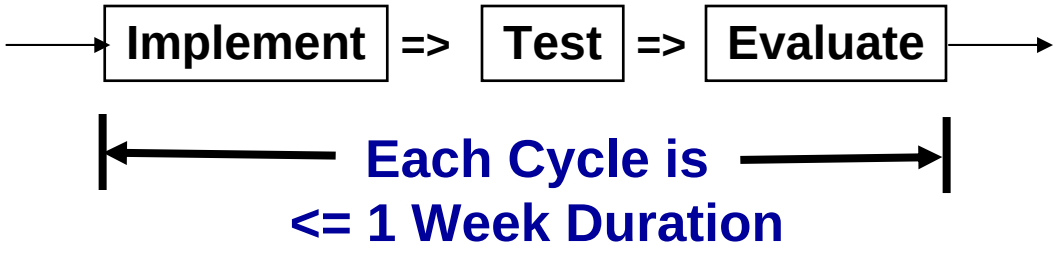
- Less than 10 software developers
- Well-defined business area
- Knowledgeable customer representative on-site
- Frequent delivery of incremental capabilities needed

The Role of Prototyping

- Prototyping involves building a “mock-up” of some part of a system
- Prototyping is typically used to
  - build a mock-up of the user interface
  - or, write a program to study a technical issue
- Prototyping is a technique, not a process model
  - prototyping does not eliminate the need for requirements analysis, design, test planning, integration, system testing, and documentation activities

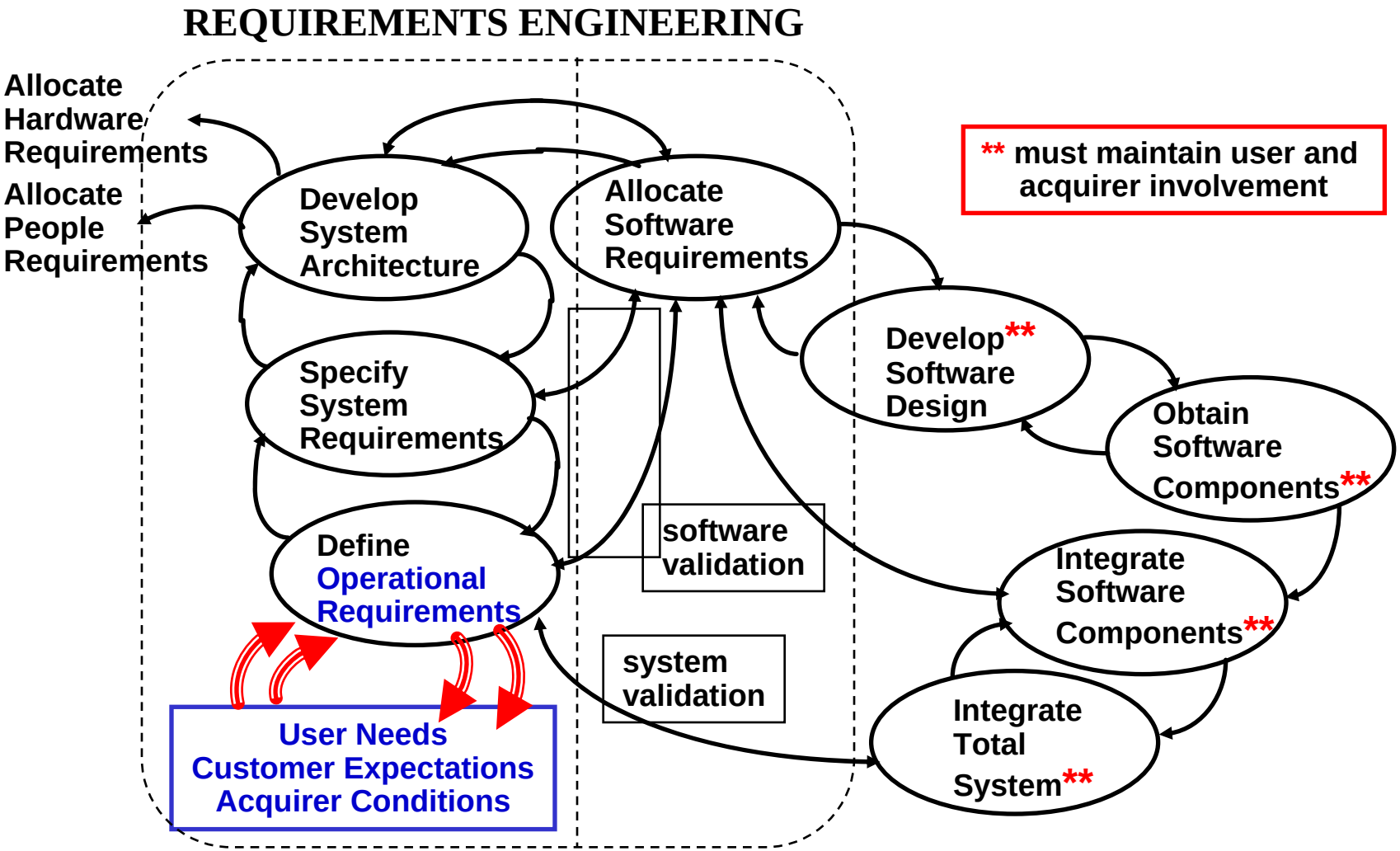
Prototyping vs Evolutionary Development

- The goal of prototyping is to gain knowledge and evolve requirements



- The goal of evolutionary development is to gain knowledge and evolve solutions

An Iterative Model of Prototyping to Elicit User Requirements



Managing Prototyping Efforts

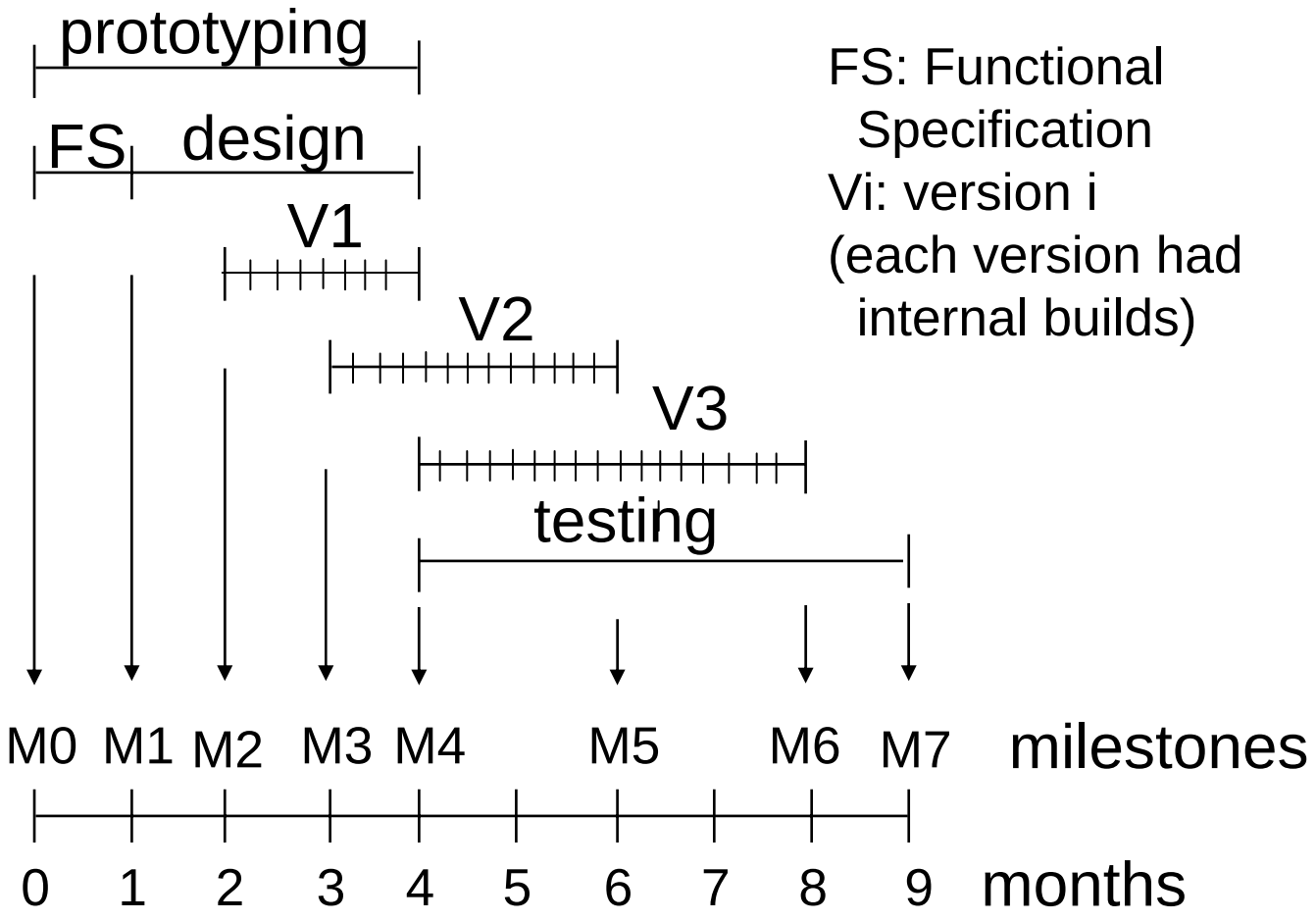
- Prototyping must not be used as an excuse for uncontrolled code-and-fix
  - specific (limited) objectives must be established for each prototyping iteration
  - the time frame of each iteration must be limited to one week or less
  - results must be evaluated and decisions made, just as in the evolutionary approach
- Evolutionary development follows a systematic process within a larger strategy
  - prototyping is a technique to study a specific problem within a limited context

Prototyping must be planned and controlled

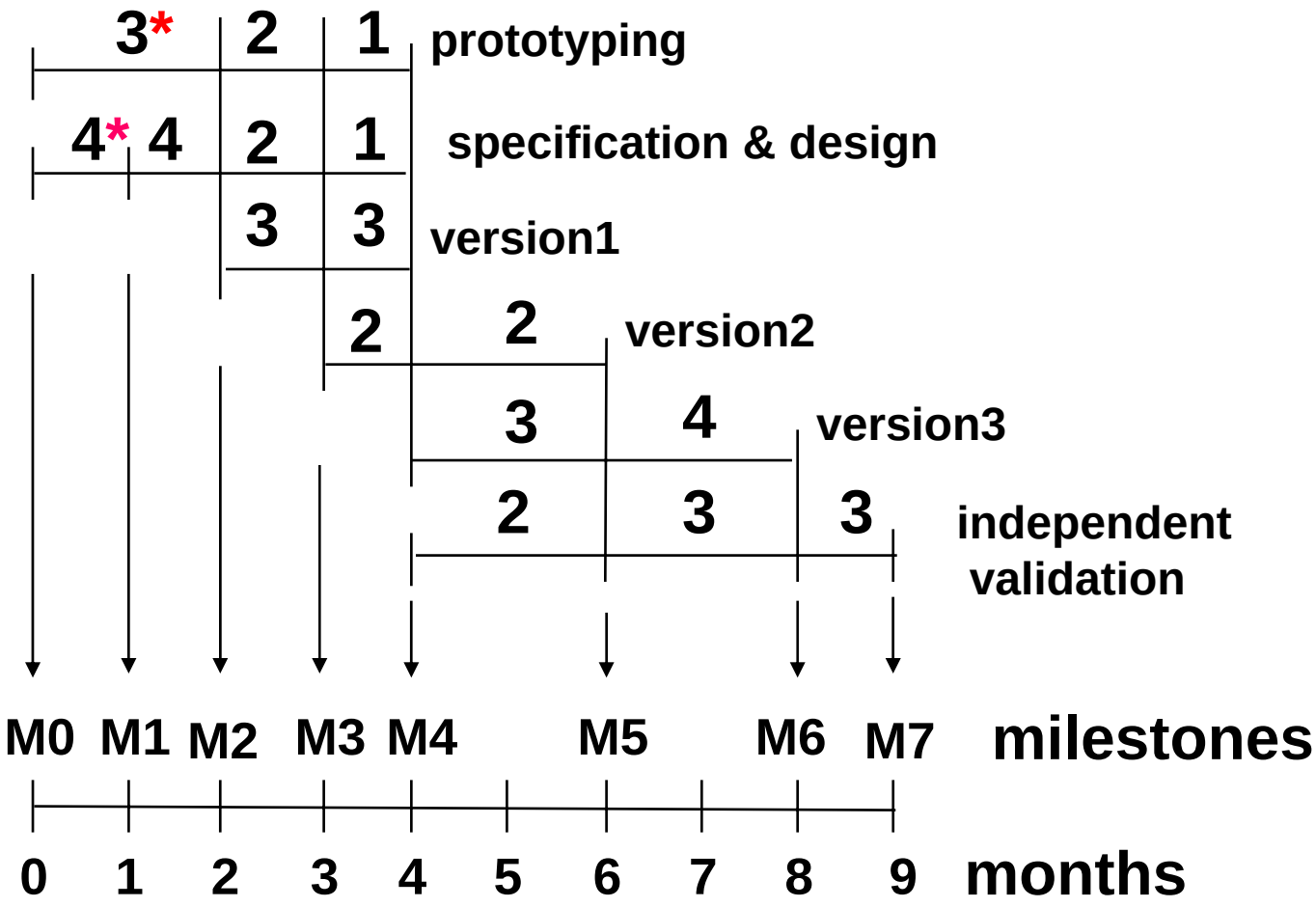
Some Distinctions

- When building a *prototype*, we keep the knowledge we have gained but
  - we do not use the code in the deliverable version of the system *unless we are willing to do additional work to develop production code*
- When using *evolutionary development*, we keep the knowledge we have gained in each cycle
  - we may, or may not, use the code we have written in the deliverable version of the system
- When using *agile or incremental development*, the goal is to keep the code we write in each build as part of the deliverable system

An Example of a Tailored Development Process



Staffing of the Tailored Process Model

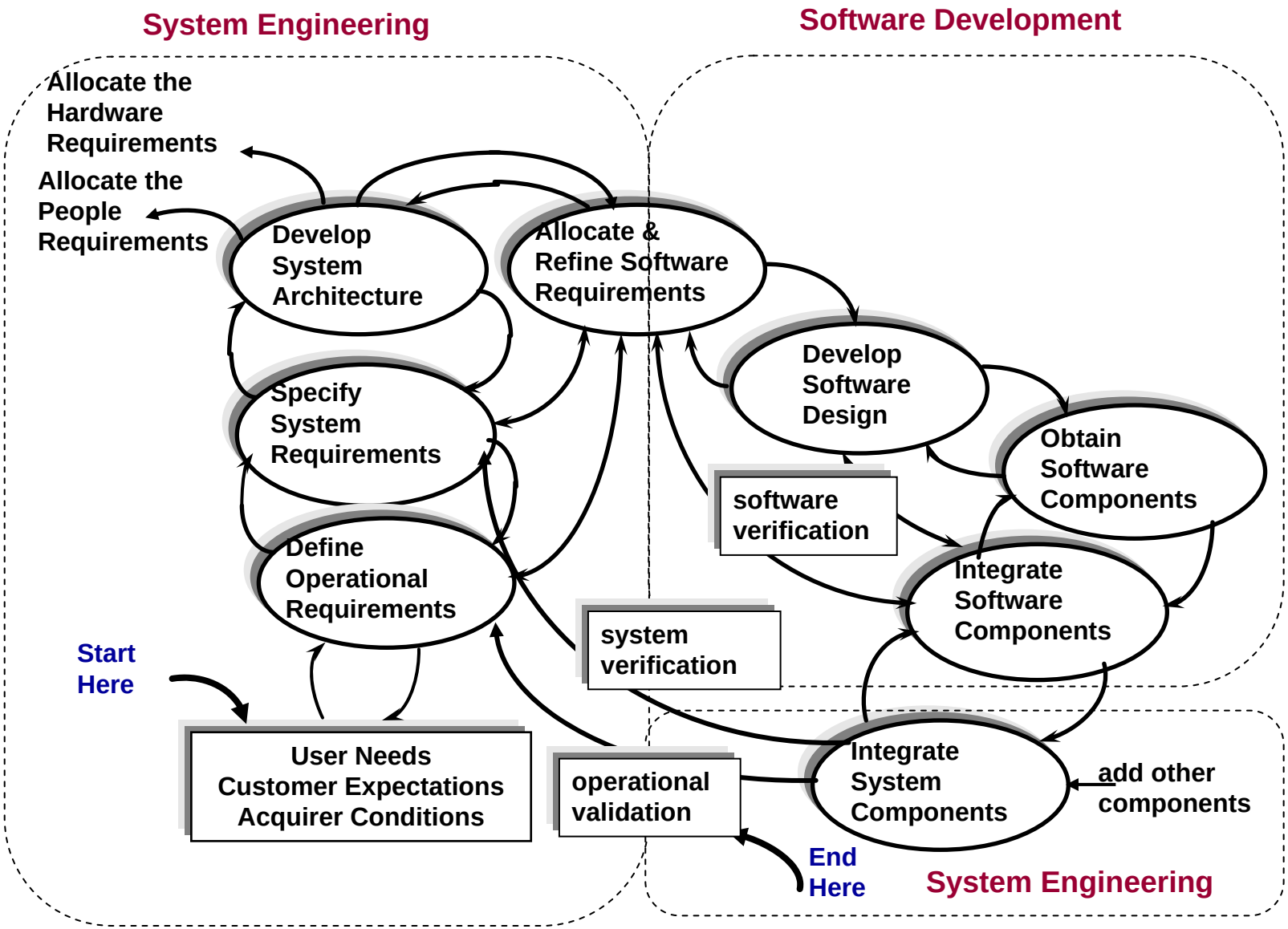


*number of people assigned:
7 staff members plus team leader

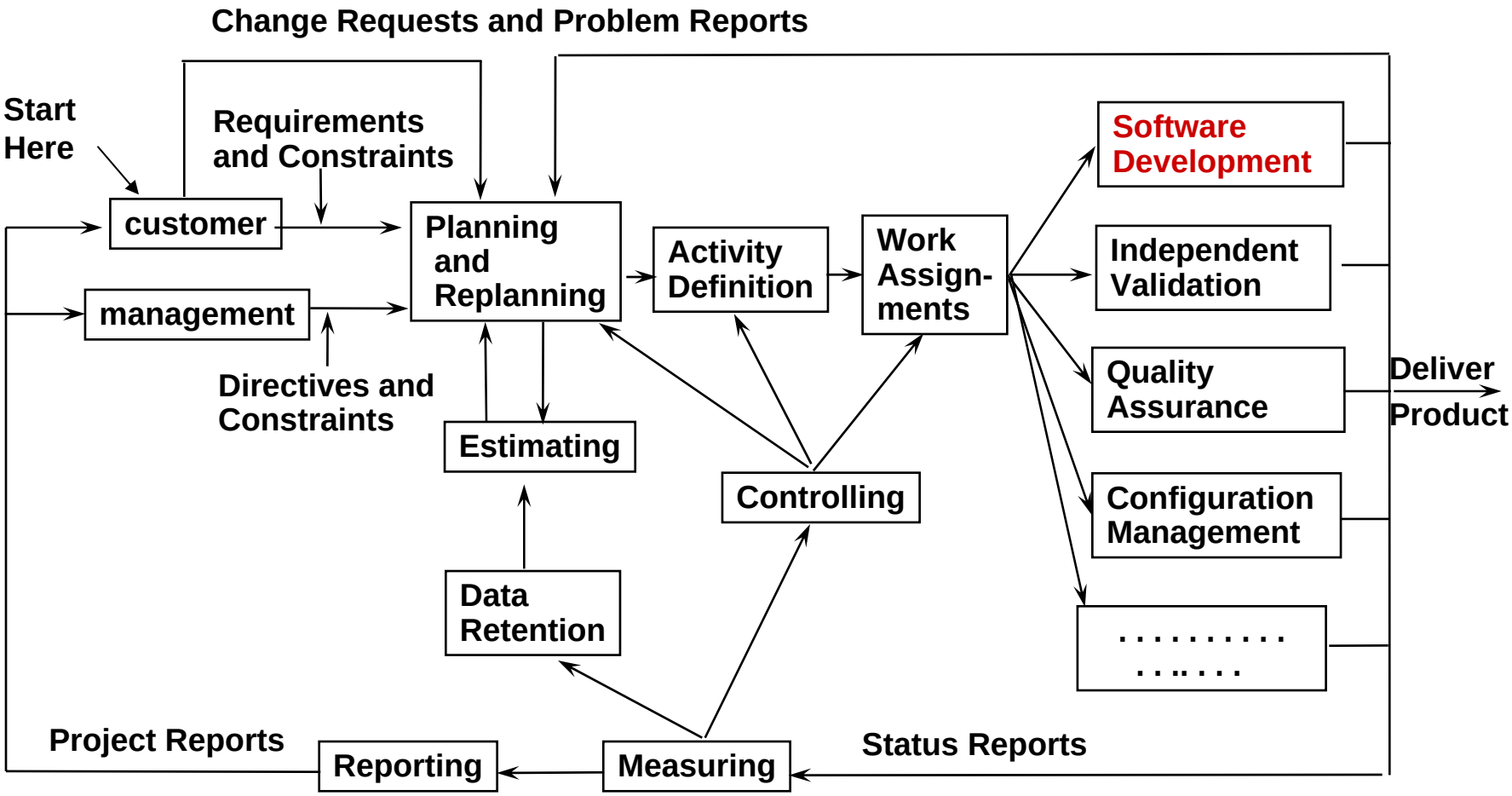
A Recommended Approach

- Use prototyping to understand technical issues and to specify the user interface
- Use an evolutionary approach to evolve solutions (as necessary)
- Use agile development when customer input on evolving requirements is important
- Use an incremental-build approach whenever possible
  - include frequent demonstrations and incremental version testing
- Embed software development in a system-level development model
- Embed the system development process in a workflow process model

A System-Level Engineering Model for Developing Software-Intensive Systems



A Workflow Model for Software Projects



Managing Iterative Development (1)

- Iterations must be pre-planned
- Iterations must be of limited duration
  - daily, weekly, or monthly
- Multiple work activities, and multiple types of work activities may be conducted concurrently
- Iterative, independent verification is required
- Customer and users must be involved in validation

Managing Iterative Development (2)

- An automated version control is essential for establishing and maintaining product baselines
  - A baseline is a work product that has been evaluated accepted by the involved parties
  - A baseline changed for one of two reasons:
    - in response to a requirements change
    - to fix a defect

A Caution

- In iterative development it is easy to lose sight of the process roadmap and the product vision
  - different people are engaged in different work activities
  - and there are typically many pieces and parts in various stages of development

- **the process roadmap and the product vision must be maintained**

> "when you're up to your neck [or whatever] in alligators it is easy to forget that your mission is to drain the swamp"

Maintaining the Vision

- The project manager maintains the vision of:
  - acceptable product, on schedule, within budget
- The software architect maintains the acceptable product vision
  - in coordination with the project manager

the project manager is the “producer”
the software architect is the “director”

The Main Points of Chapter 2 (1)

- A development-process framework is a generic process model that can be tailored and adapted to fit the needs of various projects.
- The development process for each software project must be designed with the same care used to design the product.
- Process design is best accomplished by tailoring and adapting well-known development process models and process frameworks, just as product design is best accomplished by tailoring and adapting well-known architectural styles and architectural frameworks.
- There are several well-known and widely used software development process models, including waterfall, incremental, evolutionary, agile, and spiral models.
- There are various ways to obtain the needed software components; different ways of obtaining software components require different mechanism of planning, measurement, and control.
- The development phases of a software project can be interleaved and iterated in various ways.

The Main Points of Chapter 2 (2)

- Iterative development processes provide the advantages of:
  - continuous integration,
  - iterative verification and validation of the evolving product,
  - frequent demonstrations of progress,
  - early detection of defects,
  - early warning of process problems,
  - systematic incorporation of the inevitable rework that occurs in software development, and
  - early delivery of subset capabilities (if desired).
- Depending on the iterative development process used, the duration of iterations range from 1 day to 1 month.
- Prototyping is a technique for gaining knowledge; it is not a development process.
- SEI, ISO, IEEE, and PMI, provide frameworks, standards, and guidelines relevant to software development process models (see Appendix 2A to Chapter 2)