Lesson 2

Software Engineering Best Practices: *Action in Accord with All the Laws of Nature*

Topics: Principles of Software Engineering

- The inherent challenge of software engineering
- The importance of analysis and design
- Best practices

Software Development Historical Analysis

- No Silver Bullet – Brooks (1986)
- Essential nature of SW engineering vs. the non-essential
- Non-essential -- E.g., syntax of programming languages
- Major advances already made for non-essential aspects
  - High level languages, IDEs, Application platforms,
  - Software and GUI tools, builds, source control, etc.

Essential Difficulties

- Complexity
  - Combinatorial complexity of states and processes
  - Sheer size
  - Interactive, distributed, networked
- Conformity
  - Expect software to conform to needs of hardware and business domains, even if a more "logical" approach would involve changing these external interfaces

Essential Difficulties (cont)

- Changeability
  - Expect to modify incredibly complex systems
  - Many pressures to modify software: reality changes, functionality extensions, easier than hardware, lasts longer
- Invisibility
  - Software is abstract compared to to buildings, bridges, …
  - Difficult for visual diagrams to capture

Software Development Historical Analysis

- 1994 – W.W. Gibbs, "Software's Chronic Crisis", *Scientific American*, Sept., 1994
  - 25% of large scale SW projects cancelled
  - Average delivery is late by 50% (worse for large scale projects)
  - 75% of large scale SW projects – do not function as intended or are not even used

The Need for a Software Engineering "Process"

1999 - (Jacobson, *The Unified Software Development Process,* 1999, pp.3-4)

"The software problem boils down to the difficulty developers face in pulling together the many strands of a large software undertaking. The software development community needs a controlled way of working. It needs a process that integrates the many facets of software development. It needs a common approach, a process that :

- Provides guidance to the order of a team's activities.
- Directs the tasks of individual developers and the team as a whole.
- Specifies what artifacts should be developed.
- Offers criteria for monitoring and measuring a project's products and activities.

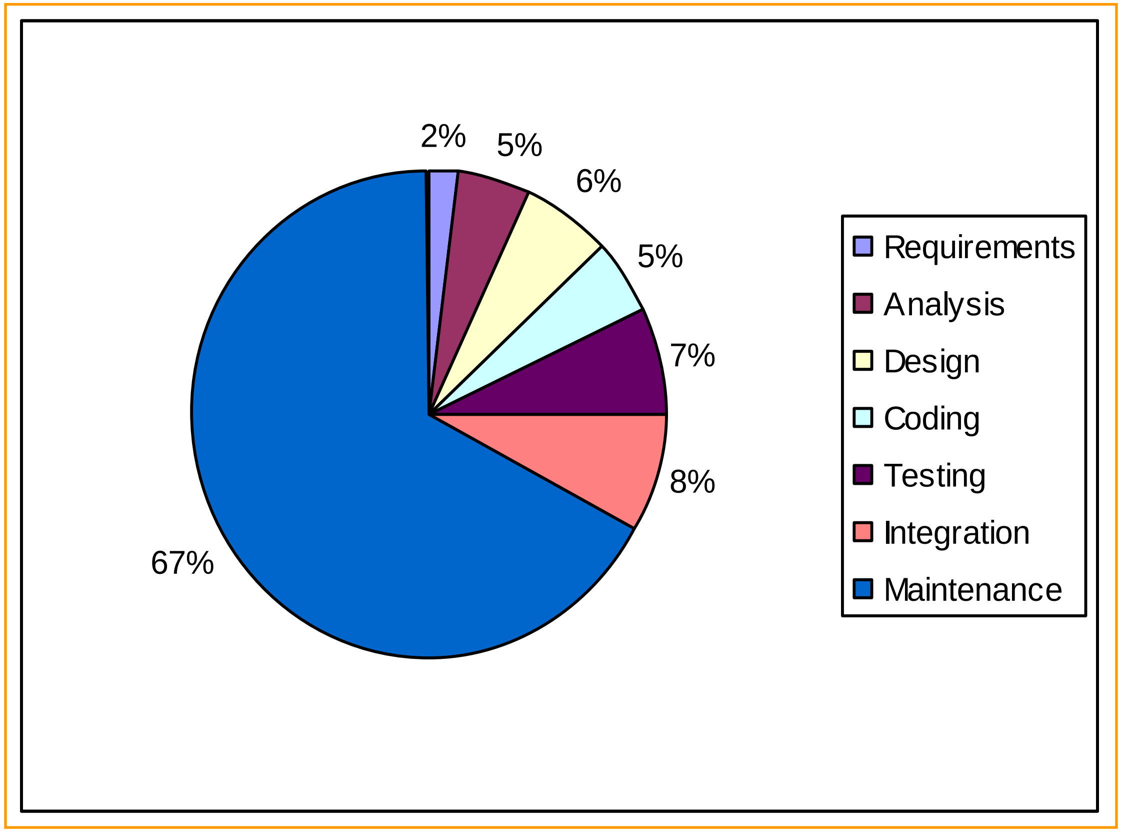
"The presence of a well-defined and well-managed process is a key discriminator between hyperproductive projects and unsuccessful ones."

Topics: Principles of Software Engineering

- The inherent challenge of software engineering
- The importance of analysis and design
- Best practices

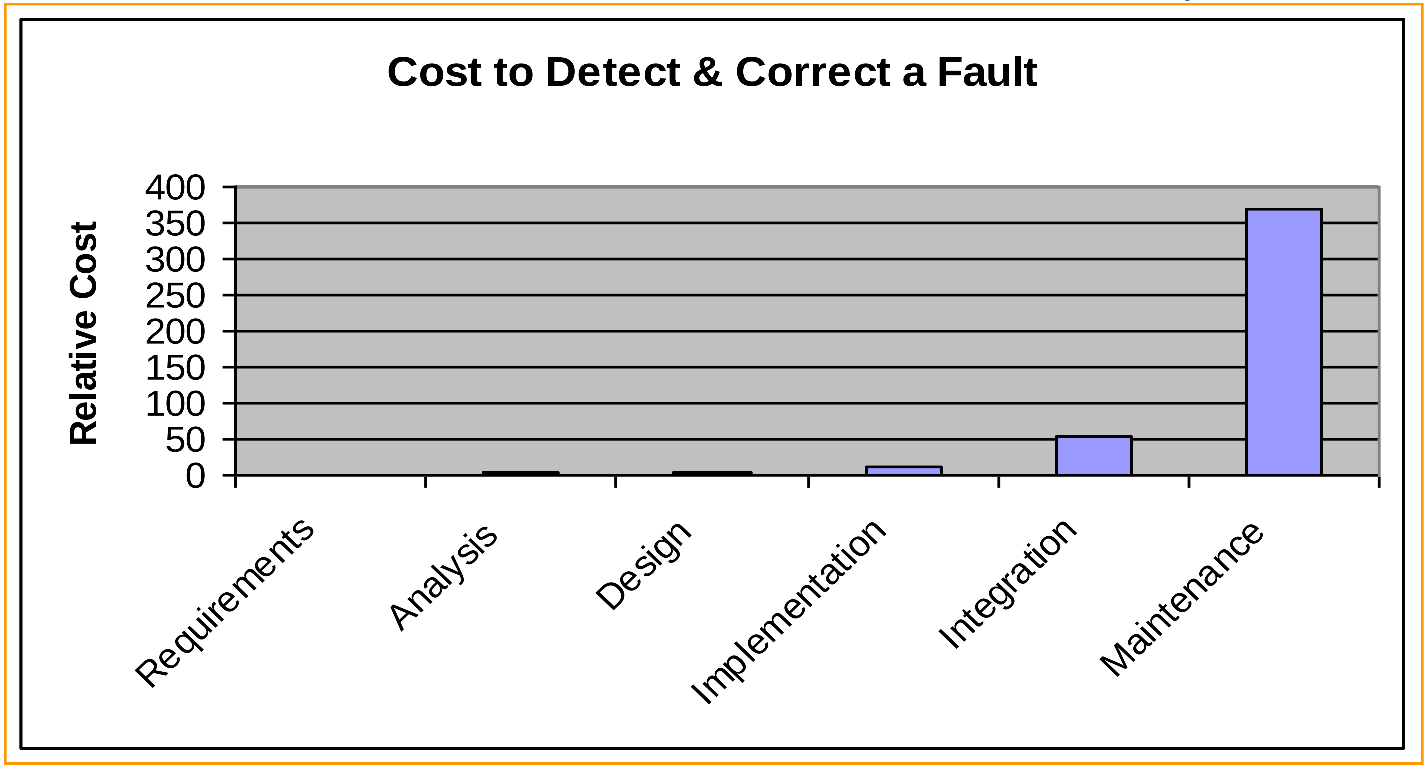
Importance of Maintainability

- 2/3 of life-cycle cost is in system maintenance (for successful projects)
- Only about 5% total costs for coding

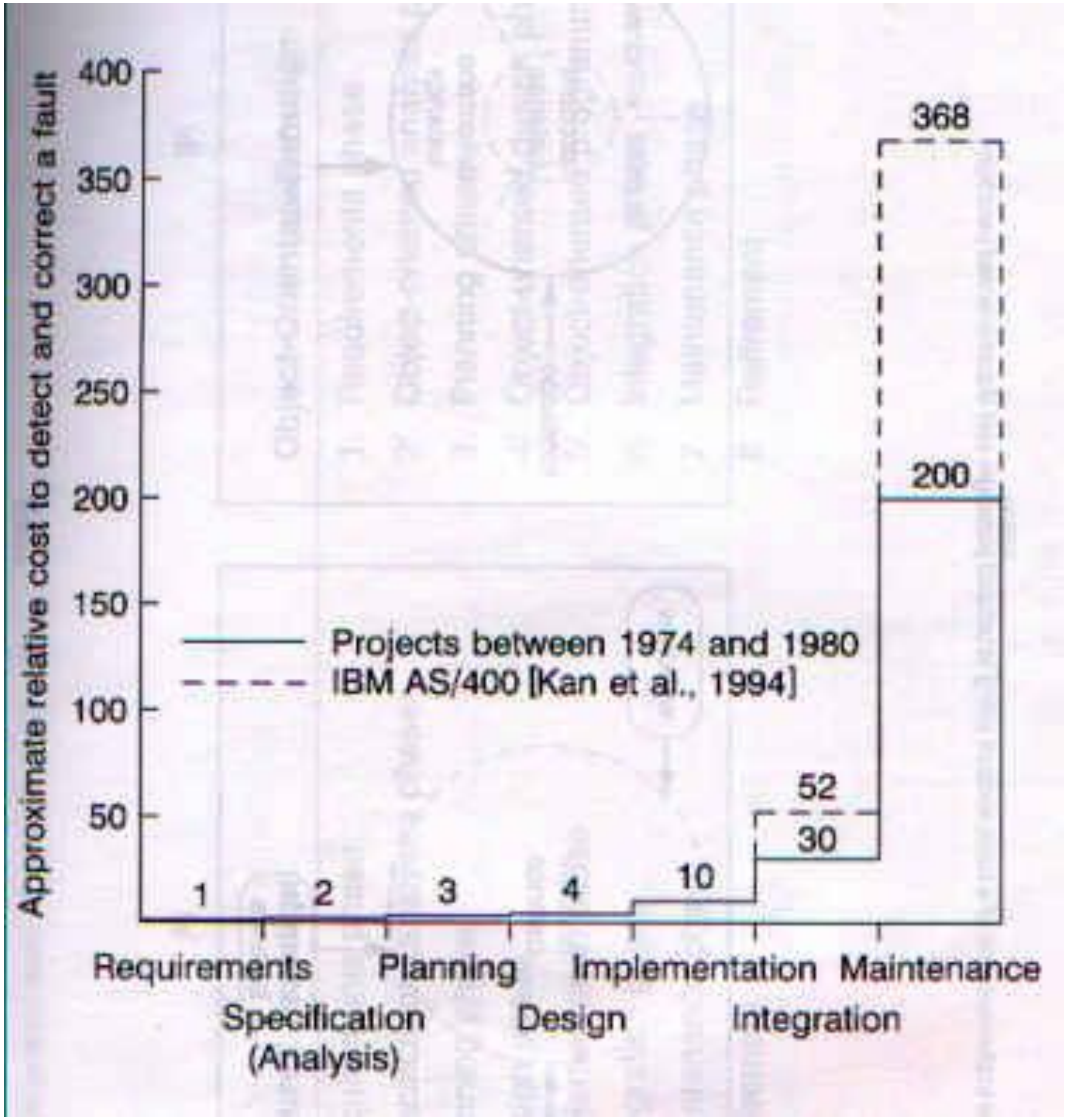


Importance of a Good Start

- Fixing faults is much cheaper earlier in life-cycle than later
- 60-70% faults in large projects specification/design faults.
  - Good A&D facilitate integration and maintenance
  - Greatest potential area for improvement and payback



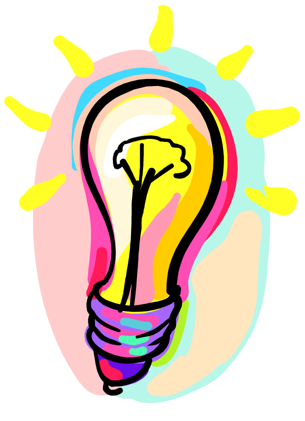
Increasing Costs to Detect and Correct Faults



Summary Points

- Efficient development of quality software in a timely manner is inherently difficult
- Two-thirds of all the faults in large scale projects have been observed to be specification or design faults. Two-thirds of the life-cycle costs of a software system are incurred during the maintenance phase.

Spec and design faults



Maintenance costs

Topics: Principles of Software Engineering

- The inherent challenge of software engineering
- The importance of analysis and design
- Best practices

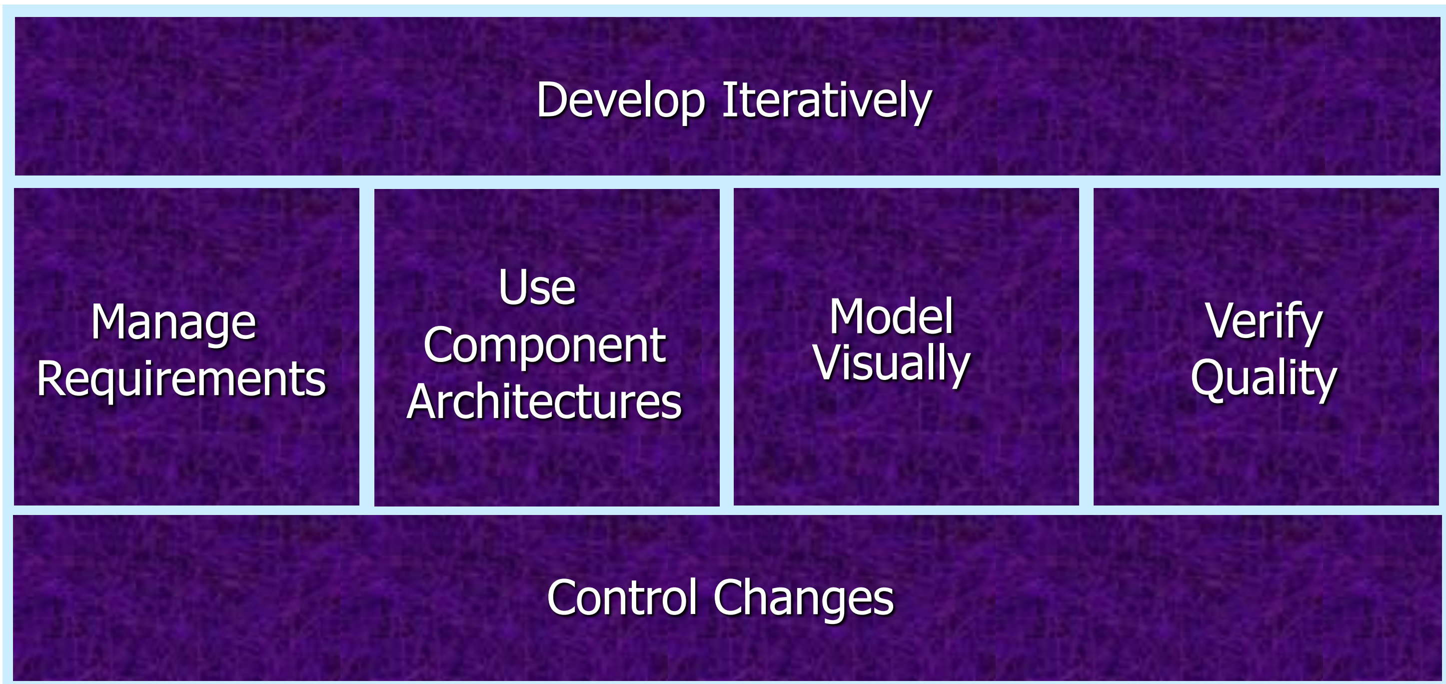
Greatest Potential (Brooks)

- Rapid prototyping
- Incremental development
- Develop great designers

Value of OO Techniques

- Widely accepted in the field as supporting good software engineering principles and practices
- Analysis and design closely integrated with implementation
  - Directly addresses 60-70% faults in specs and design
- Objects support greater integrity of software components
  - Encapsulate implementation details
  - Leads to reusability, extensibility, and maintainability
  - Addresses other area of maximum potential payback (maintenance)

Unified Process Delivers Best Practices



Visually Model Software

- Addresses one of Brooks' "essential" difficulties (inherent abstractness of software)
- Helps to visualize, specify, construct and document structure and behavior of a system
- Using a standard modeling language (like UML) helps to maintain consistency among system's artifacts and facilitates communication among team members

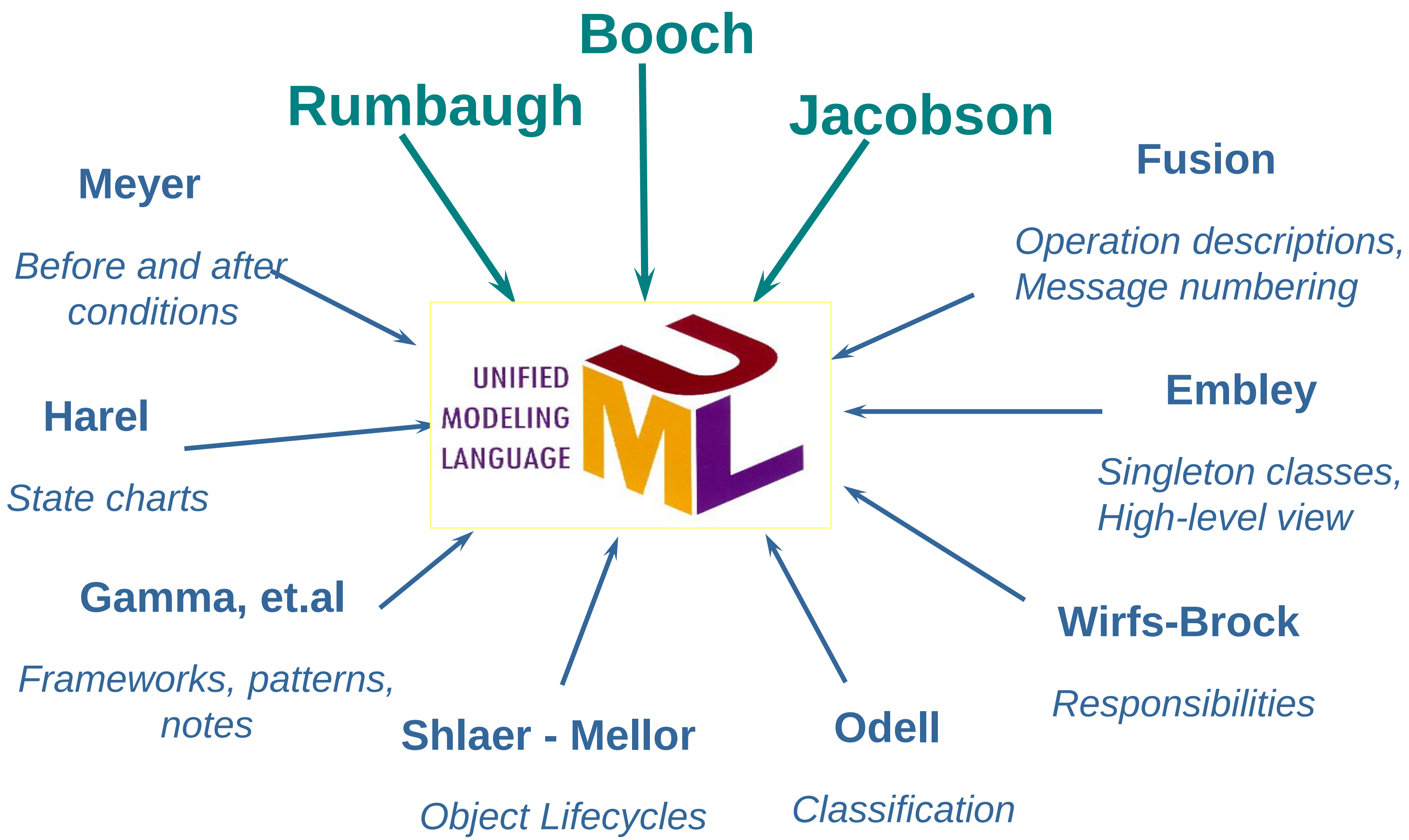
K12-13

What Is the UML?

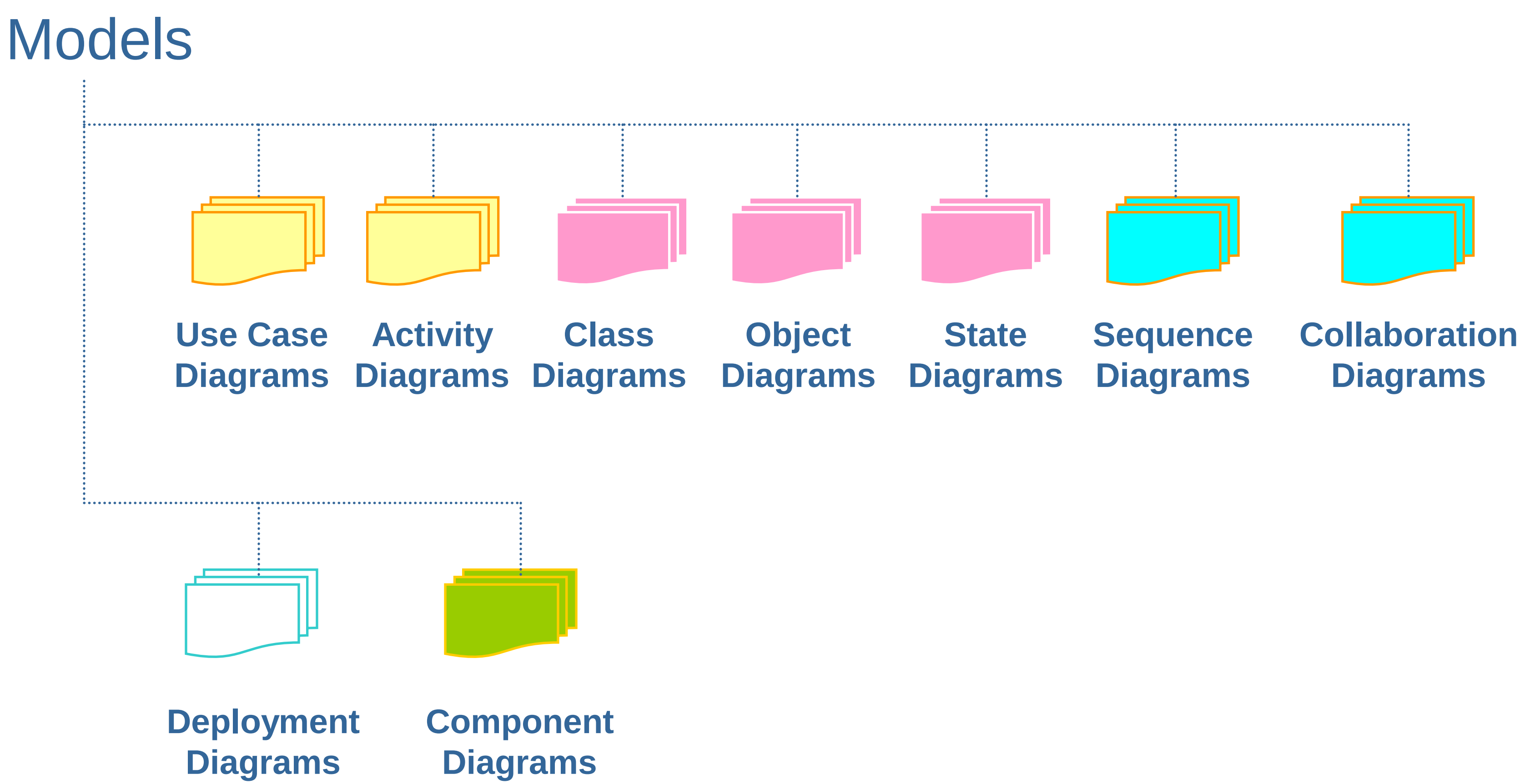
- The Unified Modeling Language (UML) is a language for
  - Specifying
  - Visualizing
  - Constructing
  - Documenting

  the artifacts of a software-intensive system

Inputs to UML



The UML Provides Standardized Diagrams



Best Practices Address Root Causes

- Visually model software
- Develop software iteratively
- Use component-based architectures
- Manage requirements
- Verify software quality
- Control changes to software

K5-6

What Is a Software Development Process?

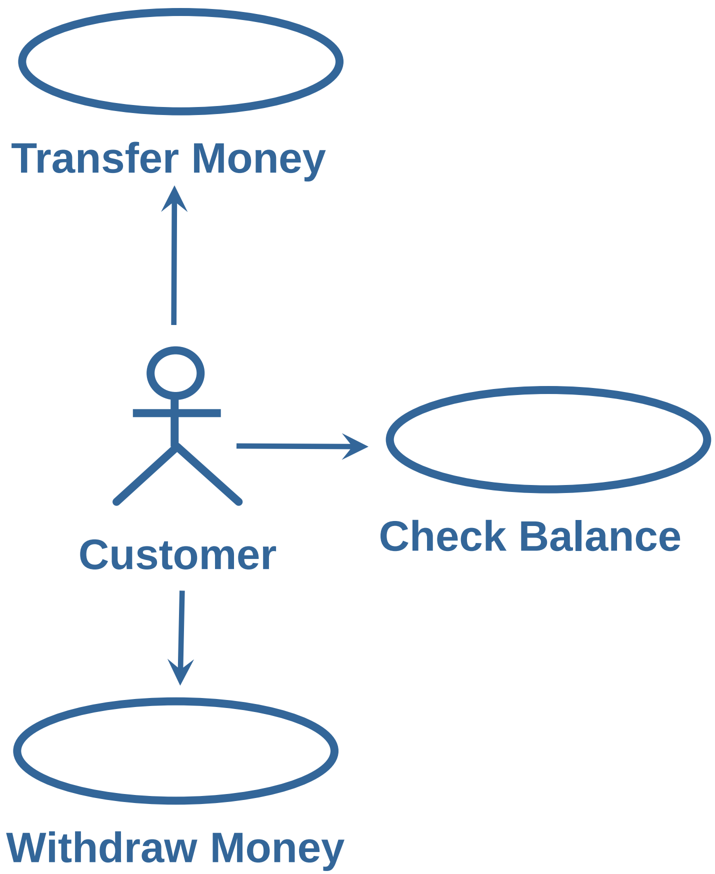
A software development process is the set of activities needed to transform a user's requirements into a software system. The process defines *Who* is doing *What, When,* as well as *How* to reach a certain goal.

J4

RUP Key Features

- Use case driven
- Iterative and incremental
  - Models, workflows, phases, and iterations
- Architecture centric

Rational Unified Process Is Use-Case Driven



Use-Cases for an Automated Teller Machine

- An **actor** is someone or something outside the system that interacts with the system

- A **Use-Case** is a sequence of actions a system performs that yields an observable result of value to a particular actor

Use-Cases Include a Flow of Events

Flow of events for the Withdraw Money Use-Case

1. The Use-Case begins when the client inserts an ATM card. The system reads and validates information on the card.
2. The system prompts for the PIN. Client enters PIN. The system validates the PIN.
3. The system asks which operation the client wishes to perform. Client selects “Cash withdrawal.” System requests amount.
4. Client enters amount. System requests the account type.
5. Client selects account type (checking, savings, credit). The system communicates with the ATM network . . .

Benefits of a Use-Case Driven Process

- Use-Cases are concise, simple, and understandable by a wide range of stakeholders
  - End users, developers and acquirers understand functional requirements of the system
- Use-Cases drive numerous activities in the process:
  - Creation and validation of the design model
  - Definition of test cases and procedures of the test model
  - Planning of iterations
  - Creation of user documentation
- Use-Cases help synchronize the content of different models

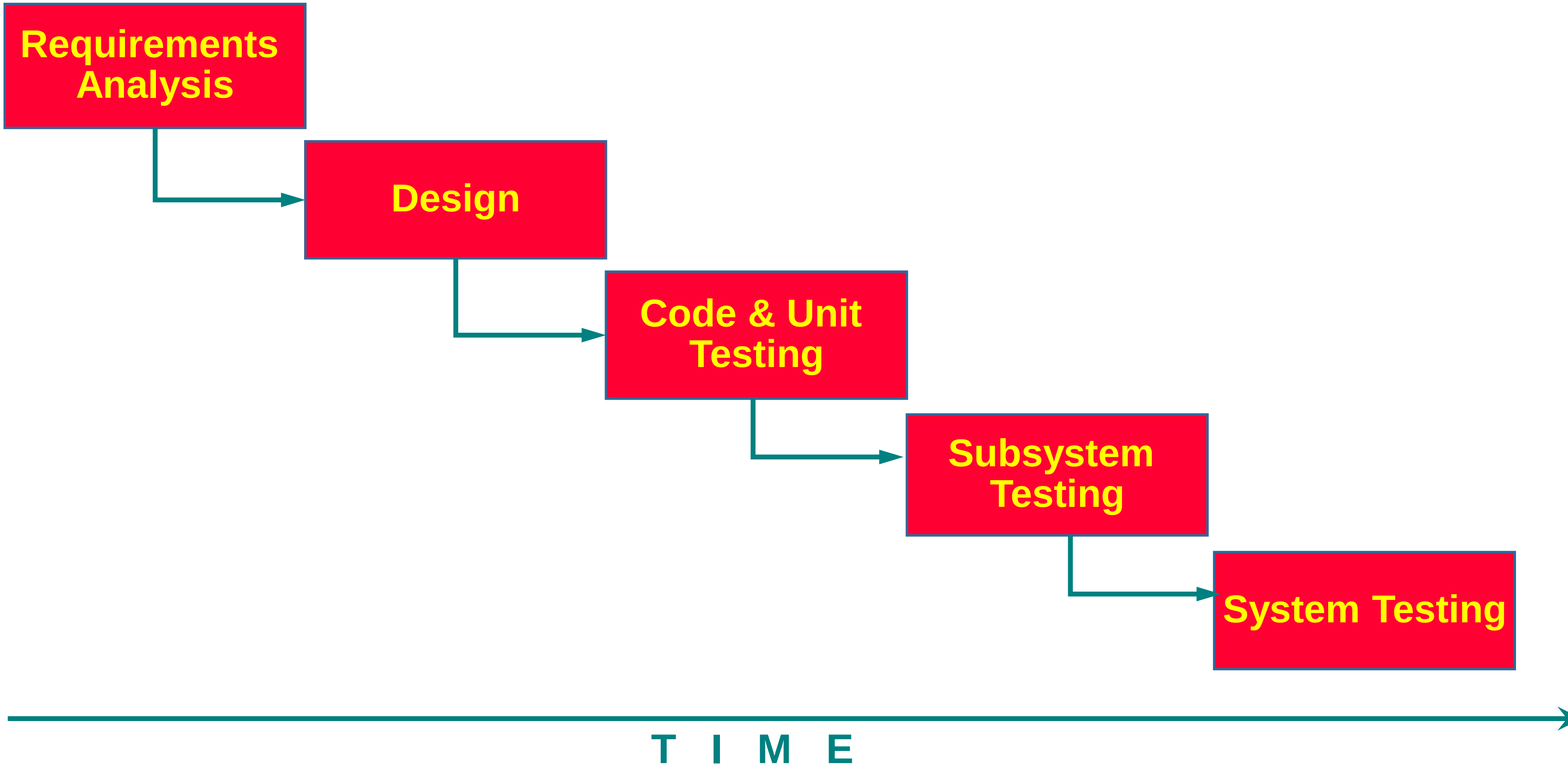
RUP Key Features

- Use case driven
- Iterative and incremental
  - phases, iterations, and workflows
- Architecture-centric

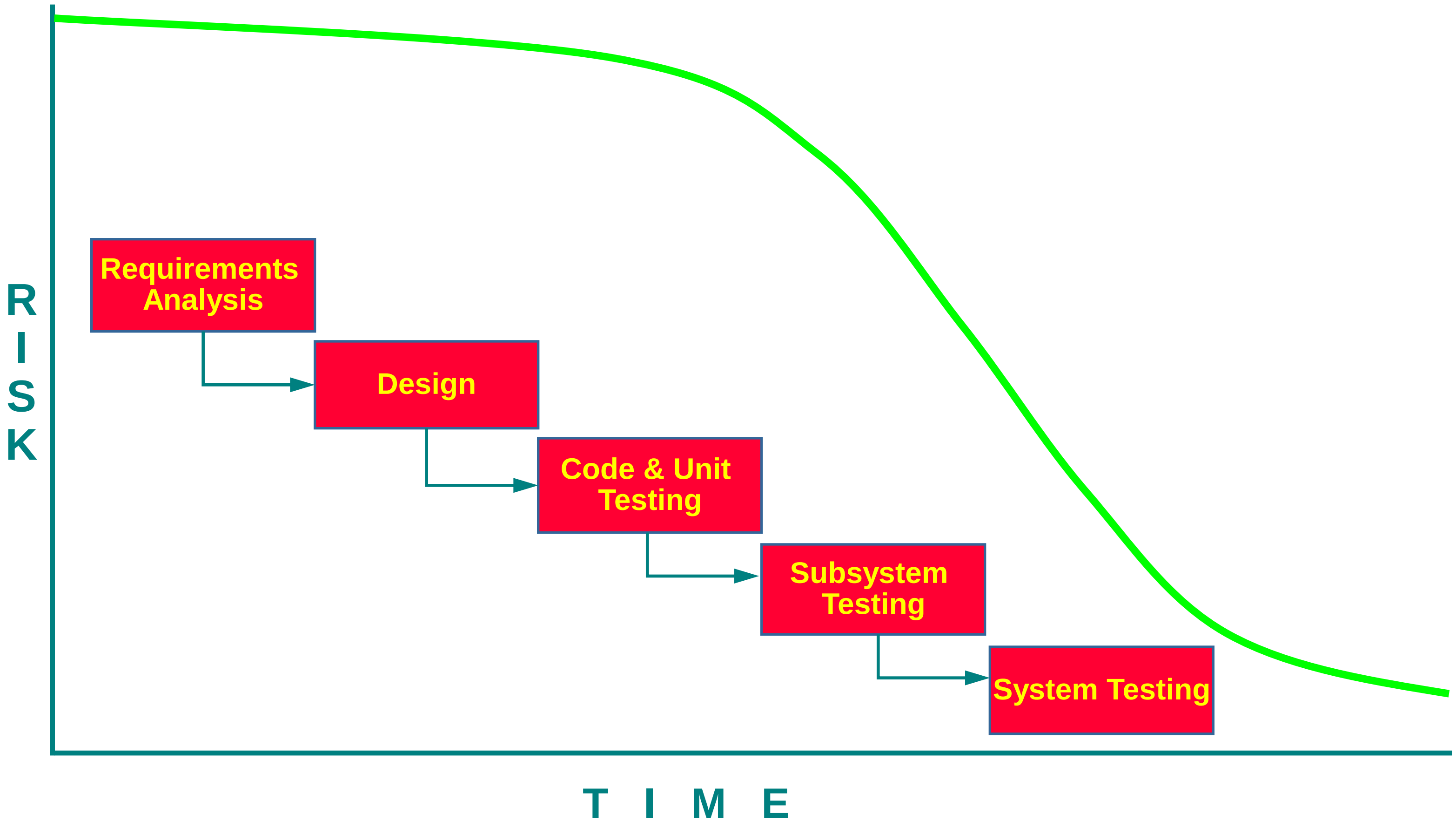
Develop Software Iteratively

- An initial design will likely be flawed with respect to its key requirements
- Late-phase discovery of design defects results in costly over-runs and/or project cancellation

Traditional Waterfall Development

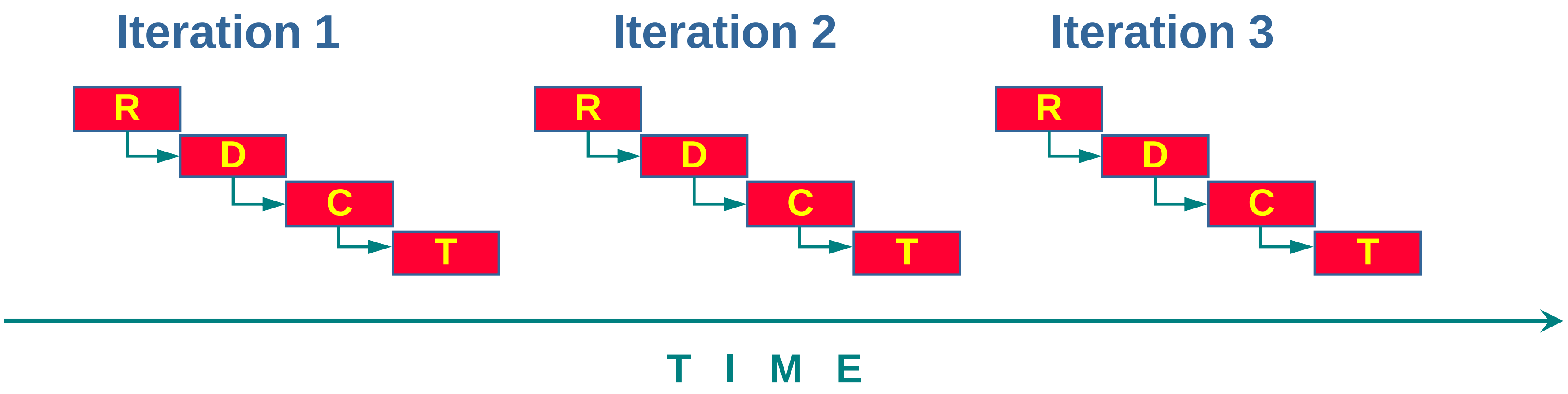


Waterfall Development Delays Reduction of Risk



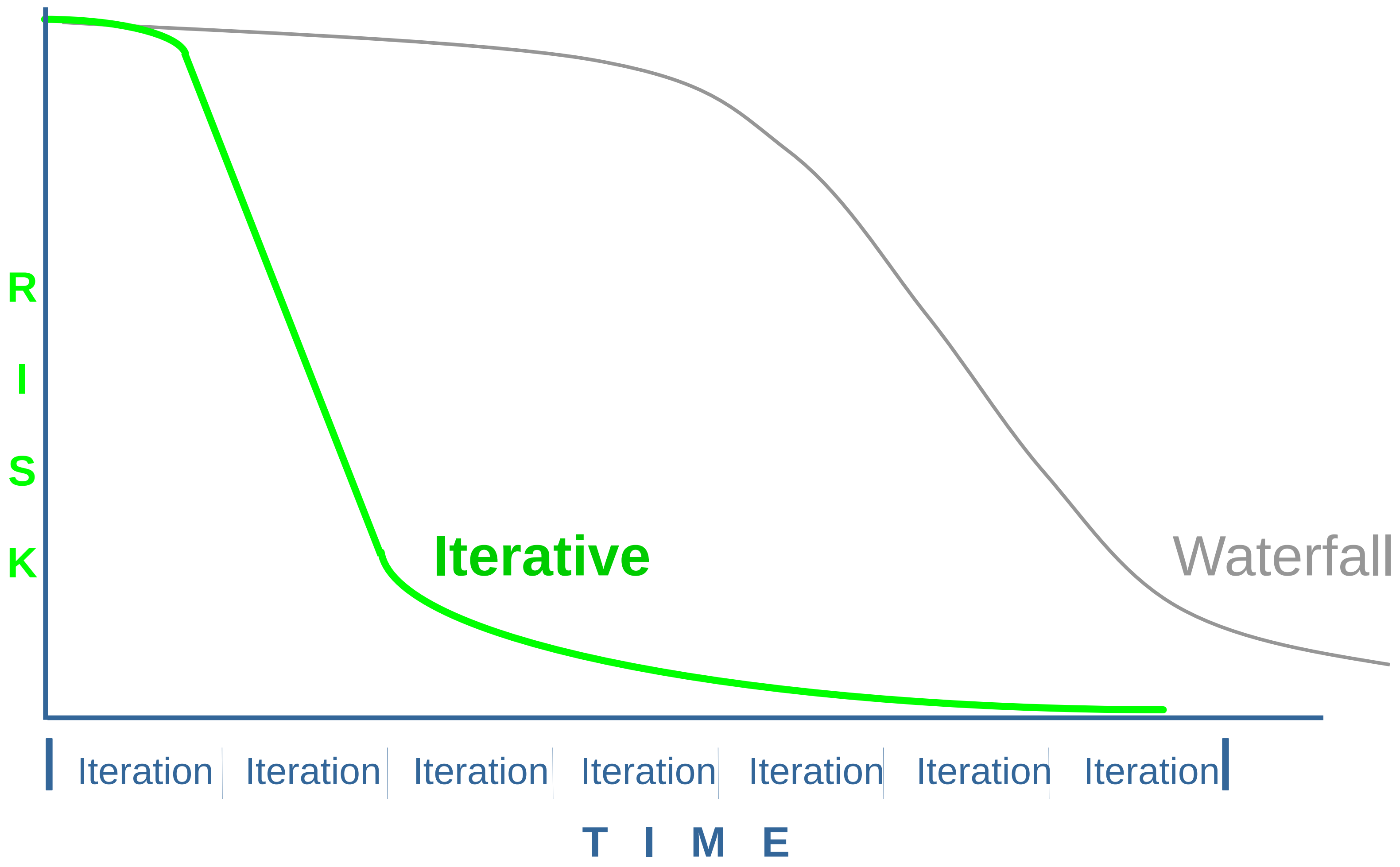
K6-7

Apply the Waterfall Iteratively to System Increments

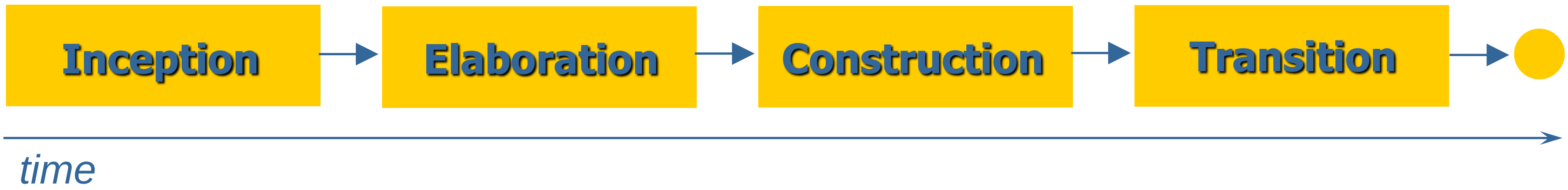


- Earliest iterations address greatest risks
- Each iteration produces an executable release, an additional increment of the system
- Each iteration includes integration and test

Iterative Development Accelerates Risk Reduction



Lifecycle Phases

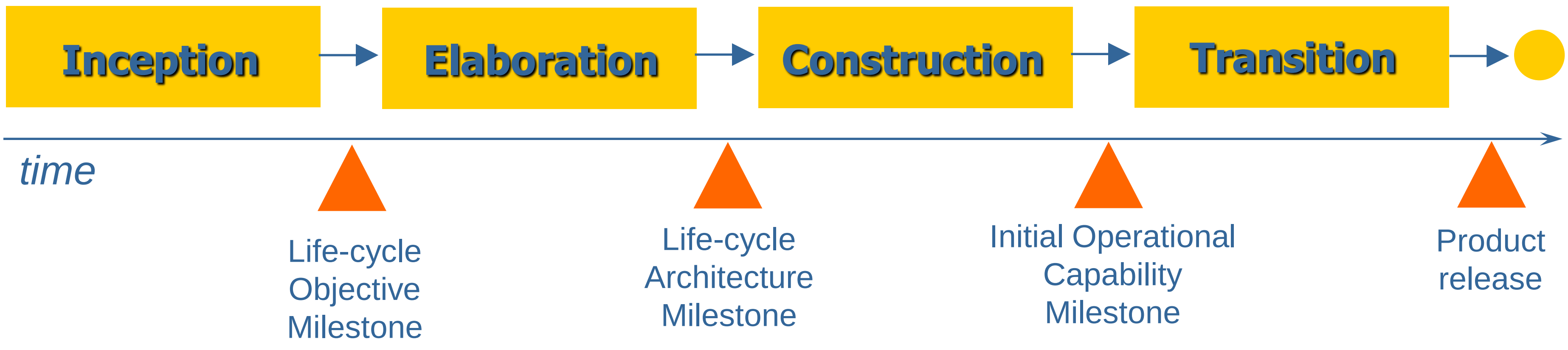


The Rational Unified Process has four phases:

- **Inception** - Define the vision and scope of project
- **Elaboration** - Plan project, specify features, baseline architecture
- **Construction** - Build product
- **Transition** - Transition product to users

K60

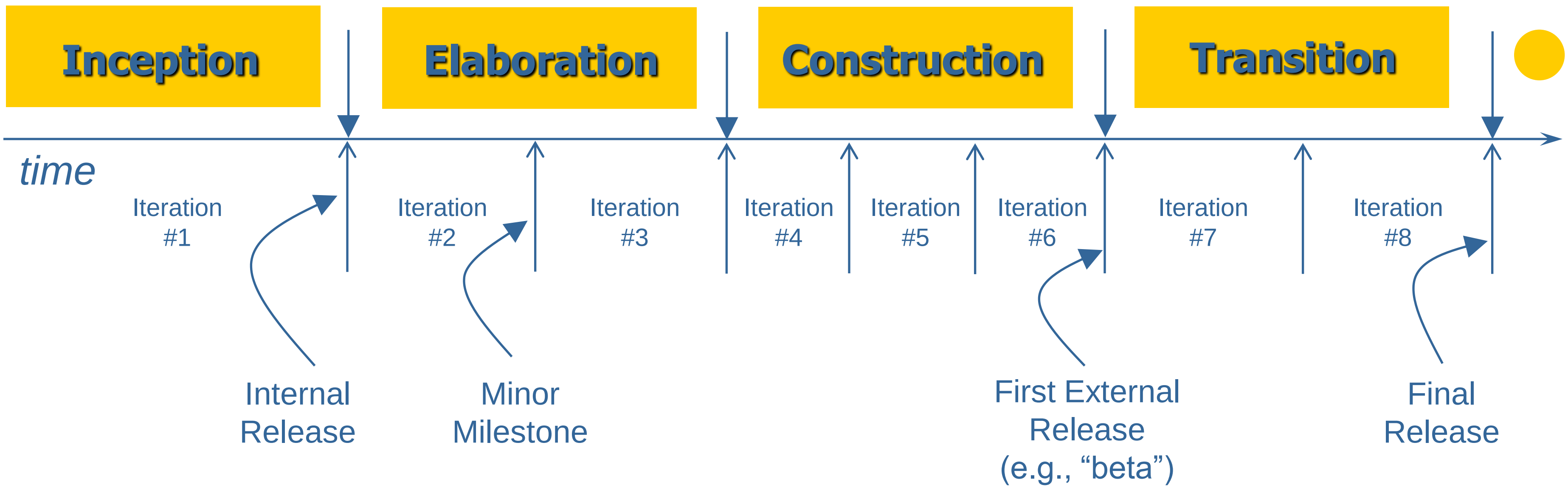
Phase Milestones



K60

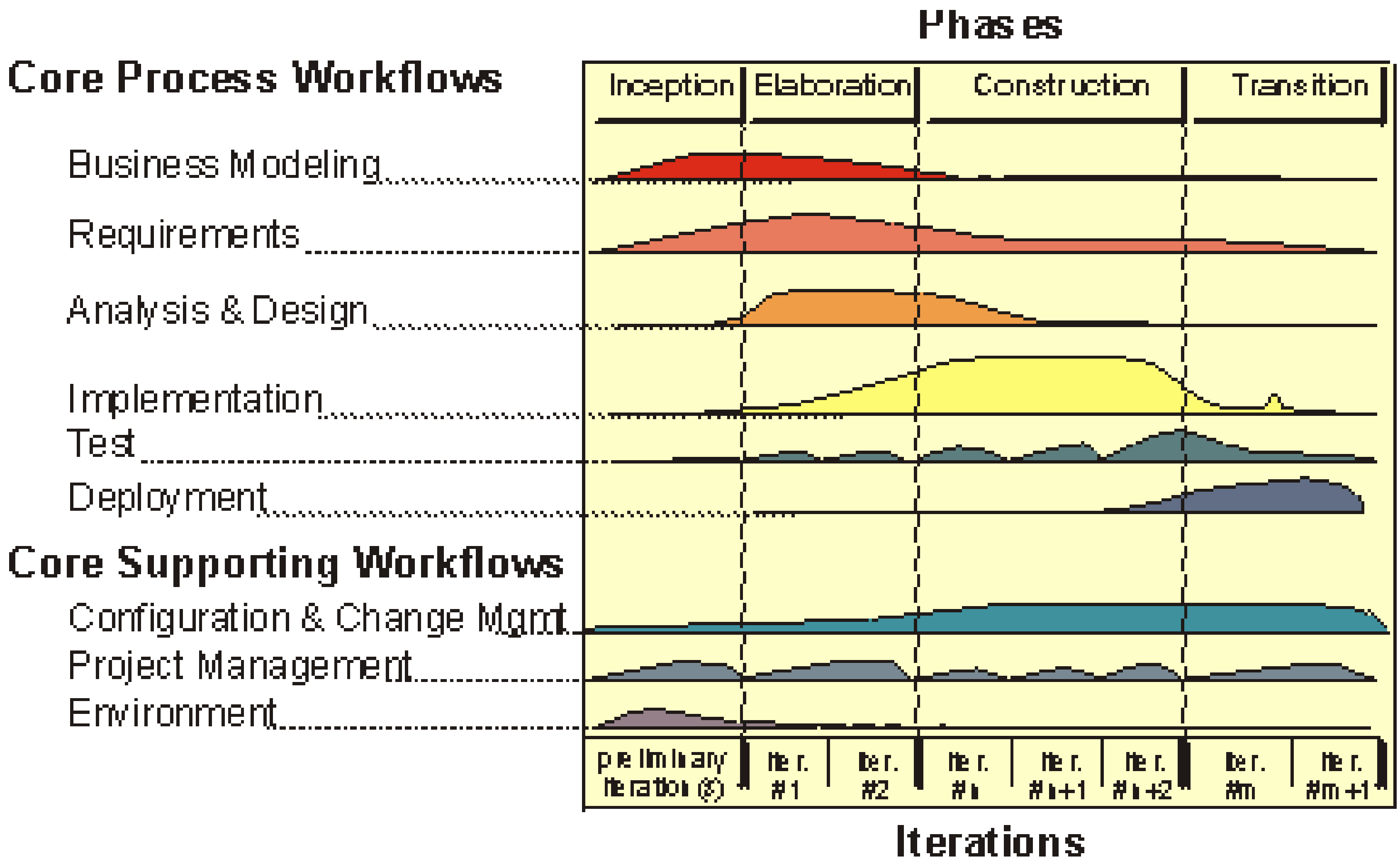
Iterations and Phases

- Each phase consists of one or several iterations.



K62, J9

Overall Architecture of the Rational Unified Process



K45, J11

Benefits of Iterative Development

- Serious misunderstandings become evident early in the life cycle
- Enables and encourages user feedback
- Development focuses on critical issues
- Objective assessment thru testing
- Inconsistencies detected early
- Workload of teams is spread out
- Leverage lessons learned earlier
- Stakeholders are kept up to date on project’s status

K8

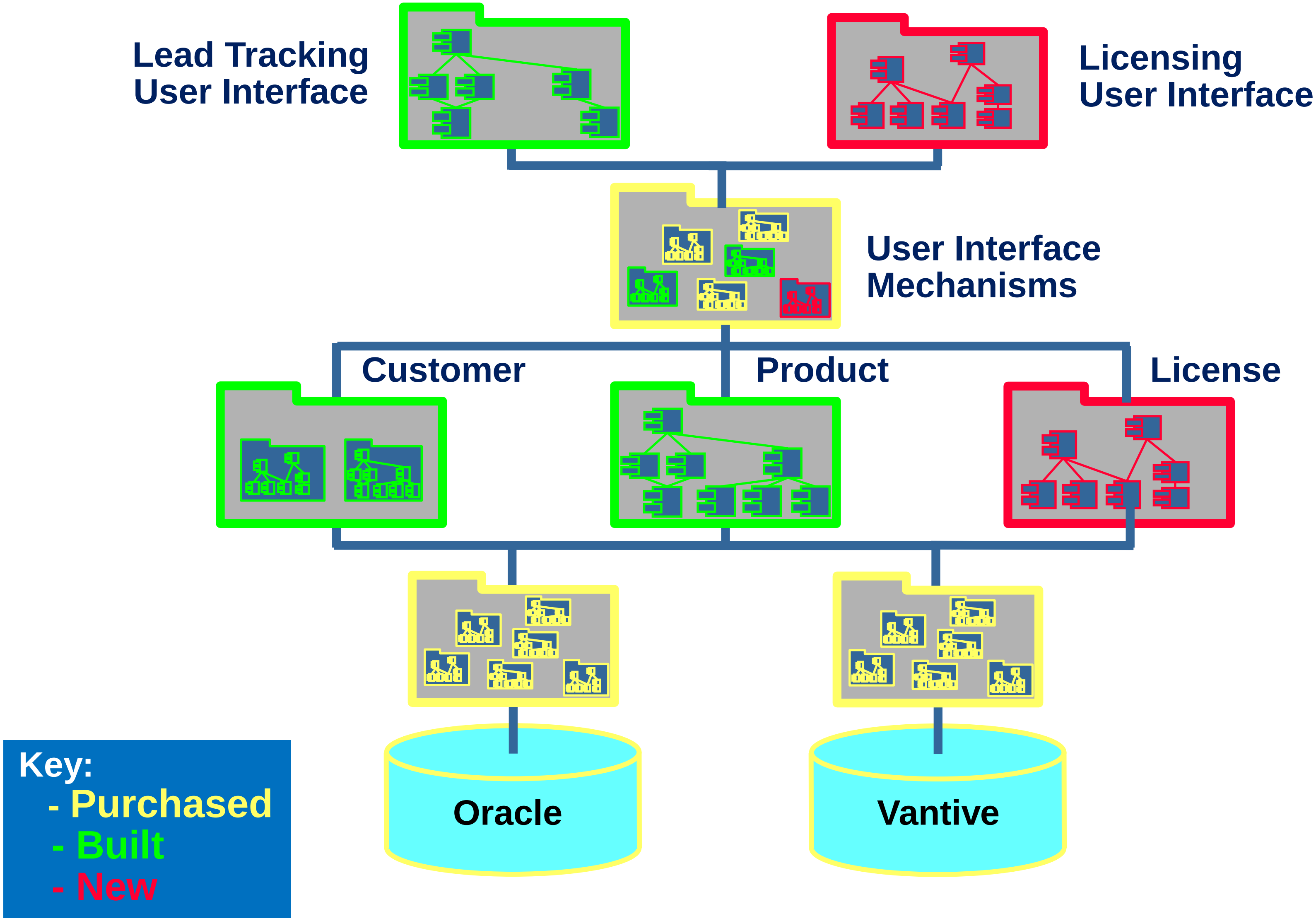
RUP Key Features

- Use case driven
- Iterative and incremental
  - Models, workflows, phases, and iterations
- Architecture-centric

Software Architecture Encompasses

- Organization of a software system
- Selection of the structural elements and their interfaces by which a system is composed
- Their behavior, as specified in collaborations among those elements
- The composition of these structural and behavioral elements into progressively larger subsystems

Example: Component-Based Architecture



K11

Benefits of using Component Architectures

- Components facilitate resilient architectures
- Modularity enables a clear separation of concerns among elements of a system that are subject to change.
- Reuse is facilitated by leveraging standardized frameworks and commercially available components

K10-11

Review Questions

- What are 3 “essential” difficulties in software development?
- What phase of the lifecycle has the highest costs?
- What phase introduces the most significant errors?

K13-14