Software Quality Assurance

Module 1

Software Engineering Practices

(Some things Testers should know about them)

Objectives

- Identify some common software development problems.
- Identify six software engineering practices for addressing common software development problems.
- Discuss how a software engineering process provides supporting context for software engineering practices.

Module 1 - Content Outline (Agenda)

➔Software development problems

- ◆ Six software engineering practices
- ◆ Supporting software engineering practices with process

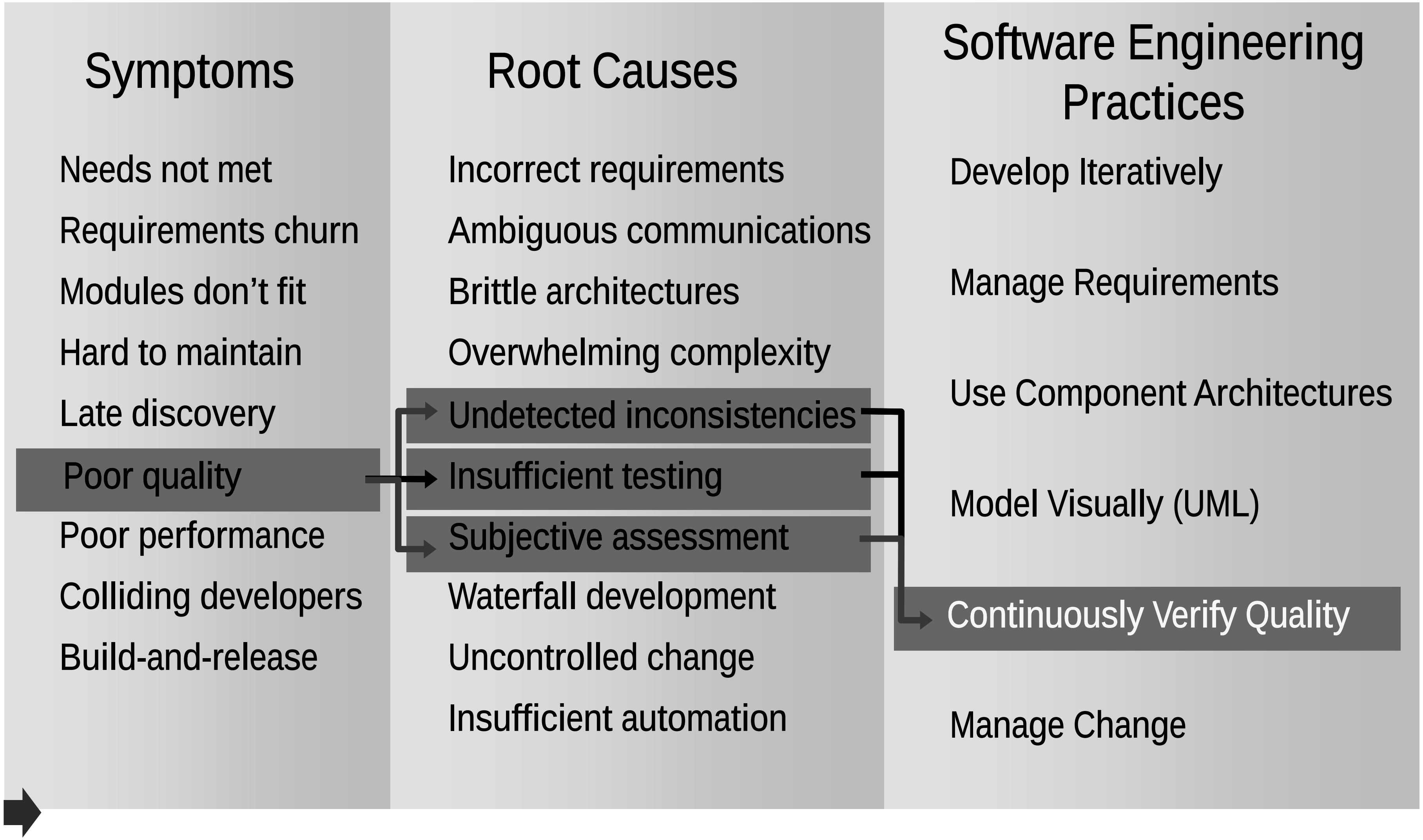
Symptoms of Software Development Problems

- **User or business needs not met**
- **Requirements churn**
- **Modules don’t integrate**
- **Hard to maintain**
- **Late discovery of flaws**
- **Poor quality or poor user experience**
- **Poor performance under load**
- **No coordinated team effort**
- **Build-and-release issues**

Discussion 1:

- Are there other software development problems?

Trace Symptoms to Root Causes



Symptoms

Needs not met

Requirements churn

Modules don’t fit

Hard to maintain

Late discovery

Poor quality

Poor performance

Colliding developers

Build-and-release

Root Causes

Incorrect requirements

Ambiguous communications

Brittle architectures

Overwhelming complexity

Undetected inconsistencies

Insufficient testing

Subjective assessment

Waterfall development

Uncontrolled change

Insufficient automation

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Module 1 - Content Outline (Agenda)

- Software development problems

➔Six software engineering practices

- Supporting software engineering practices with process

Six software engineering practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Practice 1: Develop Iteratively

Software Engineering Practices

Develop Iteratively

Manage Requirements

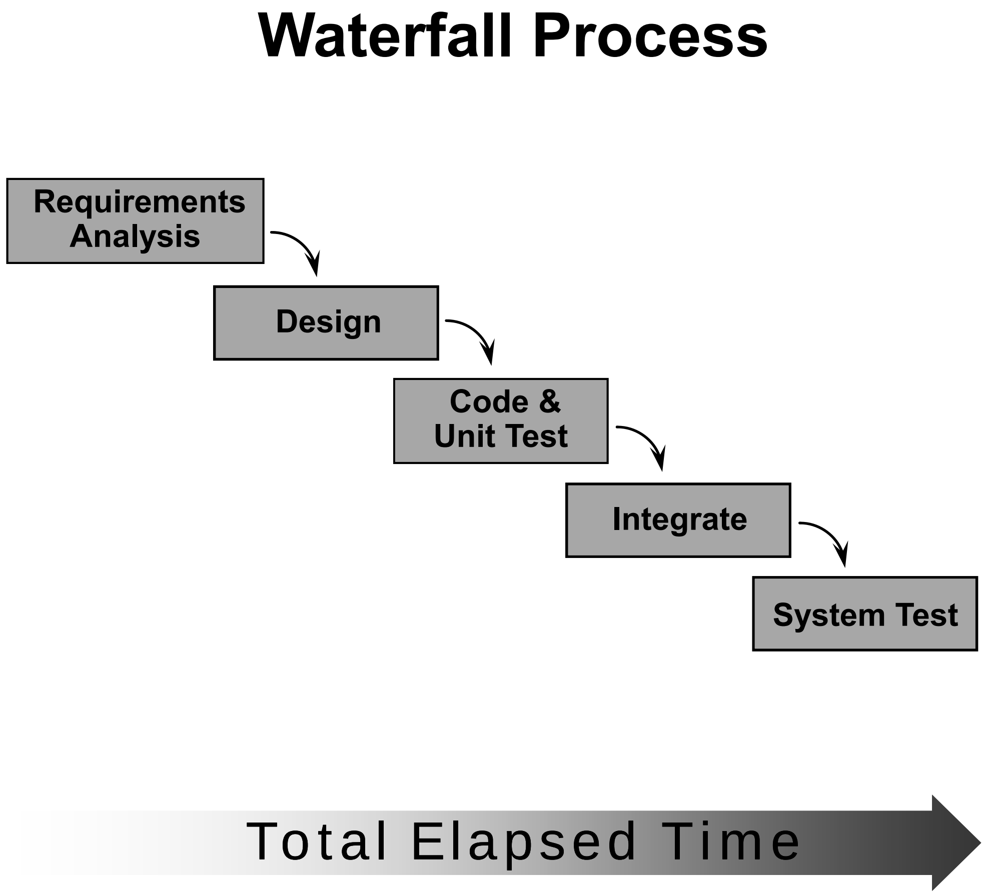
Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

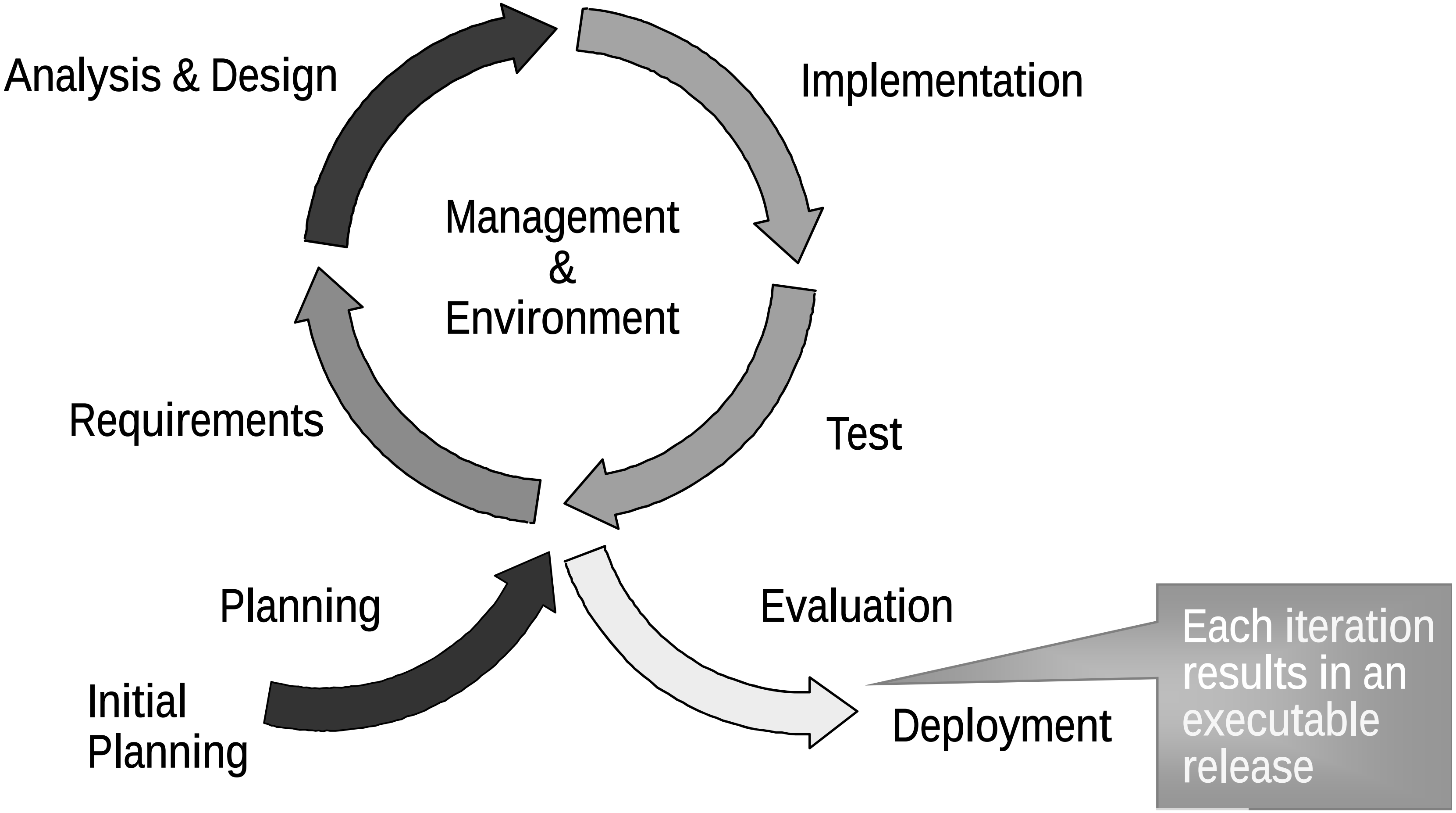
Manage Change

Waterfall Development Characteristics



- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned project extensions

Iterative Development Produces an Executable

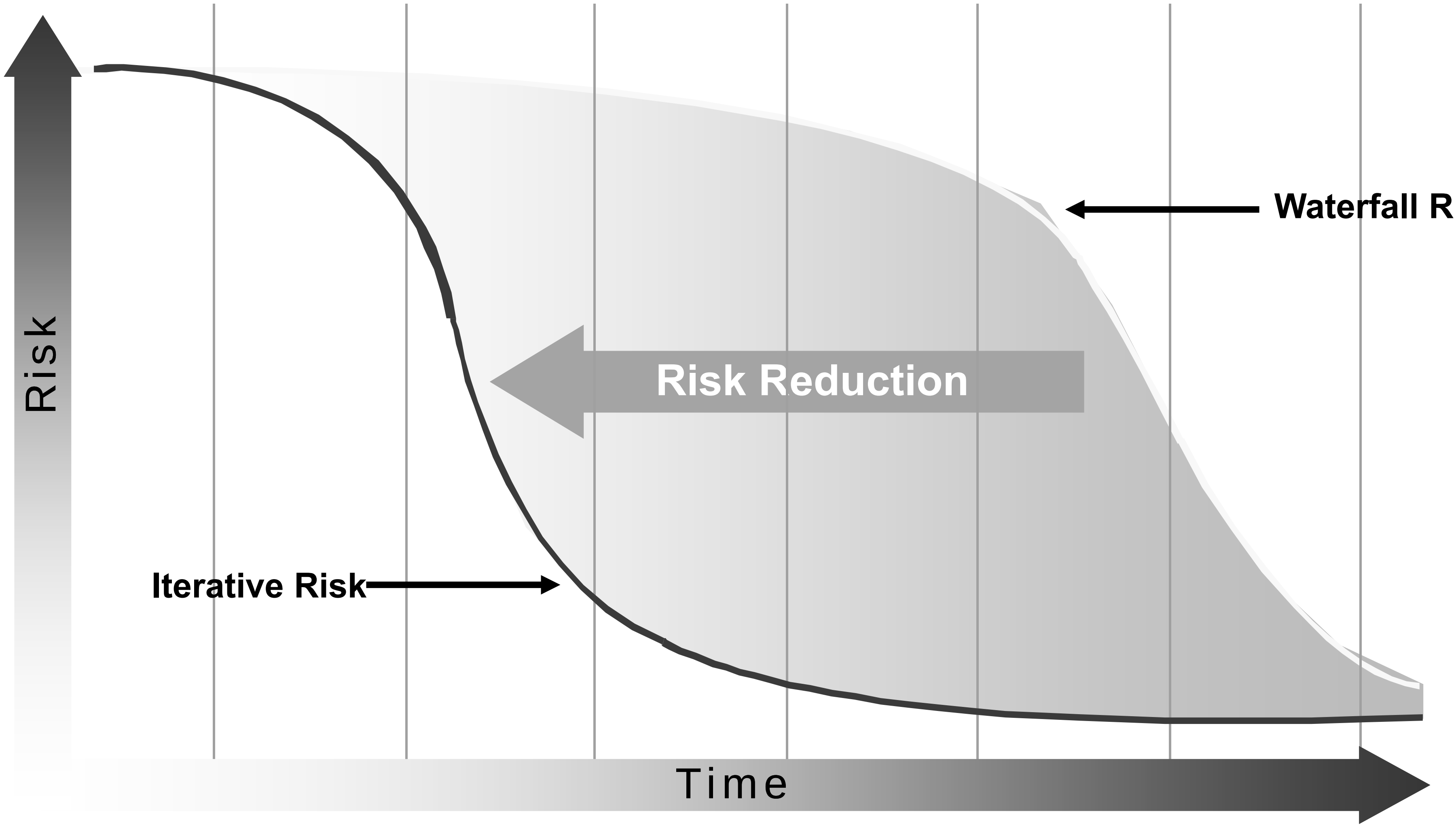


Each iteration results in an executable release

Iterative Development Produces an Executable

- ◆ The earliest iterations address greatest risks. Each iteration produces an executable release.
- ◆ Each iteration includes integration and test. Iterations help to:
  - •resolve major risks before making large investments
  - •enable early objective feedback
  - •make testing and integration continuous
  - •focus the project on achievable short-term objective milestones
  - •make it possible to deploy partial implementations of the completed final system

Risk Profiles



Iterative development drives risks out early.

Practice 2: Manage Requirements

Software Engineering Practices

Develop Iteratively

Manage Requirements

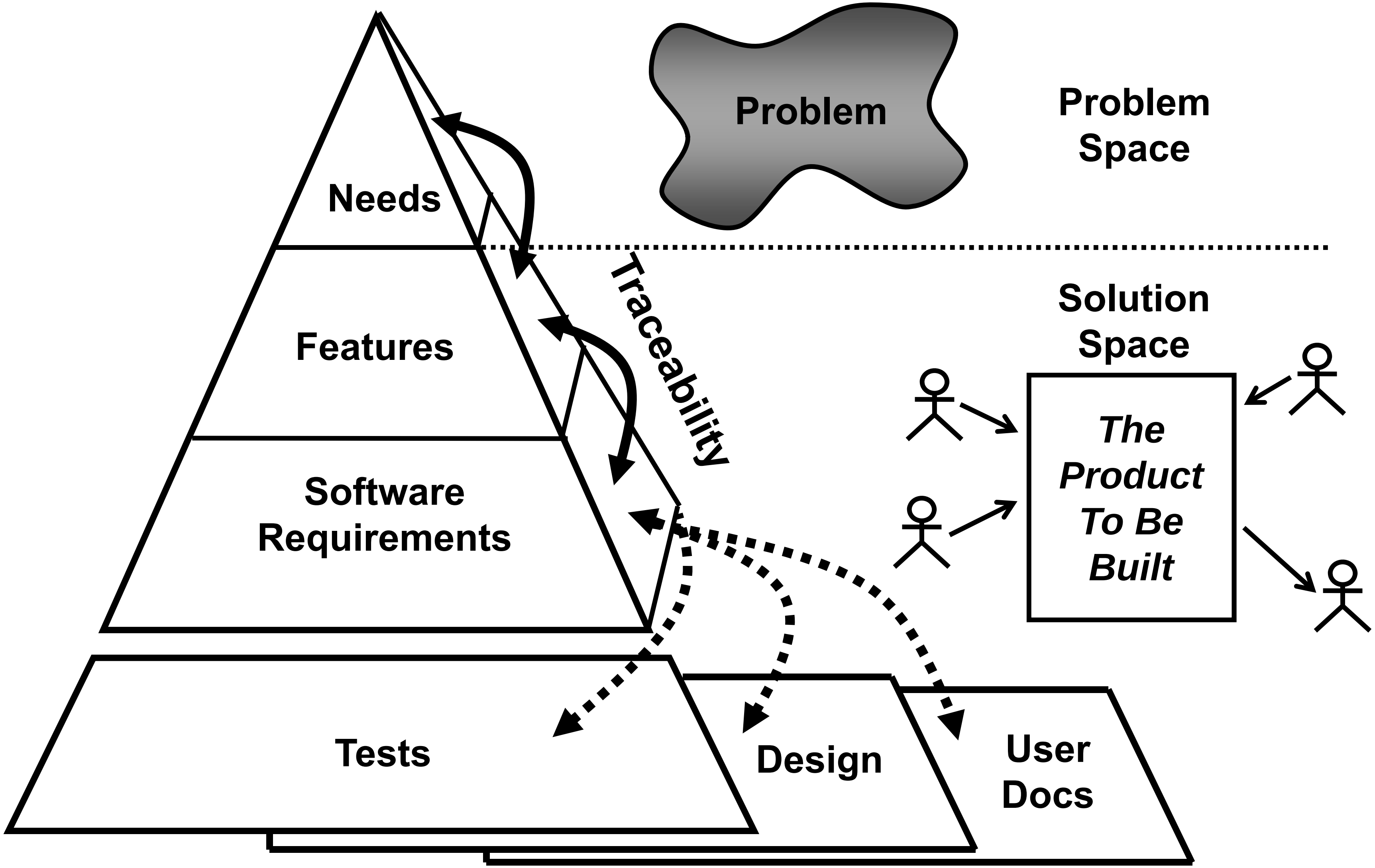
Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

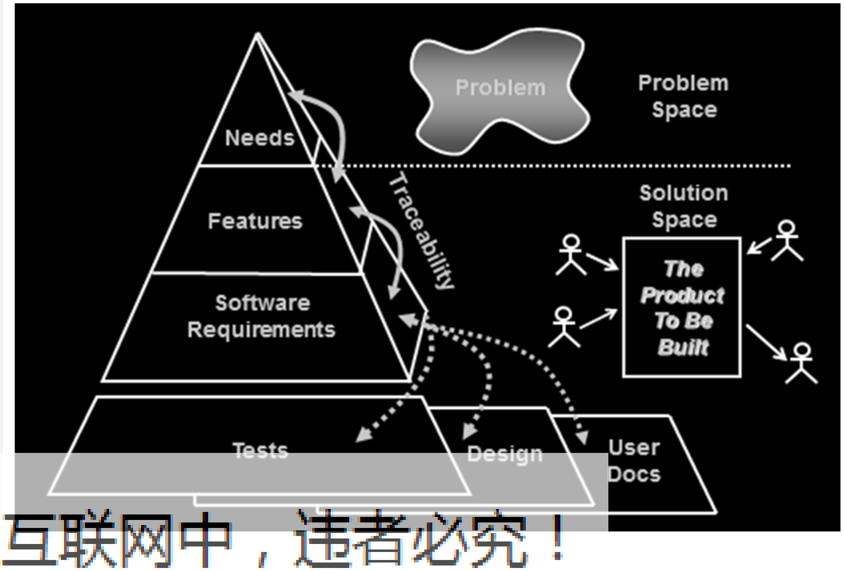
Manage Requirements - Map of the Territory



Manage Requirements - Map of the Territory

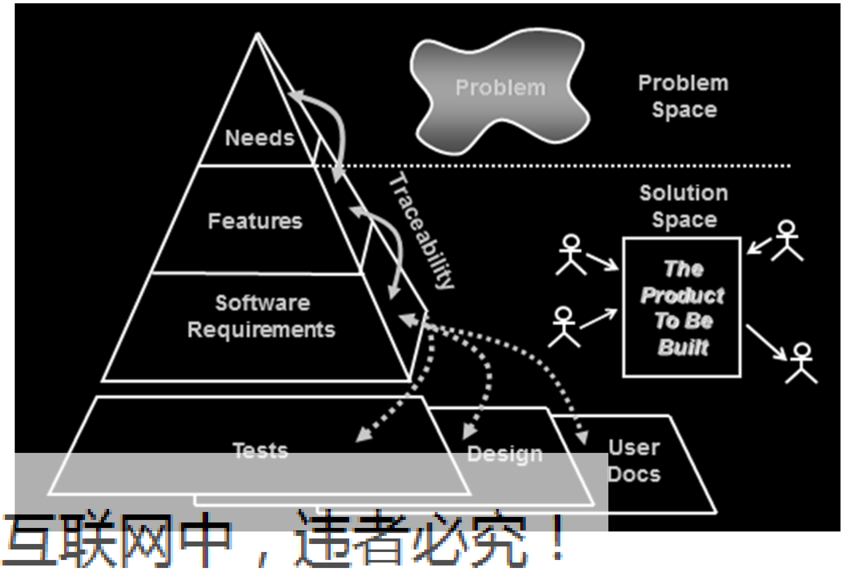
- Managing requirements involves the translation of stakeholder requests into a set of key stakeholder needs and system features.
- These in turn are detailed into specifications for functional and non-functional requirements.
- Detailed specifications are translated into a design, user documentation and tests.

- The requirements for the software are a key input to testing.
- You will often find important problems at the boundary between each section of the pyramid – for example,
  - are the needs appropriately reflected in the features?
  - Does the design appropriately reflect the requirements?

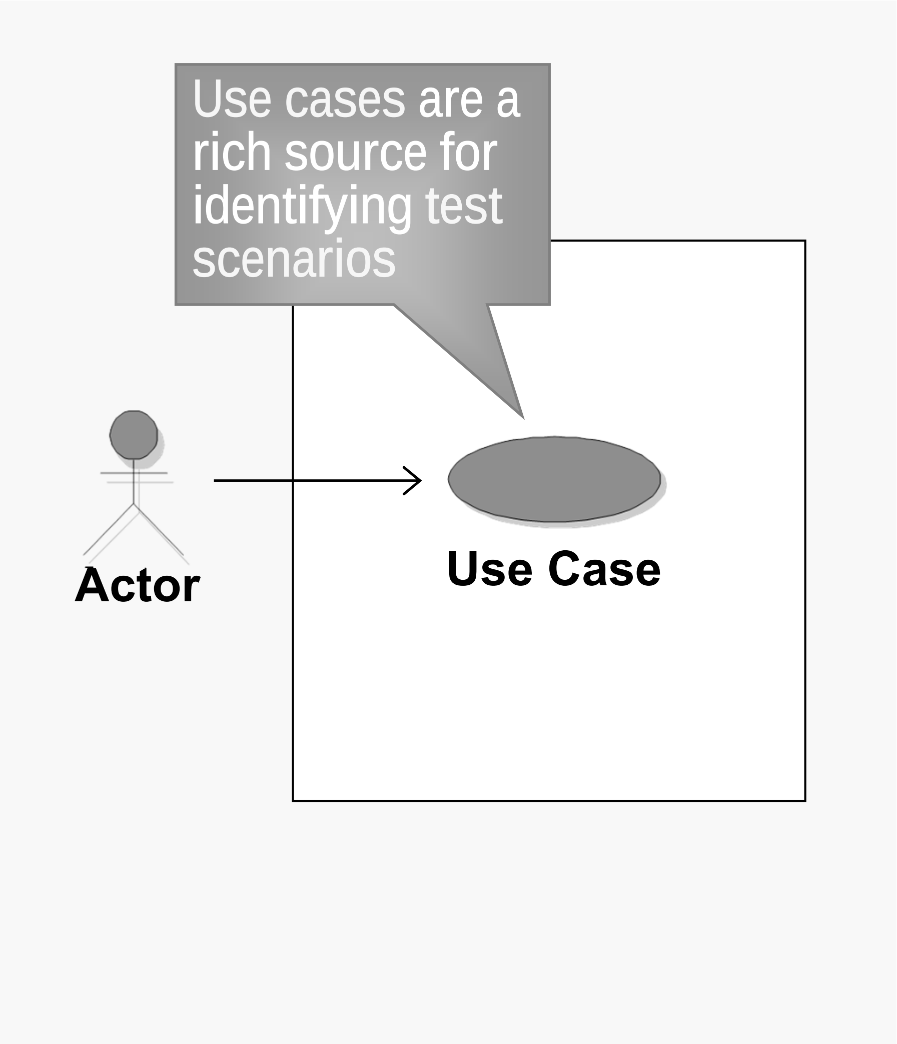


Manage Requirements - Map of the Territory

- To help manage the relationship between the requirements and the tests derived from those requirements, you can establish traceability relationships between those elements.
- Traceability assists us to do many things, including:
  - Assess the project impact of a change in a requirement
  - Assess the impact of a failure of a test on requirements (i.e., if test fails, the requirement may not be satisfied)
  - Verify that the application does only what it was intended to do
  - Verify that all requirements of the system are fulfilled by the implementation
  - Manage the scope of the project
  - Manage change



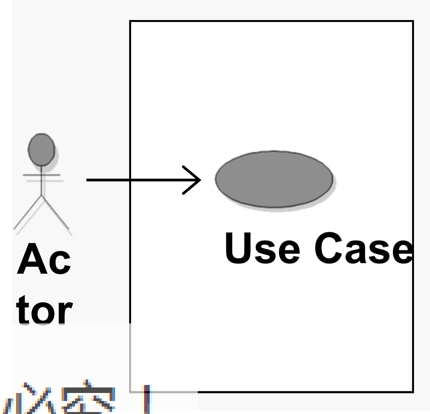
Manage Requirements - Use-Case Concepts



- Use Cases represent a technique for defining requirements in a way that focuses on the end-user goal.
- They have been popularized by iterative development processes such as the Rational Unified Process.
- However, the technique is not specific to iterative development – it can be applied just as well to eliciting and managing requirements in a waterfall development lifecycle.

Manage Requirements - Use-Case Concepts

- An **actor** represents a person or another system that interacts with the system.
  - is not part of the system. It represents a role that users of the system will play when interacting with it.
  - can actively interchange information with the system.
  - can be a passive recipient of information.
  - can be a giver of information.
  - can represent a human, a machine or another system.
- A **use case** defines a sequence of actions a system performs that yields a result of observable value to an actor.
  - specifies a dialogue between an actor and the system.
  - is initiated by an actor to invoke certain functionality in the system.
  - is a collection of meaningful, related flows of events.
  - yields a result of observable value.



Practice 3: Use Component Architectures

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

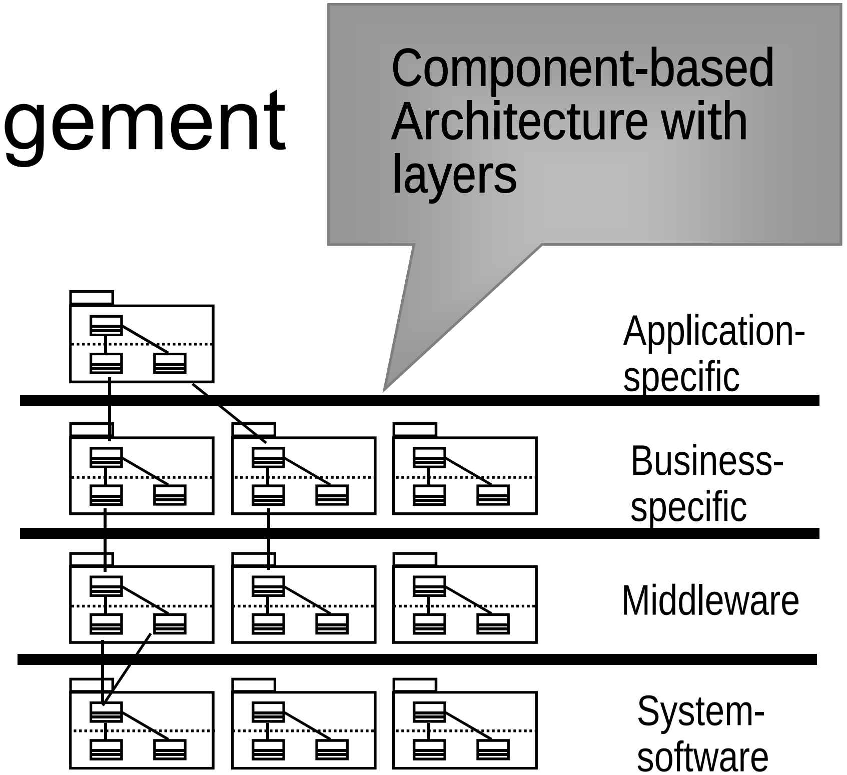
Manage Change

Resilient Component-Based Architectures

- Resilient
  - Meets current and future requirements
  - Improves extensibility
  - Enables reuse
  - Encapsulates system dependencies
- Component-based
  - Reuse or customize components
  - Select from commercially available components
  - Evolve existing software incrementally

Purpose of a Component-Based Architecture

- Basis for reuse
  - Component reuse
  - Architecture reuse
- Basis for project management
  - Planning
  - Staffing
  - Delivery
- Intellectual control
  - Manage complexity
  - Maintain integrity



Component-based Architecture with layers

Practice 4: Model Visually (UML)

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

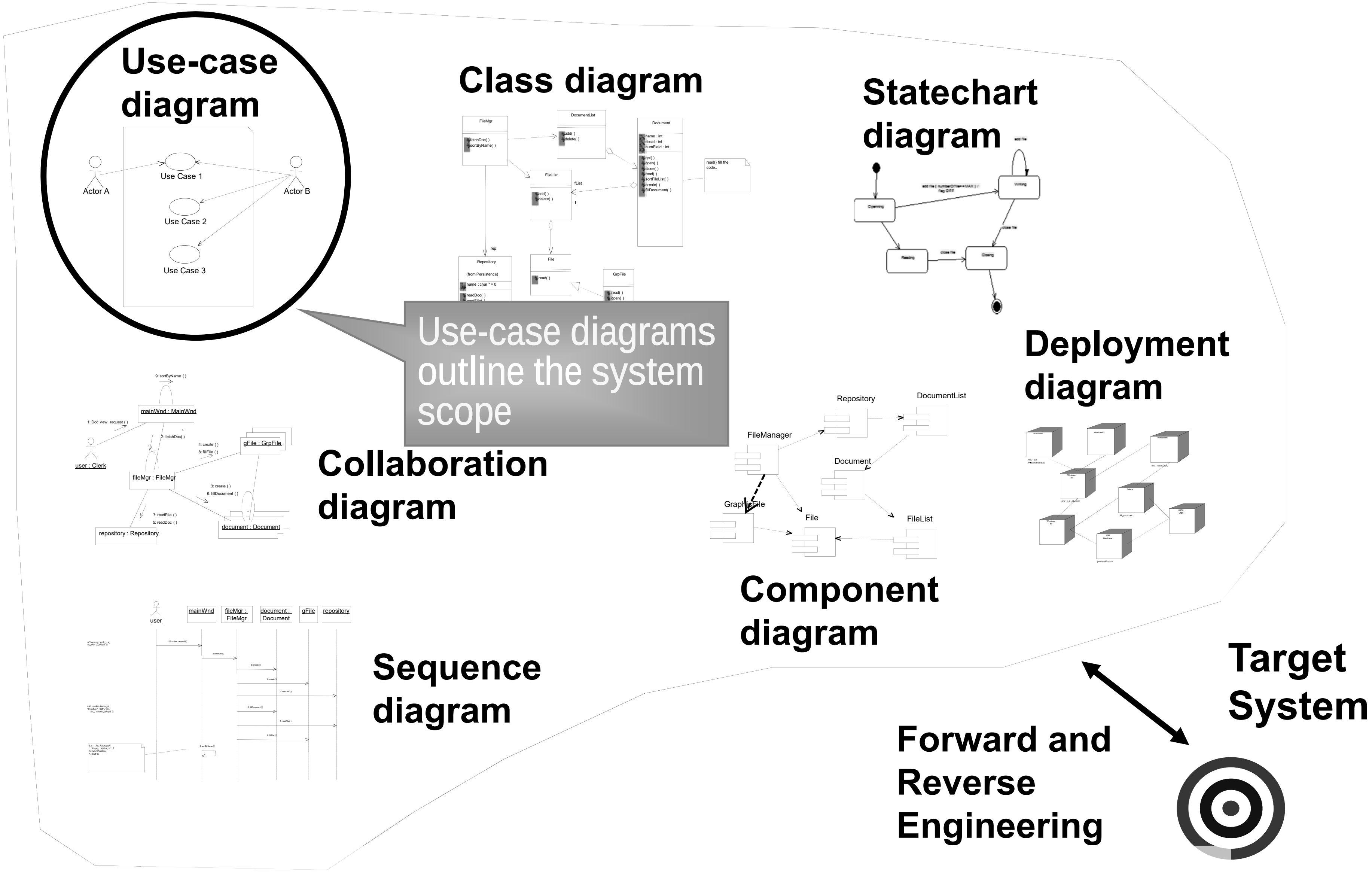
Continuously Verify Quality

Manage Change

Why Model Visually?

- To help manage complexity
    - To capture both structure and behavior
    - To show how system elements fit together
    - To hide or expose details as appropriate
- To keep design and implementation consistent
- To promote unambiguous communication
    - UML provides one language for all practitioners

Visual Modeling Using UML Diagrams



Use-case diagram

Class diagram

Statechart diagram

Use-case diagrams outline the system scope

Deployment diagram

Collaboration diagram

Component diagram

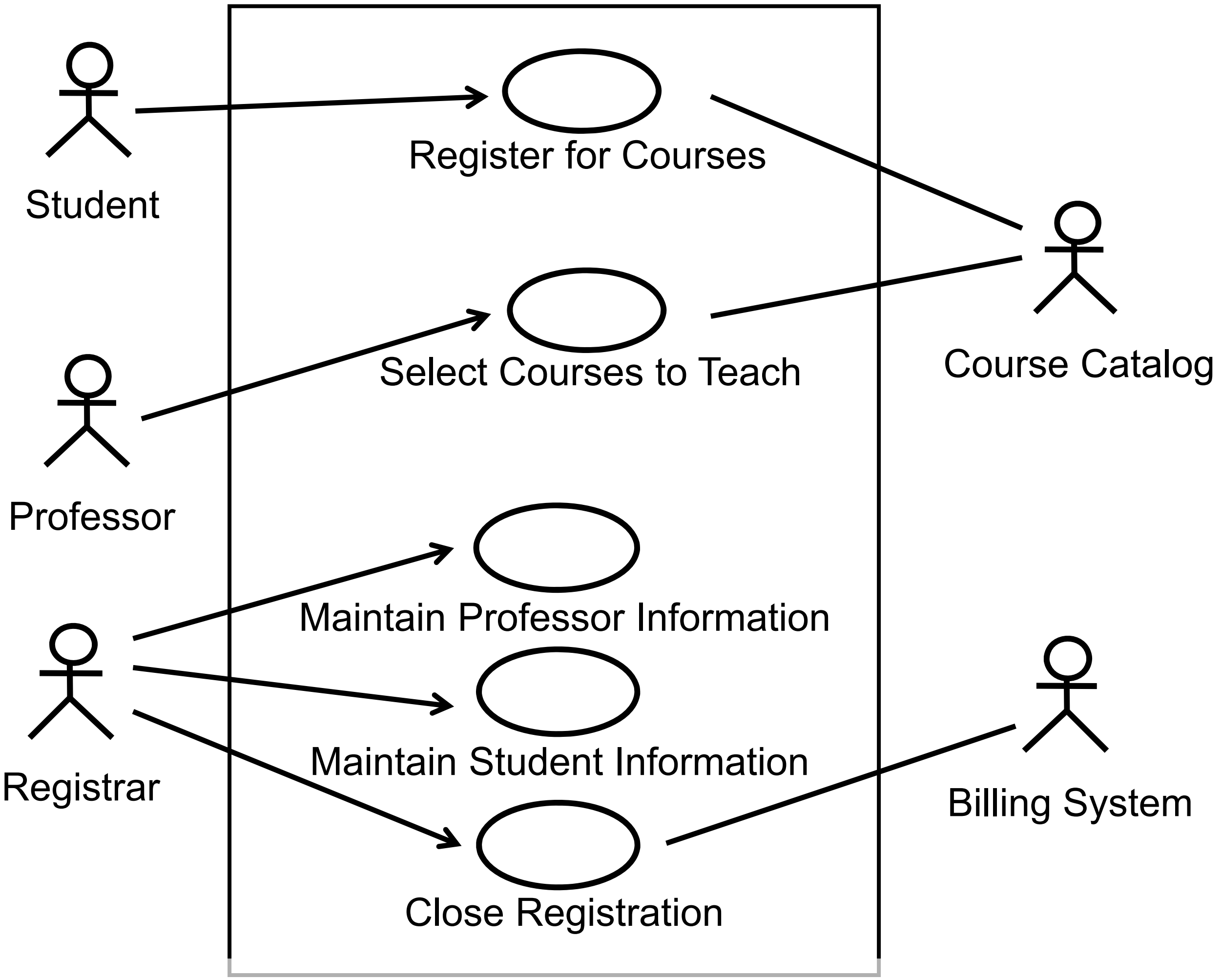
Sequence diagram

Target System

Forward and Reverse Engineering

Workbook Page: A Sample UML Diagram – Use Cases

A University Course Registration System



Practice 5: Continuously Verify Quality

Software Engineering Practices

Develop Iteratively

Manage Requirements

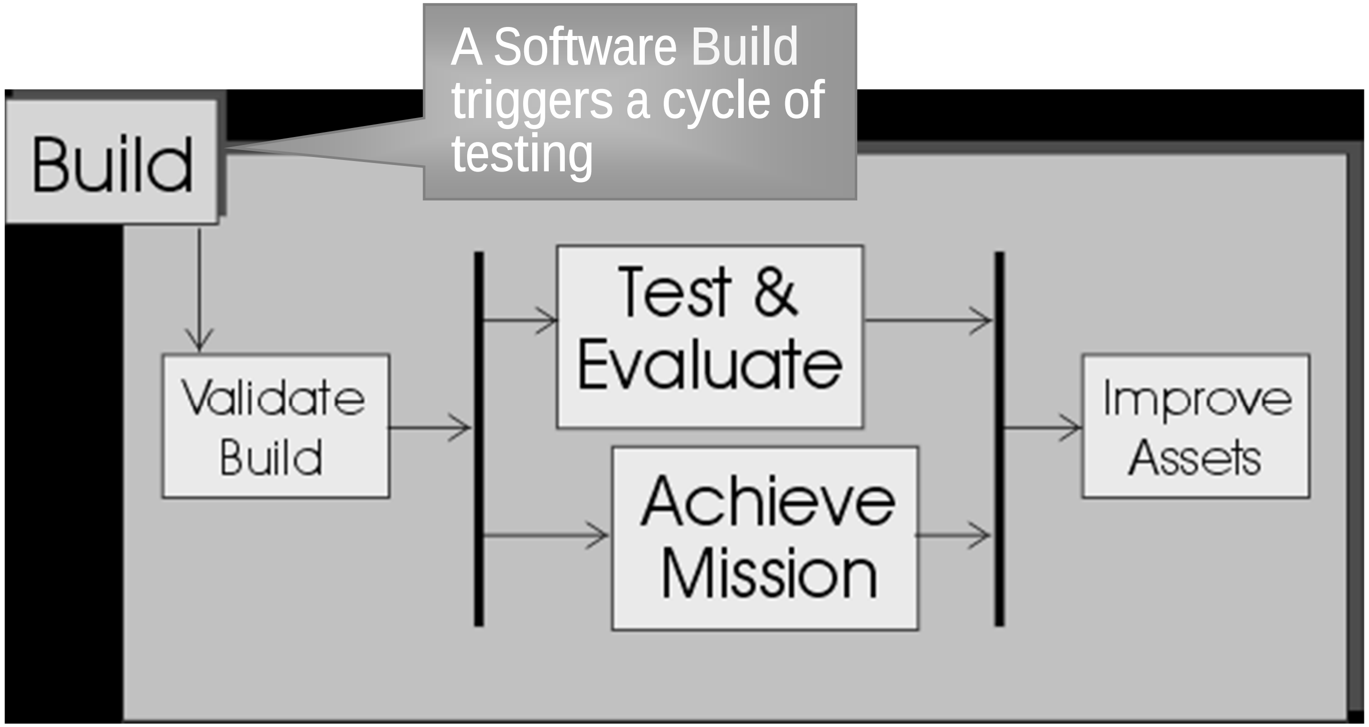
Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

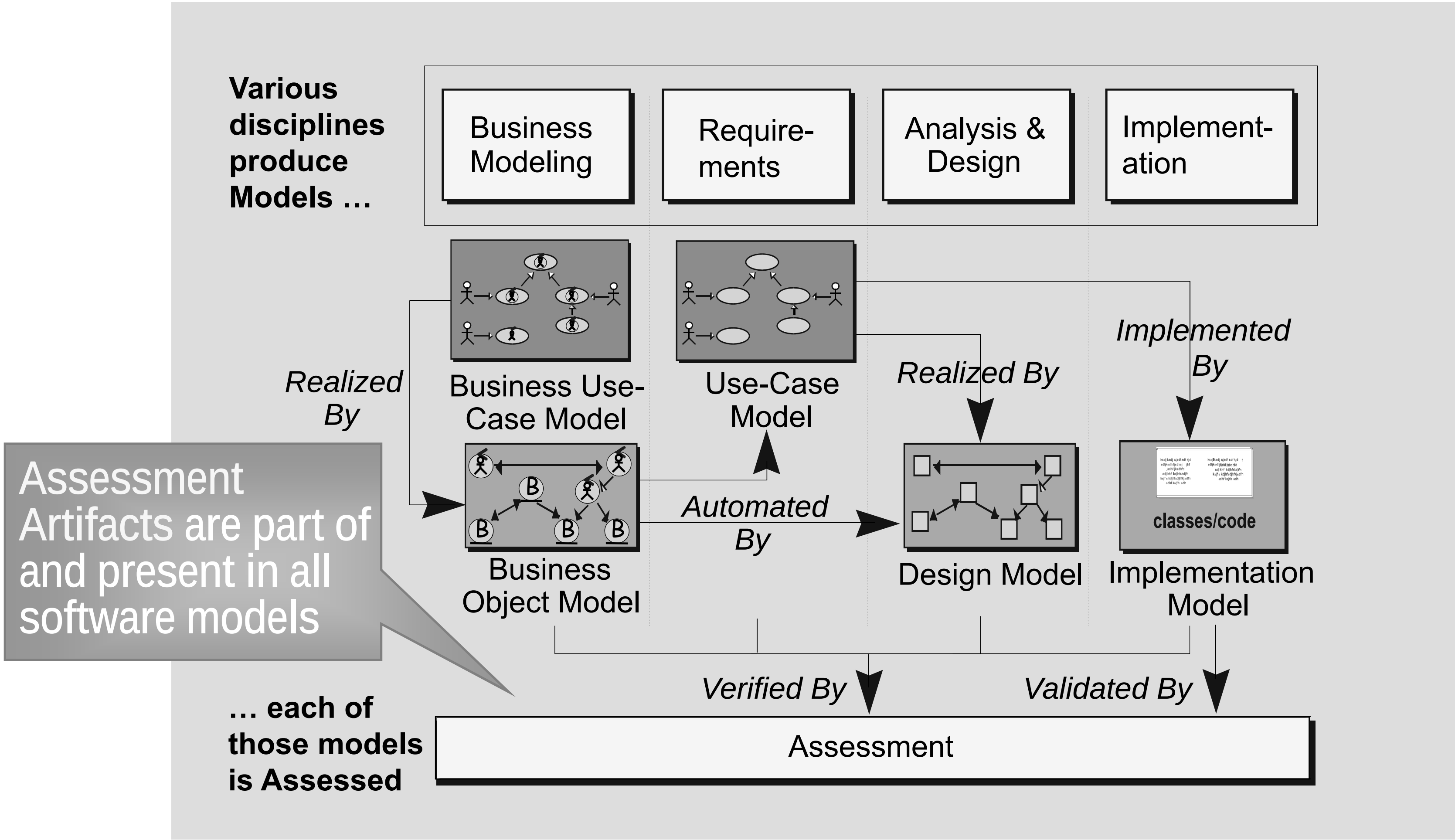
Manage Change

Continuously Verify Quality – in each Iteration



A Software Build triggers a cycle of testing

Continuously Verify Quality – Software Models



Various disciplines produce Models …

Business Modeling

Require-ments

Analysis & Design

Implement-ation

Business Use-Case Model

Use-Case Model

Realized By

Realized By

Implemented By

Automated By

Business Object Model

Design Model

classes/code

Implementation Model

Assessment Artifacts are part of and present in all software models

Verified By

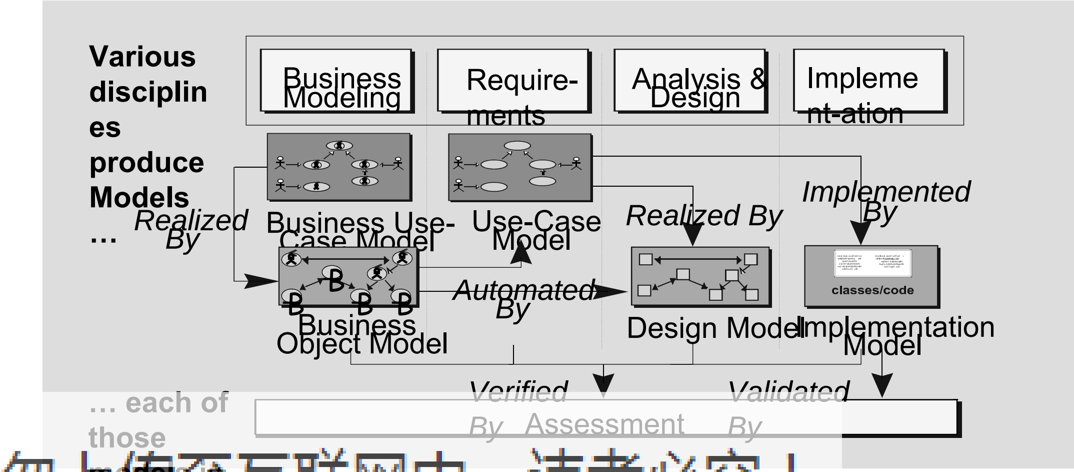
Validated By

… each of those models is Assessed

Assessment

Continuously Verify Quality – Software Models

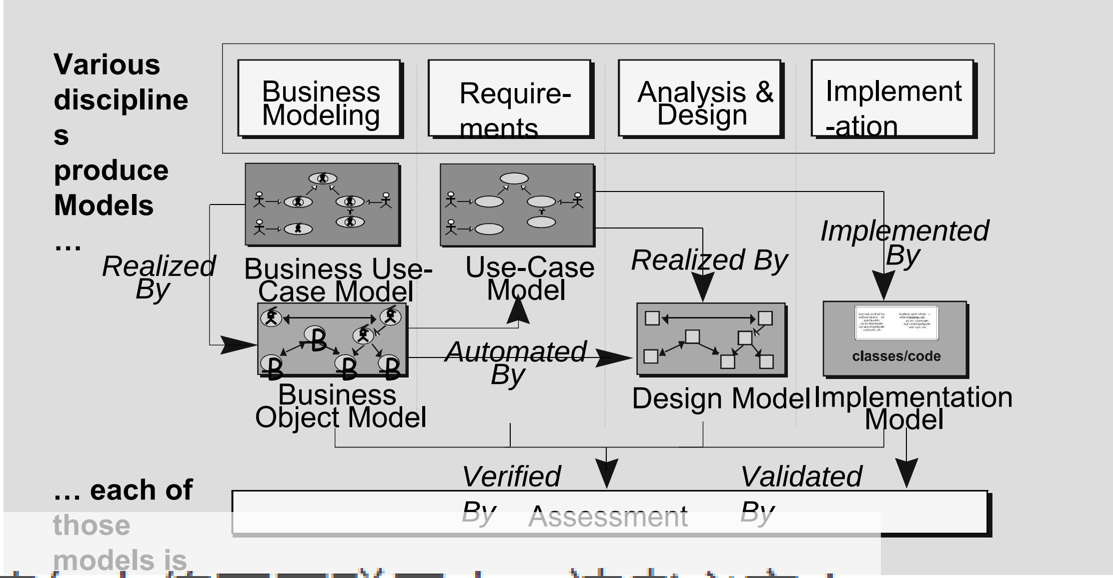
- The **Business** Model is a model of what the business processes are and of the business environment. It is primarily used to gain a better understanding of the software requirements in the business context.
- The Use-Case Model is a model of the value the system represents to the external users of the system environment. It describes the “external services” that the system provides.
- The Design Model is a model that describes how the software will “realize” the services described in the use cases. It serves as a conceptual model (or abstraction) of the implementation model and its source code.
- The Implementation Model represents the physical software elements and the implementation subsystems that contain them.


Various disciplines produce Models …

… each of those models is

Continuously Verify Quality – Software Models

- Assessment involves both Verification and Validation activities:
  - verifying that the software product is being built right
  - validating that the right software product is being built.
- This distinction refers to assessing both the appropriateness of the process by which the software product is built (verification) and the appropriateness of the resulting software product that will be delivered to the customer (validation).



Various disciplines produce Models …

Business Modeling

Require-ments

Analysis & Design

Implement-ation

Business Use-Case Model

Use-Case Model

Realized By

Realized By

Implemented By

Business Object Model

Automated By

Design Model

classes/code

Implementation Model

… each of those models is

Verified By

Assessment

Validated By

Practice 6: Manage Change

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

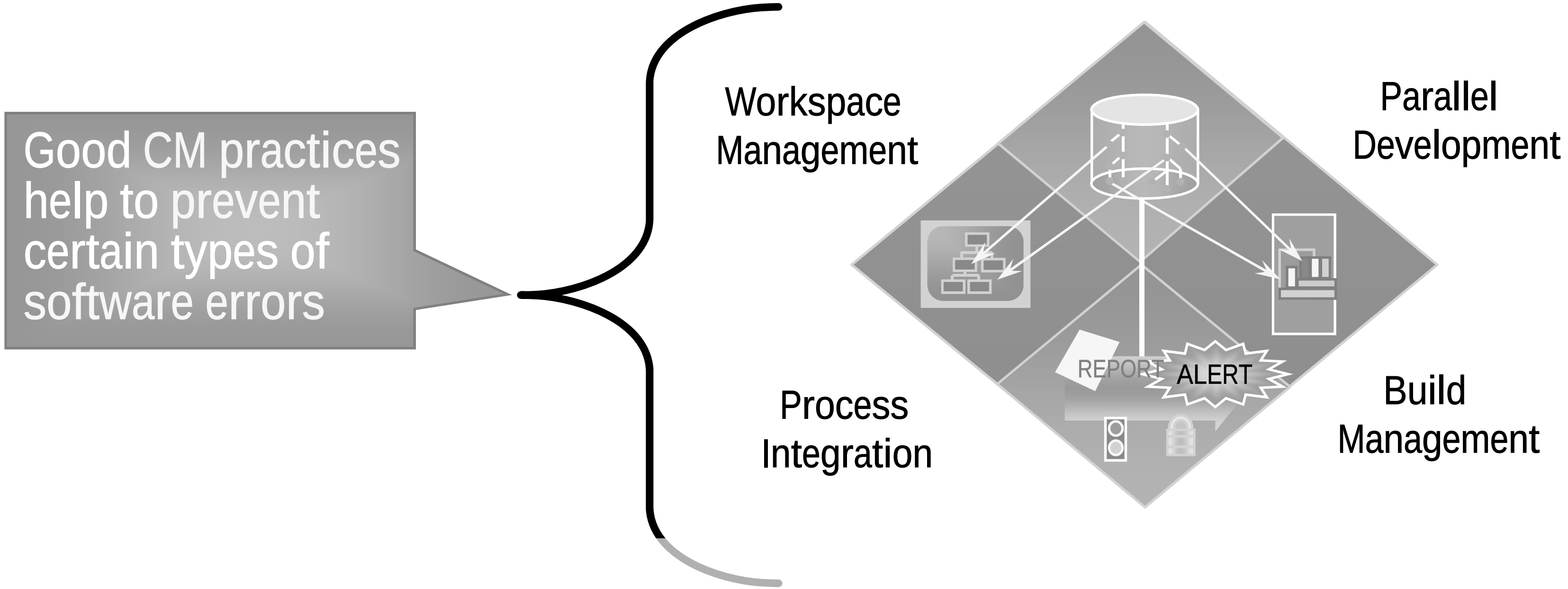
Manage Change

What Do You Want to Control?

- Establishing secure workspaces for each worker on the project provides isolation from changes made in other workspaces and control of all software artifacts -- models, code, docs, tests etc.
- A key challenge to developing software-intensive systems is the need to cope with multiple workers, organized into different teams, possibly at different sites, all working together on multiple iterations, releases, products, and platforms. In the absence of disciplined control, the development process rapidly degrades into chaos. Progress can come to a stop.
- Three common problems that result are:
  - Simultaneous update -- When two or more workers separately modify the same artifact, the last one to make changes destroys the work of the former.
  - Limited notification -- When a problem is fixed in shared artifacts, some of the workers are not notified of the change.
  - Multiple versions -- It is feasible to have multiple versions of an artifact in different stages of development at the same time.

What Do You Want to Control?

- Changes to enable iterative development
  - Secure workspaces for each worker
  - Parallel development possible
- Automated integration/build management



Good CM practices help to prevent certain types of software errors

Workspace Management

Parallel Development

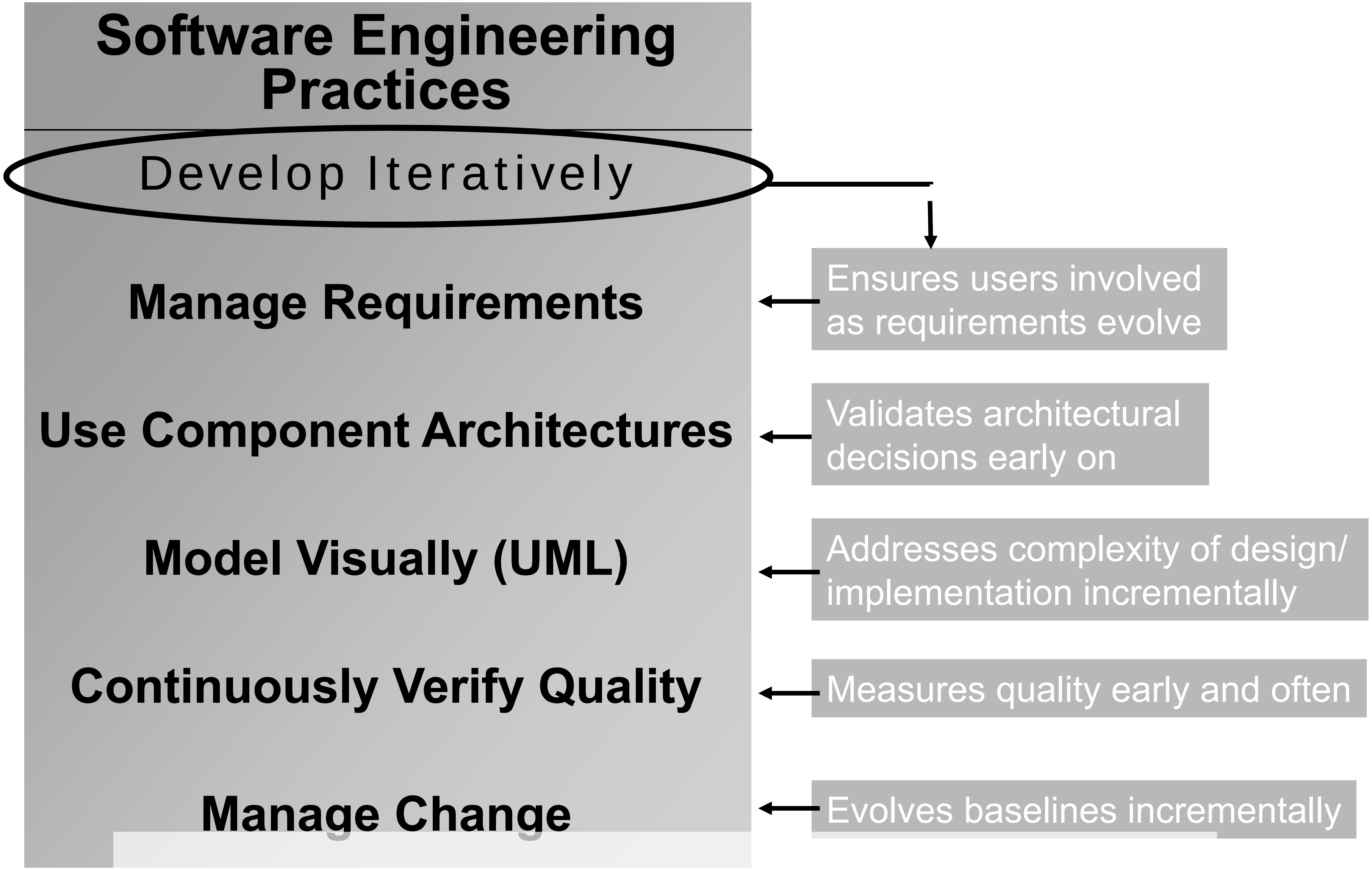
Process Integration

Build Management

Workbook Page: Aspects of a CM System

- Change Request Management (CRM)
- Configuration Status Reporting
- Configuration Management (CM)
- Change Tracking
- Version Selection
- Software Manufacture

Software Engineering Practices Reinforce Each Other



Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Ensures users involved as requirements evolve

Validates architectural decisions early on

Addresses complexity of design/ implementation incrementally

Measures quality early and often

Evolves baselines incrementally

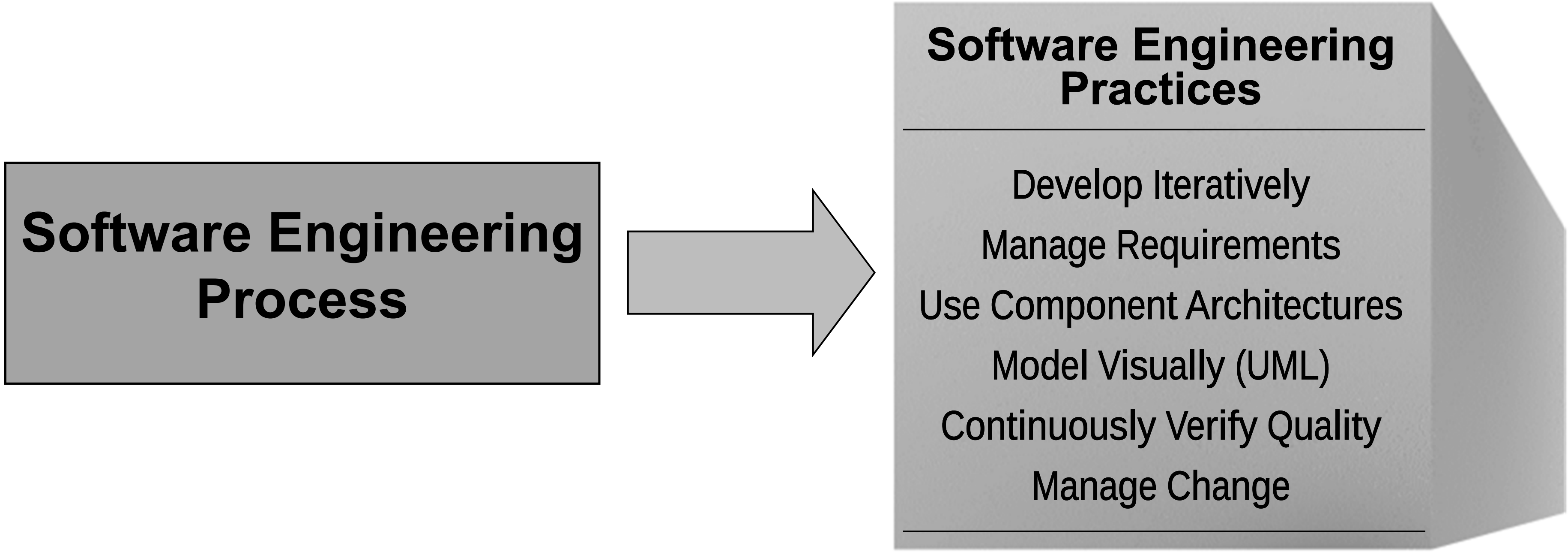
Discussion 2:

- Are there other software engineering practices?

Module 1 - Content Outline (Agenda)

- ◆ Software development problems
- ◆ Six software engineering practices
- ➔Software engineering process and software engineering practices

An Engineering Process Implements Engineering Practices



Software Engineering Process

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Why have a process?

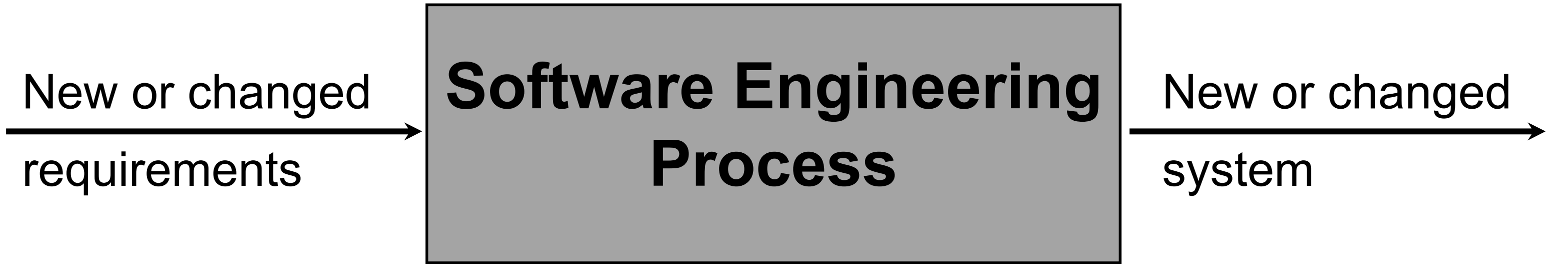
- Provides guidelines for efficient development of quality software
- Reduces risk and increases predictability
- Promotes a common vision and culture
- Harvests and institutionalizes software engineering practices

Why have a process?

- A software engineering process should provide a disciplined yet flexible approach to assigning tasks and responsibilities within a software development organization.
- The goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget.
- Successful iterative development also requires employing a repeatable engineering process.

A Team-Based Definition of Process

A process defines **Who** is doing **What When,** and **How**, in order to reach a certain goal.



This course is mainly about the What, When and How of Testers' activities in the process.

Workbook Page: Implementing Software Engineering Practices

A modern engineering process ideally:

- Supports a controlled, iterative approach
- Supports the use of user-focused requirements to coordinate and drive the work in requirements, design, implementation and test
- Enables architectural concerns to be addressed early
- Allows the process to be configured to suit the context of the individual project
- Provides guidance for conducting work (activities) and producing work products (artifacts)

Module 1 - Review

- Software engineering practices guide software development by addressing root causes of problems.
- Software engineering practices reinforce each other.
- Process guides a team on who does what when and how.
- A software engineering process provides context and support for implementing software engineering practices.