IBM

IBM Software Group

Mastering Object-Oriented Analysis and Design with UML

Module 1: Best Practices of Software Engineering

Rational software

Objectives

- Identify activities for understanding and solving software engineering problems.
- Explain the Six Best Practices.
- Present the Rational Unified Process (RUP) within the context of the Six Best Practices.

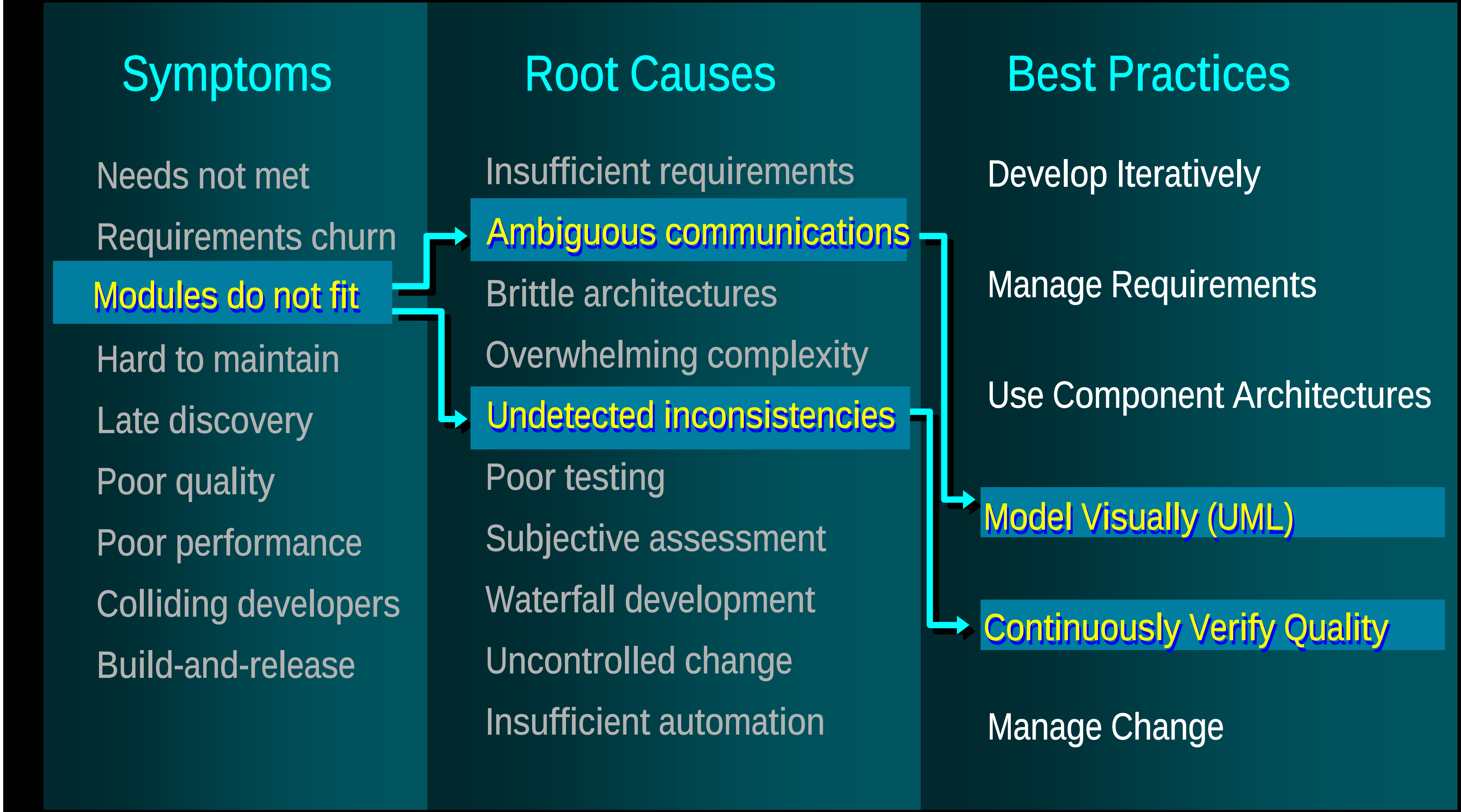
Module 1 Content Outline

- ★ Software development problems
- The Six Best Practices
- RUP within the context of the Six Best Practices

Symptoms of Software Development Problems

- ✓User or business needs not met
- ✓Requirements not addressed
- ✓Modules not integrating
- ✓Difficulties with maintenance
- ✓Late discovery of flaws
- ✓Poor quality of end-user experience
- ✓Poor performance under load
- ✓No coordinated team effort
- ✓Build-and-release issues

Trace Symptoms to Root Causes



Symptoms

- Needs not met
- Requirements churn
- Modules do not fit
- Hard to maintain
- Late discovery
- Poor quality
- Poor performance
- Colliding developers
- Build-and-release

Root Causes

- Insufficient requirements
- Ambiguous communications
- Brittle architectures
- Overwhelming complexity
- Undetected inconsistencies
- Poor testing
- Subjective assessment
- Waterfall development
- Uncontrolled change
- Insufficient automation

Best Practices

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

Module 1 Content Outline

- Software development problems
- ★ The Six Best Practices
- RUP within the context of the Six Best Practices

Practice 1: Develop Iteratively

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

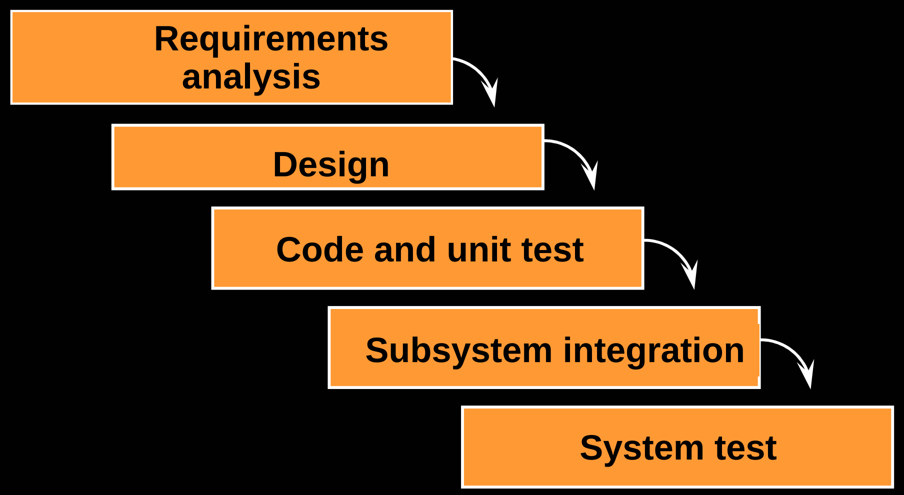
Model Visually (UML)

Continuously Verify Quality

Manage Change

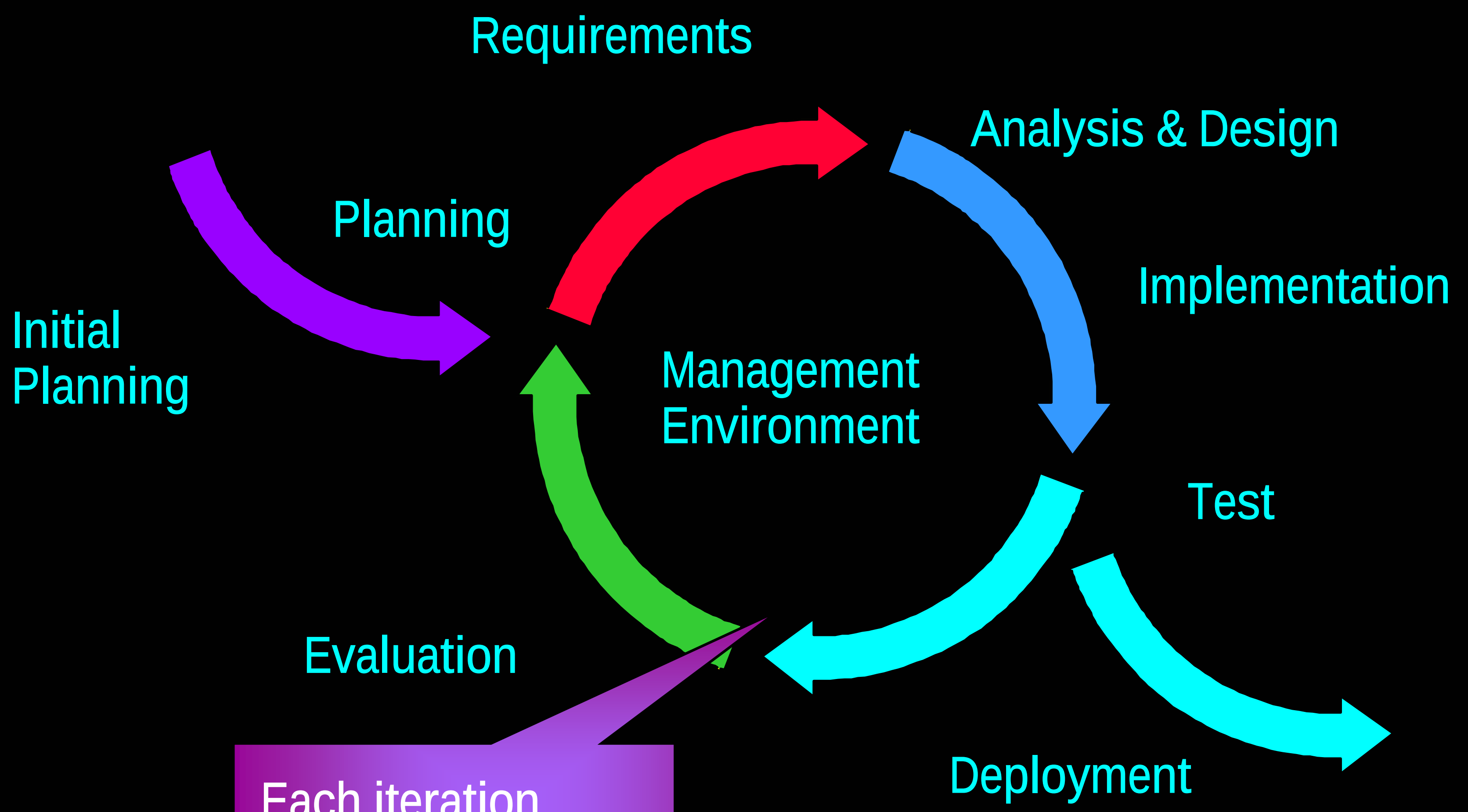
Waterfall Development Characteristics

Waterfall Process



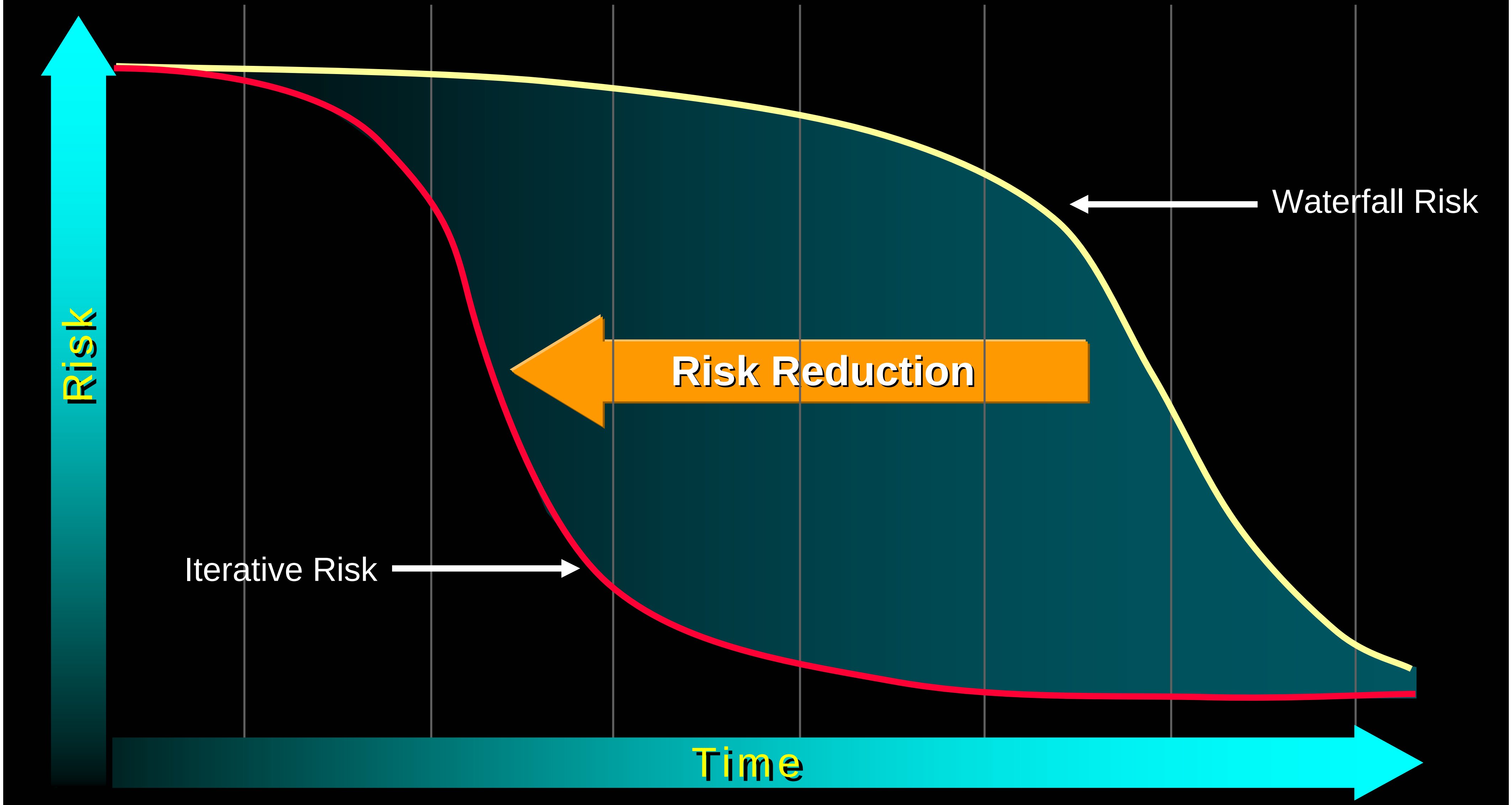
- Delays confirmation of critical risk resolution
- Measures progress by assessing work products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

Iterative Development Produces an Executable



Each iteration results in an executable release

Risk Profiles



Practice 2: Manage Requirements

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Requirements Management

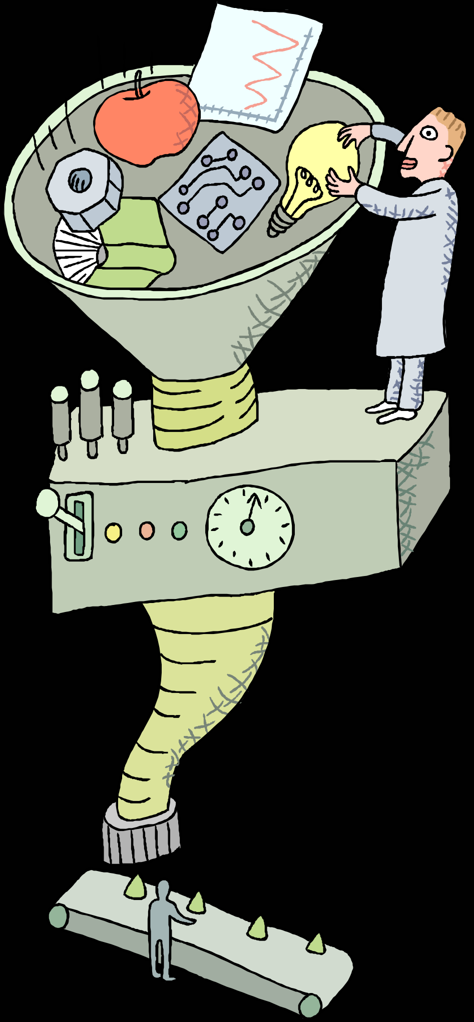
Making sure you

- solve the right problem
- build the right system

by taking a systematic approach to

- eliciting
- organizing
- documenting
- managing

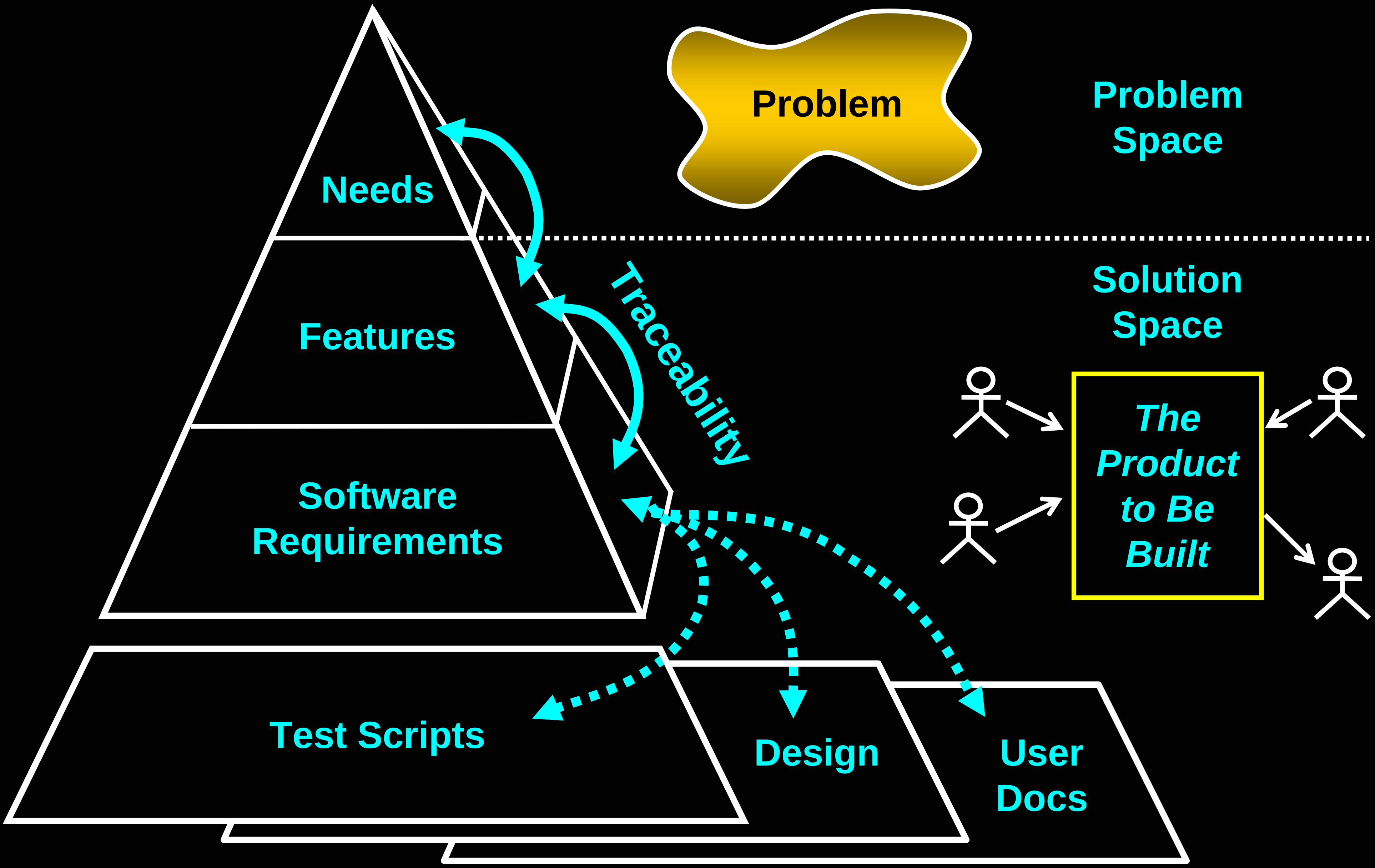
the changing requirements of a software application.



Aspects of Requirements Management

- Analyze the Problem
- Understand User Needs
- Define the System
- Manage Scope
- Refine the System Definition
- Manage Changing Requirements

Map of the Territory



Practice 3: Use Component Architectures

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

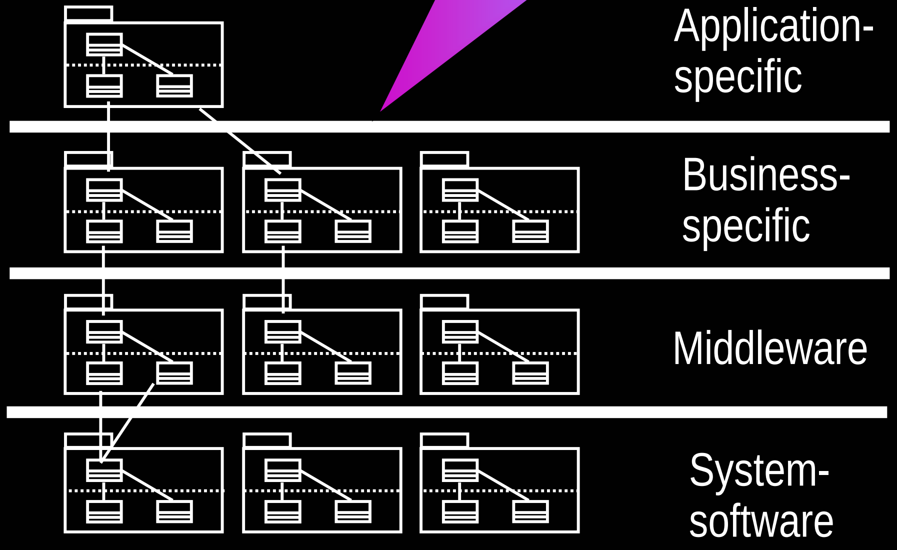
Manage Change

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)

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

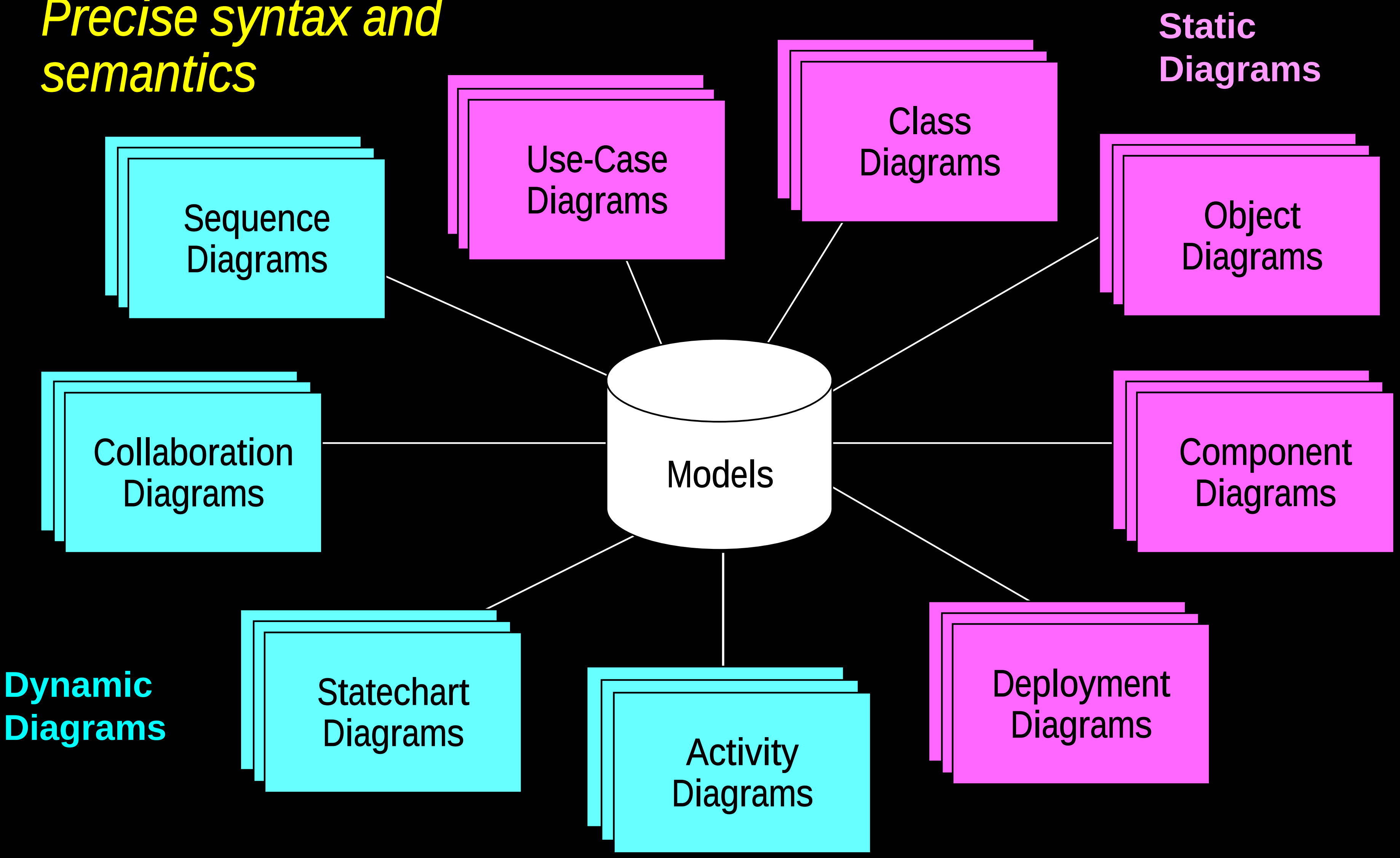
Manage Change

Why Model Visually?

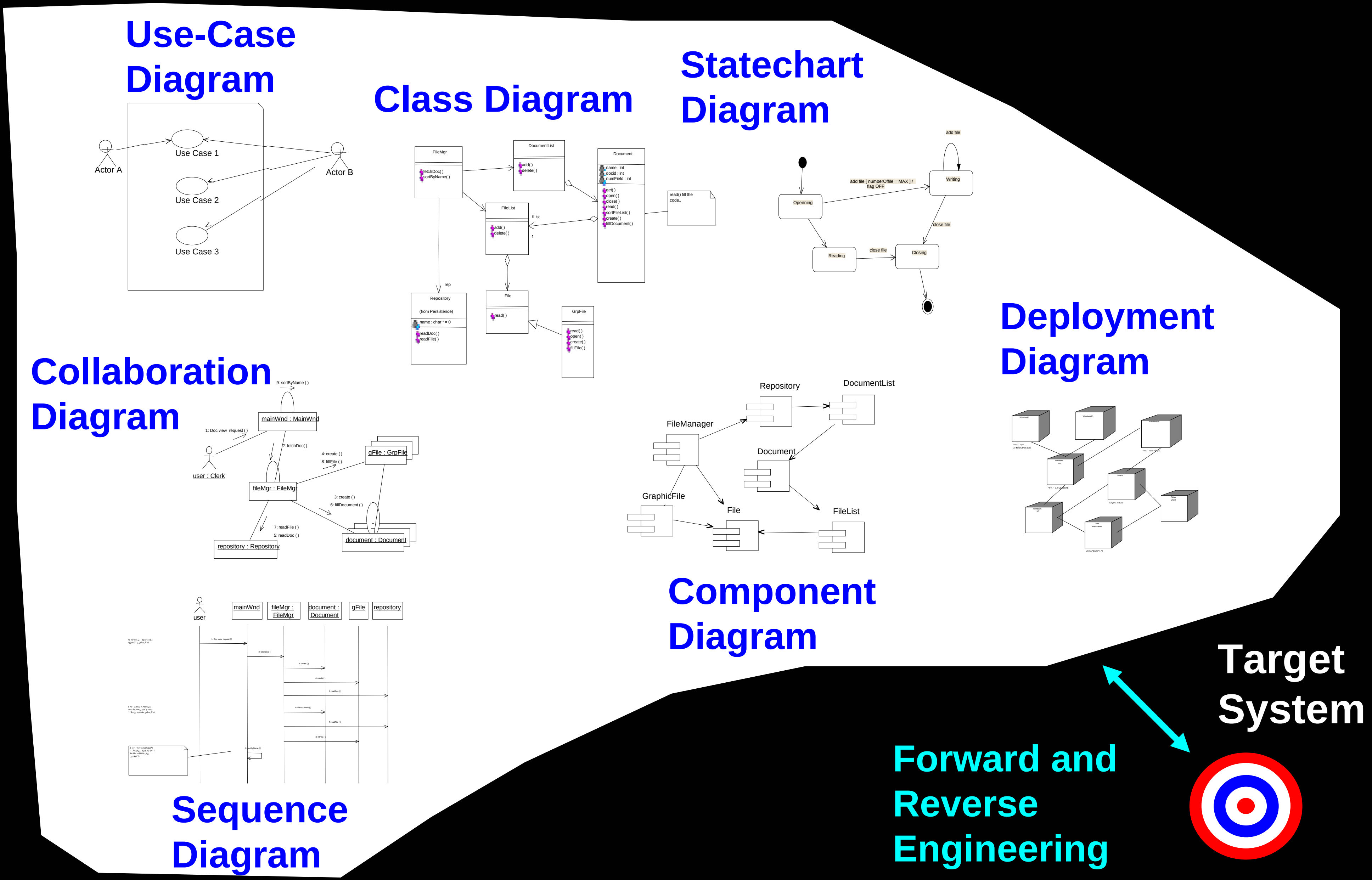
- Captures structure and behavior
- Shows how system elements fit together
- Keeps design and implementation consistent
- Hides or exposes details as appropriate
- Promotes unambiguous communication
  - The UML provides one language for all practitioners

Visual Modeling With the Unified Modeling Language

- *Multiple views*
- *Precise syntax and semantics*



Visual Modeling Using UML Diagrams



Use-Case Diagram

Class Diagram

Statechart Diagram

Deployment Diagram

Collaboration Diagram

Component Diagram

Sequence Diagram

Target System

Forward and Reverse Engineering

Practice 5: Continuously Verify Quality

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

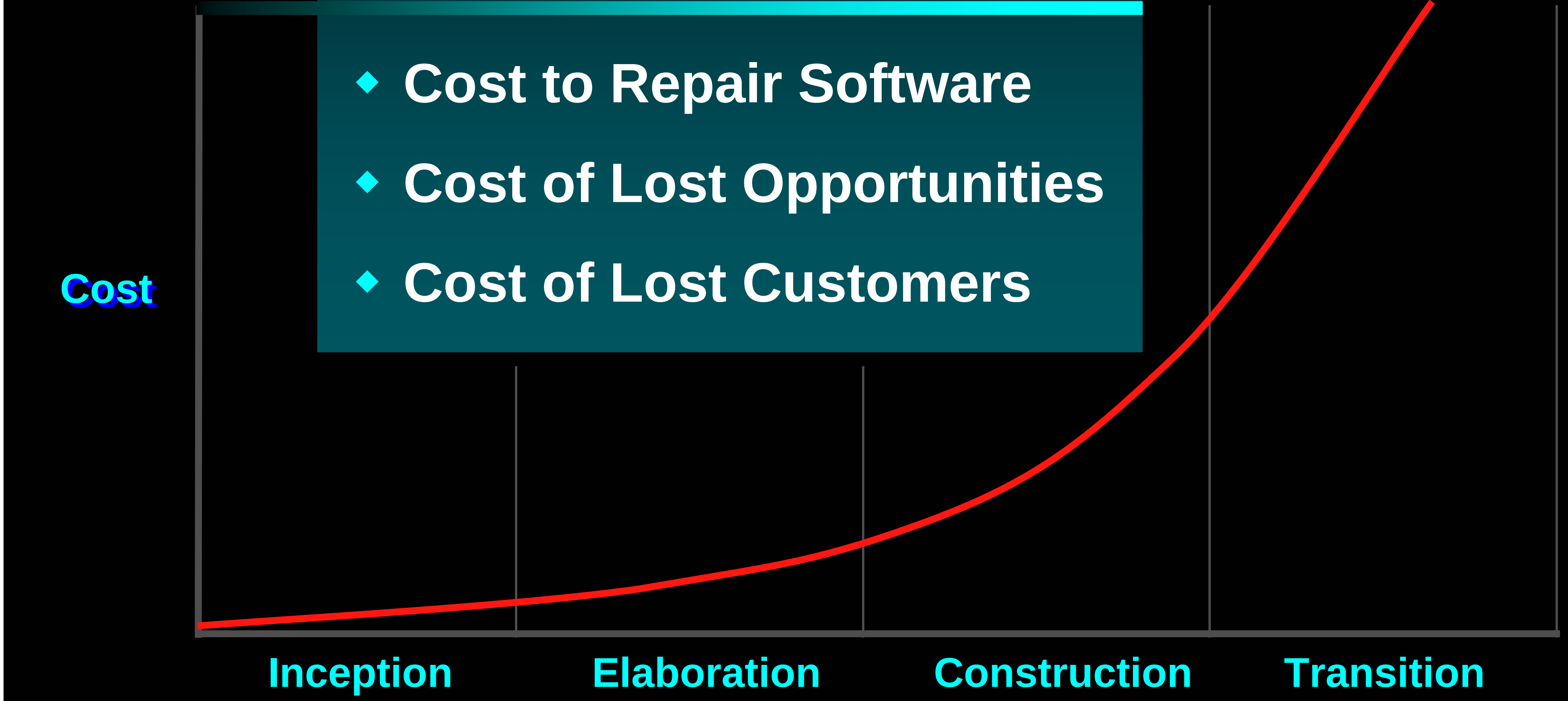
Continuously Verify Quality

Manage Change

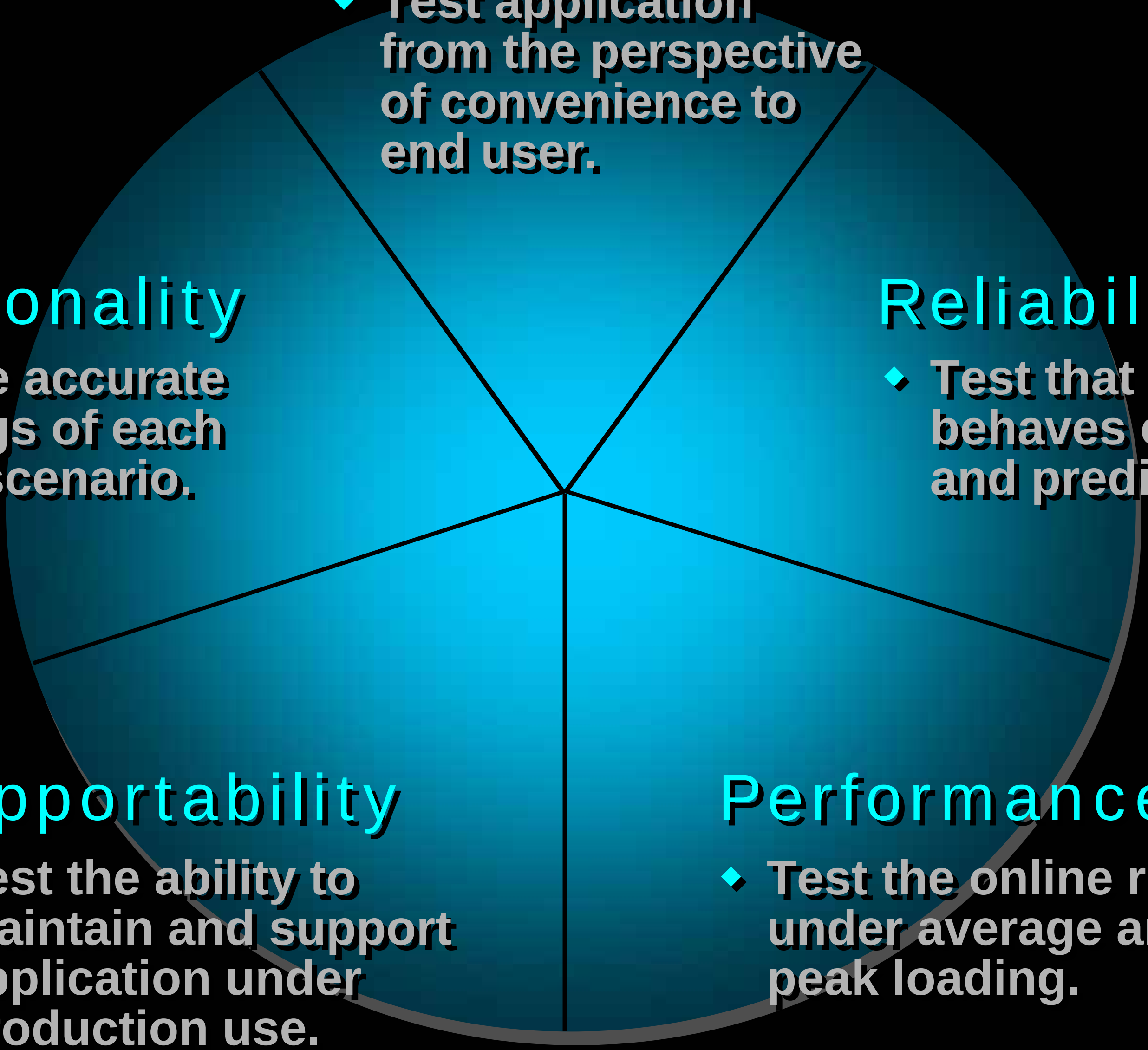
Continuously Verify Your Software's Quality

Software problems are
100 to 1000 times more costly
to find and repair after deployment

- Cost to Repair Software
- Cost of Lost Opportunities
- Cost of Lost Customers



Testing Dimensions of Quality



Usability

- Test application from the perspective of convenience to end user.

Functionality

- Test the accurate workings of each usage scenario.

Reliability

- Test that the application behaves consistently and predictably.

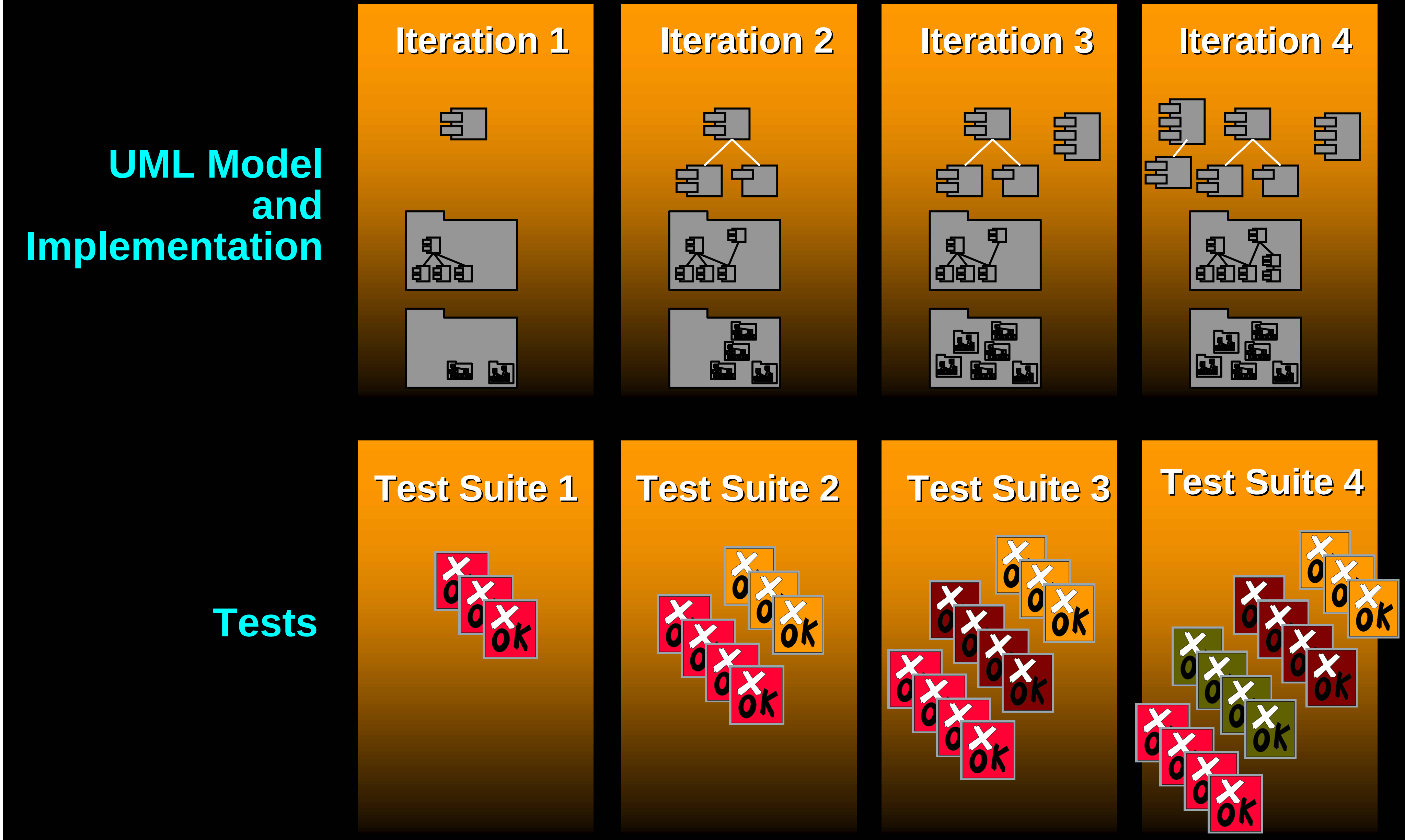
Supportability

- Test the ability to maintain and support application under production use.

Performance

- Test the online response under average and peak loading.

Test Each Iteration



UML Model and Implementation

Tests

Iteration 1

Iteration 2

Iteration 3

Iteration 4

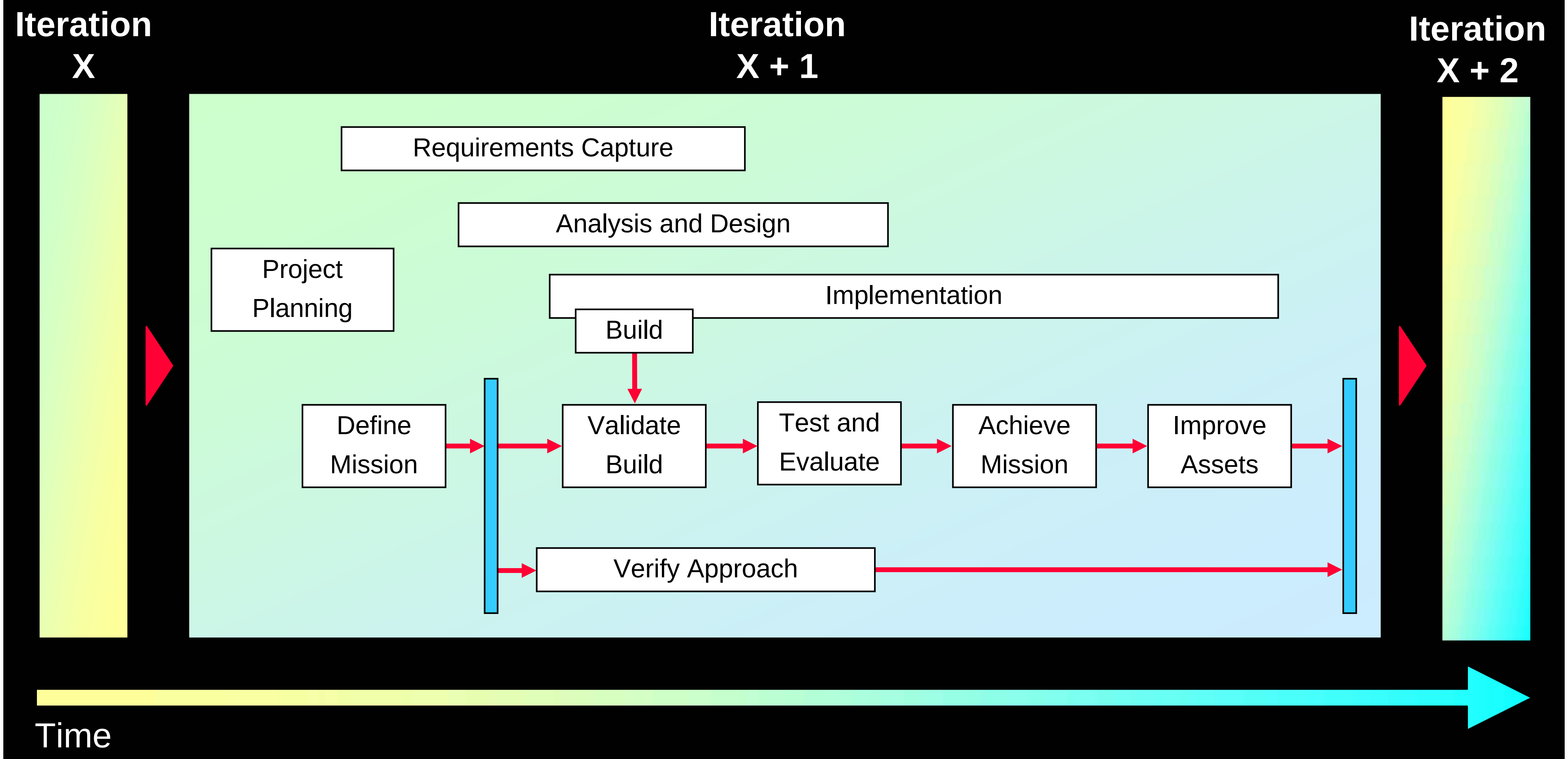
Test Suite 1

Test Suite 2

Test Suite 3

Test Suite 4

Test Within the Product Development Lifecycle



Iteration X

Iteration X + 1

Iteration X + 2

Requirements Capture

Analysis and Design

Project Planning

Implementation

Build

Define Mission

Validate Build

Test and Evaluate

Achieve Mission

Improve Assets

Verify Approach

Time

Practice 6: Manage Change

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

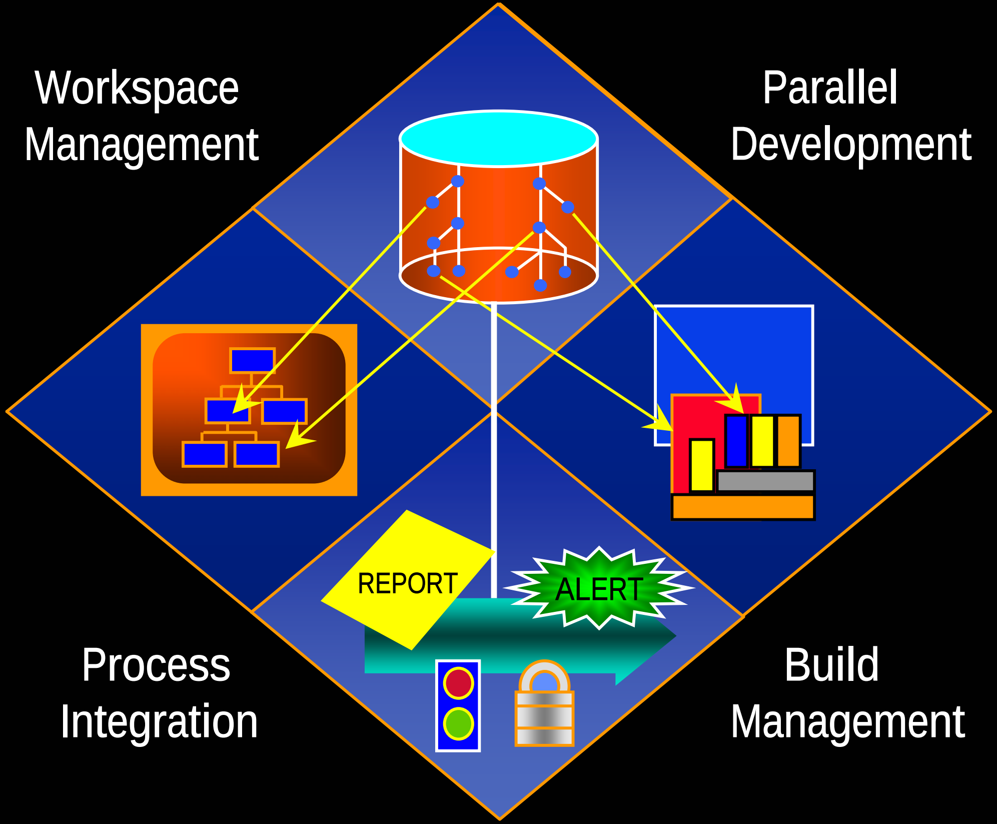
Continuously Verify Quality

Manage Change

What Do You Want to Control?

- Secure workspaces for each developer
- Automated integration/build management
- Parallel development

Configuration Management is more than just check-in and check-out



Aspects of a CM System

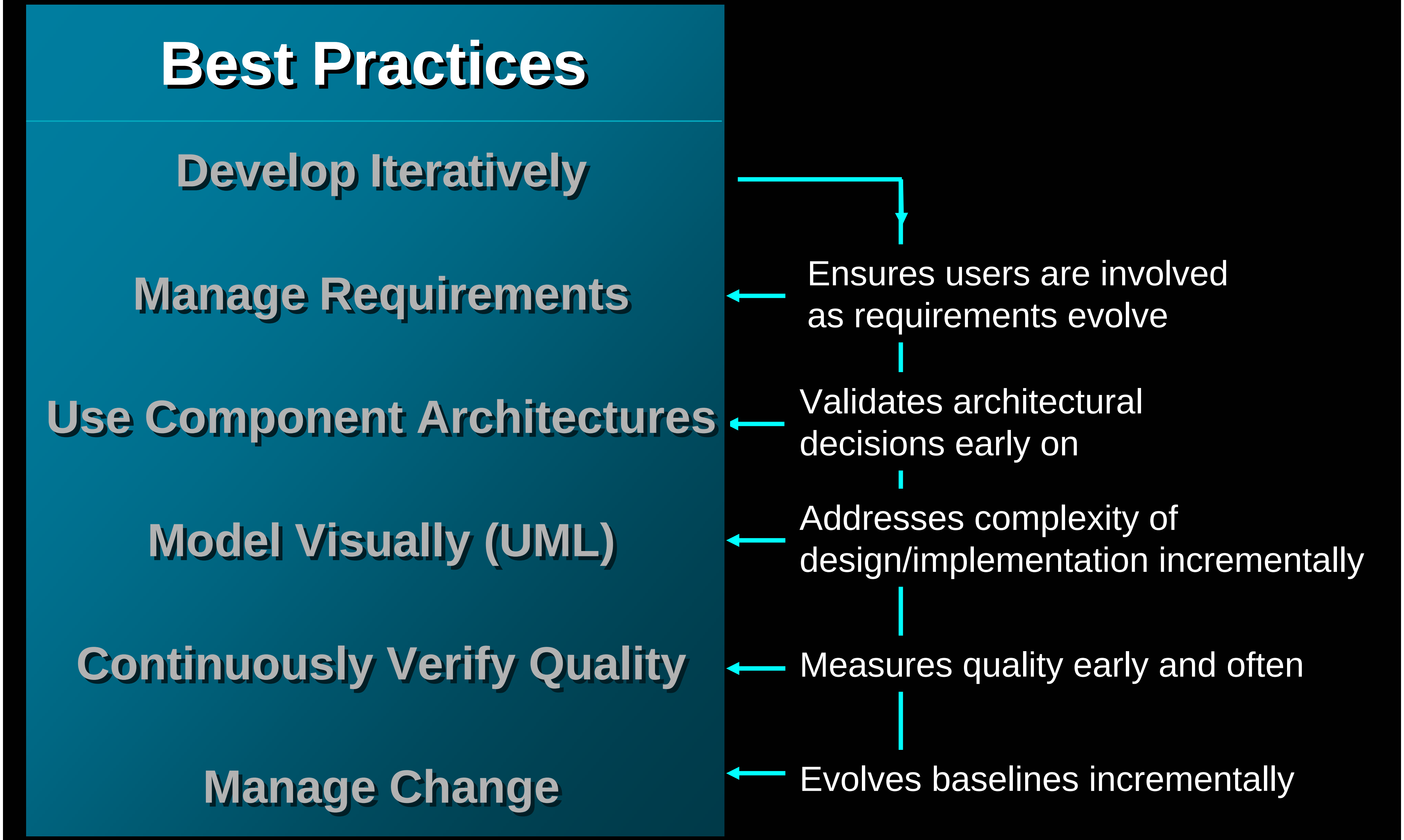
- Change Request Management (CRM)
- Configuration Status Reporting
- Configuration Management (CM)
- Change Tracking
- Version Selection
- Software Manufacture

Unified Change Management (UCM)

UCM involves:

- Management across the lifecycle
  - System
  - Project Management
- Activity-Based Management
  - Tasks
  - Defects
  - Enhancements
- Progress Tracking
  - Charts
  - Reports

Best Practices Reinforce Each Other



Best Practices

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

Ensures users are involved as requirements evolve

Validates architectural decisions early on

Addresses complexity of design/implementation incrementally

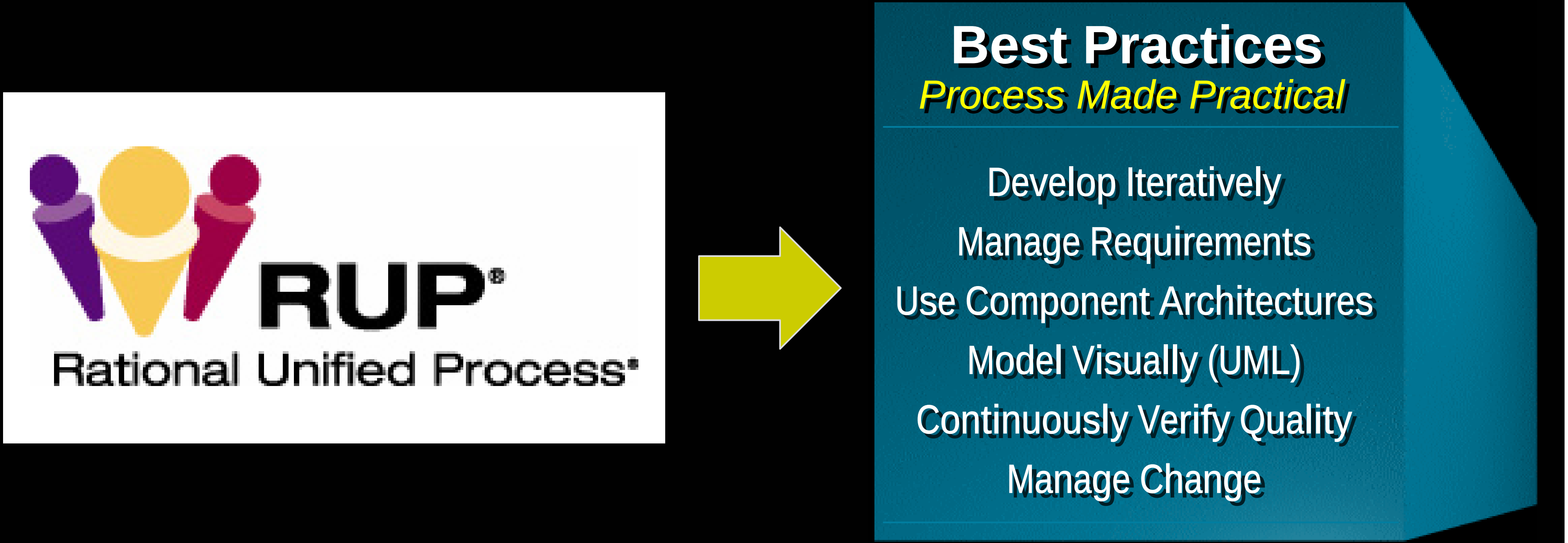
Measures quality early and often

Evolves baselines incrementally

Module 1 Content Outline

- Software development problems
- The Six Best Practices
- ★ RUP within the context of the Six Best Practices

Rational Unified Process Implements Best Practices



Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

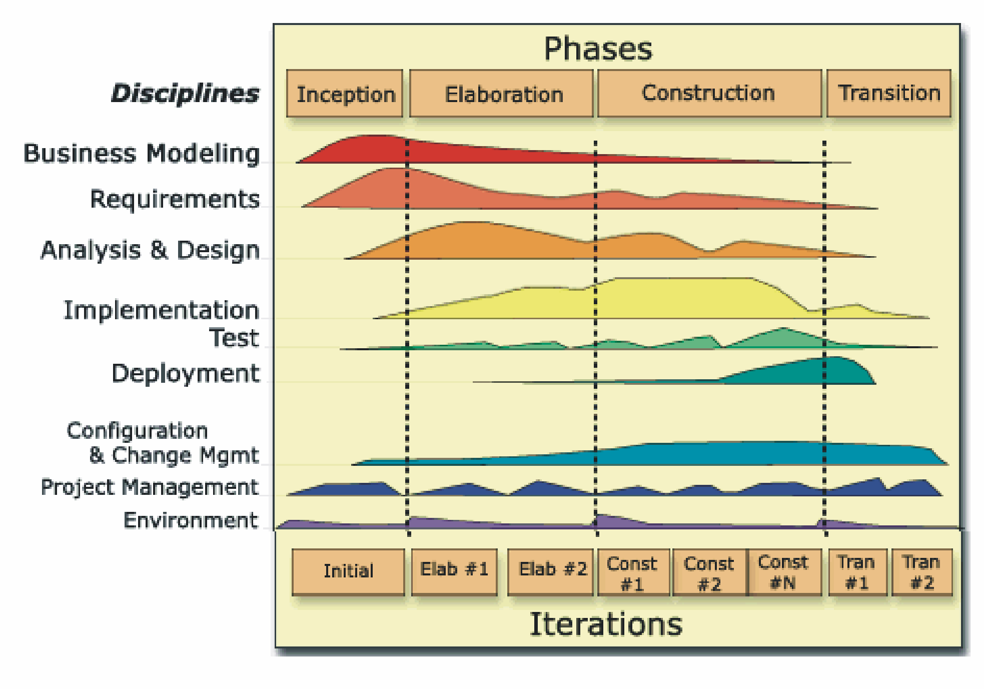
Model Visually (UML)

Continuously Verify Quality

Manage Change

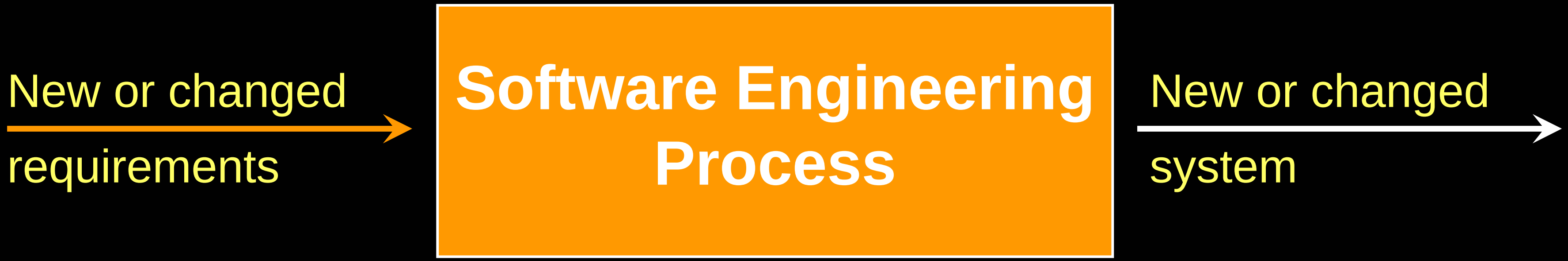
Achieving Best Practices

- Iterative approach
- Guidance for activities and artifacts
- Process focus on architecture
- Use cases that drive design and implementation
- Models that abstract the system



A Team-Based Definition of Process

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

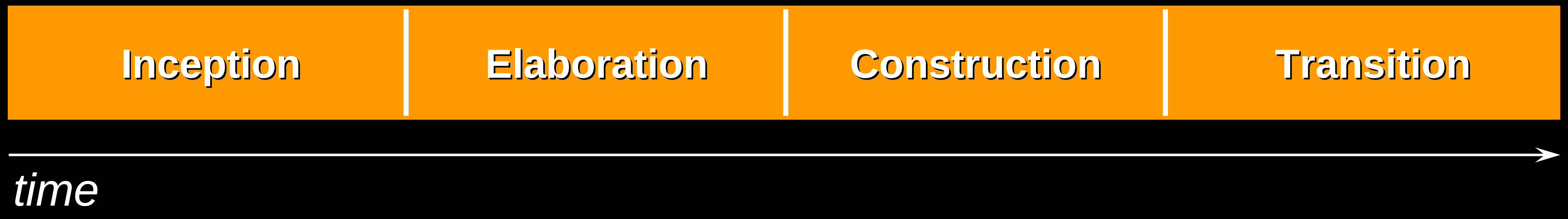


New or changed requirements

Software Engineering Process

New or changed system

Process Structure - Lifecycle Phases



Inception

Elaboration

Construction

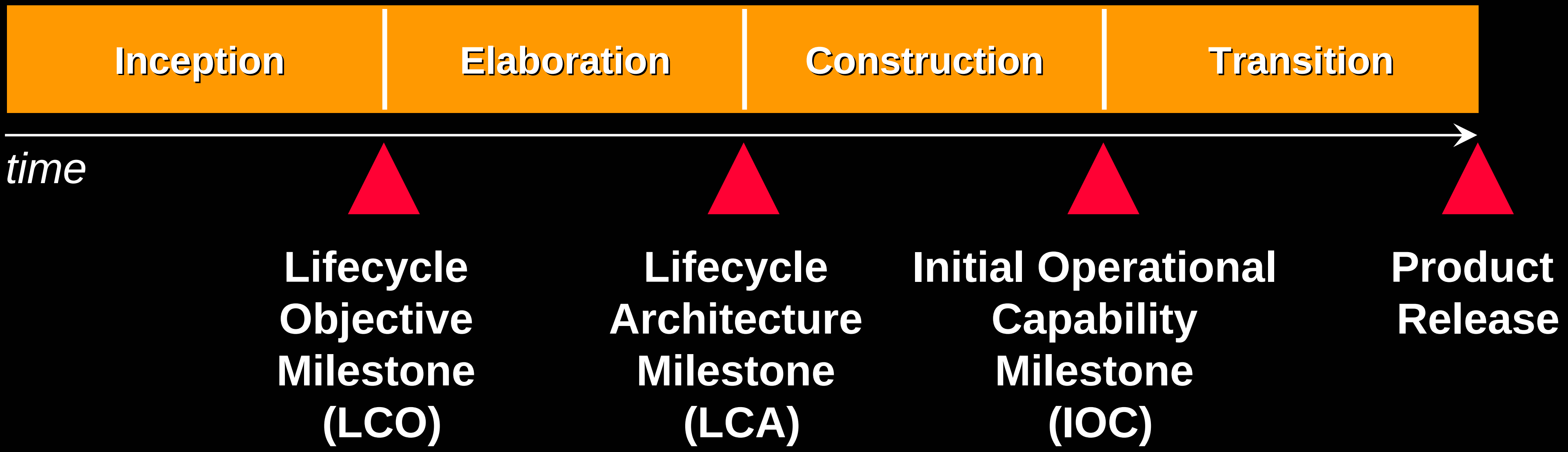
Transition

time

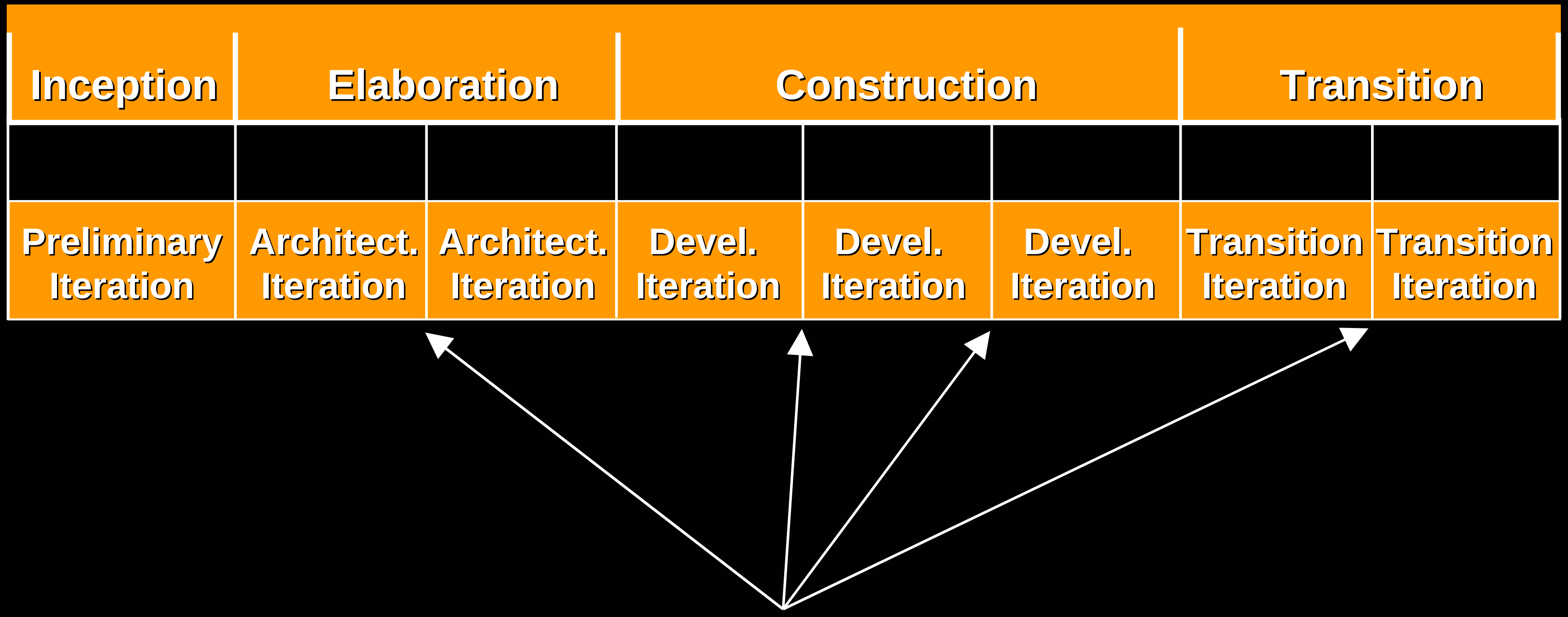
Rational Unified Process has four phases:

- **Inception** - Define the scope of project
- **Elaboration** - Plan project, specify features and baseline architecture
- **Construction** - Build the product
- **Transition** - Transition the product into end-user community

Phase Boundaries Mark Major Milestones


Inception
Elaboration
Construction
Transition
time
Lifecycle Objective Milestone (LCO)
Lifecycle Architecture Milestone (LCA)
Initial Operational Capability Milestone (IOC)
Product Release

Iterations and Phases

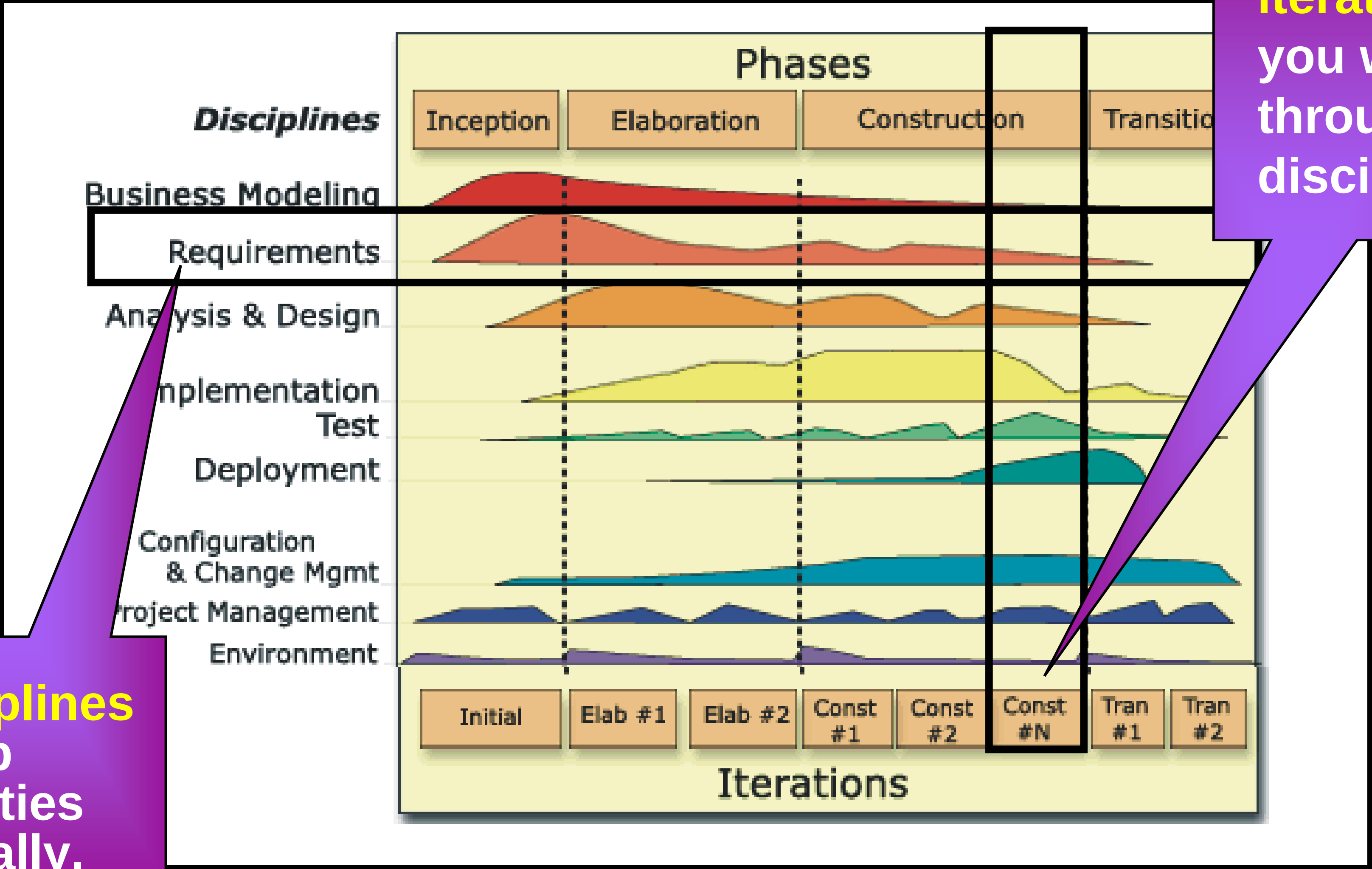


Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

Minor Milestones: Releases

An **iteration** is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external).

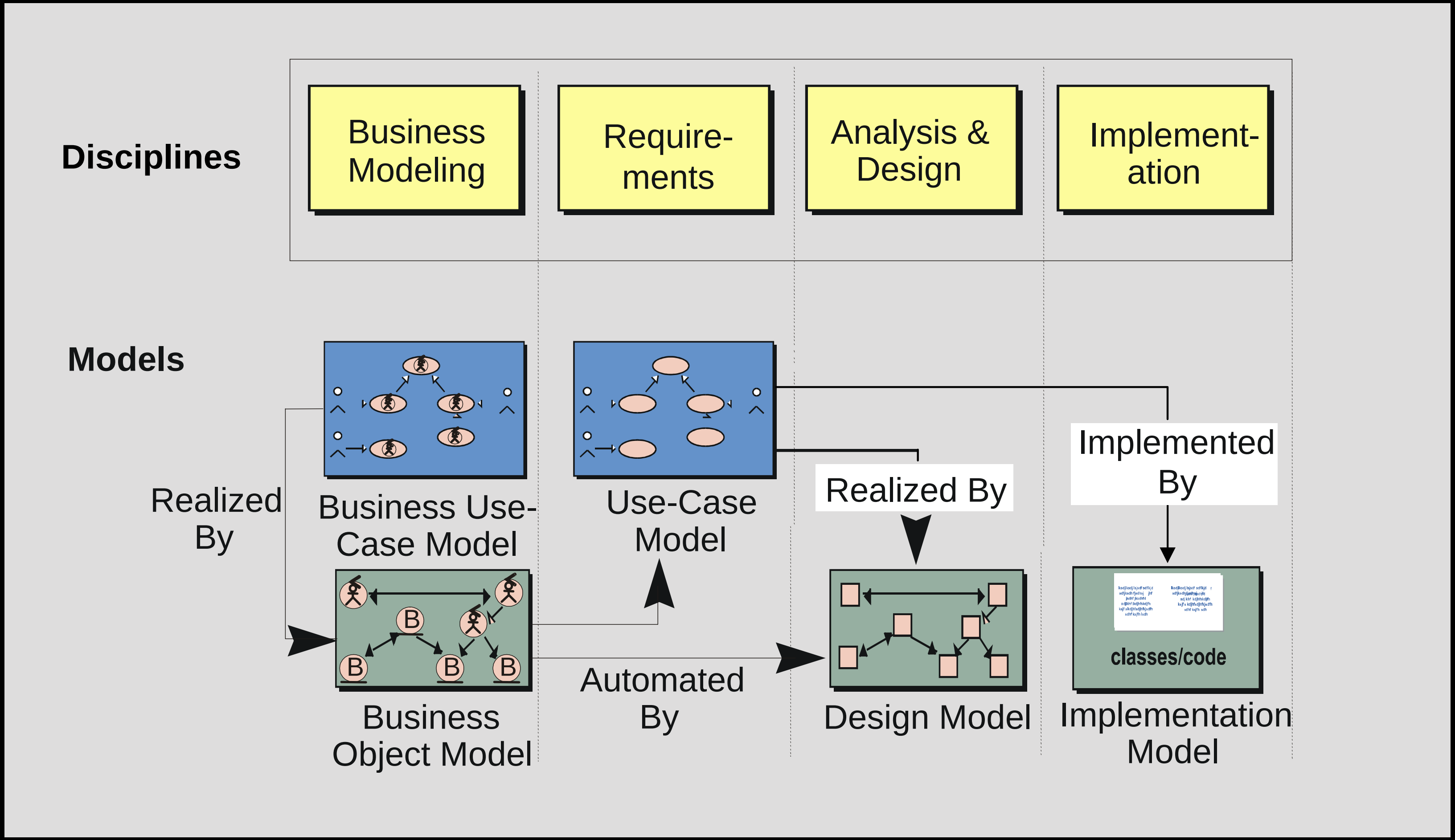
Bringing It All Together: The Iterative Approach



In an **iteration**, you walk through all disciplines.

Disciplines group activities logically.

Disciplines Produce Models



Disciplines

Business Modeling

Requirements

Analysis & Design

Implementation

Models

Realized By

Business Use-Case Model

Use-Case Model

Realized By

Implemented By

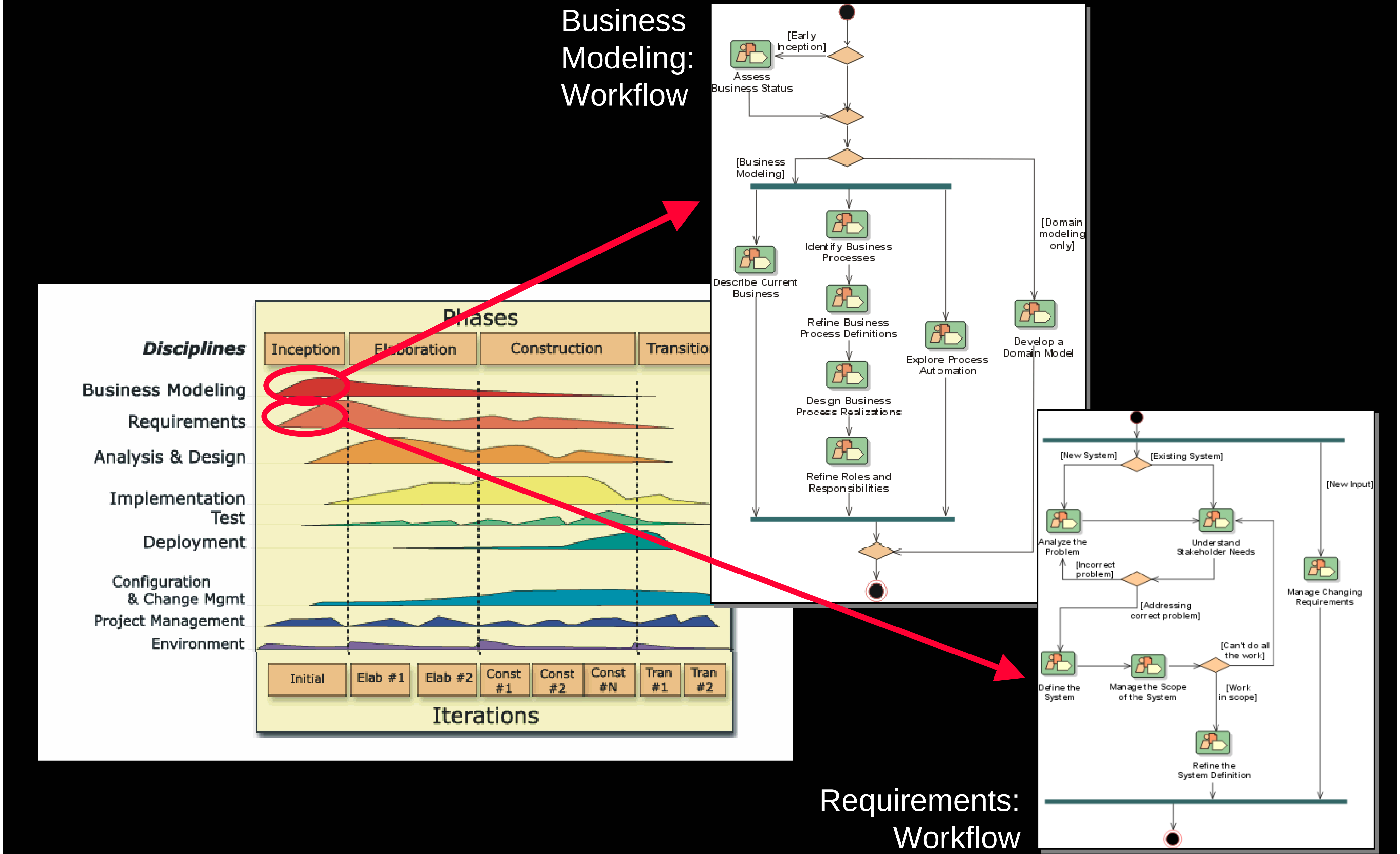
Business Object Model

Automated By

Design Model

Implementation Model

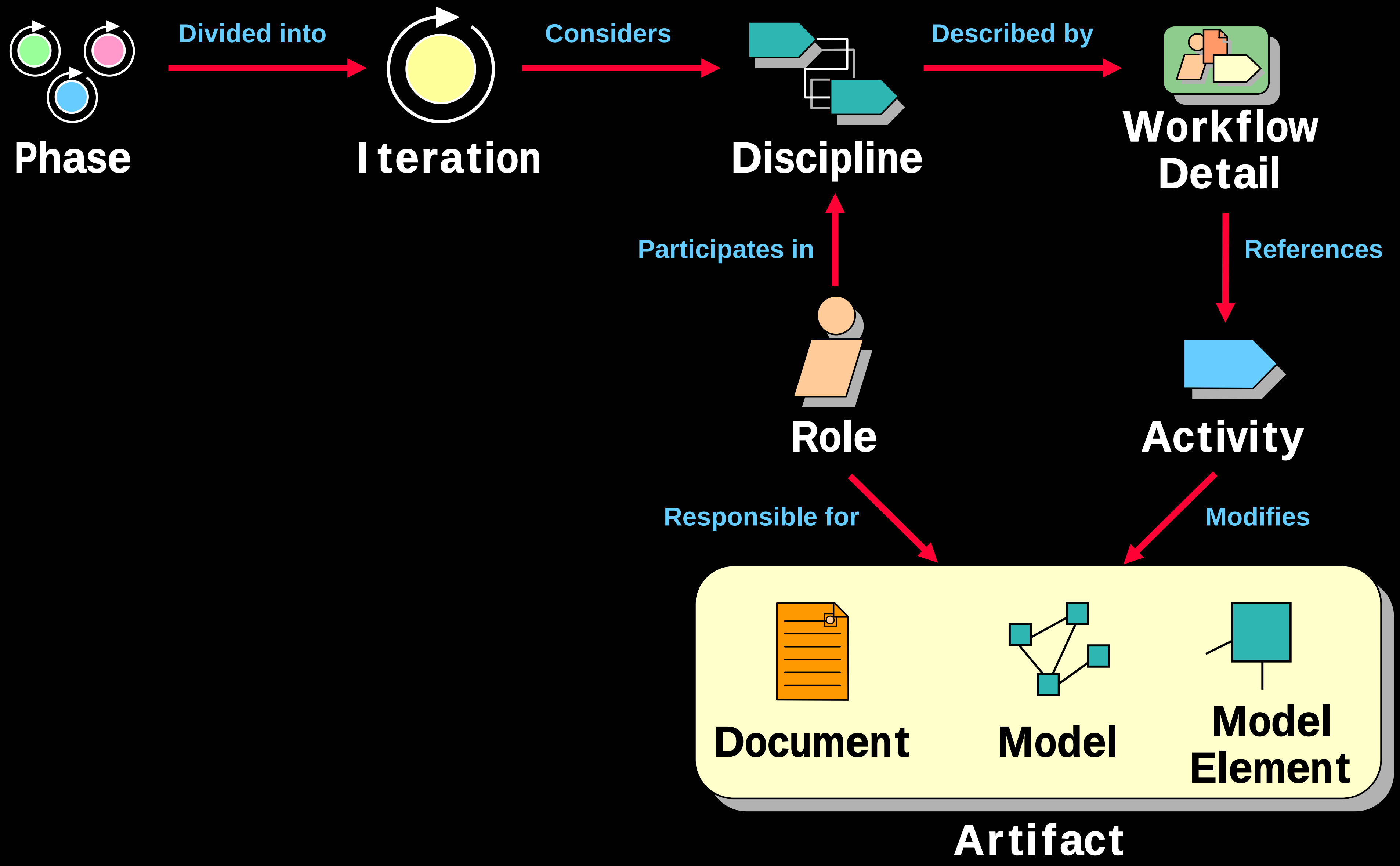
Disciplines Guide Iterative Development



Business Modeling: Workflow

Requirements: Workflow

Overview of Rational Unified Process Concepts



Review

- Best Practices guide software engineering by addressing root causes.
- Best Practices reinforce each other.
- Process guides a team on who does what, when, and how.
- The Rational Unified Process is a means of achieving Best Practices.