Software analysis and design

Module 1: Best practices of Software Engineering

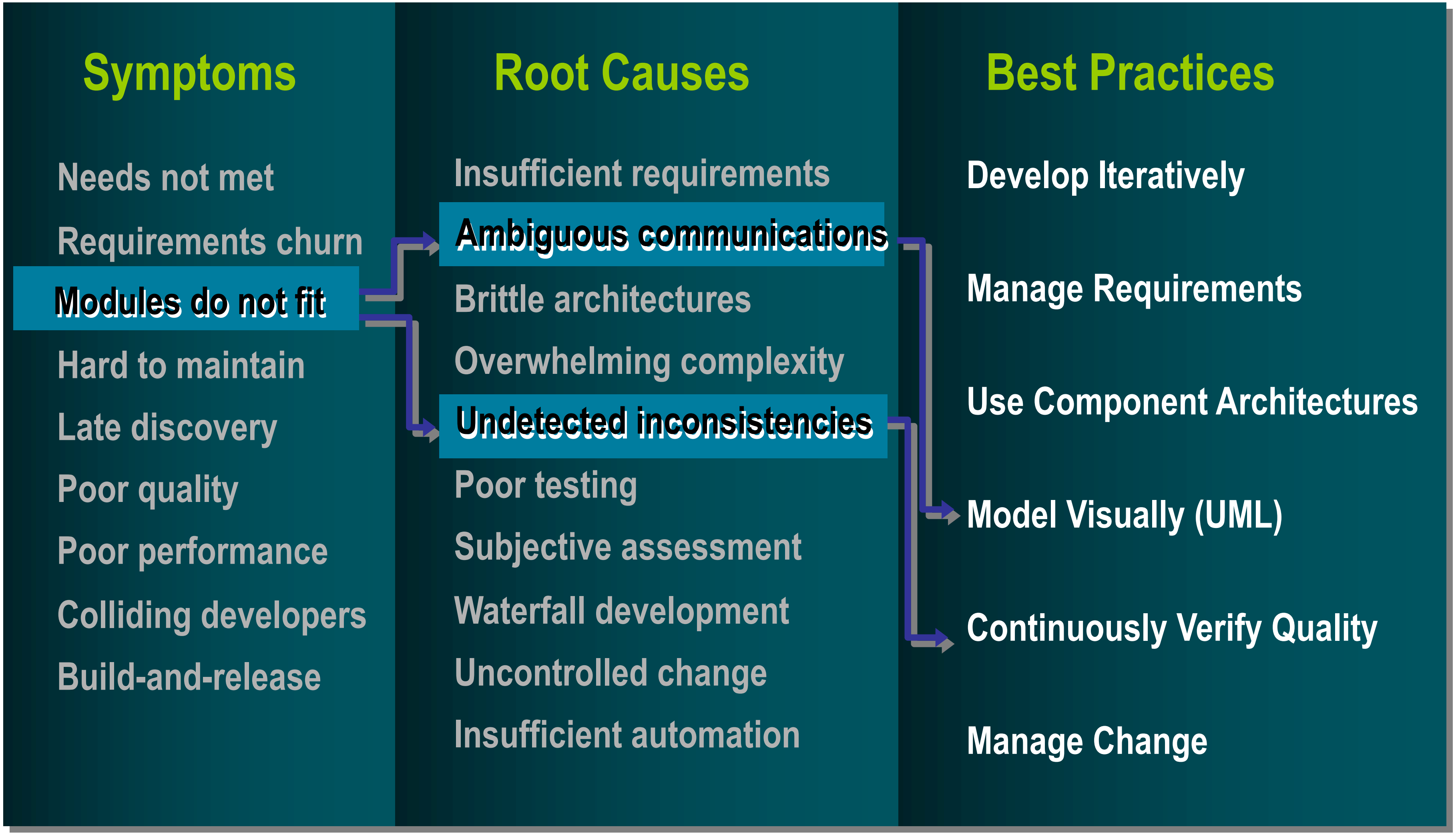
Objectives: Best Practices

- Identify symptoms of software development problems
- Explain the Best Practices
- Present the Rational Unified Process (RUP) within the context of the 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

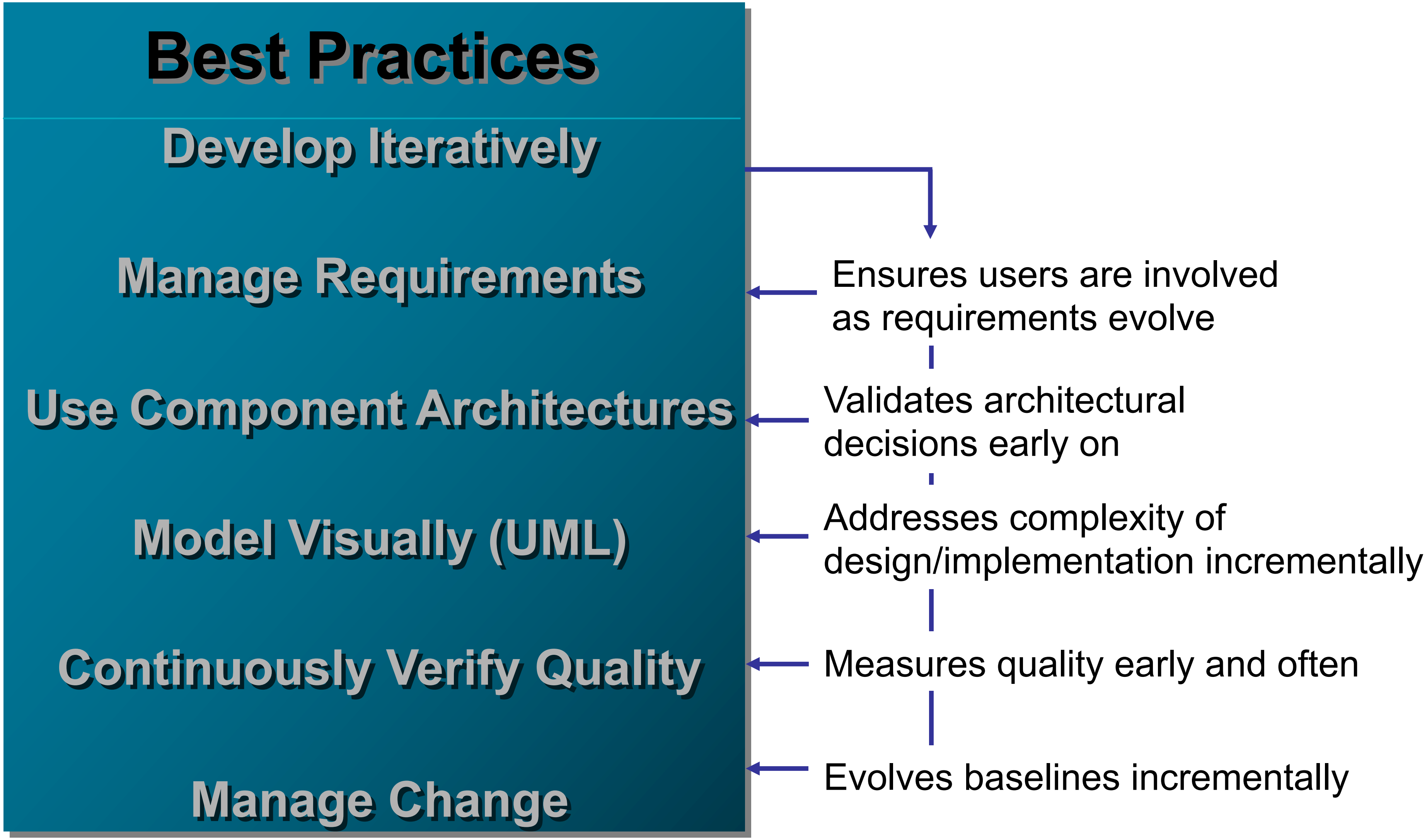
Root Causes

- 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

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

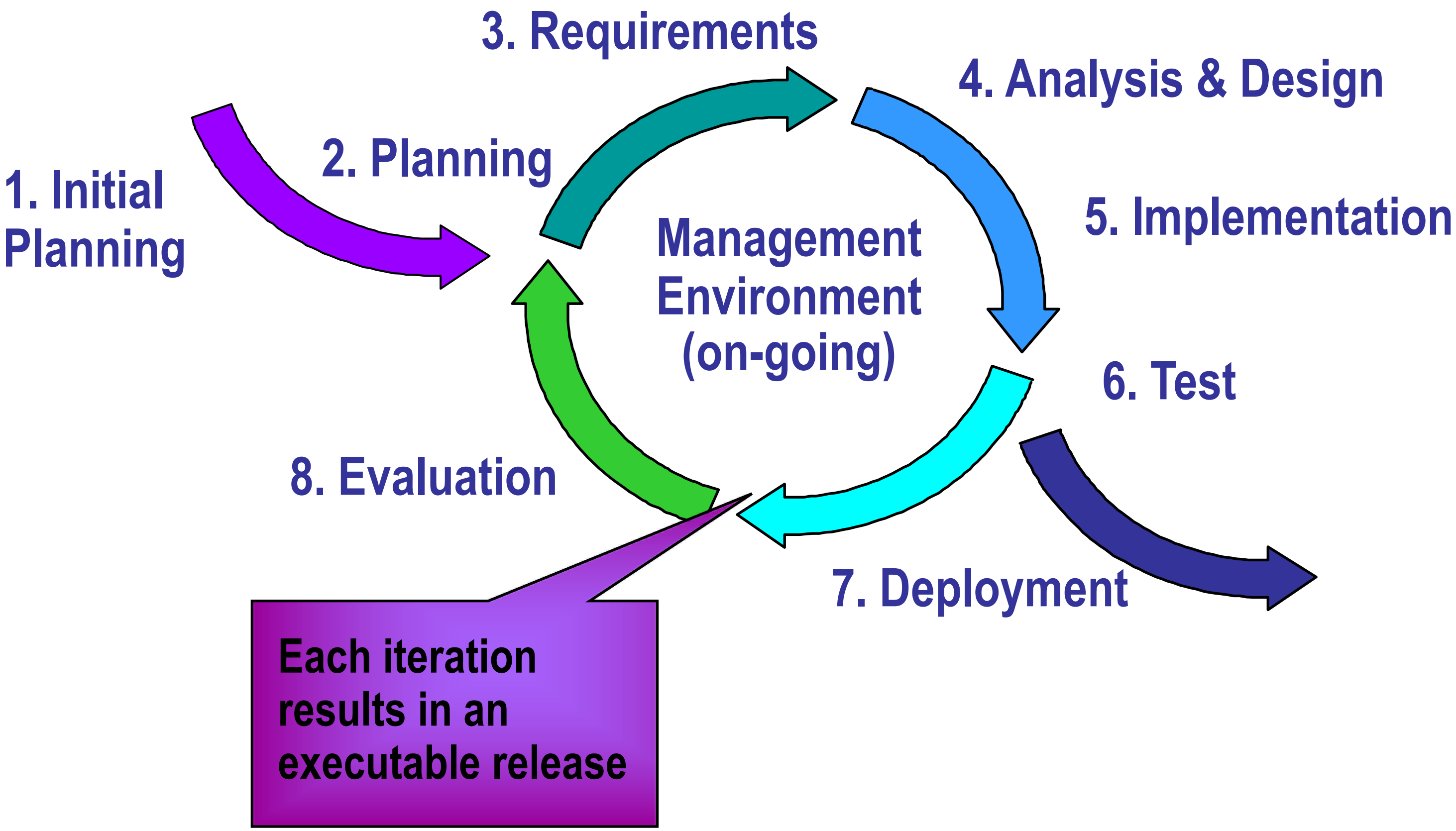
Addresses complexity of design/implementation incrementally

Measures quality early and often

Evolves baselines incrementally

Develop Iteratively

- Iterative development produces an executable



Each iteration results in an executable release

Managing Requirements

Ensures that 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.

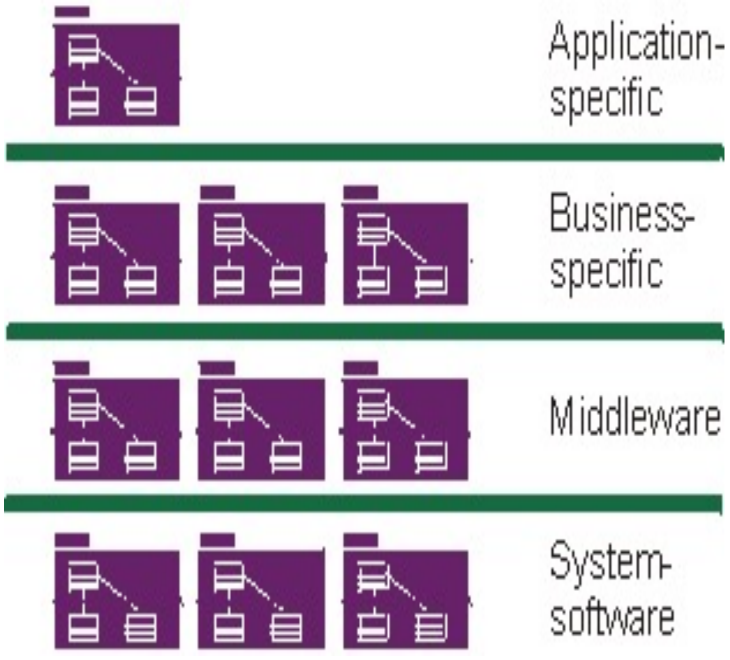
Use Component Architectures

- Software architecture needs to be:

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

Purpose of a Component-Based Architecture

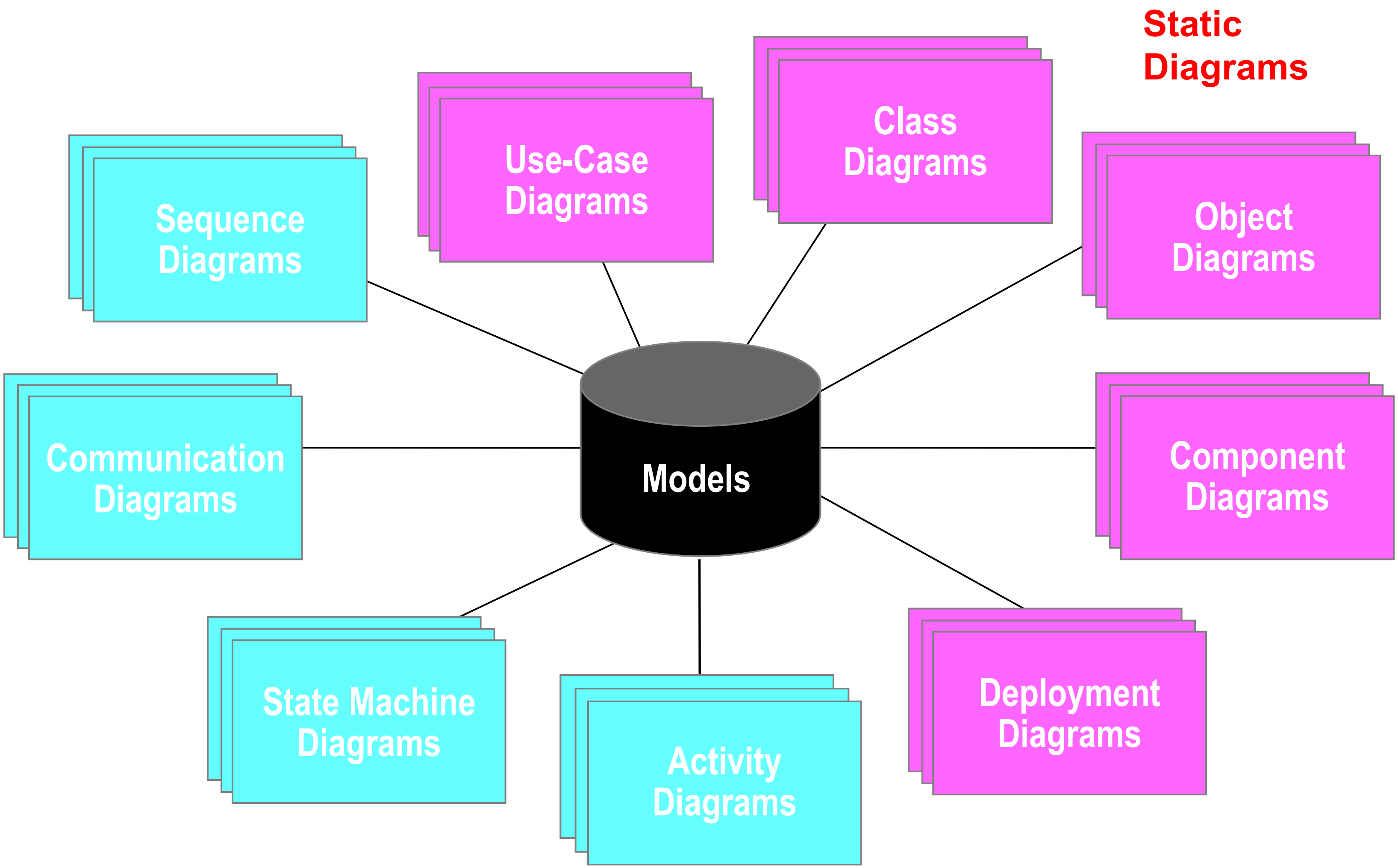
- Basis for reuse
  - Component reuse
  - Architecture reuse
- Basis for project management
  - Planning
  - Staffing
  - Delivery
- Intellectual control
  - Manage complexity
  - Maintain integrity



Model Visually (UML)

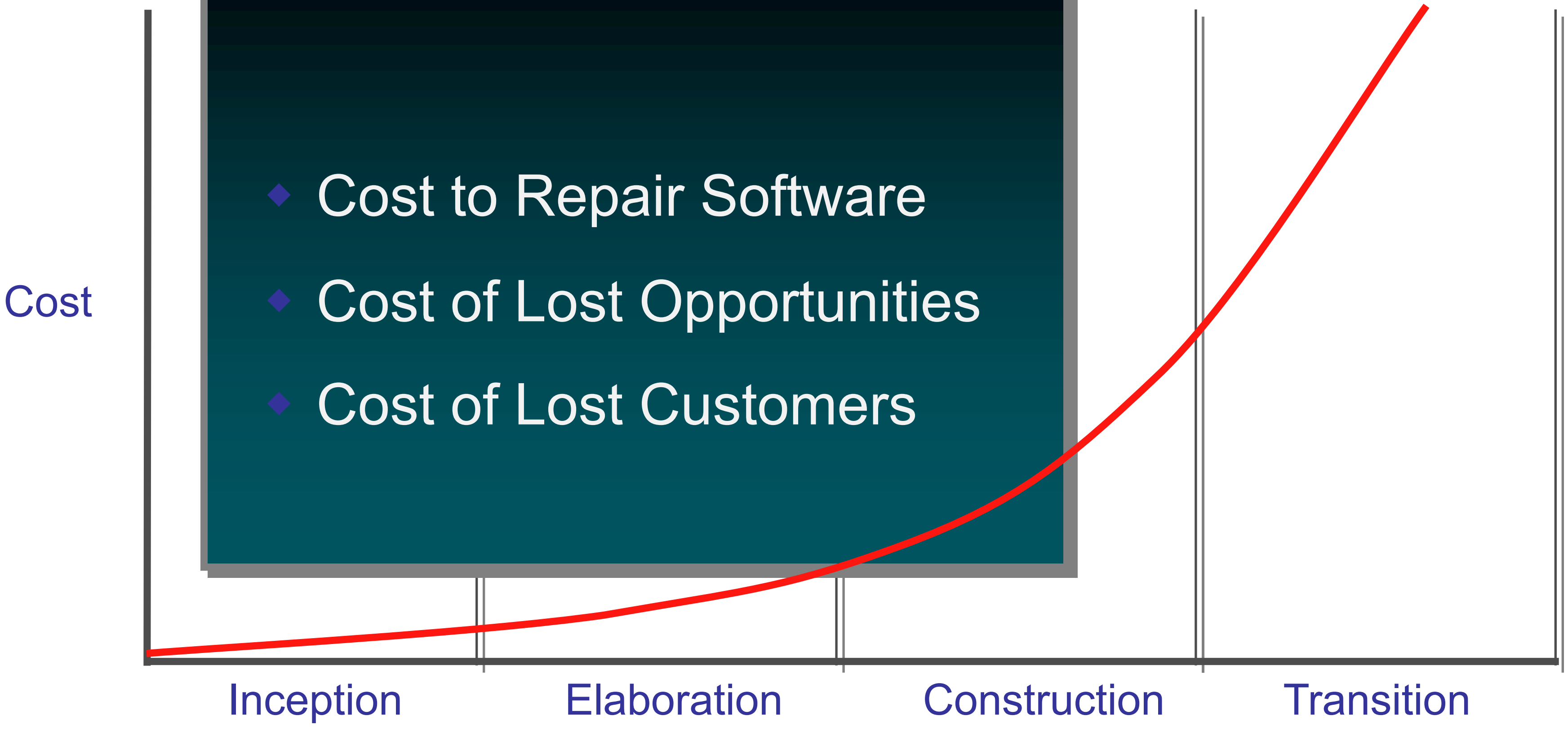
- Captures structure and behavior
- Shows how system elements fit together
- Keeps design and implementation consistent
- Hides or exposes details as appropriate
- Promotes unambiguous communication
  - The UML provides one language for all practitioners.

Visual Modeling with the Unified Modeling Language



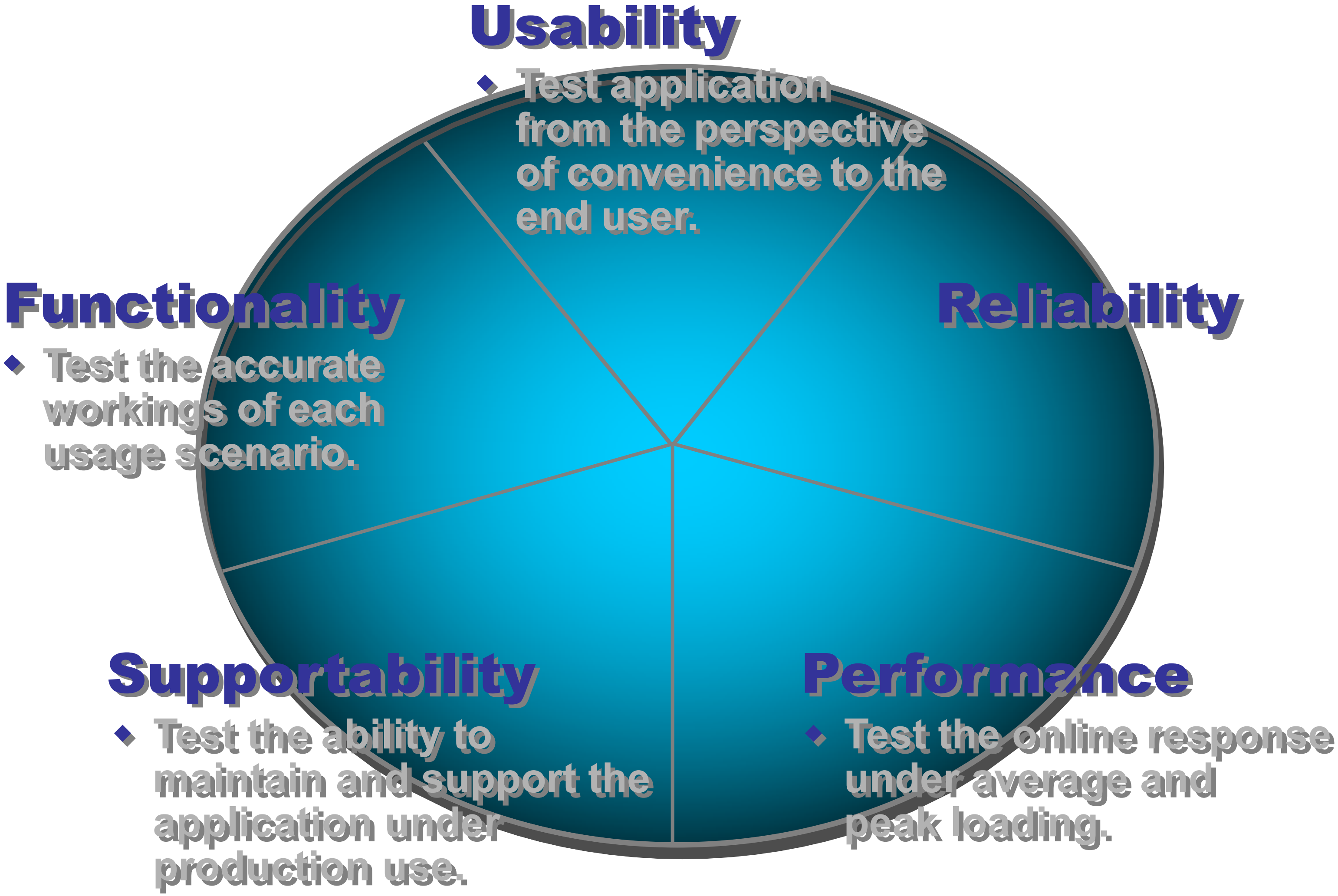
Continuously Verify Quality

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 the end user.

Functionality

- Test the accurate workings of each usage scenario.

Reliability

Supportability

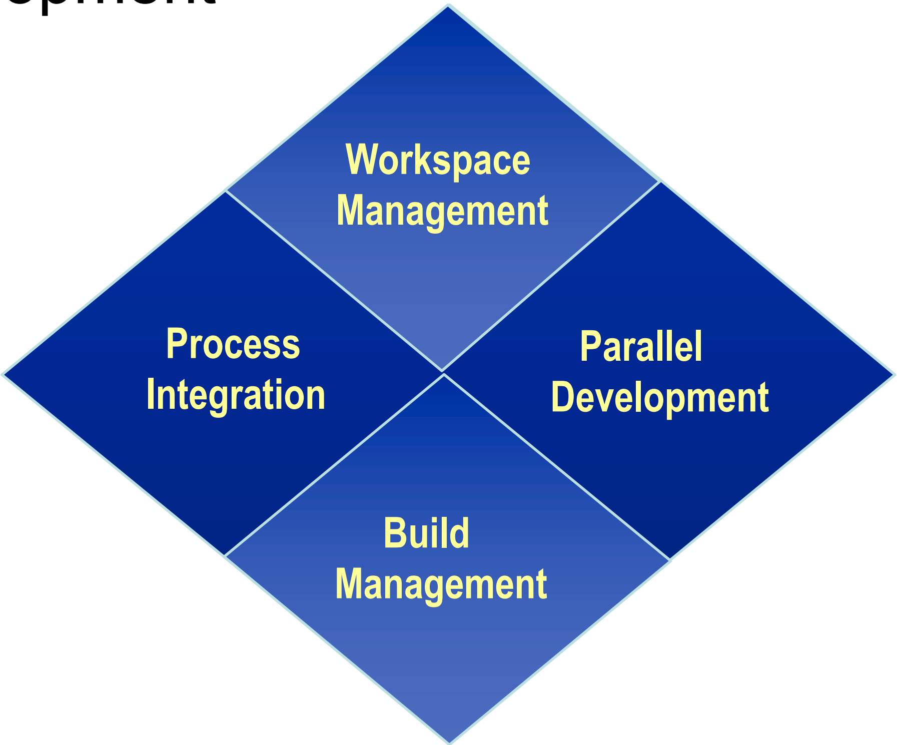
- Test the ability to maintain and support the application under production use.

Performance

- Test the online response under average and peak loading.

Manage Change

- To avoid confusion, have:
  - Secure workspaces for each developer
  - Automated integration/build management
  - Parallel development



Manage Change (continued)

- Unified Change Management (UCM) involves:
  - Management across the lifecycle
    - System
    - Project management
  - Activity-based management
    - Tasks
    - Defects
    - Enhancements
  - Progress tracking
    - Charts
    - Reports

Rational Unified Process Implements Best Practices

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

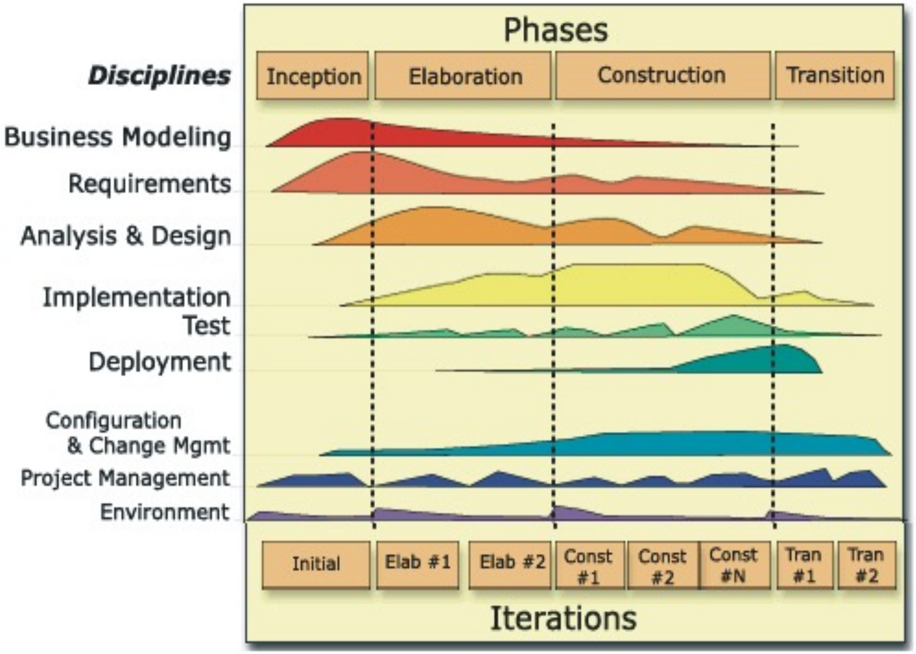
Model Visually (UML)

Continuously Verify Quality

Manage Change

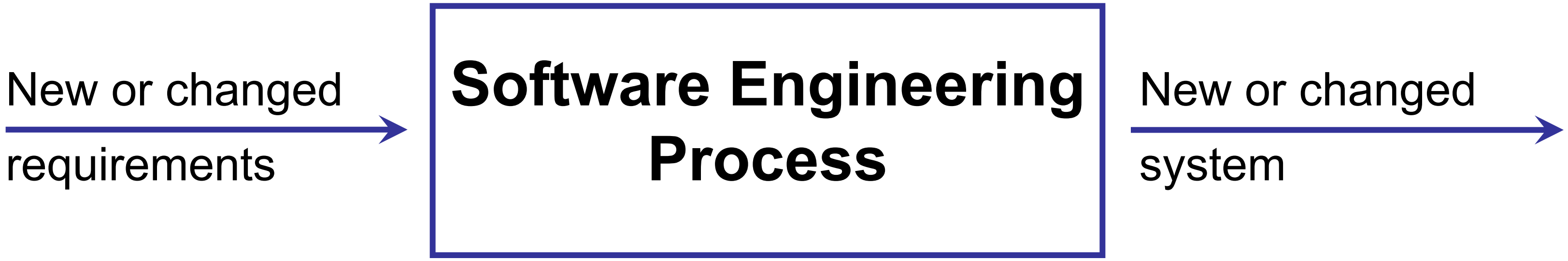
Achieving Best Practices

- 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

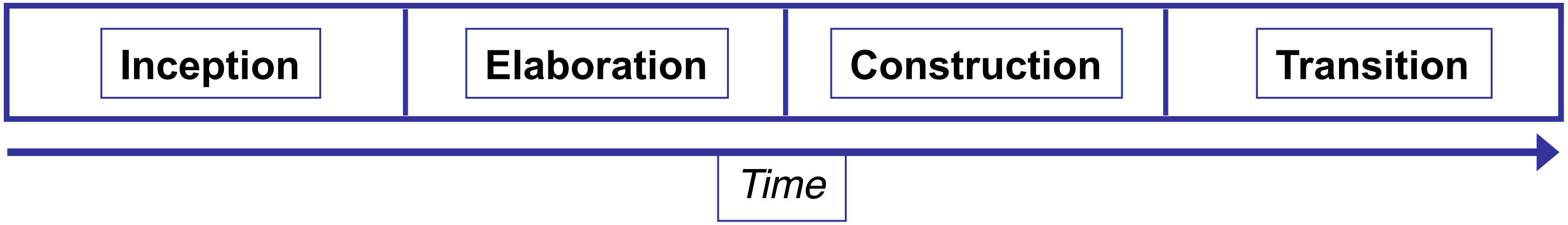
Software Engineering Process

New or changed system

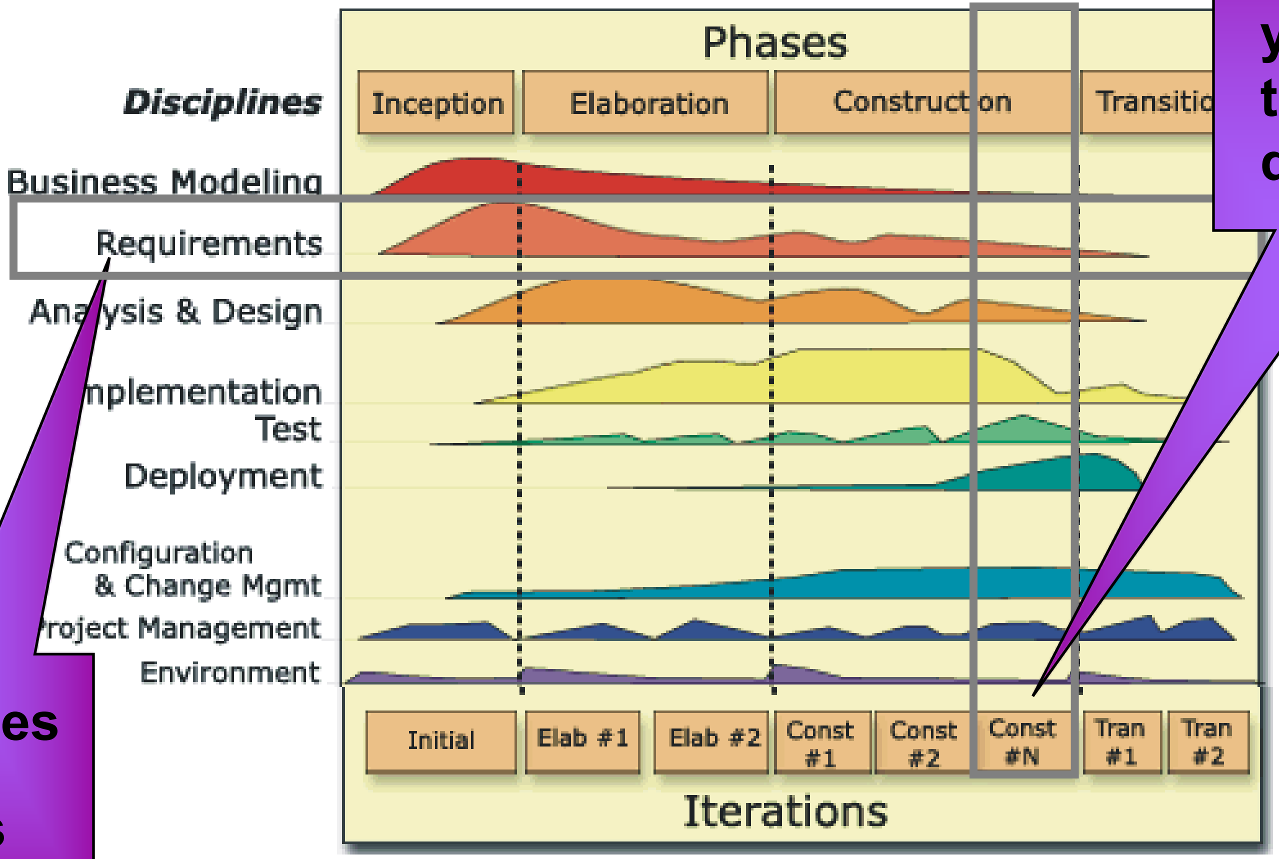
Process Structure - Lifecycle Phases

The Rational Unified Process has four phases:

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



Bringing It All Together: The Iterative Approach



In an iteration, you walk through all disciplines.

Disciplines group activities logically.

Summary

- 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.