Principles of Software Testing for Testers

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.

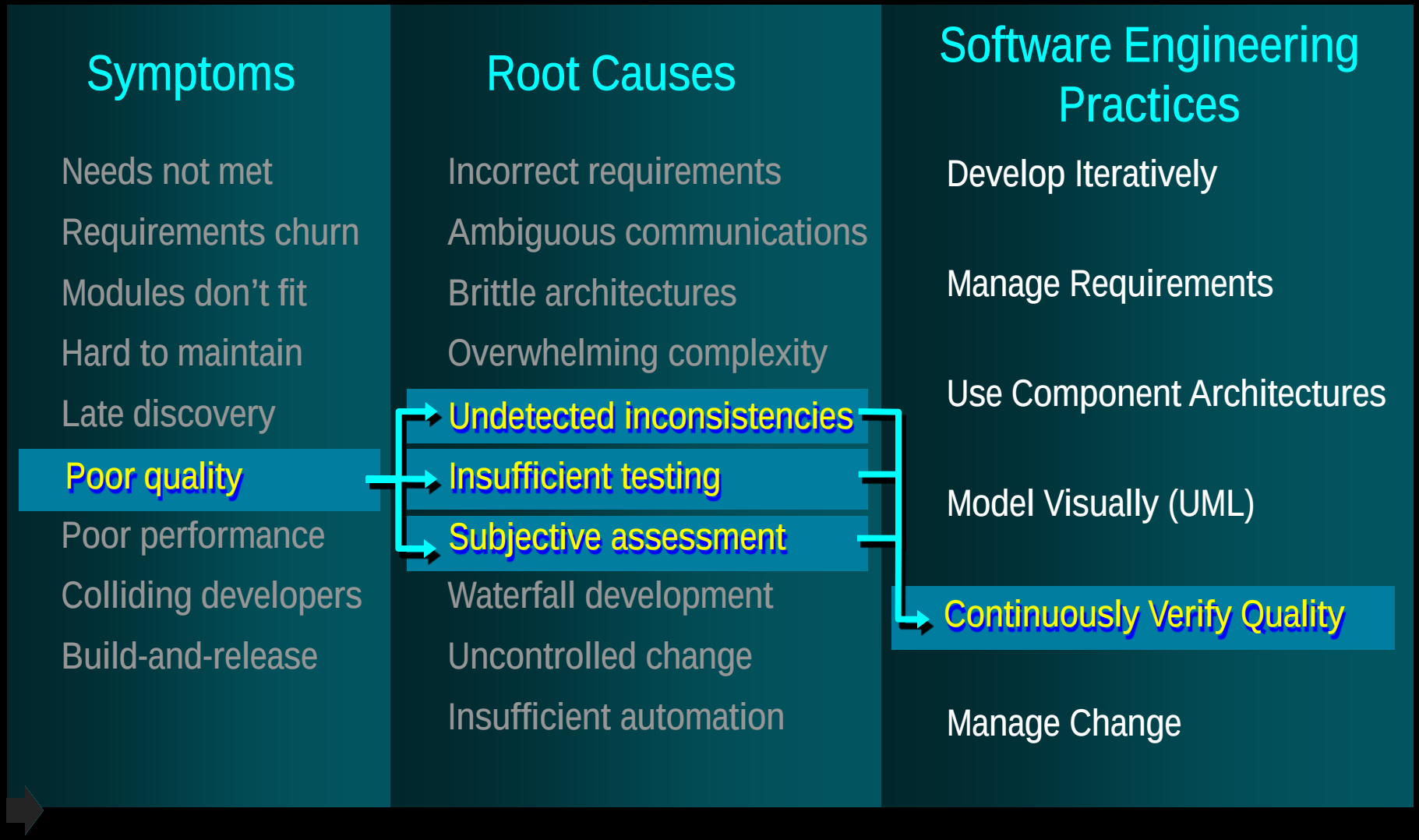
Module 1 - Content Outline (Agenda)

- ➔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

Trace Symptoms to Root Causes



Module 1 - Content Outline (Agenda)

- Software development problems
- ➔Six software engineering practices
- Supporting software engineering practices with process

Practice 1: Develop Iteratively

Software Engineering Practices

Develop Iteratively

Manage Requirements

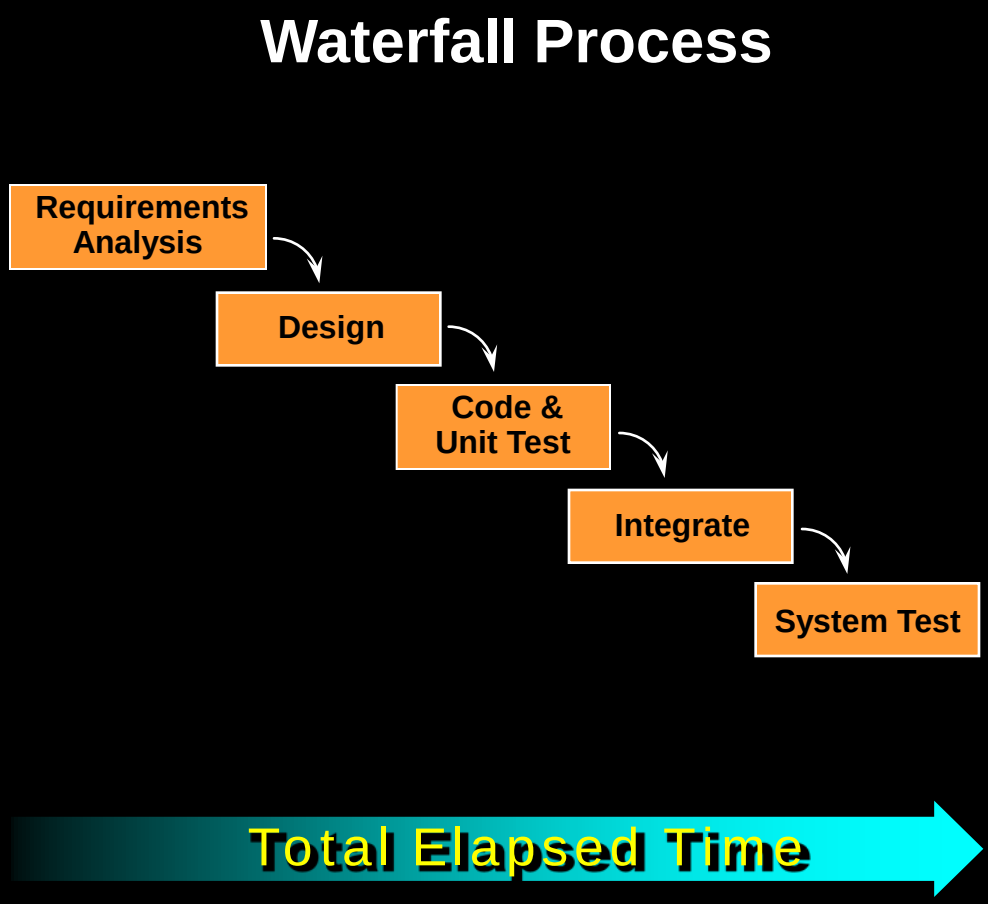
Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

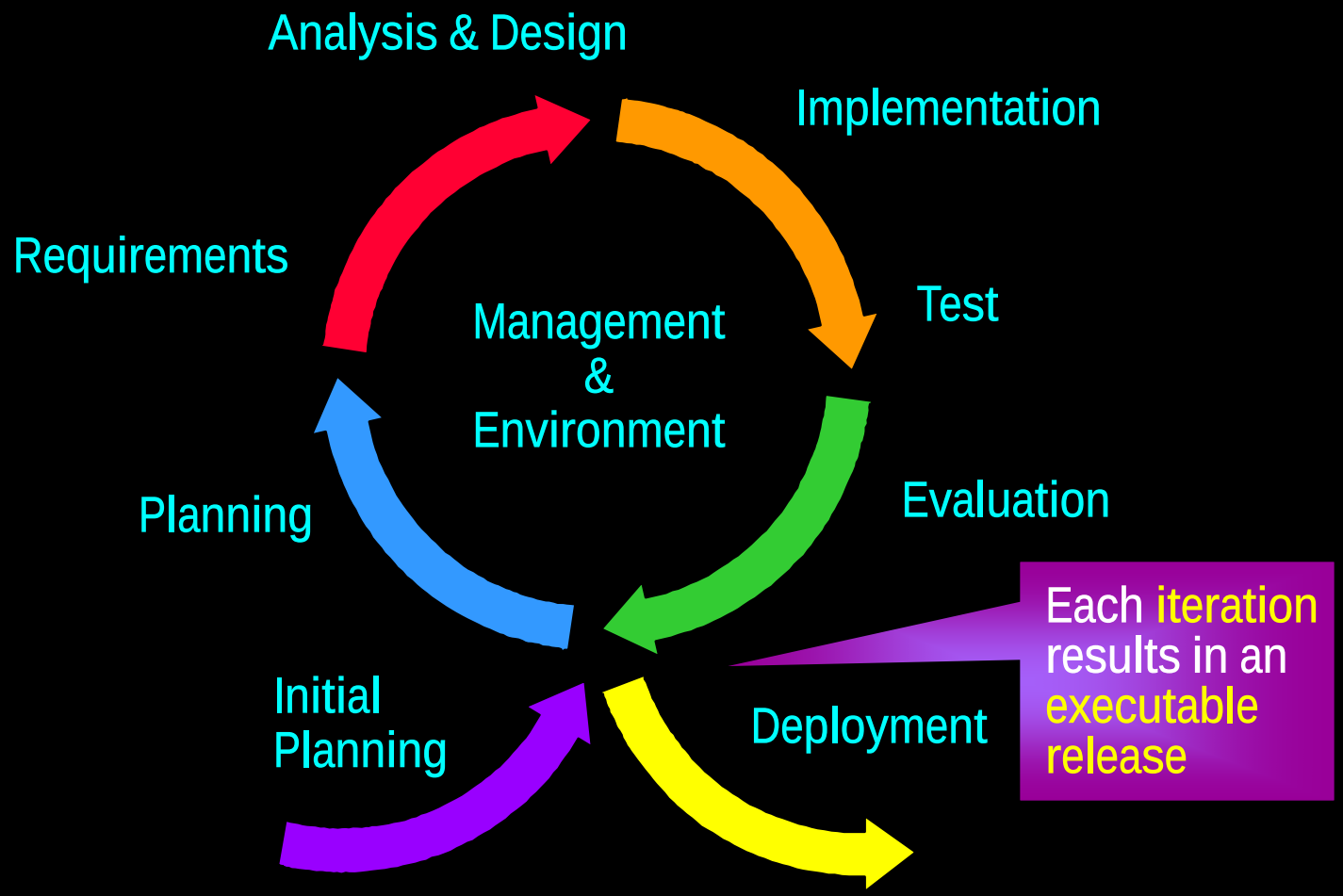
Manage Change

Waterfall Development Characteristics

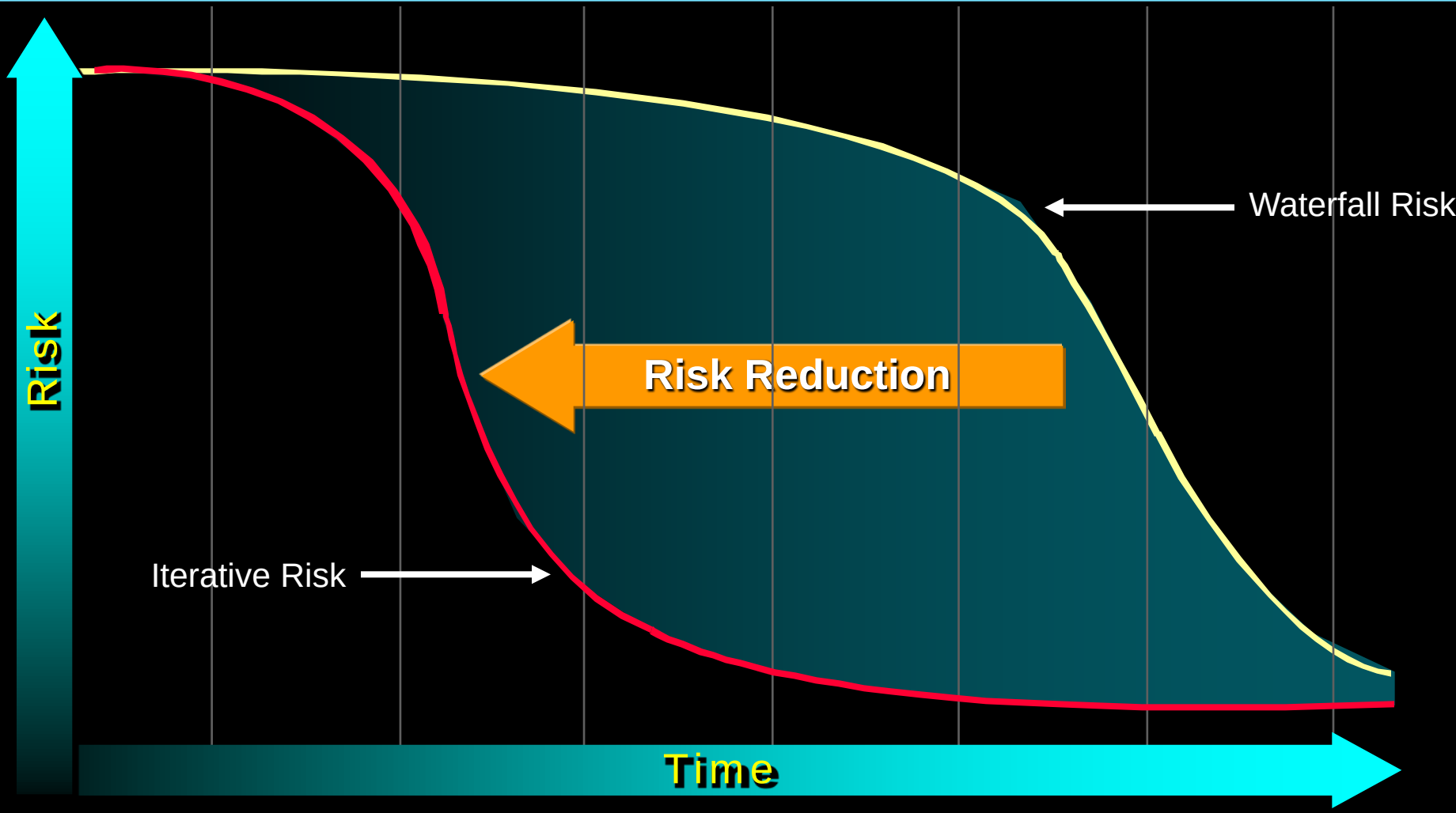


- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned project extensions

Iterative Development Produces an Executable



Risk Profiles



Iterative development drives risks out early.

Practice 2: Manage Requirements

Software Engineering Practices

Develop Iteratively

Manage Requirements

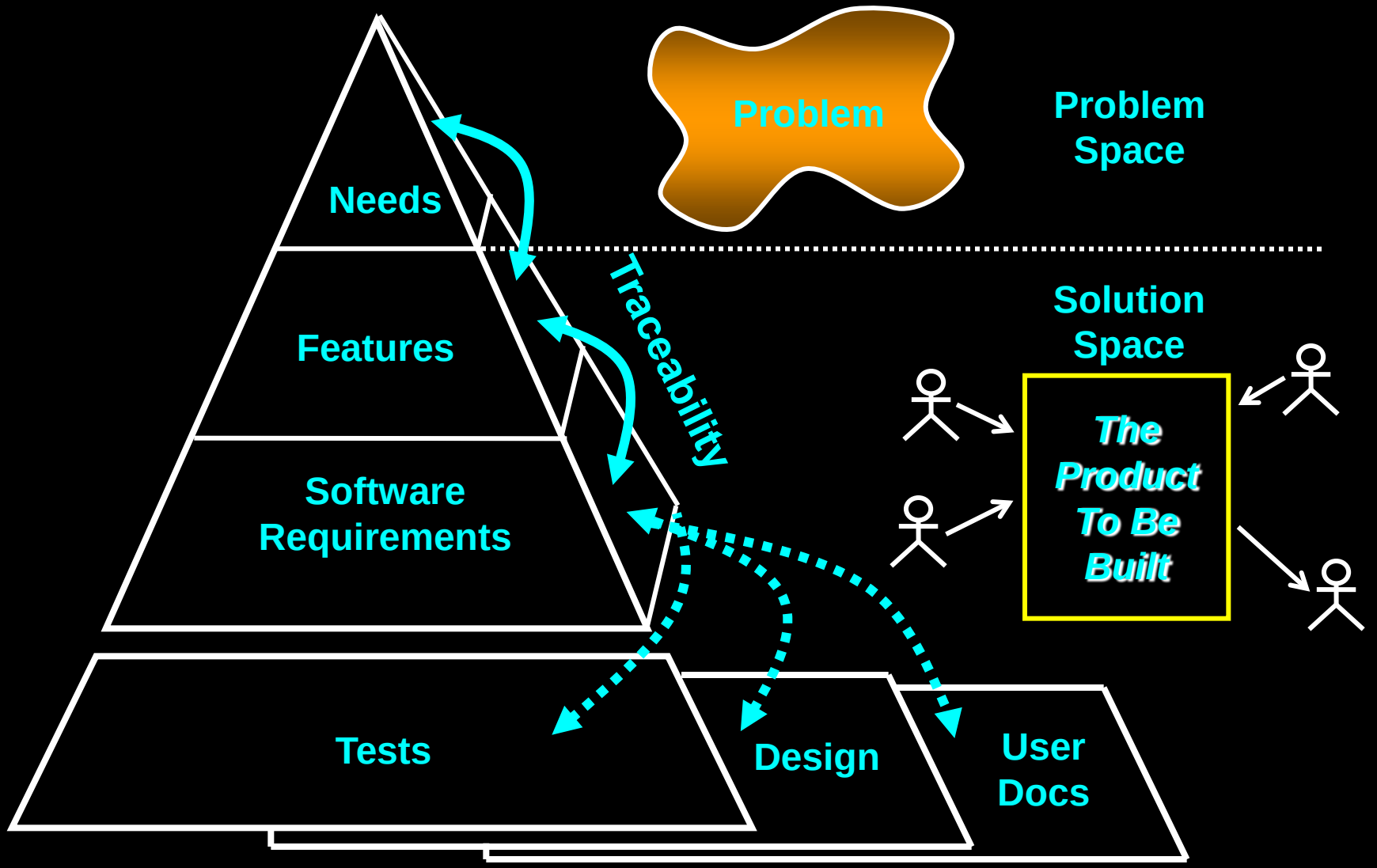
Use Component Architectures

Model Visually (UML)

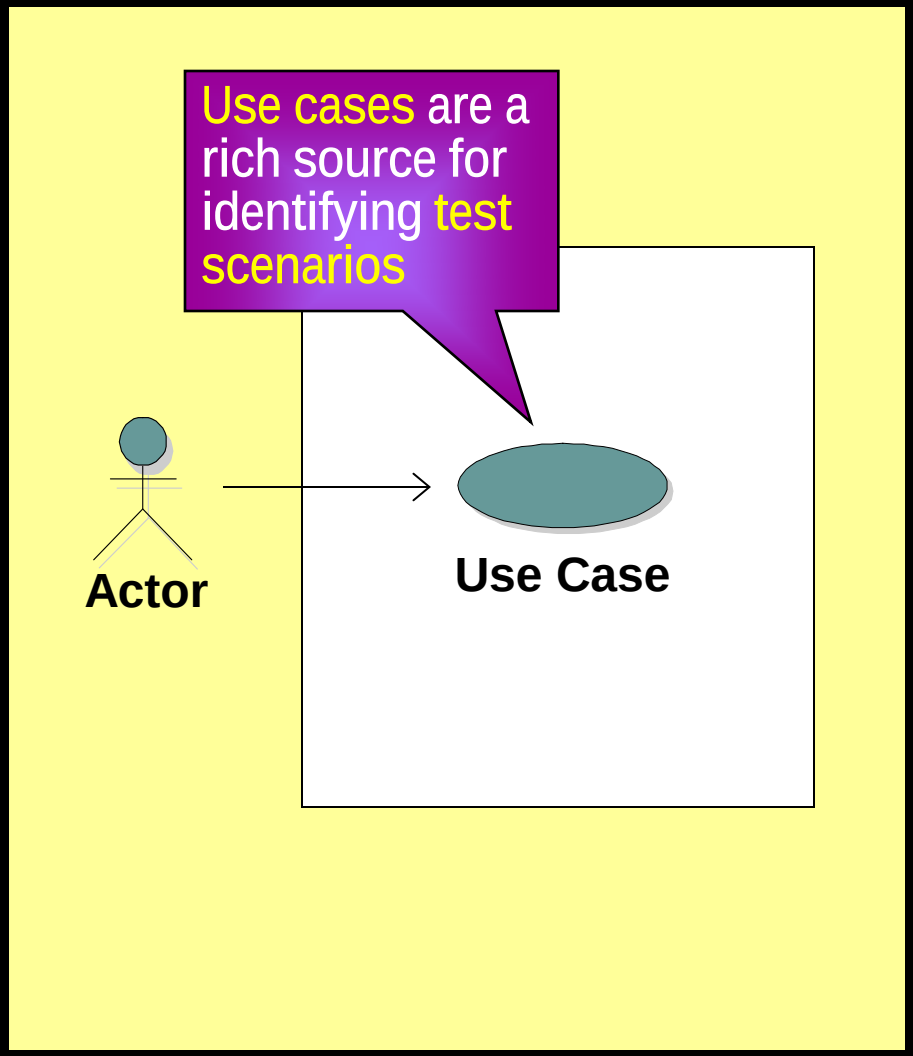
Continuously Verify Quality

Manage Change

Manage Requirements - Map of the Territory



Manage Requirements - Use-Case Concepts



An **actor** represents a person or another system that interacts with the system.

A **use case** defines a sequence of actions a system performs that yields a result of observable value to an actor.

Practice 3: Use Component Architectures

Software Engineering Practices

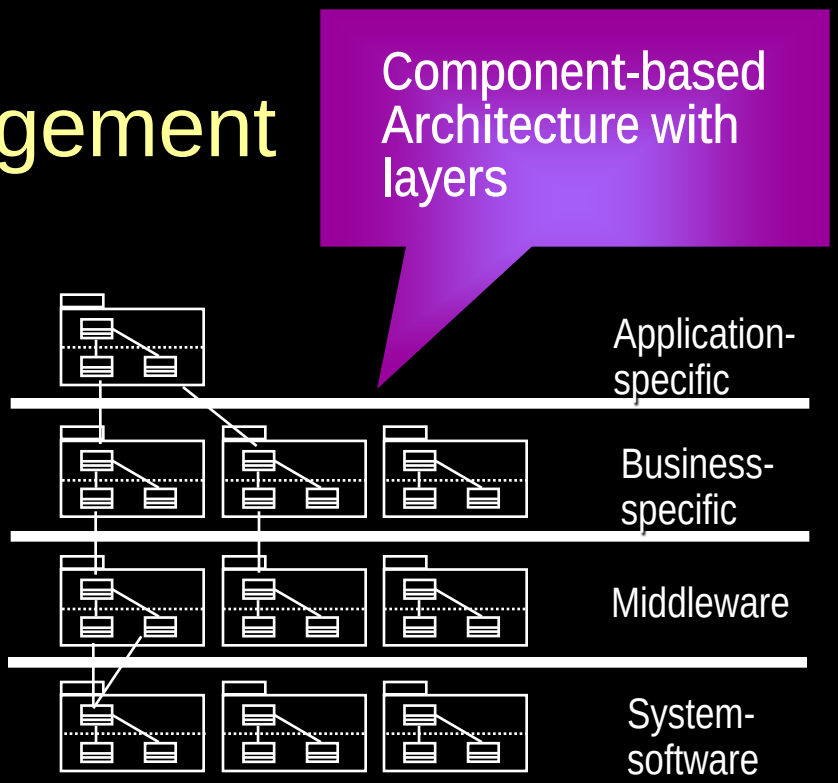
- Develop Iteratively
- Manage Requirements
- Use Component Architectures
- Model Visually (UML)
- Continuously Verify Quality
- Manage Change

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



Practice 4: Model Visually (UML)

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

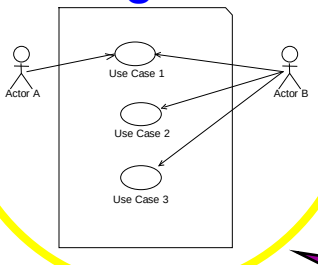
Manage Change

Why Model Visually?

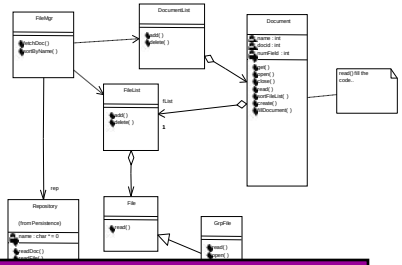
- To help manage complexity
  - To capture both structure and behavior
  - To show how system elements fit together
  - To hide or expose details as appropriate
- To keep design and implementation consistent
- To promote unambiguous communication
  - UML provides one language for all practitioners

Visual Modeling Using UML Diagrams

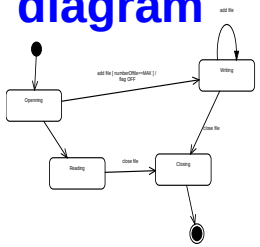
Use-case diagram



Class diagram

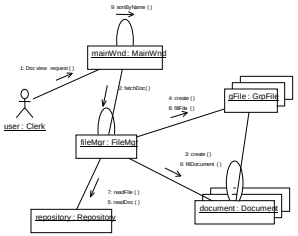


Statechart diagram

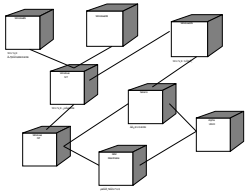


Use-case diagrams outline the system scope

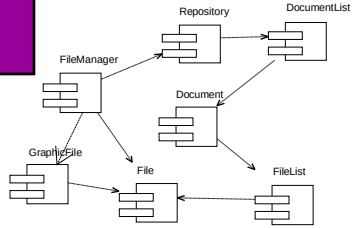
Collaboration diagram



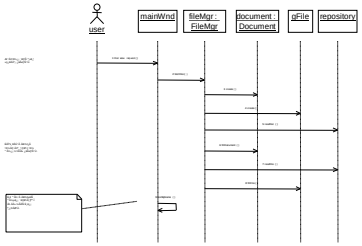
Deployment diagram



Component diagram



Sequence diagram



Forward and Reverse Engineering

Target System

Practice 5: Continuously Verify Quality

Software Engineering Practices

Develop Iteratively

Manage Requirements

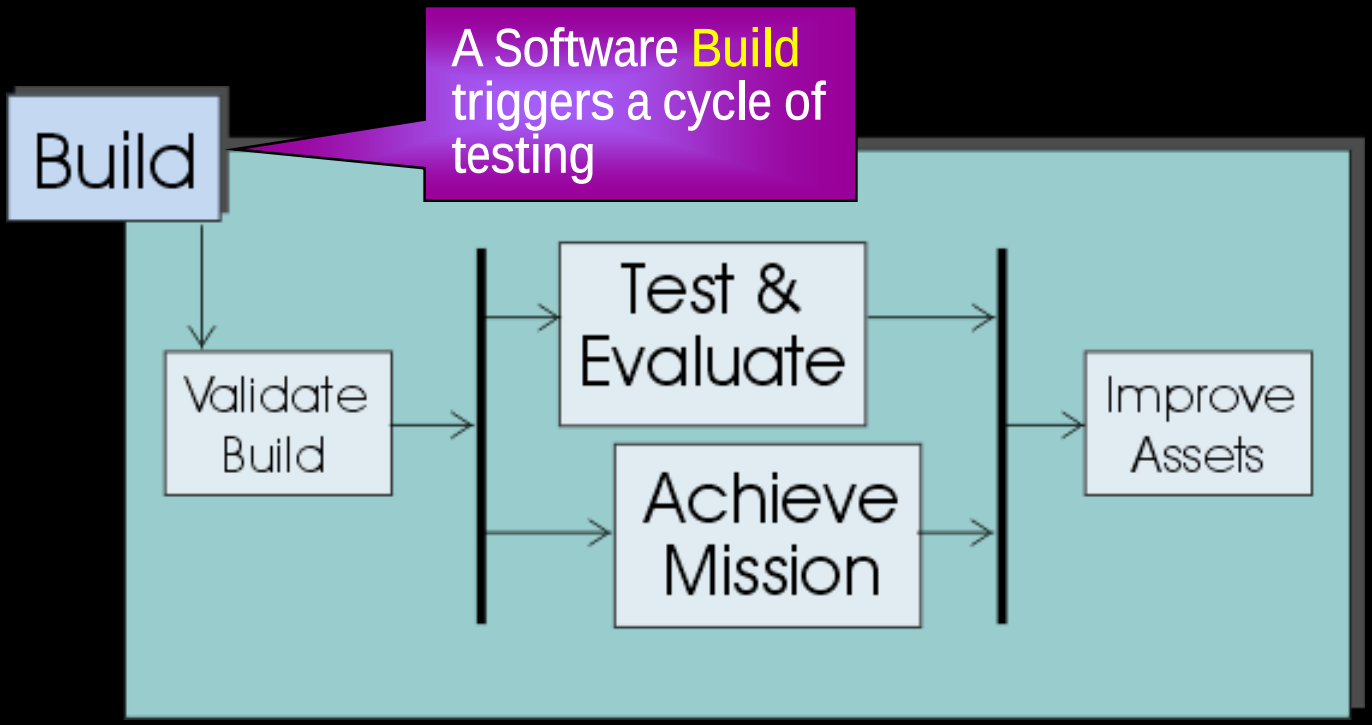
Use Component Architectures

Model Visually (UML)

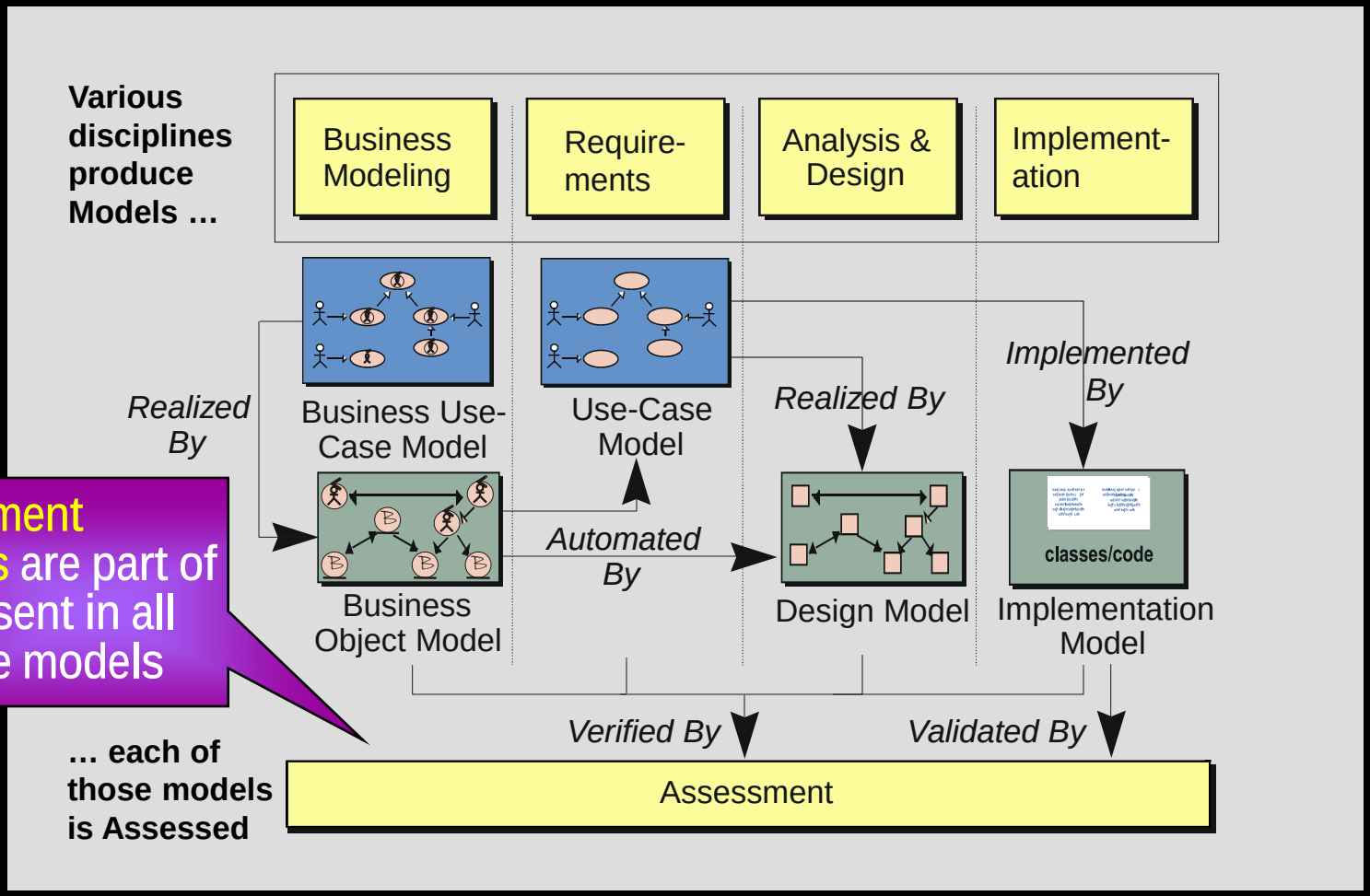
Continuously Verify Quality

Manage Change

Continuously Verify Quality – in each Iteration



Continuously Verify Quality – Software Models



Practice 6: Manage Change

Software Engineering Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

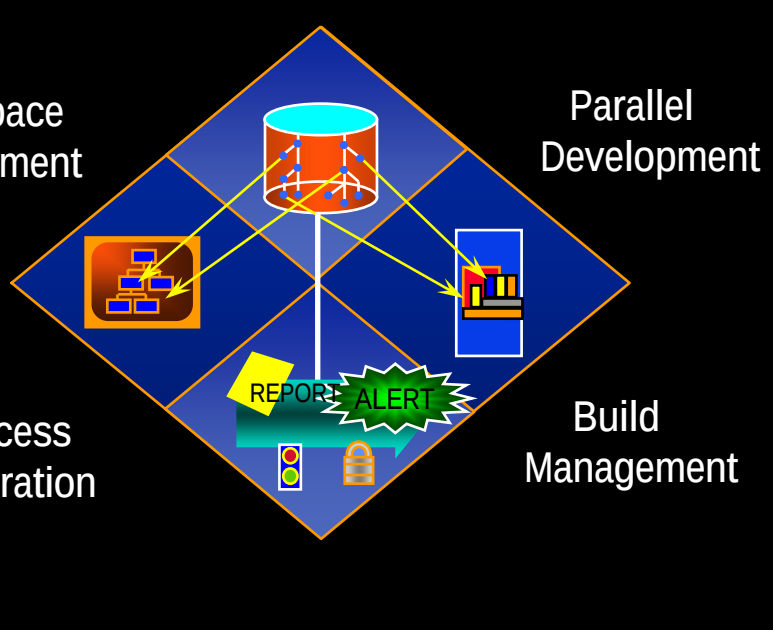
Continuously Verify Quality

Manage Change

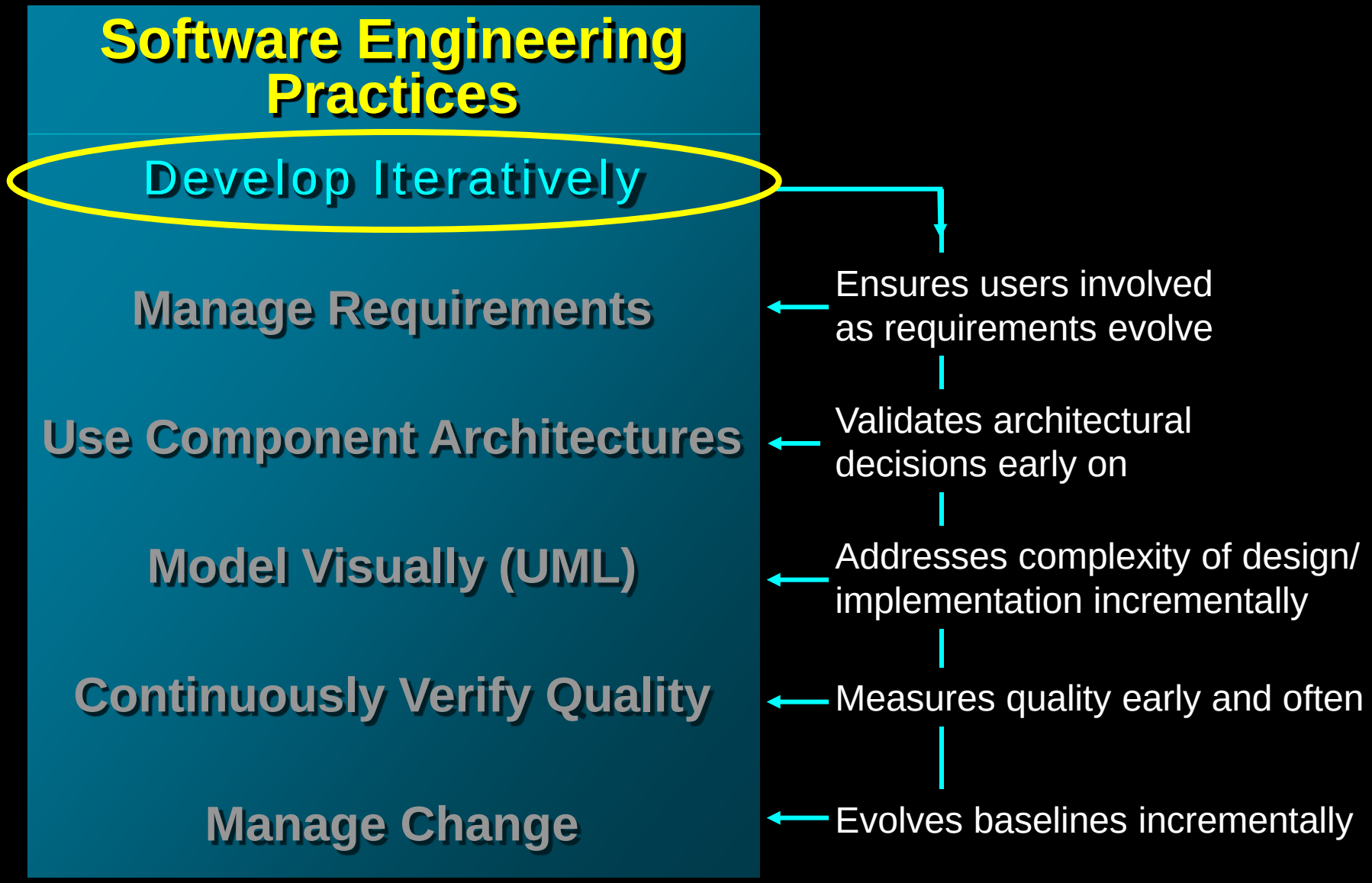
What Do You Want to Control?

- 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



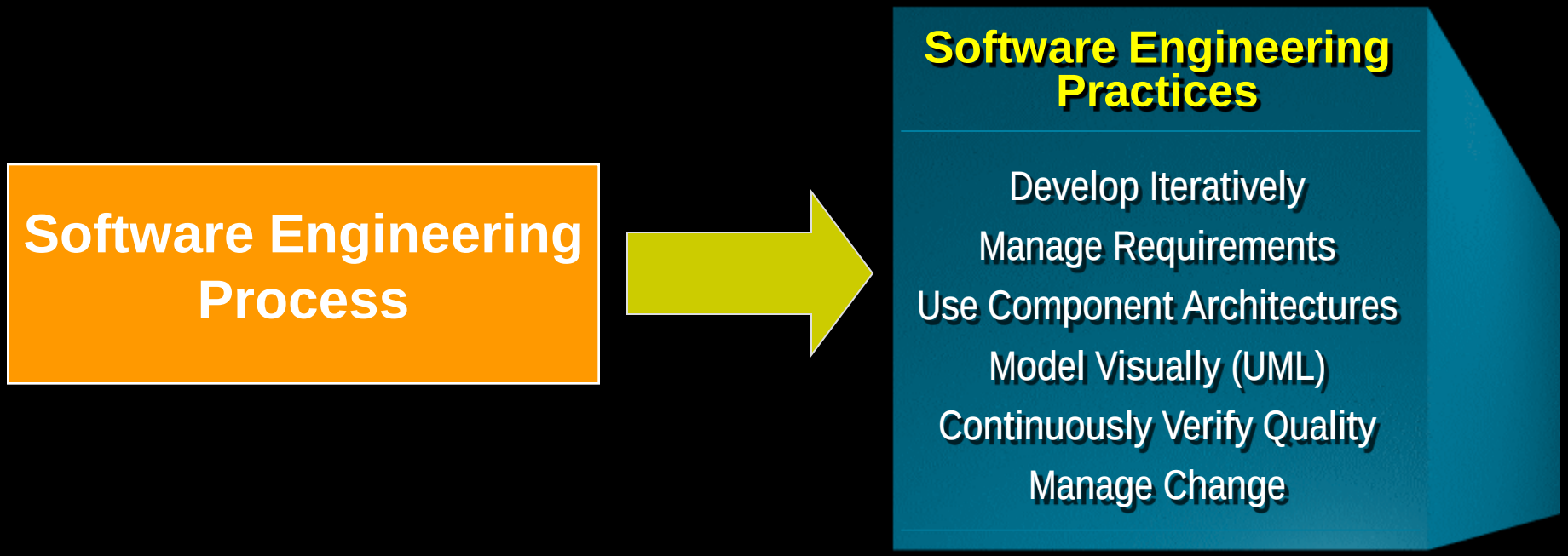
Software Engineering Practices Reinforce Each Other



Module 1 - Content Outline (Agenda)

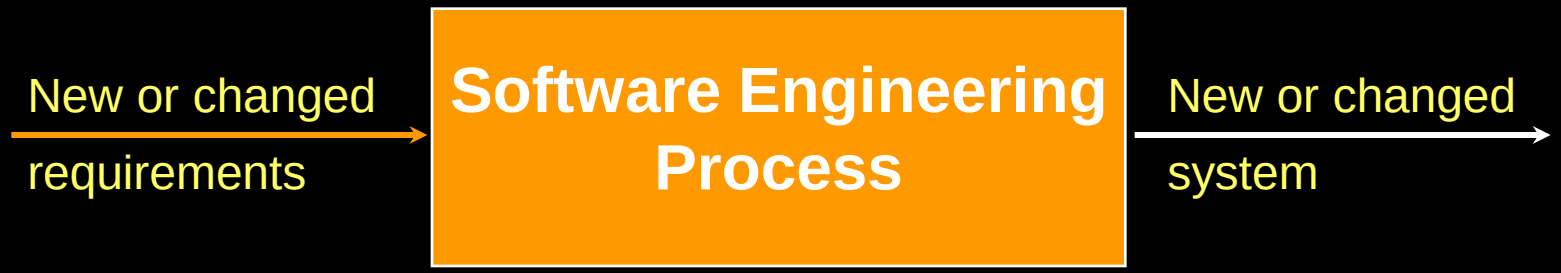
- Software development problems
- Six software engineering practices
- ➔Software engineering process and software engineering practices

An Engineering Process Implements Engineering Practices



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 about the What, When and How of Testers' activities in the process.

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.