Software Architecture

Lecture 3

Software Processes

Néstor Cataño

Innopolis University

Spring 2016

Plan

1. **Software development models**
2. Software Processes Activities
3. Methodological frameworks

Software Processes

- **A software process** is a set of related **activities** that leads to the production of a software product.
- These activities may involve the development of software from scratch in a **standard programming language** like **Java** or **C**.

Software Activities

- **Software specification** → The functionality of the software and constraints on its operation must be defined.
- **Software design and implementation** → The software to meet the specification must be produced.
- **Software validation** → The software must be validated to ensure that it does what the customer wants.
- **Software evolution** → The software must evolve to meet changing customer needs.

Processes Descriptions

- **Products ("artifacts")** → **outcomes of a process activity**, for example, the outcome of the activity of architectural design may be a model of the software architecture.
- **Roles** → **reflect the responsibilities of the people involved in the process**, for example, **project manager**, **configuration manager**, **programmer**, etc.

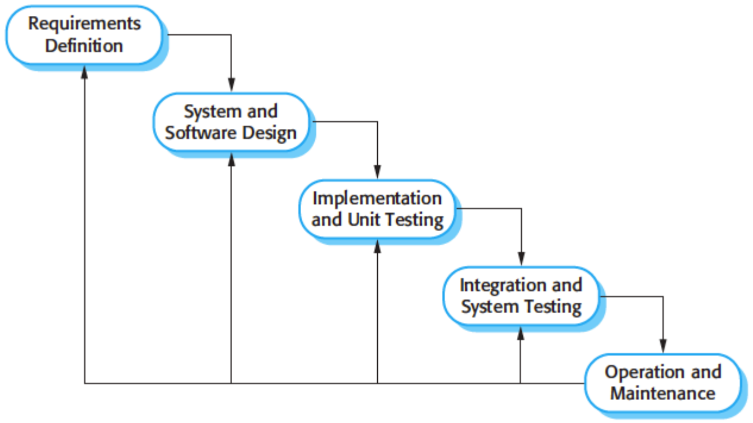
Processes Descriptions

- **Contracts (pre- and post-conditions)** → statements that are true **before** and **after** a process activity has been enacted or a product produced.
  - For example, **before** architectural design begins, a **pre-condition** may be that all requirements have been approved by the customer; this activity is finished, a **post-condition** might be that the UML models describing the architecture have been reviewed.

Lyfe Cycle Models

- **Software process model**; which activities is the process model composed and how they articulate
- Software process model (life cycle model)
  - **Waterfall**
  - **Incremental**
  - **Cleanroom (Formal Software Development)**
  - **The Cluster model of Software Development**
  - **Evolutive, spiral, V, Based on components**
  - **Concurrent**

Waterfall


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- **Winston Royce** (1970)
- **waterfall metaphor**; before **Agile …**
- you must plan and schedule all of the processes activities before starting work on them
- The following phase cannot be started before the previous one has started

Waterfall

- **Requirements analysis** → The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a **system specification**.
- **System and software design** → The system's design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships

Waterfall

- **Implementation and unit testing** → During this stage, the software design is realized as a set of programs or program units. **Unit testing** involves verifying that each unit meets its specification.
- **Integration and system testing** → The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

Waterfall

- **Operation and maintenance →**
  - Normally (although not necessarily), this is the longest life cycle phase.
  - The system is installed and put into practical use. **Maintenance involves correcting errors** which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered

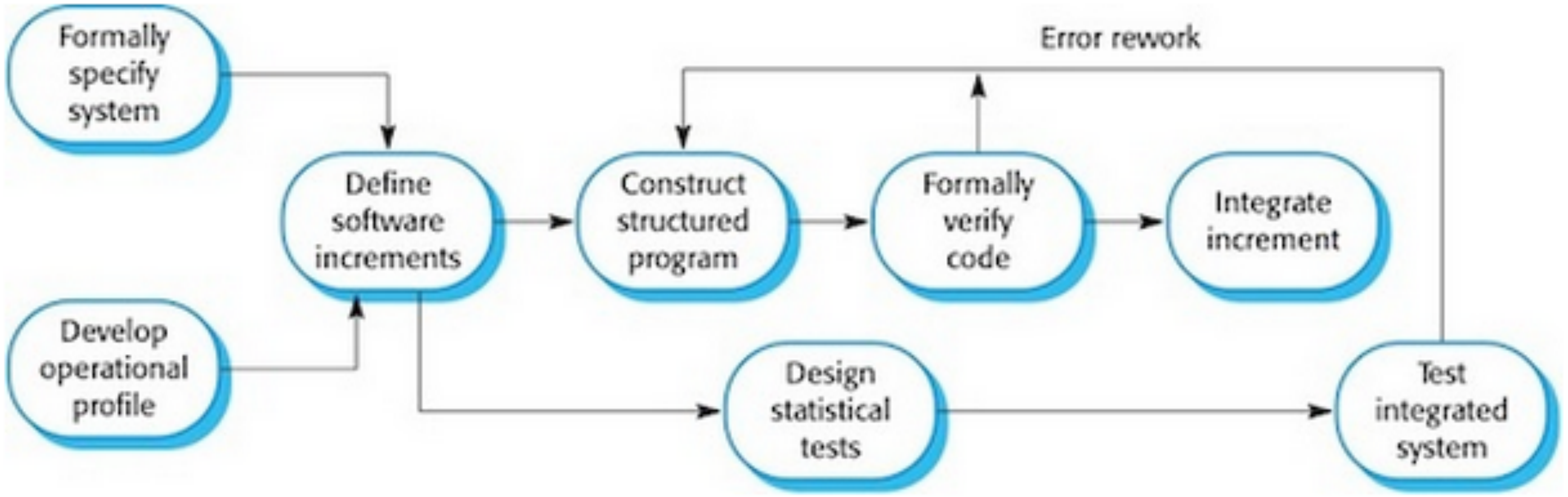
When should one use Waterfall for software development?

- → when all the **software requirements** are **well- and fully-understood**.

Cleanroom software development

- originally developed by **Harlan Mills at IBM**.
  - defect prevention rather than defect removal
  - no need of conducting testing
  - based on **formal methods**, e.g. embedded application
- Software is **formally specified** and this specification is transformed into an **implementation**, e.g. **B Method** (**next slides**)

Cleanroom software development



Cleanroom software development

```
machine Facebook sees ctx
variables person content owner page
invariants
 @inv1 person ⊆ PERSON /\ content ⊆ CONTENT
 @inv2 owner ∈ content ↔ person
 @inv3 page ∈ content ↔ person

event upload any c p where
 @grd1 p ∈ person /\ c ∈ CONTENT \ content
then
 @act1 content := content ∪ c
 @act2 owner := owner ∪ {c ↦ p}
 @act3 pages := pages ∪ {c ↦ p}
end
```

Cleanroom software development

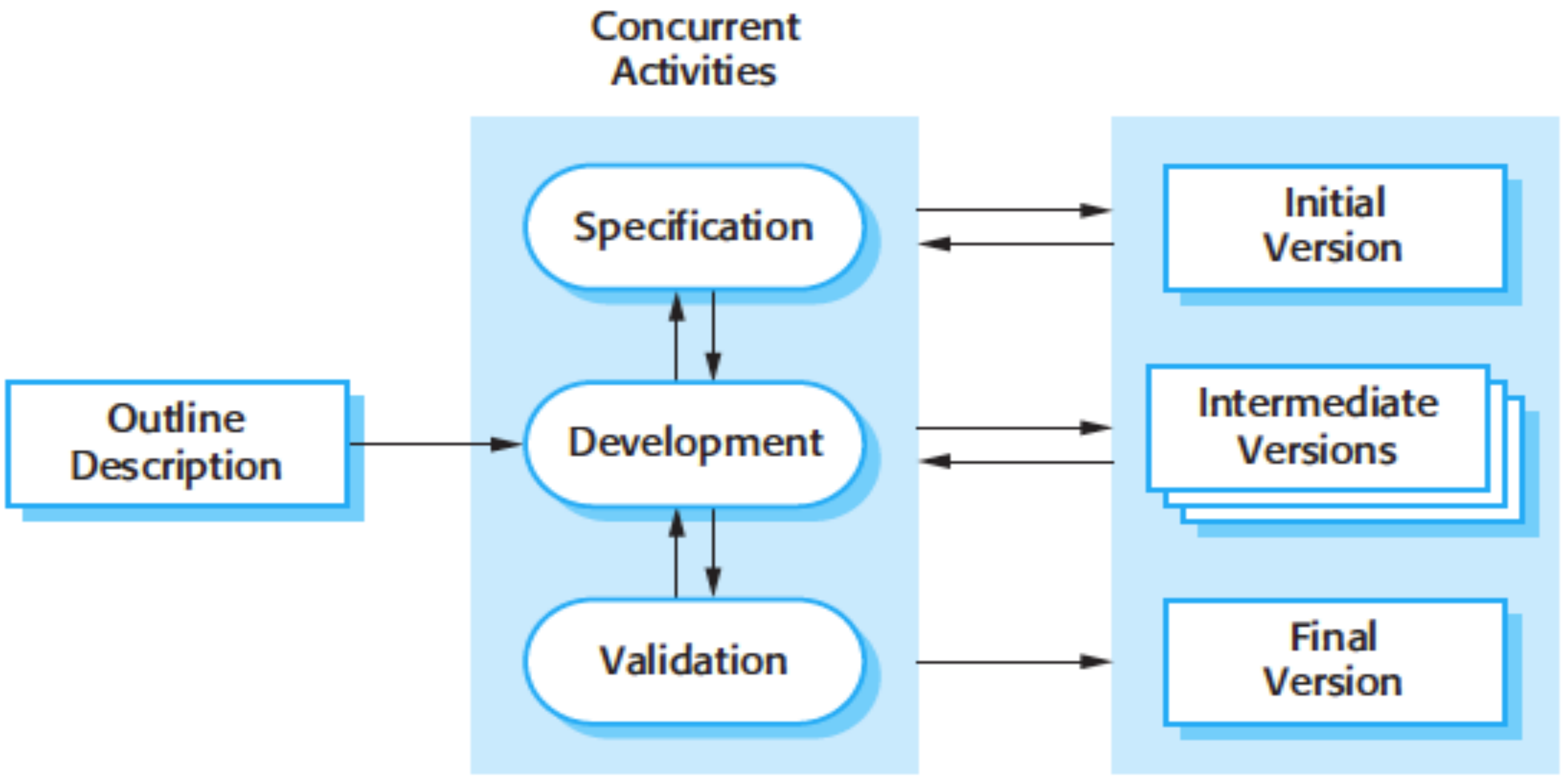
```
machine permissions refines Facebook sees ctx
variables person content owner page viewp editp
invariants
@invr1 viewp ∈ content ↔ persons
@invr2 editp ∈ content ↔ persons
@invr3 editp ⊆ viewp /\ owner ⊆ viewp
@invr5 owner ⊆ editp /\ viewp ⊆ page

event upload extends upload then
 @act1r1 viewp := viewp ∪ {c ↦ p}
 @act2r1 editp := editp ∪ {c ↦ p}
end
```

Incremental model

- Mcdermid (1993) "**interleaves**" the **activities** of **specification**, **development**, and **validation**.
- It's based on the idea of developing an initial implementation, exposing it to user comment and evolving it through several versions until an adequate system has been developed
- The system is developed as a series of versions (**increments**), with each version adding functionality to the previous version
- Similar to **Agile software development**

Incremental software development



Incremental software development

- **Agile software development**
- each increment or version of the system incorporates some of the functionality that is needed by the customer
- usually the early **increments** include the most **urgent** functionality needed by the customer

Advantages of Incremental Over Waterfall

- The cost of accommodating changing customer requirements is reduced.
- The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.

Advantages of Incremental Over Waterfall

- Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included.
- Customers are able to use and gain value from the software earlier than is possible with a **waterfall** process

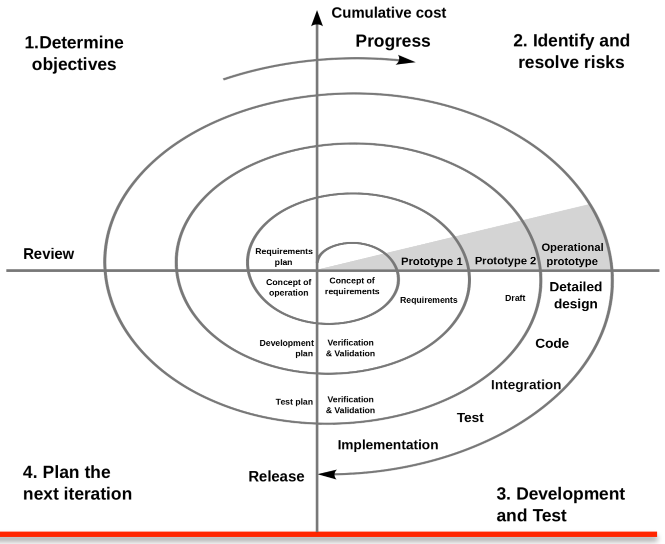
Disadvantages of Incremental Over Waterfall

- **System structure tends to degrade** as new increments are added.
- Unless time and money is spent on **refactoring** to improve the software, regular change tends to **corrupt its structure**.

"The problems of incremental development become particularly acute for **large**, **complex**, **long-lifetime systems**, where different teams develop different parts of the system"

Spiral and iterative development

- **Bohem** (1988) → The **whirlpool** metaphor
- Each loop in the spiral represents a phase in the software development process
  - **loop** → system feasibility, requirements definitions, system design, etc.
- Loop is divided in 4 sectors
  - **Objective setting**
  - **Risk assessment**
  - **Development and validation**
  - **Planning**



Concurrent software engineering

- **River delta** as a metaphor; speed through **parallelism**.
- derived from concurrent engineering concepts in **aerospace** projects.
- Potential concurrent activities $\rightarrow$ **analysis**, **design**, **implementation**.

Building large software systems

- Sub-systems within a larger system may developed using different approaches
  - well-understood parts → waterfall-based process
  - non well-understood parts (e.g. Interfaces) → incremental approach (e.g. Agile Methodologies)

Reuse software engineering

- Reuse takes place irrespective of the development process that is used
- Typical software components used in re-used oriented processes →
  - **Web-services**
  - **Collection of objects** within a framework such as **.NET** or **J2EE**
  - **Standalone software systems**

Reuse software engineering

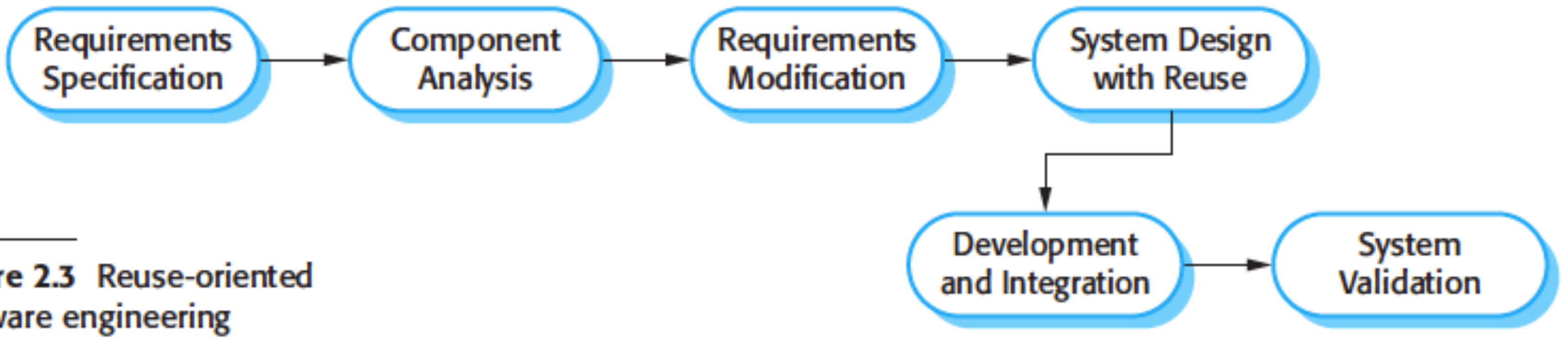

Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Figure 2.3 Reuse-oriented software engineering

The Cluster model of software development

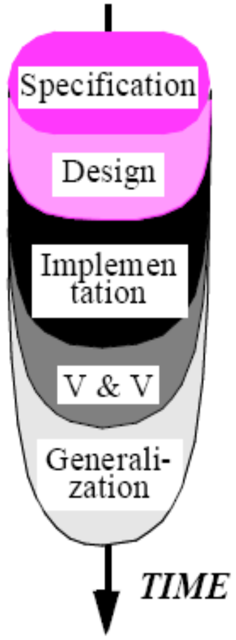
- **Source: Object Oriented Software Construction, Chapter 28, by B. Mayer**

The Cluster model of software development

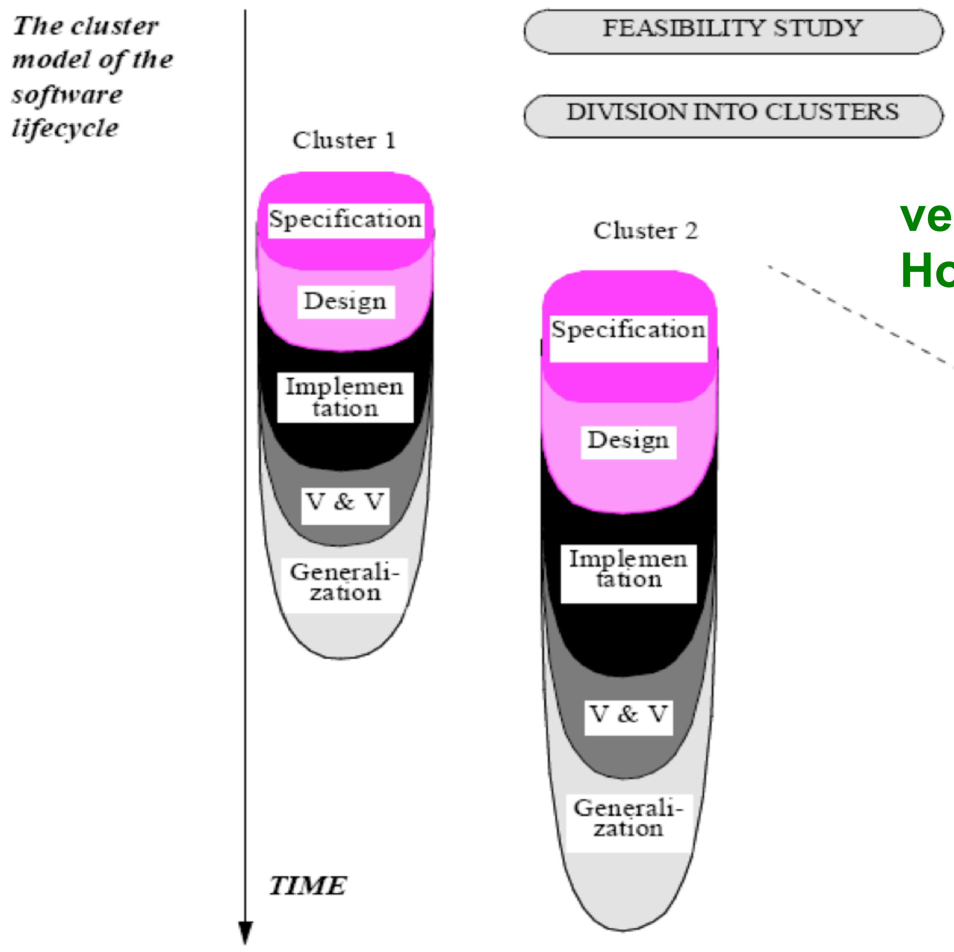
- **reuse** software oriented engineering
- **cluster** →
  - A cluster is a group of related classes or, recursively, of related clusters
  - similar to the notion of **package** in Java.
  - e.g. a **syntactical analyzer**
  - , **5** to **40** classes.
- Successful **object-oriented** (**O-O**) development needs to support a **concurrent engineering** scheme

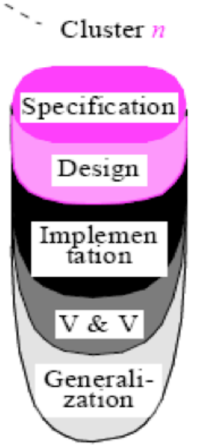
Mini-lifecycle governing the development of a cluster

- **Specification**: identify the classes (data abstractions) of the **cluster** and their major features and constraints
- **Design**: define the architecture of the classes and their relations.
- **Implementation**: finalize the classes, with all details added.
- **Verification & Validation**: check that the cluster's classes perform satisfactorily (through **static examination**, **testing** and other techniques).
- **Generalization**: prepare for **reuse**.



The cluster model of software development





vertical axis → Sequence
Horizontal axis → Concurrence

Plan

1. Software development models
2. **Software Processes Activities**
3. Methodological frameworks

Software Process Activities

1. Software Specification
2. Software Design and Implementation
3. Software Validation
4. Software Evolution

Software specfication

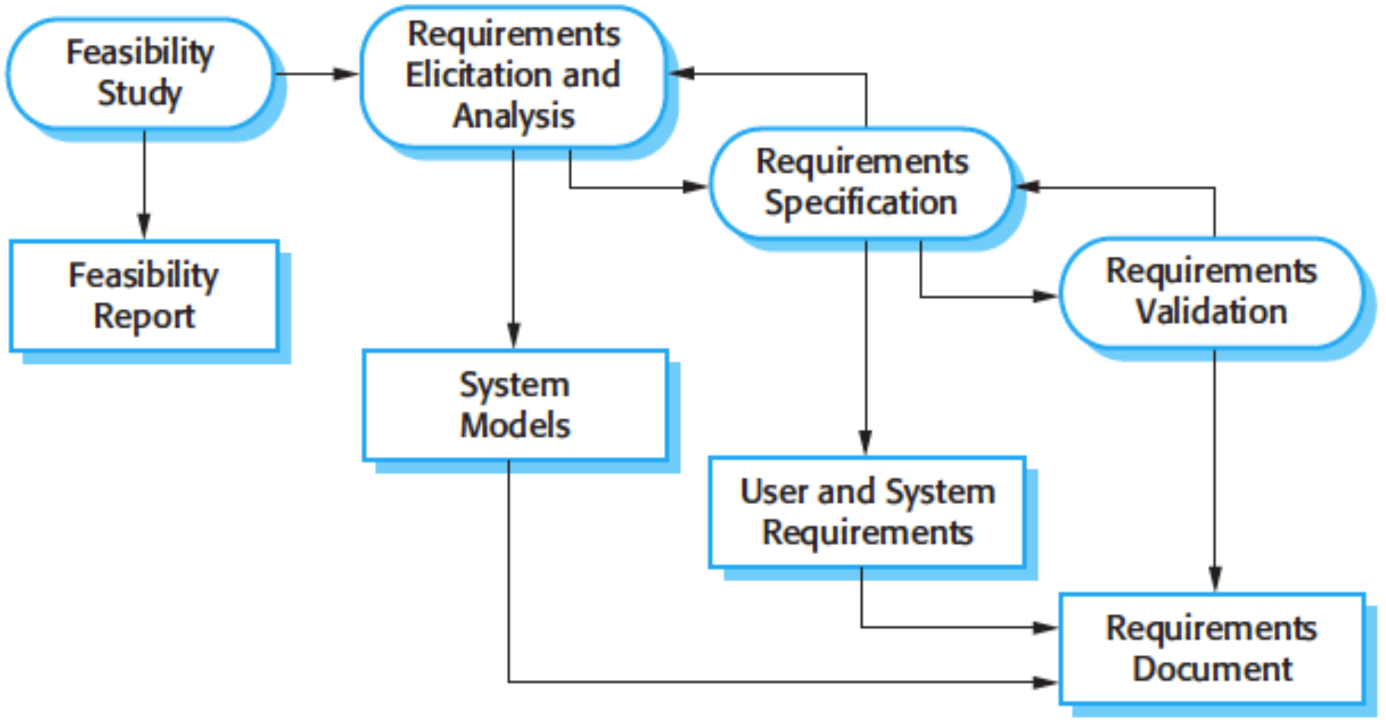


Figure 2.4 The requirements engineering process

Software design and implementation

- **Architectural design** → overall structure of the system, main components, their relationships
- **Interface design** → interfaces between system component
- **Component design** → how each component will operate?
- **Database design** → how will data structures will be represented in a database

Software design and implementation

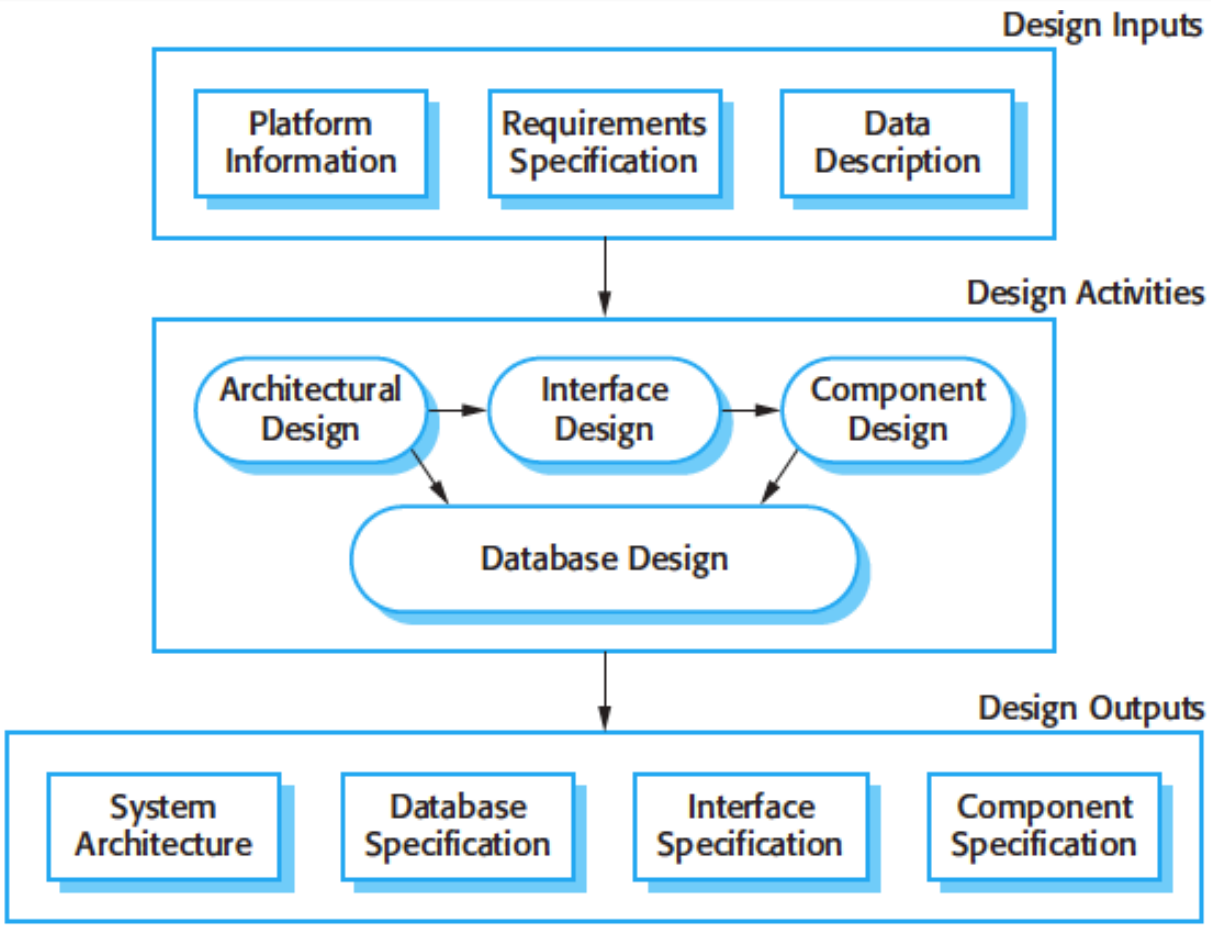


Figure 2.5 A general model of the design process

Software validation

- **Verification and Validation** (**V&V**) is intended to show that a system
  - **conforms to its specification**
  - **meets user's expectations**
- Main techniques →
  - **Validation Testing**
  - **Inspections and reviews**

Software validation

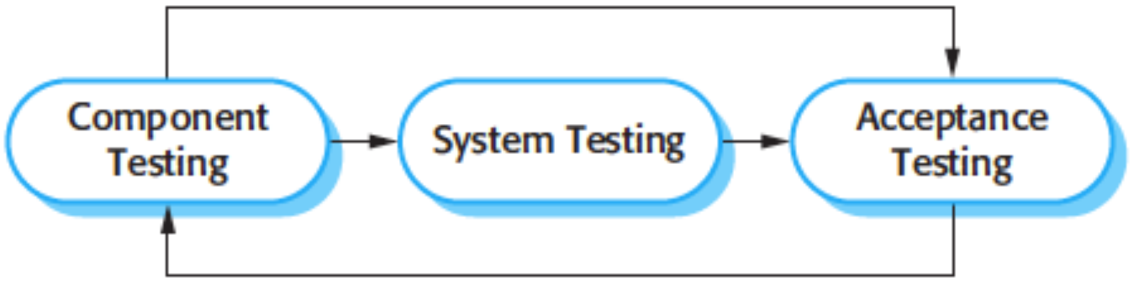


Figure 2.6 Stages of testing

Plan

1. Software development models
2. Software Processes Activities
3. **Methodological frameworks**

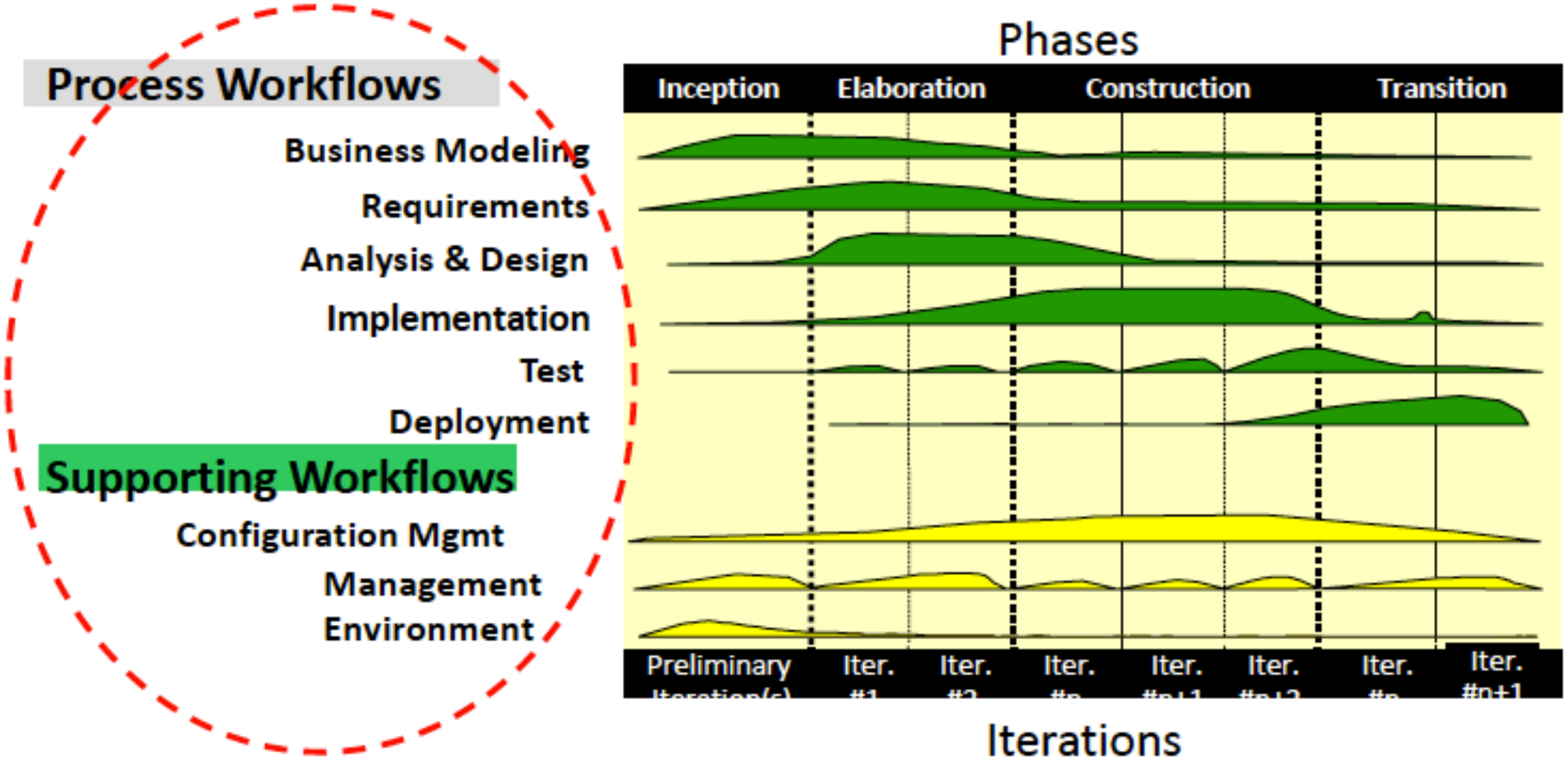
Unified process

- attempt to “**unify**” all things to all projects.
- Combined **sequential** and **iterative** with overlapping activities.
- Many variants, adaptations, including **RUP (Rational Unified Process)** and **Agile RUP (AUP)**

Rational Unified Process (RUP)

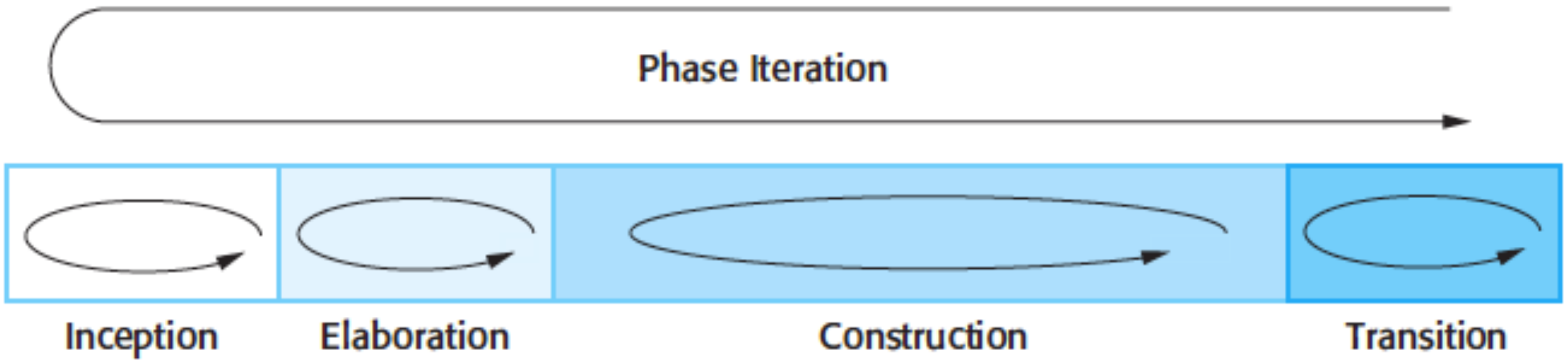
- architecture centered approach
- uses cases oriented
- iterative and incremental

Rational Unified Process (RUP)



Rational Unified Process (RUP)

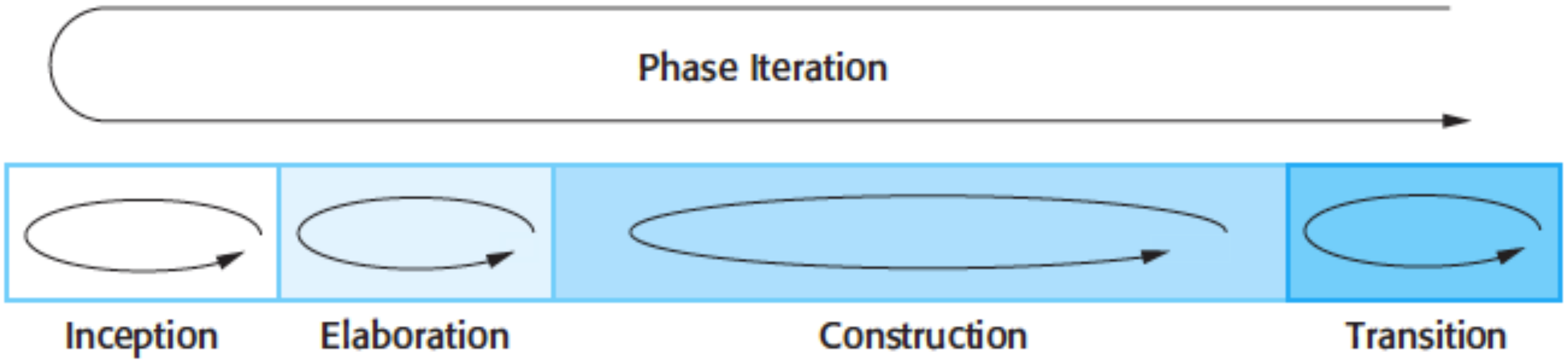
- **Inception →**
  - to establish a **business case** for the system
    - success criteria, requirements, resources of the project
  - **output** (**“artifacts”**) →
    - **project definition document**



Rational Unified Process (RUP)

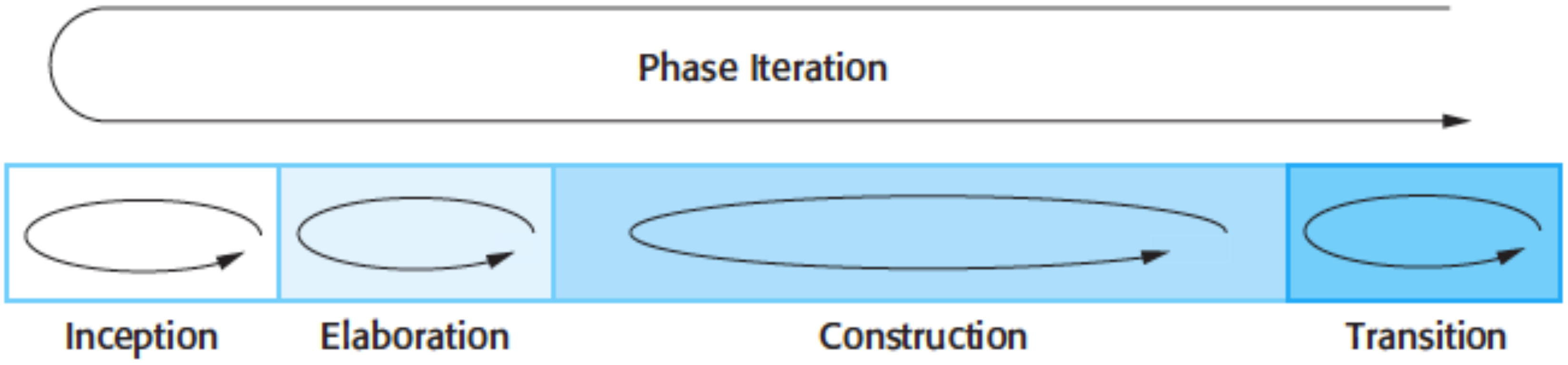
- **Elaboration →**
  - to establish an **arch. framework** of the system
  - to develop the **project** and **risk plans**
  - **Output →**

    use cases, **business class model, system design**



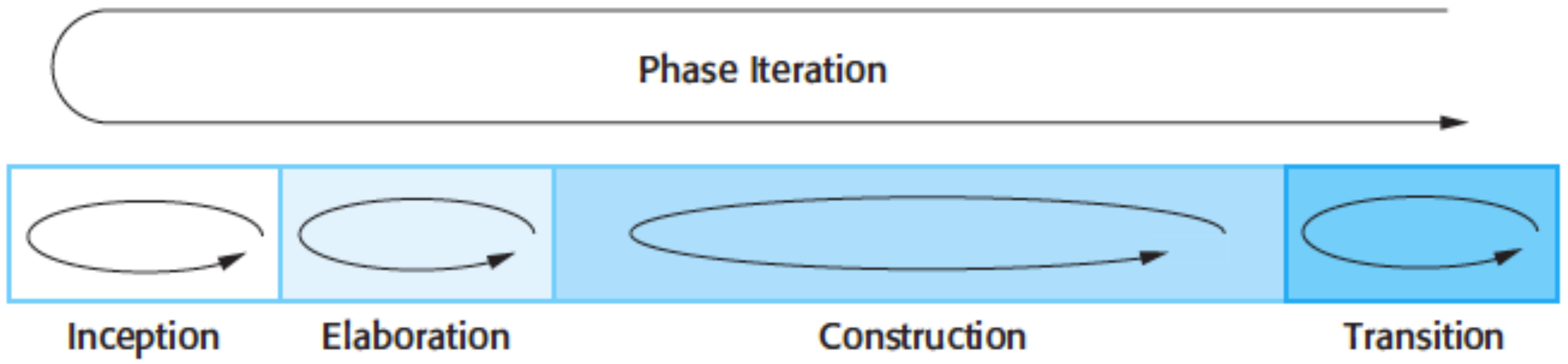
Rational Unified Process (RUP)

- **Construction →**
- to build the system over several iterations
- **Output →**
  - implementation of the system design
  - testing

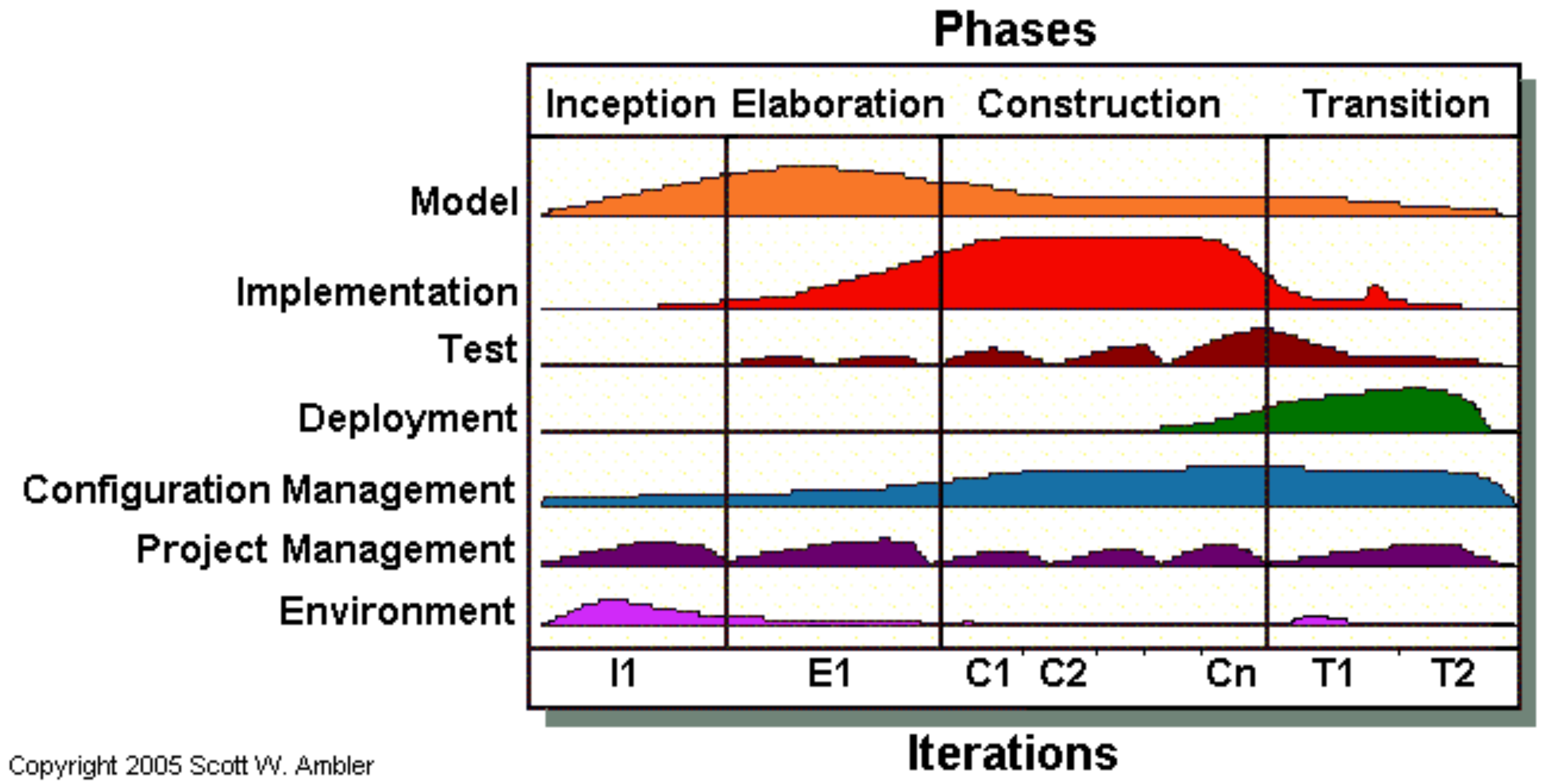


Rational Unified Process (RUP)

- **Transition ("Transición")→**
  - moving the system from the **development community** to the **user community**
  - **activities → beta testing, user training**

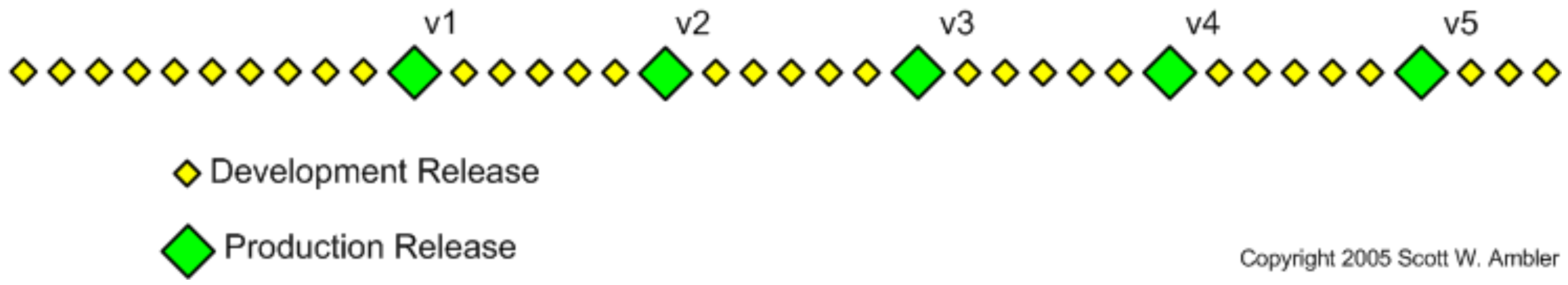


The Agile Unified Process (AUP) Lifecycle



Source: http://www.ambysoft.com/unifiedprocess/agileUP.html

Incremental releases over time



Source: http://www.ambysoft.com/unifiedprocess/agileUP.html

A Tour through AUP

http://www.ambysoft.com/unifiedprocess/agileUP.html

Next Class

- Agile software development

Today's Lab

1. Requirements Engineering
2. Exercise on writing software requirements