Lesson 2

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

Topics: Principles of Software Engineering

- Software development processes
- The importance of analysis and design
- Best practices

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.

Main Points

1. Software engineering studies the principles for creating reliable and easily maintainable software, on time and within budget. A month-long project will be used in the course to provide hands-on experience of some of the most important software engineering best practices.
2. Likewise, SCI studies the fundamental principles that underlie life itself and brings out effective techniques for making individual and collective life as rich and fully developed as possible. Through the practical application of SCI, TM, we gain direct experience of a best practice for engaging Nature's intelligence in our software development.

Software Development History

- 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."

Software Process Results

IT Success Rate in 2010 – Some Survey Data
http://www.drdobbs.com/architecture-and-design/2010-it-project-success-rates/226500046

- **Iterative projects**: 61% are successful, 28% are challenged, and 11% are failures.
- **Agile projects**: 60% are successful, 28% are challenged, and 12% are failures.
- **Ad-hoc projects**: 49% are successful, 37% are challenged, and 14% are failures.
- **Traditional projects**: 47% are successful, 36% are challenged, and 17% are failures.

Software Process Results

What is success?

What is challenged?

What is failure?

Software Process Results

What is success? Delivered with expected functionality, on time, and within budget.(per organization's tolerances)

What is challenged? Missed one or more of the above (but still delivered a solution.)

What is failure? No solution delivered.

Software Process Results

What are the development processes used?

- **Iterative**. Development divided into multiple iterations or time boxes. e.g. Rational Unified Process (RUP)

- **Agile**. iterative process with emphasis on:
  - Lightweight
  - highly collaborative
  - self-organizing
  - quality focused.
  - e.g. OpenUP, Scrum, and Extreme Programming (XP).

Software Process Results

Development process (cont.)

- **Ad-hoc -** team does not follow a defined process.

- **Traditional/Waterfall** a staged/serial process:
  - Requirements
  - Architecture/Design
  - Coding
  - Testing
  - Deployment

Software Process Results

Team Size Success Rates:

1. Small teams < 10
2. Medium size teams 11 to 25
3. Large teams > 26

- **Iterative projects**: 80% for small teams, 68% for medium-sized teams, and 55% for large teams.
- **Agile projects**: 83% for small teams, 70% for medium-sized teams, and 55% for large teams.
- **Ad-hoc projects**: 74% for small teams, 58% for medium-sized teams, and 40% for large teams.
- **Traditional projects:** : 69% for small teams, 61% for medium-sized teams, and 50% for large teams.

Software Process Results (comparison)

Small teams success vs all teams success

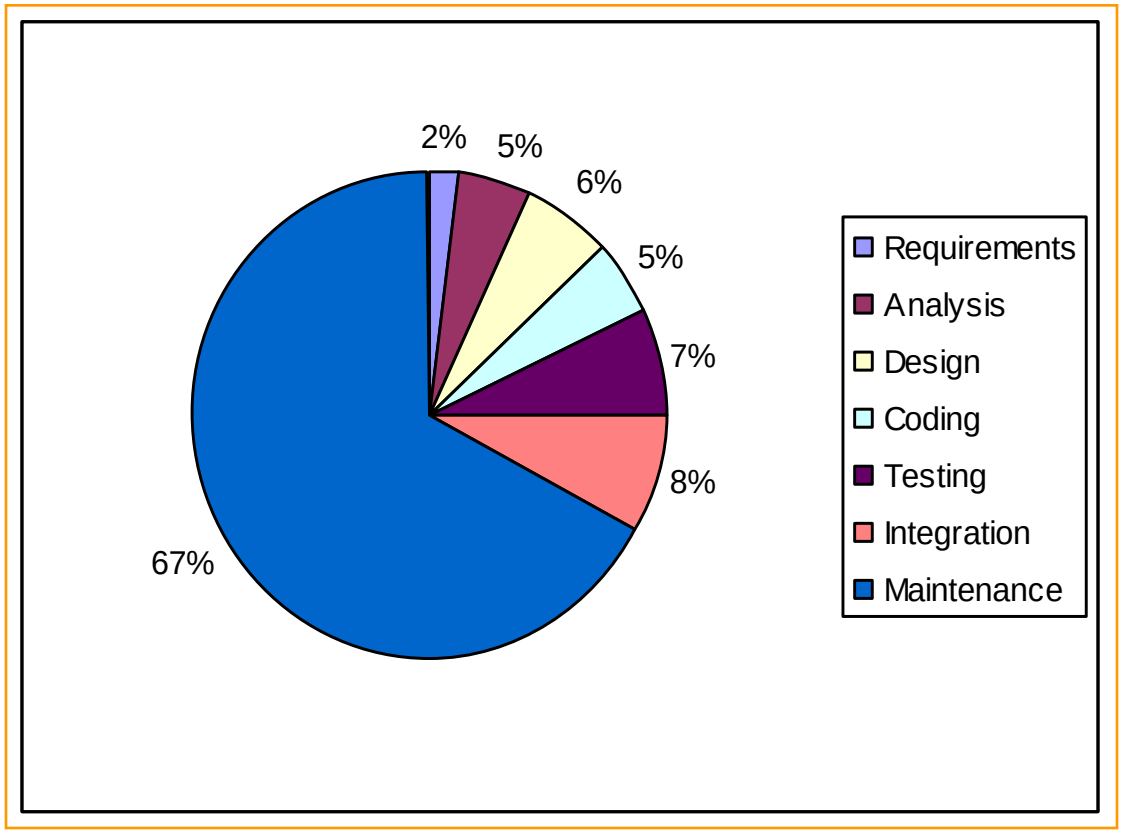
- **Iterative projects**: 80% vs 61%
- **Agile projects**: 83% vs 60%
- **Ad-hoc projects**: 74% vs 49%
- **Traditional projects**: 69% vs 47%.

Topics: Principles of Software Engineering

- Software development processes
- The importance of analysis and design
- Best practices

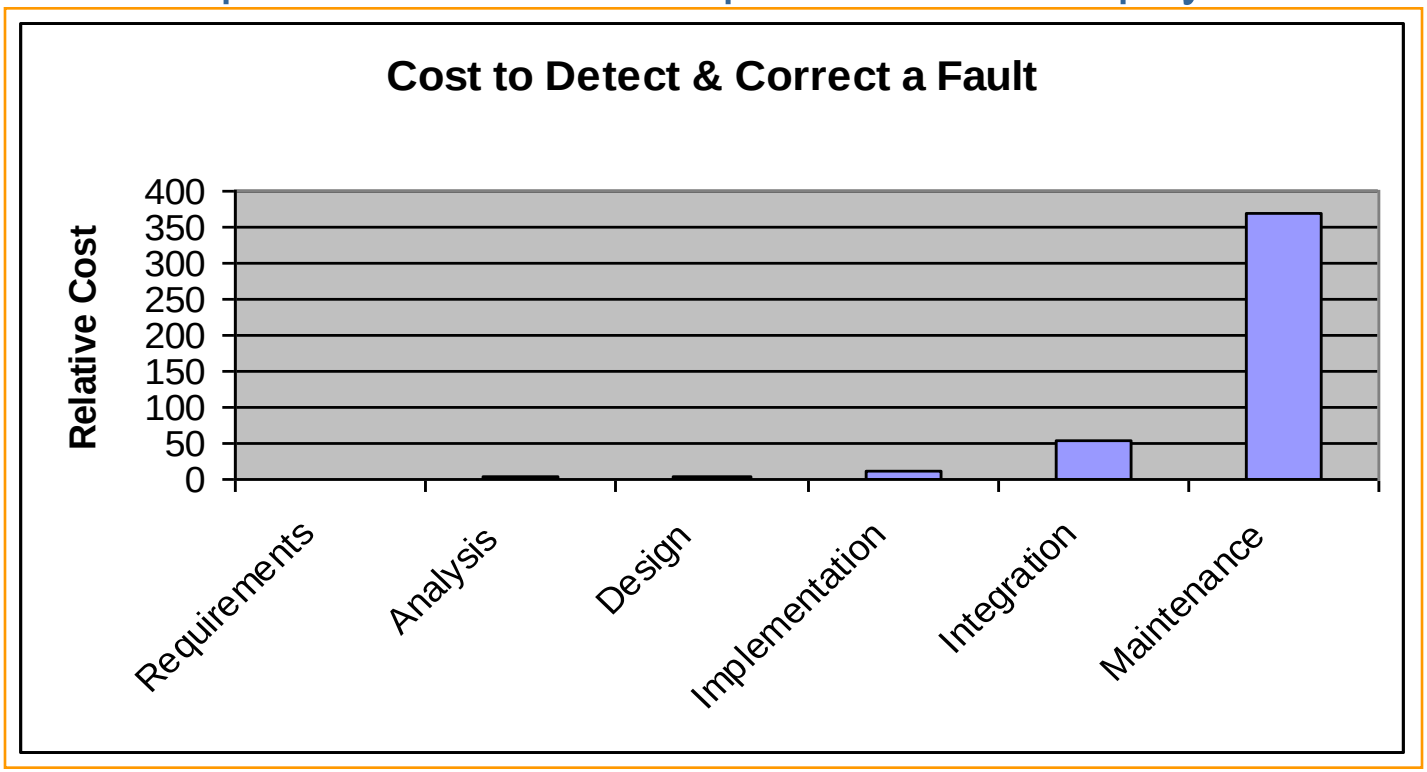
Importance of Maintainability

- 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



Summary Points

- 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

- Software development processes
- The importance of analysis and design
- Best practices

Best Practices Address Root Causes

We will follow these best practices for our project:

- Visually model software -- we will use UML diagrams
- Develop software iteratively – RUP and Agile processes
- Use component-based architectures – Enterprise Architectures
- Manage requirements – use cases and user stories

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)

Visually Model Software

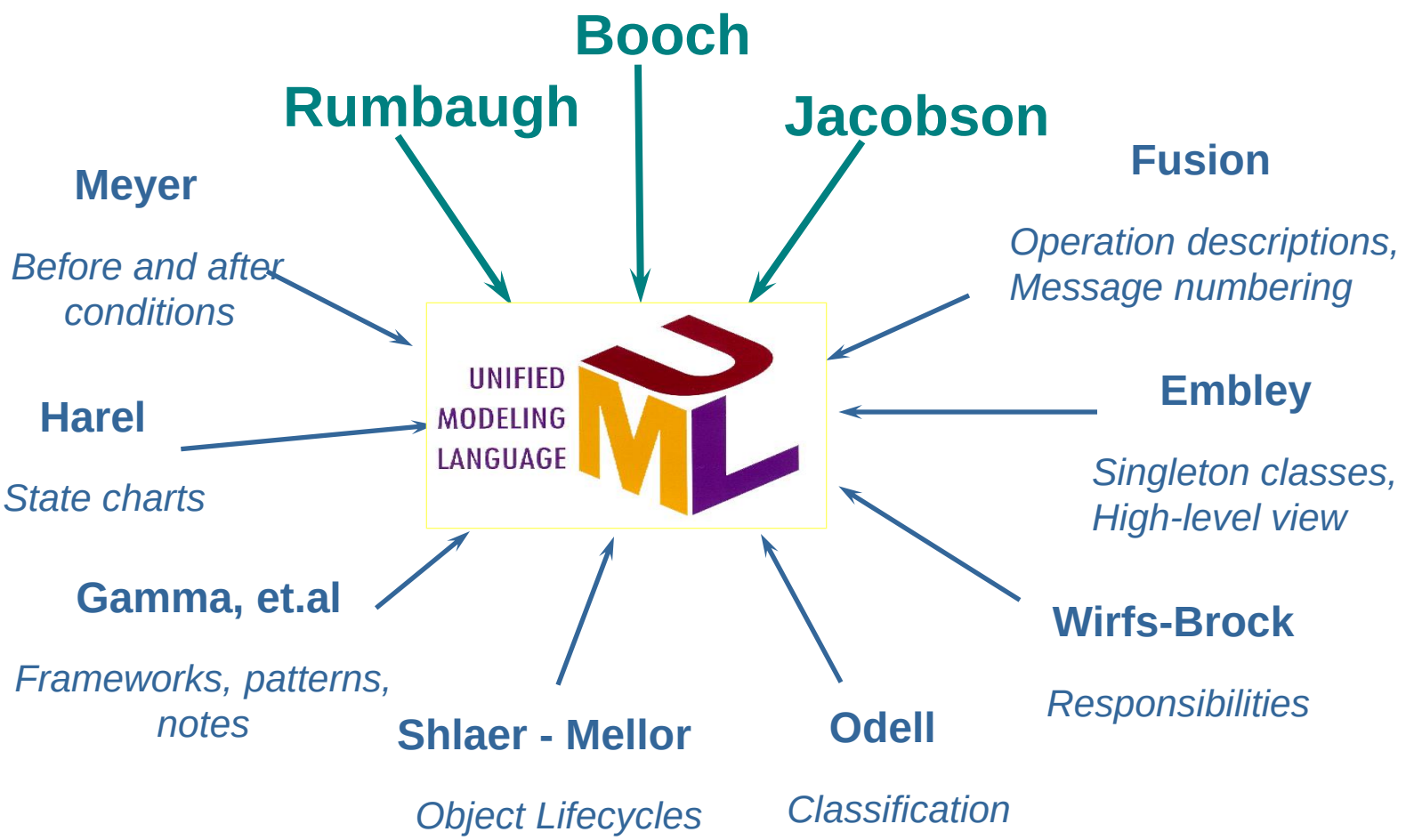
- Addresses the difficulty of the abstractness of software
- Helps to visualize, specify, construct, and document the structure and behavior of a system
- Using a standard modeling language (like UML) helps to maintain consistency among system's artifacts and facilitates communication among team members

What Is the UML?

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

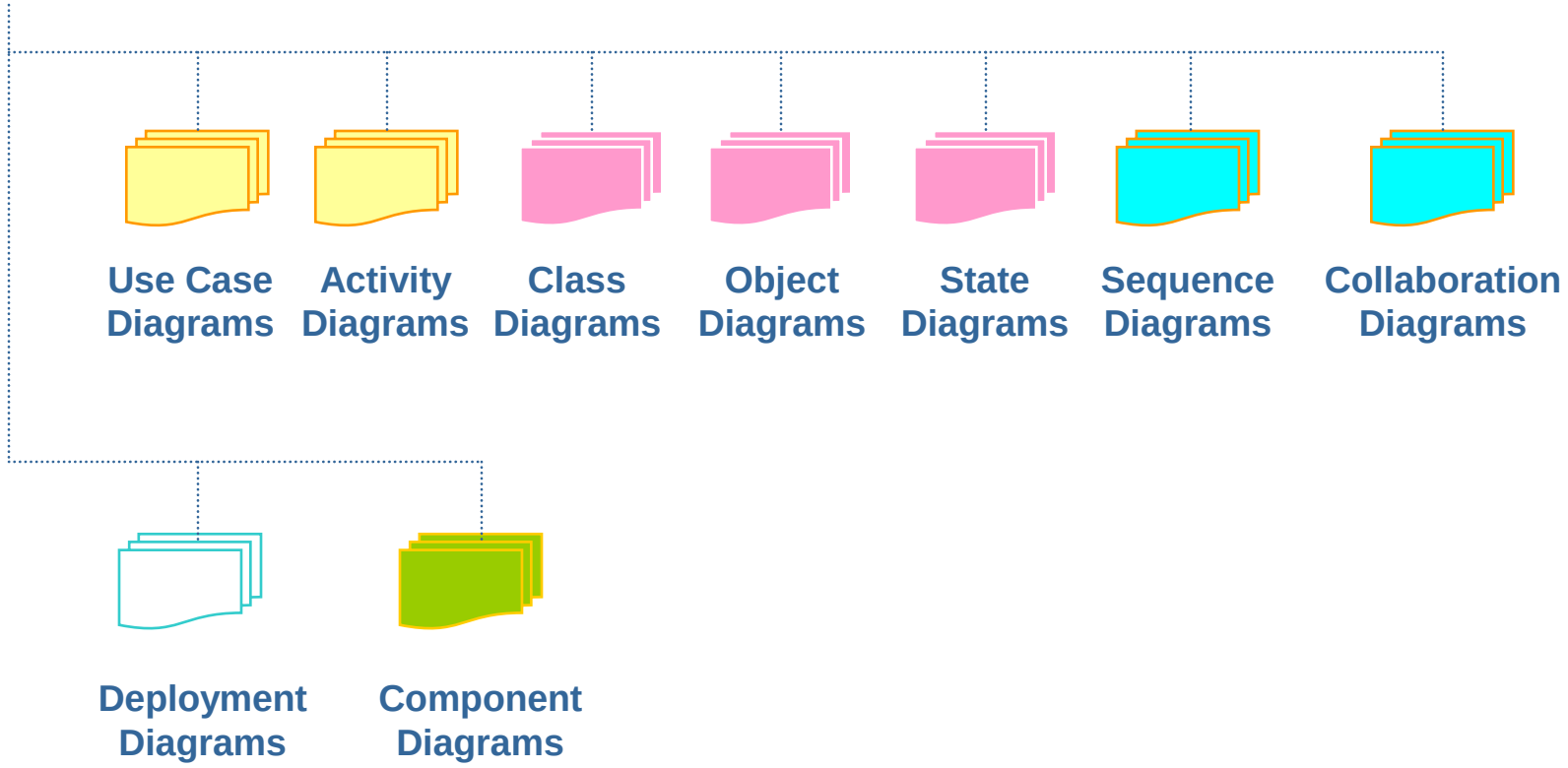
  the artifacts of a software-intensive system

Inputs to UML



The UML Provides Standardized Diagrams

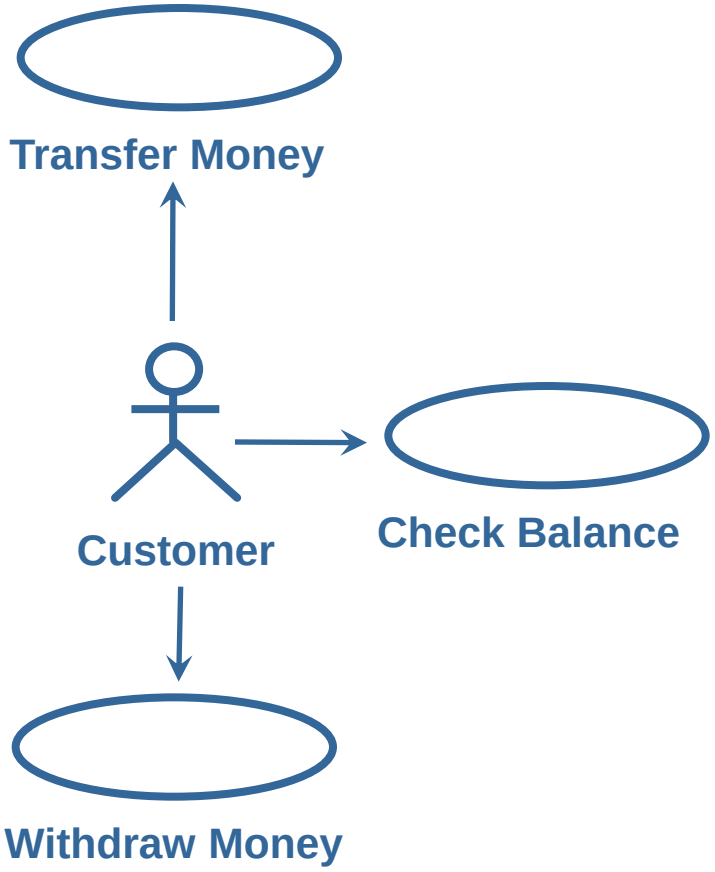
Models



RUP Key Features

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

Rational Unified Process Is Use-Case Driven



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 for an Automated Teller Machine

What common use case is missing?

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

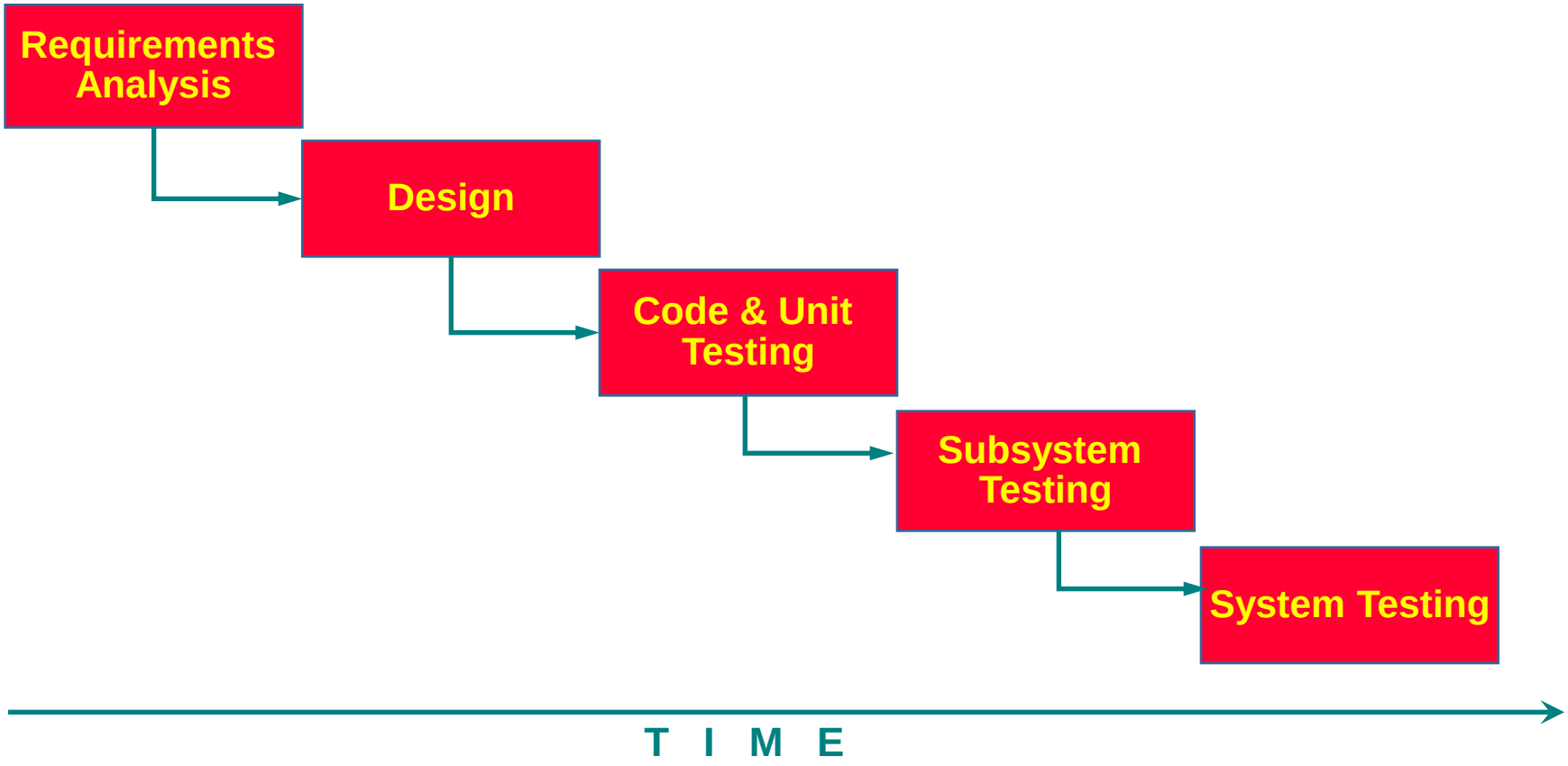
RUP Key Features

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

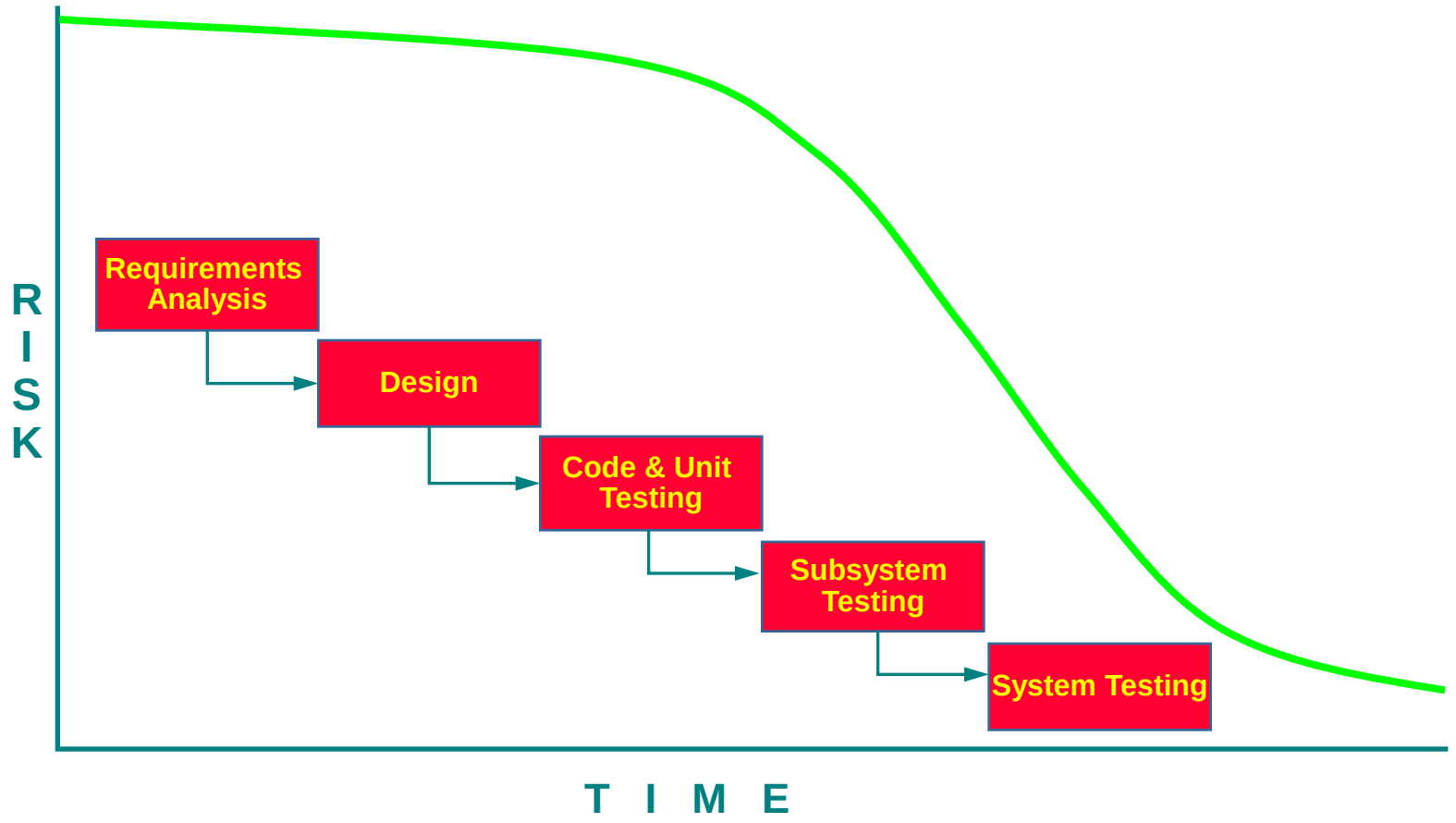
Develop Software Iteratively

- 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

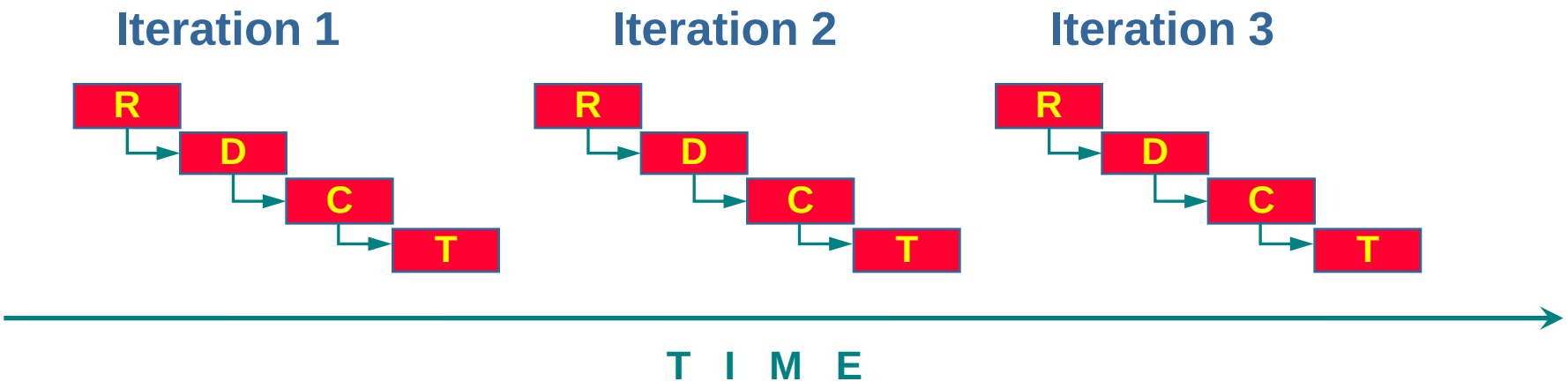


Waterfall Development Delays Reduction of Risk



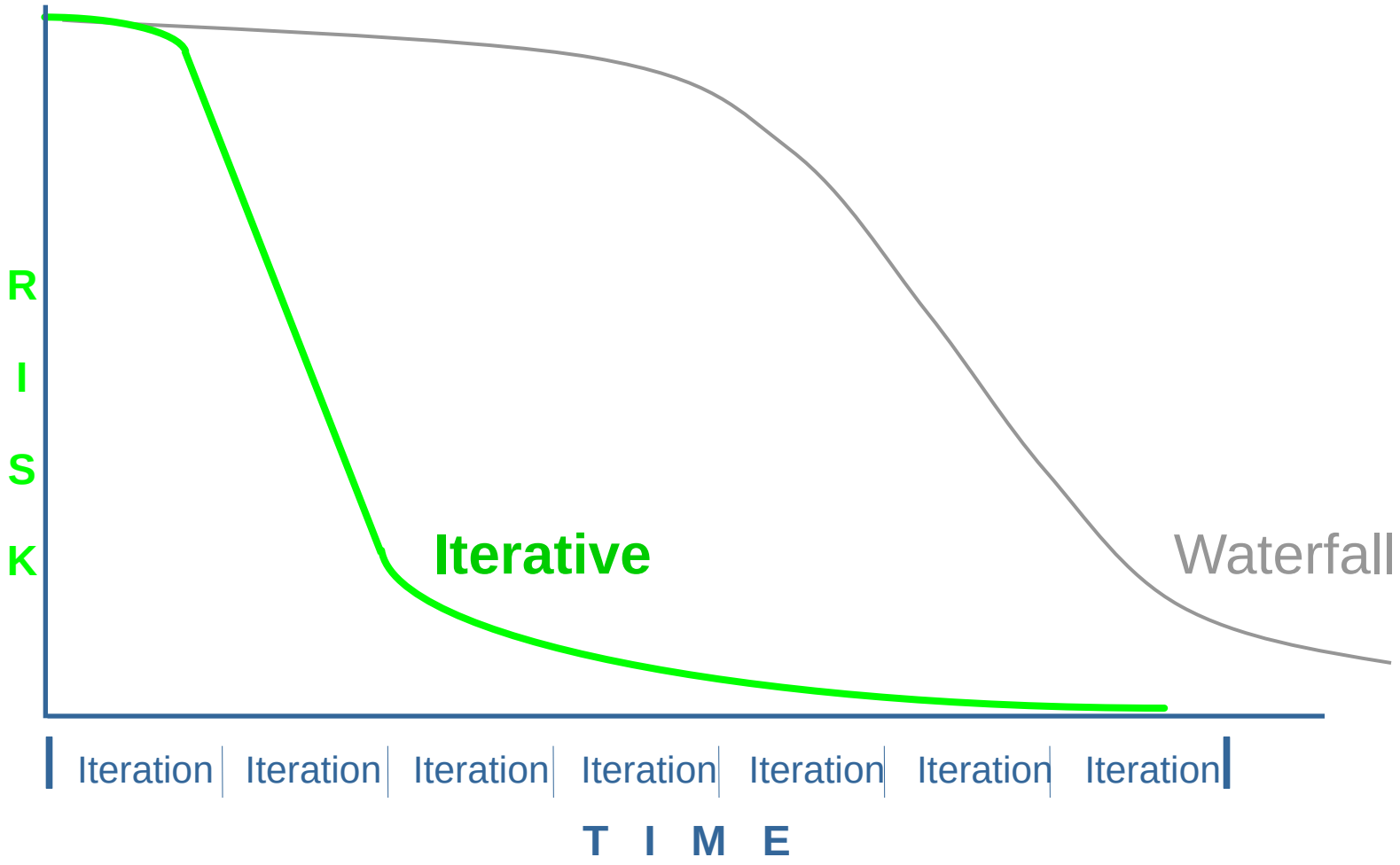
K6-7

Apply the Waterfall Iteratively to System Increments

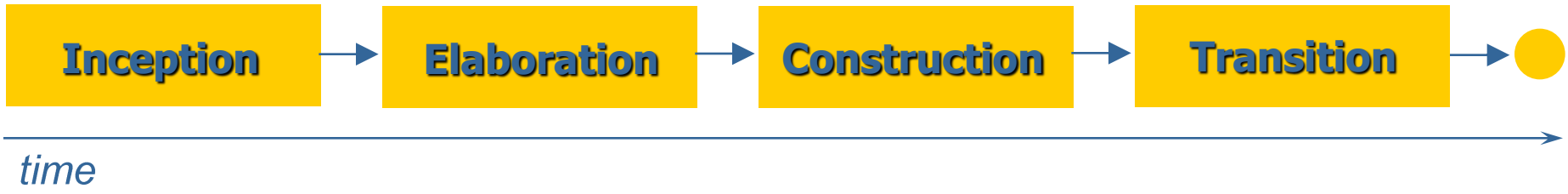


- Earliest iterations address greatest risks
- Each iteration produces an executable release, an additional increment of the system
- Each iteration includes integration and test

Iterative Development Accelerates Risk Reduction



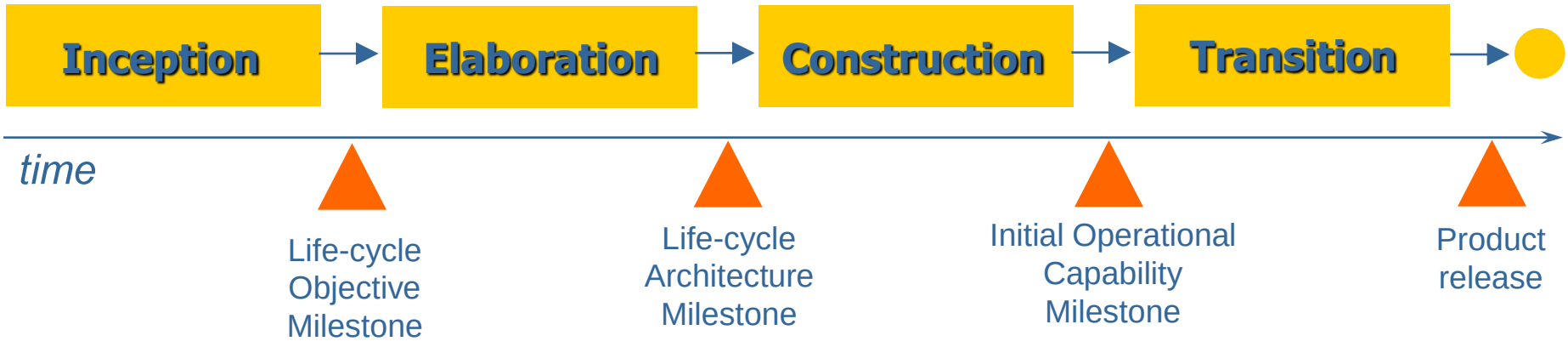
Lifecycle Phases



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

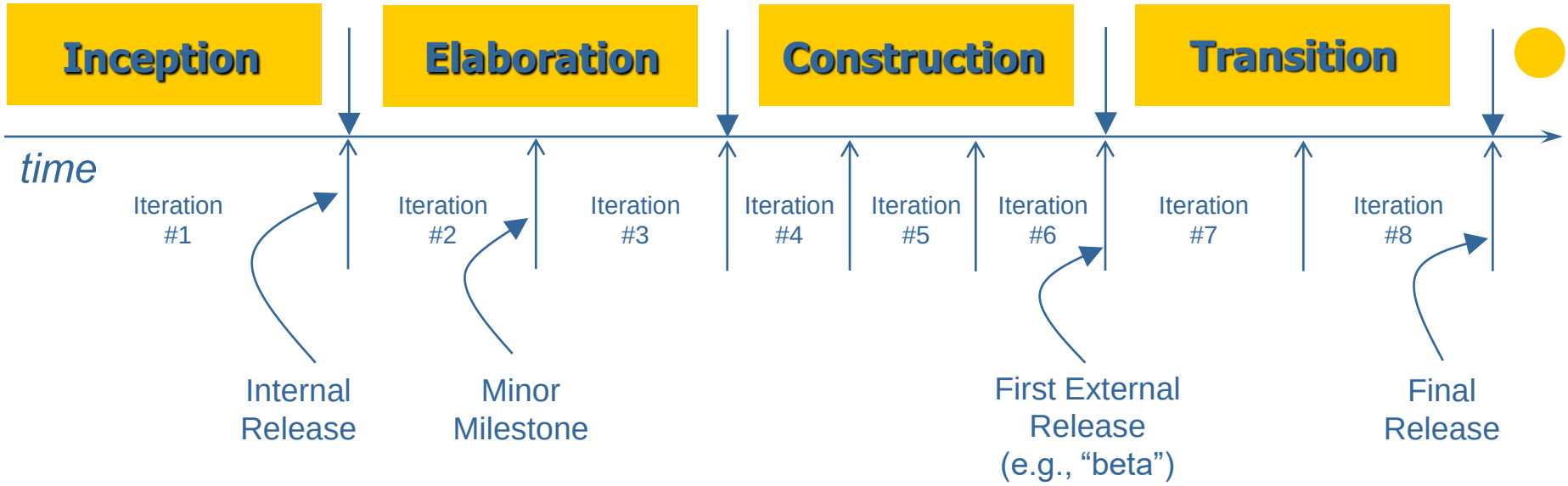
Phase Milestones



K60

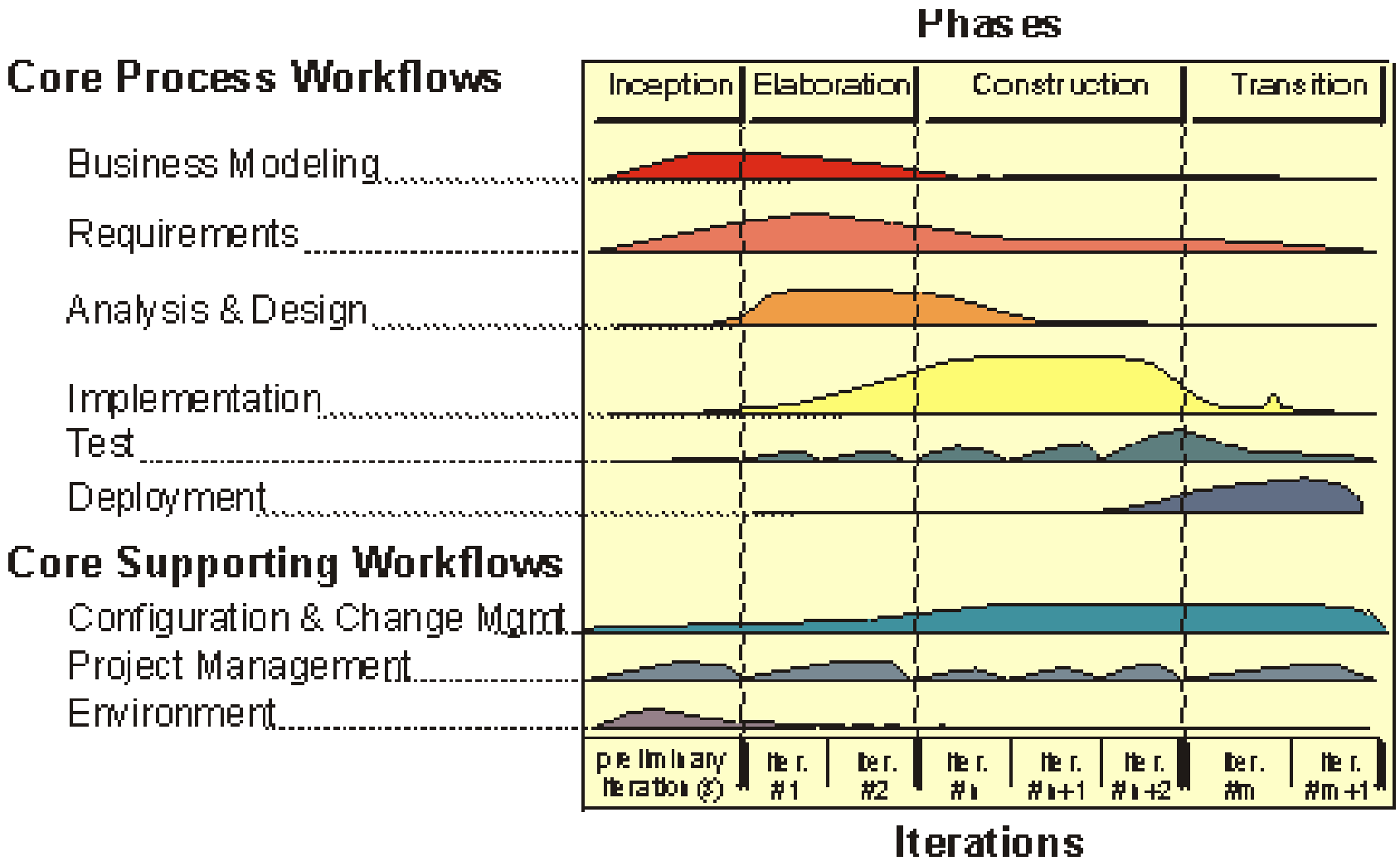
Iterations and Phases

- 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

Review Questions

- What are 4 best practices for software development?
- What phase of the lifecycle has the highest costs?
- What phase introduces the most significant errors?
- What are four different software development processes?

K13-14

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. Software Engineering is a discipline that presents best practices and optimized methodologies for software development. It organizes the insights and discoveries of computer scientists and software engineers over the past several decades.
2. The deeper truth about the principles of good software engineering, elaborated in the Software Engineering discipline is that the processes are only valuable to the extent they fit with our own creative intelligence. RUP and Agile processes have gained in acceptance because they are the most natural fit to the way we understand and build software.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

3. The practice of TM improves our software development skills because is uses the natural tendencies of our minds to reduce stress and increase creativity through the deep rest we experience in our daily meditations.

SCI question: Of the four different software development processes we discussed today which one shares the most characteristics with our practice of TM?