System Analysis and Design

(IT3120E)

Quang Nhat Nguyen

quang.nguyennhat@hust.edu.vn

Hanoi University of Science and Technology

School of Information and Communication Technology

Academic year 2022-2023

Content:

- Introduction of object-oriented system analysis and design
- Introduction of the modeling language UML
- **Introduction of software development process**
- Analysis of the environment and needs
- Function analysis
- Structure analysis
- Interaction analysis
- Behavior analysis
- Design of the system’s overall architecture
- Class detail design
- User interface design
- Data design

Introduction of software development process

- Definition of Software development process (SDP)
- Popularly used SDPs
- Software development process RUP

Definition of SDP

- Software development process (SDP)
  - A structured (ordered) set of activities required to develop a software system
- There exist several SDPs
  - Example: Waterfall, Prototyping, Spiral, etc.
  - There does not exist a single ideal SDP process suitable for all practical problems and requirements

Select a suitable SDP

- Type of software system to be built
  - Build the system from scratch >< Upgrade, update from an existing system
  - Common (popular) >< Customized or unique
  - Defined software requirements >< Software requirements change (quickly)
  - Critical system >< Business system
- Size of the software development project, Size of the development team, Project implementation time
- Characteristics of the software development team
  - Experience, Motivation (+ encouragement), Work attitude (effort)
- Budget of the software development project

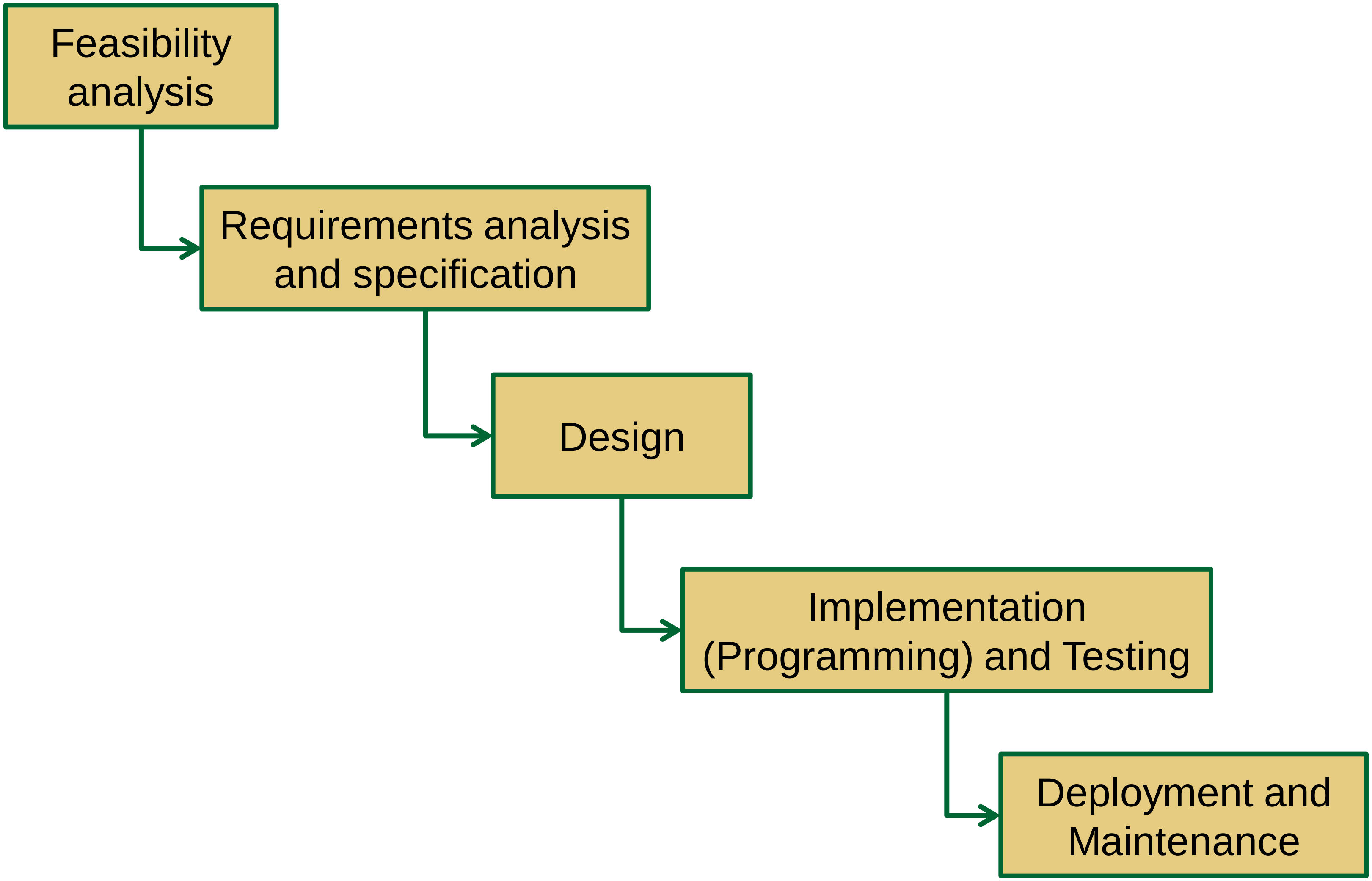
Main activities of an SDP

- Feasibility study
- Requirements analysis and specification
- Design
- Implementation
- Testing
- Deployment
- Maintenance

Popularly used SDPs

- Waterfall model
- Prototyping model
- Spiral model
- Agile model
- Unified model

Waterfall model



Feasibility analysis

Requirements analysis and specification

Design

Implementation (Programming) and Testing

Deployment and Maintenance

Waterfall model

- Introduced by Winston Royce in 1970, and is still the most used model in software development projects
- Software development is based on a set of sequentially ordered phases
  - The order of the phases is deterministic, and the results of a previous phase will be used as input for the following ones
- Once the software development process ends and the software system is handed over (signed off) to the customer, the software system cannot be changed or adjusted
  - The software development process can only be reopened (in response to adjustments/changes) through a formal change process
- The most important feature of the Waterfall model: **non-overlapping, non-repeating phases** (in a software development process)
  - The *Design* phase cannot start before the *Analysis* phase completes, and the *Testing* phase cannot start before the *Implementation* phase completes

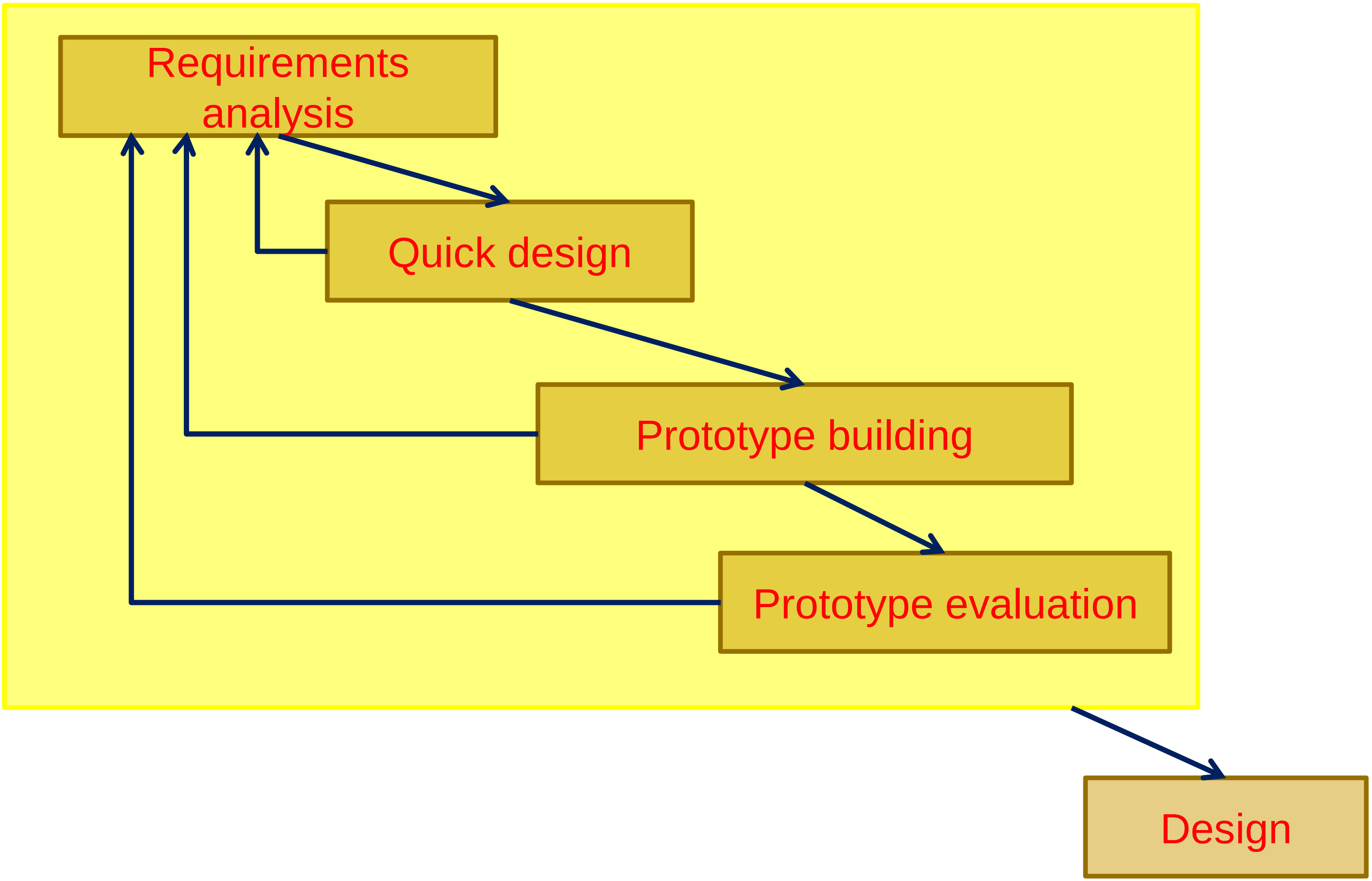
Waterfall model

- Advantages
  - Simple, easy to understand, and easy to use
  - The documents are completed after each phase
  - Software requirements are provided early to the testers
  - Allows the project manager (PM) to easily plan and control execution
  - This process is also very well known and known even by non-professional-software-development persons, making it easy to use for communication
- Disadvantages
  - Only suitable for real problems **when the software requirements are clearly defined, complete and fixed from the beginning** (before the Design phase)
  - Not suitable for long-running projects
  - Those projects that may have risk or uncertainty factors
  - Hard (or even impossible) to have initial results (versions) of the software soon

Waterfall model

- When to use?
  - **Software requirements are clearly defined, complete and fixed**
  - The definition of the product (software system) does not change
  - The related and necessary technologies are mastered
  - The resources and experience of the software development team are sufficient
  - The time of the project execution is short (not long)

Prototyping model



Prototyping model

- Instead of fixing requirements before proceeding with design or implementation (programming), **a prototype (or several) is built to understand the exact software requirements**
- Each prototype is built upon current software requirements (obtained from evaluation of previous prototypes)
- By using prototypes, customers can get a “real feel” of the software system, because interactions with the prototype allow customers to have a better, more precise understanding of the requirements of the desired software system
- Using prototypes is reasonable for the development of large and complex software systems (when there is no requirement gathering process or inbuilt system to help define the software requirements)
- A prototype is usually not a complete software system, and many details are not implemented in the prototype

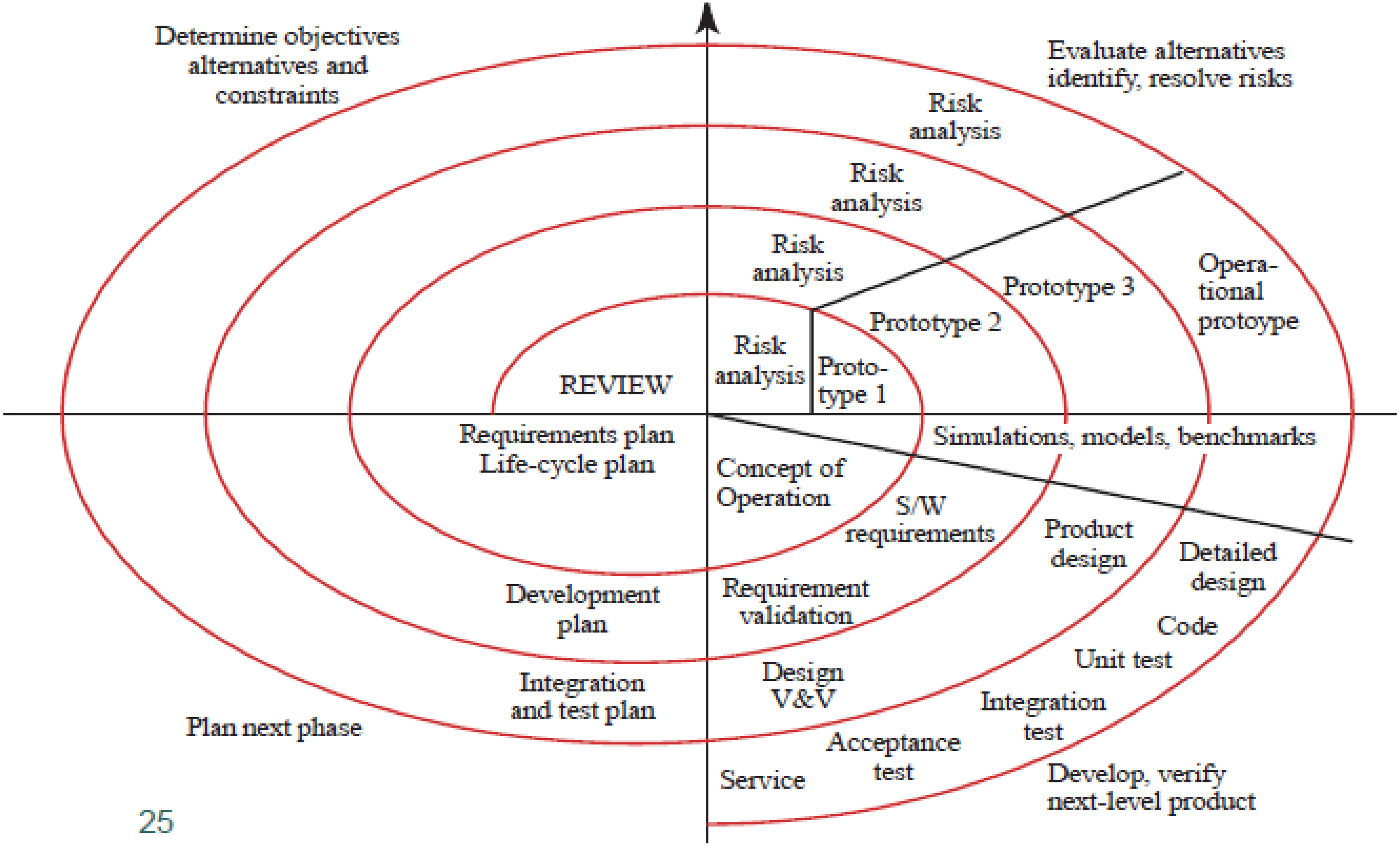
Prototyping model

- Advantages
  - Users are actively involved in the software development process
  - Using a prototype as a working model of the system, the users gain a better understanding of the system being built
  - Errors, problems can be detected (very) early
  - Early to get evaluation feedback from users for better software development solutions
  - Missing functions can be discovered early
  - Functions that are unclear or difficult to operate may be detected
- Disadvantages
  - Users may think that software development is easy, and thus become inconsistent in the expression of requirements
  - Planning is not done at the beginning of the project, which may lead to project management problems: undefined deadlines, budgets and deliverables
  - This prototyping model often leads to prolong software development process
  - Developers tend to deliver a basic working prototype, rather than a real complete product

Prototyping model

- When to use the prototyping model?
  - **When software requirements cannot be determined at the time of project initiation**
  - **When users (for various reasons) cannot express their requirements clearly**
  - This prototyping model is well suited for developing the "look and feel" or user interface of the system, because these features are *difficult to describe by documentation*, but often obtained through trial use
  - When customers ask for proof of feasibility
  - When demos are needed for senior management board
  - When technology problems need to be tried and tested

Spiral model



Spiral model

- Proposed by Barry Boehm
- An evolutionary development model, based on a hybrid combination of the iterative development feature of the Prototyping model and the sequential development feature of the Waterfall model
  - **Focus on risk analysis**
- In the spiral model, the software system is developed through a series of incremental releases
  - In the initial iterative steps of development, versions of the software system can be simply sketched models on paper or prototypes
  - In later iterative development steps, increasingly mature versions of the software system are created

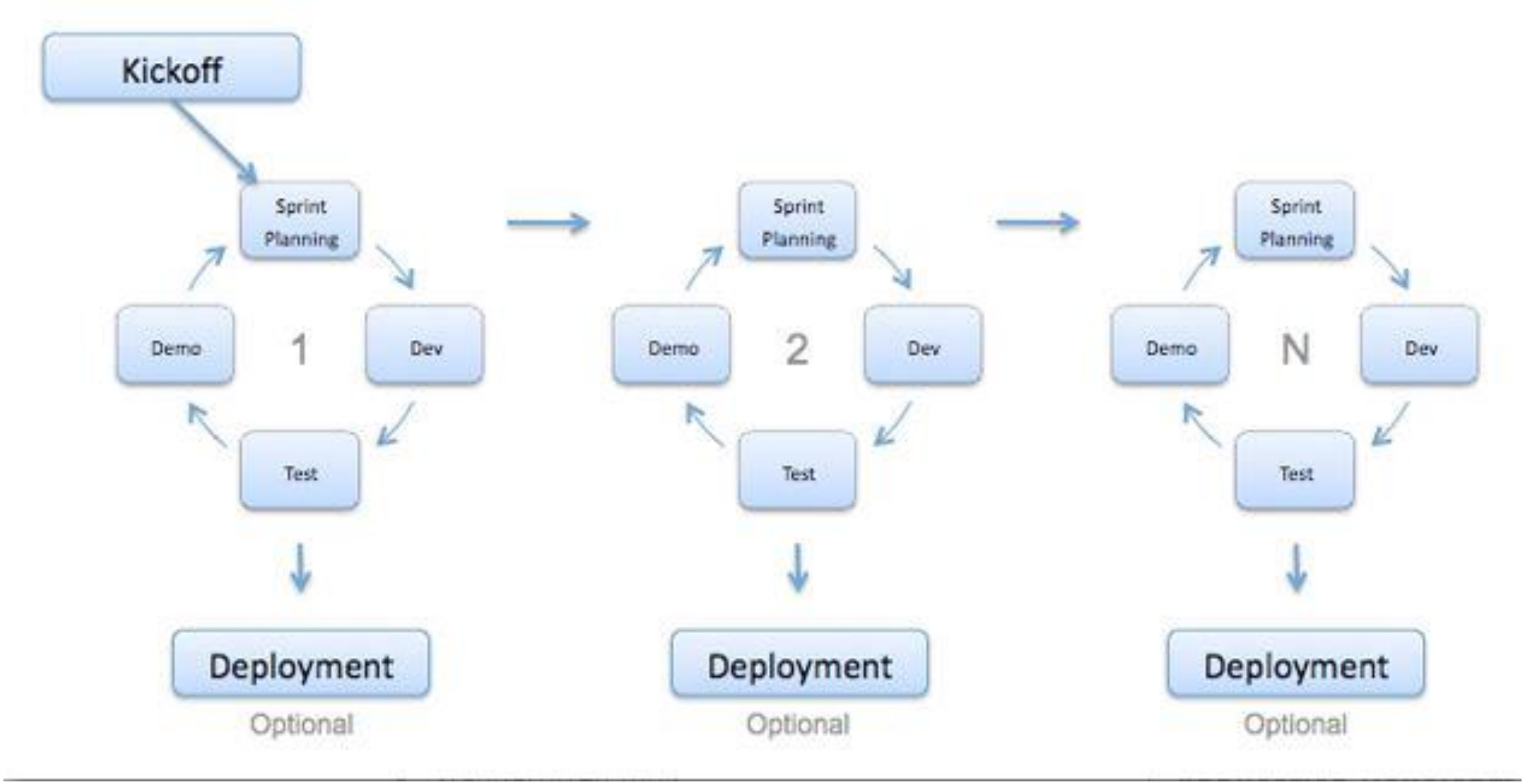
Spiral model

- Advantages
  - Focus on risk analysis, therefore help reduce risks in software development projects
  - Suitable for large and particularly important projects
  - New functions may be added later
  - Initial versions of the software system are created early
- Disadvantages
  - High cost (time, resources, money) to apply
  - Risk analysis requires high skills and experience
  - The success of the project depends strongly on the risk analysis phase
  - Not suitable for small projects

Spiral model

- When to use the spiral model?
  - **When the assessment (analysis) of costs and risks is important**
  - For medium- to high-risk projects
  - Users are uncertain about their needs
  - Complex and large software requirements
  - Need to develop a new product line
  - Desire for significant changes (careful research and investigation is required)

Agile model


Kickoff
Sprint Planning
Demo
1
Dev
Test
Deployment
Optional
Sprint Planning
Demo
2
Dev
Test
Deployment
Optional
Sprint Planning
Demo
N
Dev
Test
Deployment
Optional

Agile model

- Is an incremental and iterative type of model
- The software system is developed through incremental and rapid cycles
- Helps create small-scale enhanced versions (of the software system) where a next version is built on the features of the previous one
- Each enhanced version is carefully tested to ensure software quality
- **Used for software development projects that require quick completion times**
  - Extreme Programming (XP) is one of the famous software development methods belonging to the agile model

Agile model

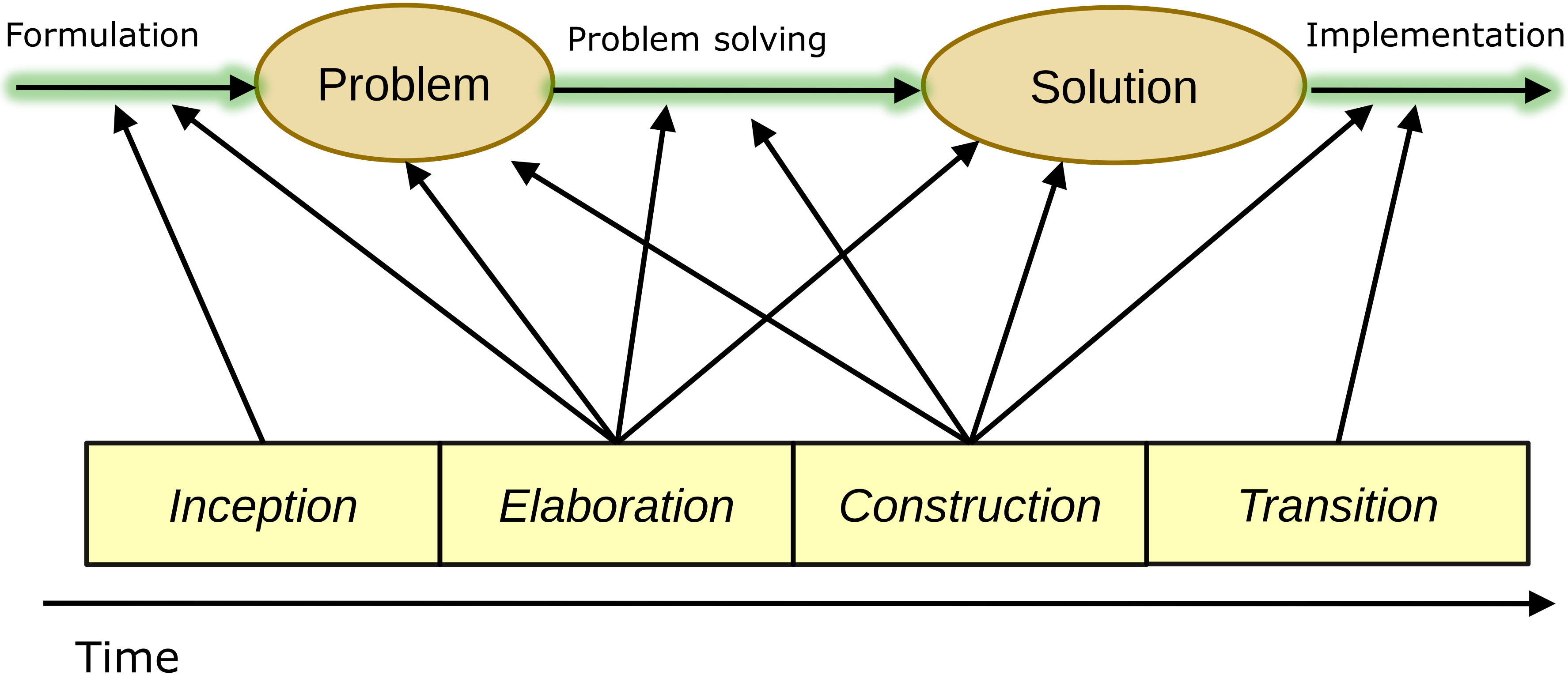
- Advantages
  - Satisfy customers with agile enhanced software versions
  - Emphasis on interactions between actors rather than processes and tools (i.e., customers, developers, and testers constantly interact with each other)
  - Frequent communication between the business analysis team and the programming team
  - **Adapt (response) quickly to changing requests**
  - Even allowing changes of the software requirements are added later
- Disadvantages
  - For large software systems, this agile model makes it difficult to estimate the necessary costs and effort at the beginning of the software development process
  - Less emphasis on required design and documentation
  - Usually only senior developers can make the necessary decisions during development (i.e., not suitable for inexperienced developers, unless working in conjunction with experienced ones)

Agile model

- When to use the agile model?
  - New changes can be implemented at low cost, by the frequent creation of enhanced versions
  - To implement a new feature, developers only need a few days, or even a few hours to implement
  - Unlike the waterfall model, in the agile model, planning is (much) less cost. **The agile model assumes that user needs will (often) change. Changes can always be requested, and features can always be added or removed based on customer feedback.** This helps give customers the software system they need and want to use
  - Both developers and users of the system find they have more freedom of time and choices than models that strictly follow a sequence of steps (e.g., the waterfall model)

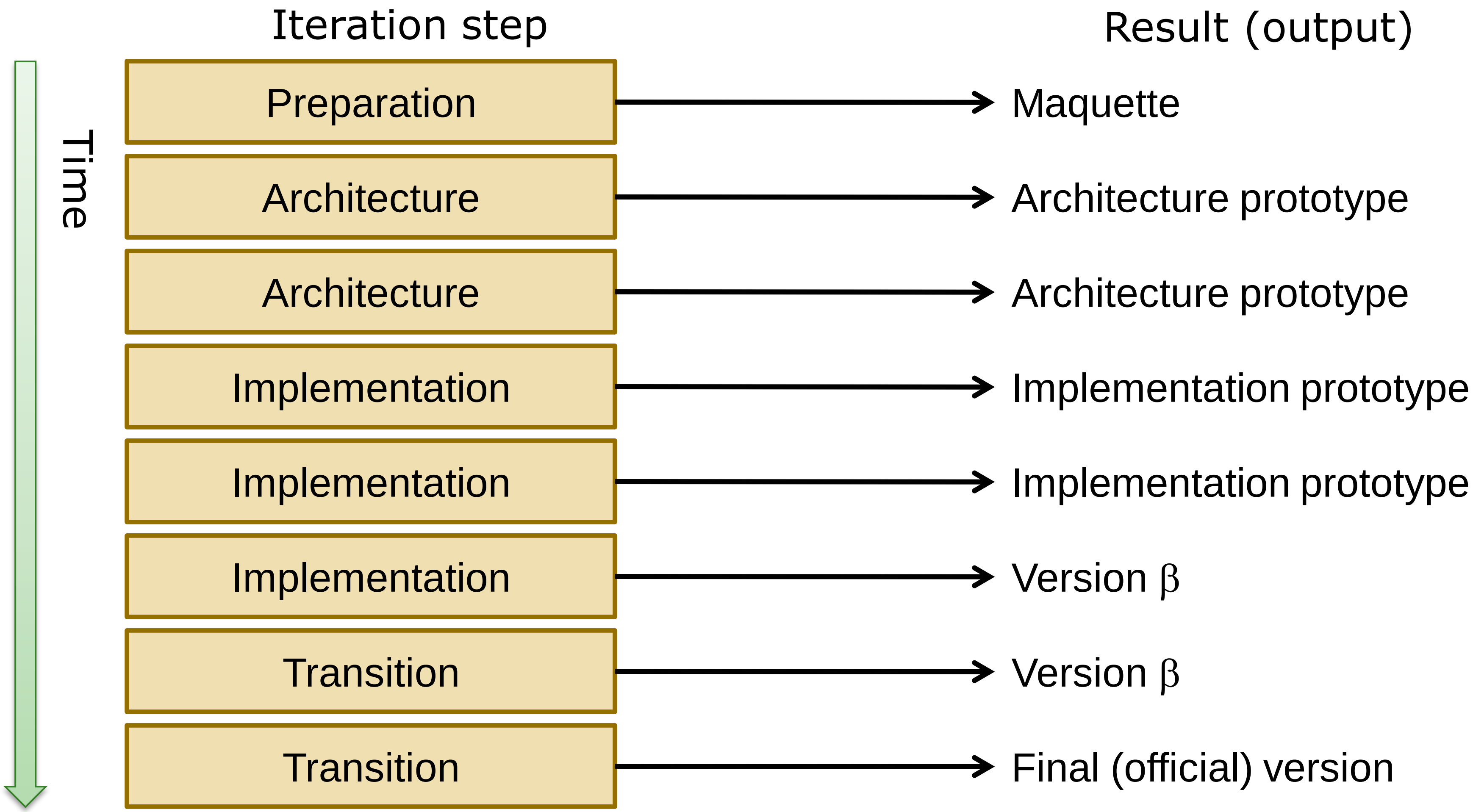
Unified model

- Project-management view



Unified model

- Technical view

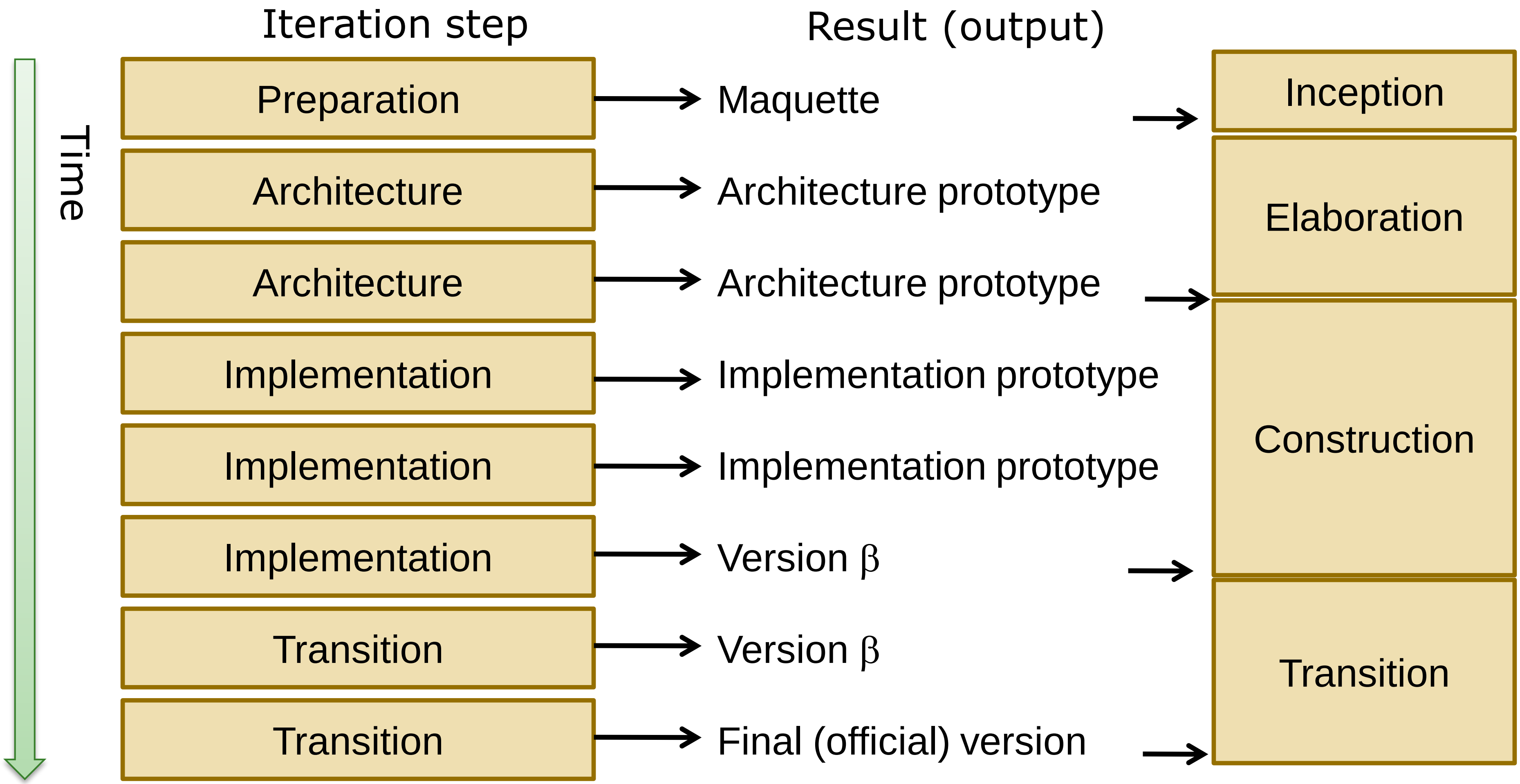


Time

Iteration step	Result (output)
Preparation	Maquette
Architecture	Architecture prototype
Architecture	Architecture prototype
Implementation	Implementation prototype
Implementation	Implementation prototype
Implementation	Version β
Transition	Version β
Transition	Final (official) version

Unified model

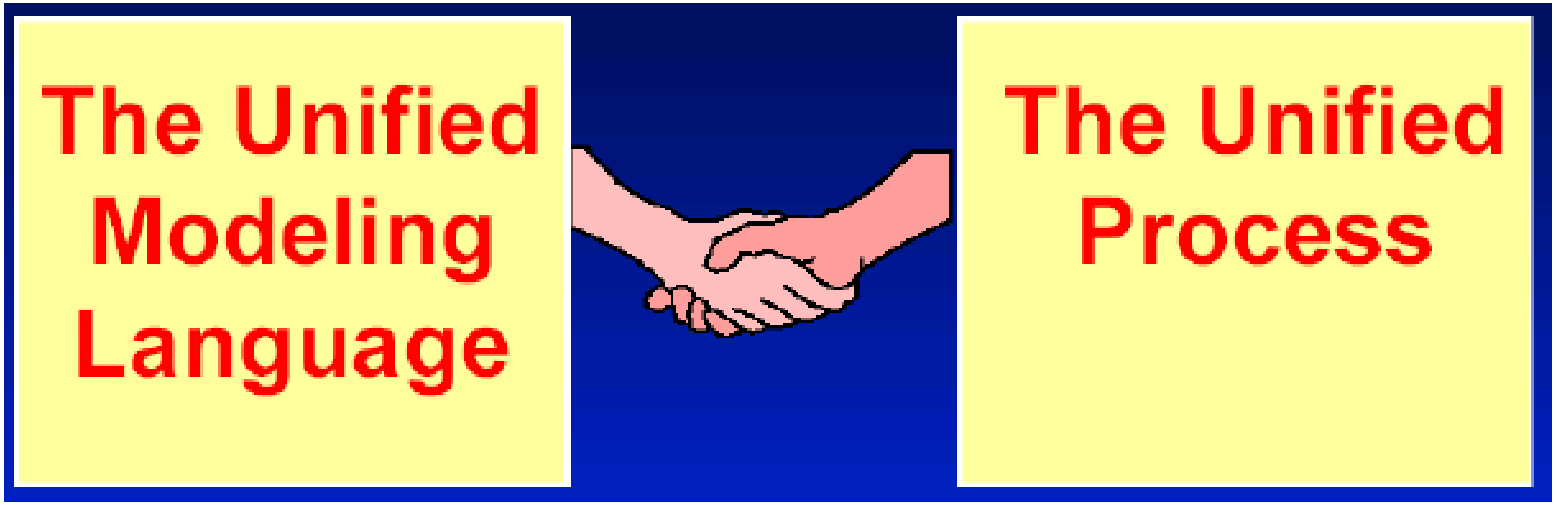
- Combination of the 2 views



Time

Iteration step	Result (output)	
Preparation	Maquette	Inception
Architecture	Architecture prototype	Elaboration
Architecture	Architecture prototype	
Implementation	Implementation prototype	Construction
Implementation	Implementation prototype	
Implementation	Version β	
Transition	Version β	Transition
Transition	Final (official) version	

Unified model and UML



The Unified Modeling Language

The Unified Process

RUP

RUP (Rational Unified Process) is a modeling process using the modeling language UML:

- Basic principles
- Main phases
- Main steps

Basic principles (1)

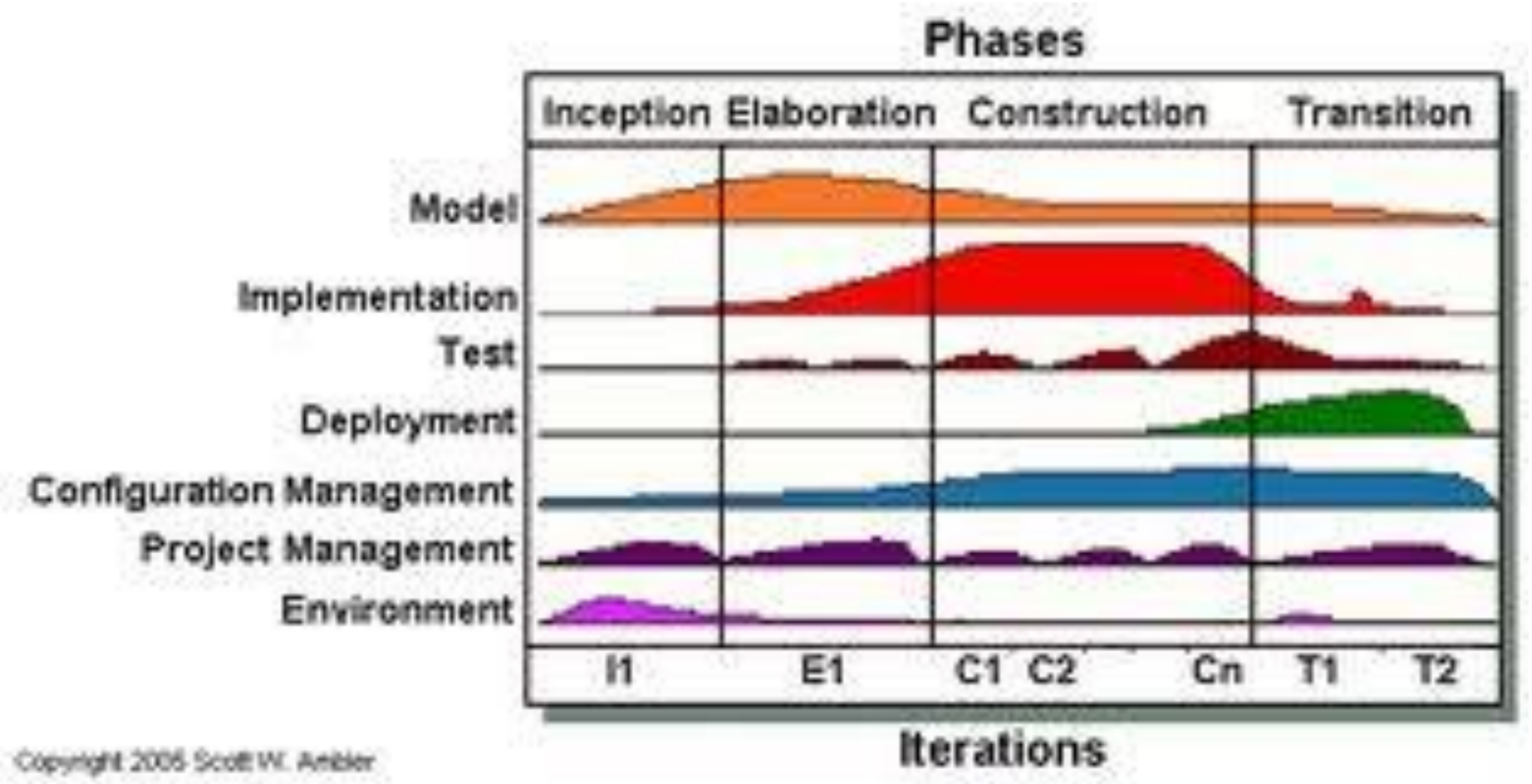
- Iteration and incremental
  - The project is divided into short loops or stages for easy control
  - At the end of each loop, the executable part of the software system is produced in a gradually added
- Focus on architecture
  - The complex system is divided into modules for easy deployment and maintenance
  - This architecture is presented in 5 different views

Basic principles (2)

- Led by use cases
  - Use cases influence every phase of the system development, are the basis for defining loops and enhancement, and are the basis for dividing work within the team
  - **Needs understanding**: Detect the use cases
  - **Analysis**: Dive into description (i.e., specification) of the use cases
  - **Design and implementation**: Build the system according to the use cases
  - **System testing and acceptance**: Follow the use cases
- Control the risks
  - Early detect and eliminate risks to the software development project

Main phases of RUP (1)

- RUP is organized into 4 phases: Inception, Elaboration, Construction, and Transition



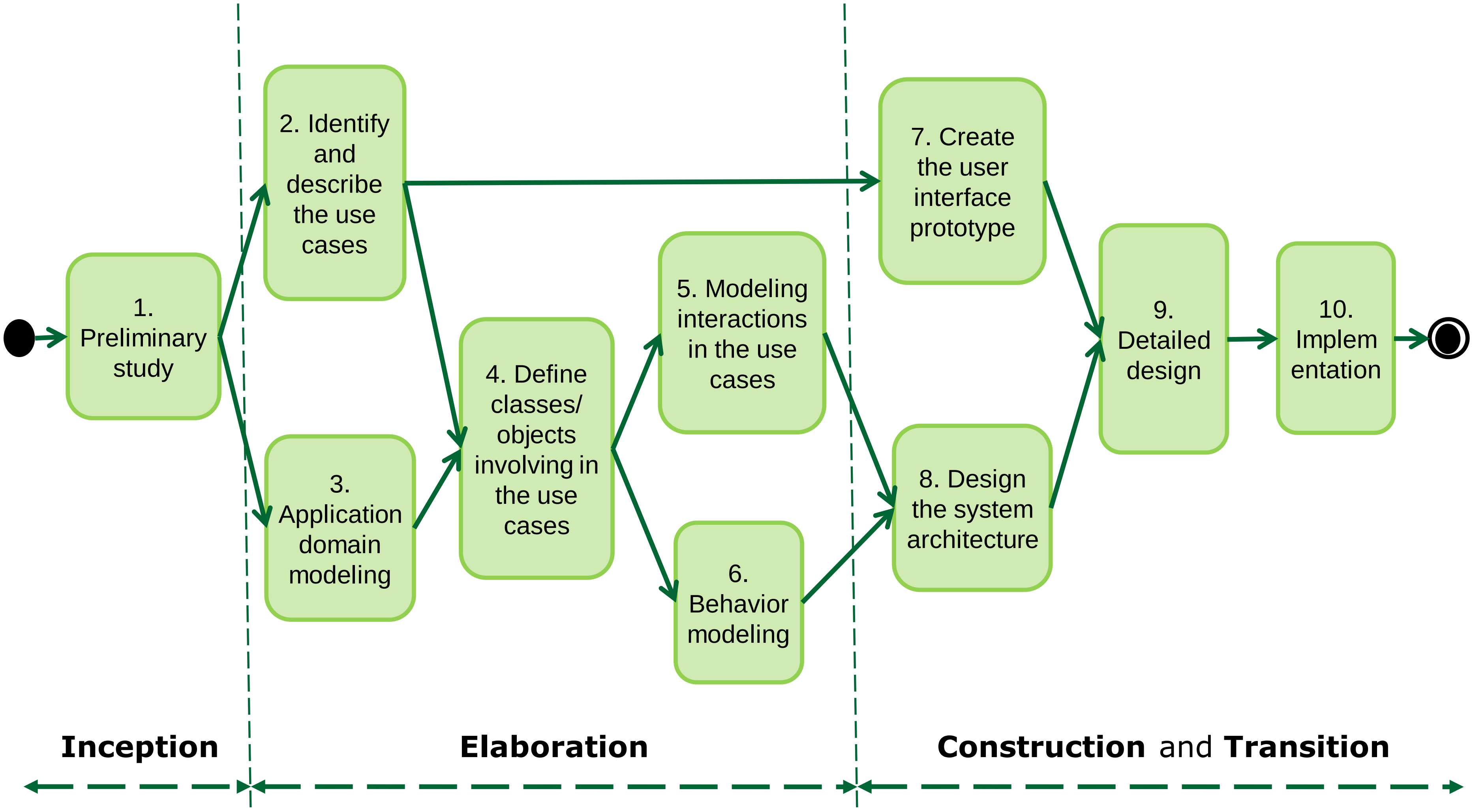
Main phases of RUP (2)

- **Inception**
  - Give *an overview of the software system* (functions, performance, technology, ...) and *the software development project* (scope, goals, feasibility, etc.) => *Make conclusion of whether to develop or give up the project*?
- **Elaboration**
  - *More detailed analysis of the system* (functions and static structures)
  - Propose a system *architecture prototype*

Main phases of RUP (3)

- **Construction**
  - Focus on *the system design* and *implementation*
  - *The system architecture is detailed and edited*
  - Finishes when a complete system with accompanying technical documentation is created
  - This is the phase that takes the most time and effort
- **Transition**
  - *Transfer the system* to the end users: data conversion, installation, testing, training, etc.

Main steps of RUP (1)


1. Preliminary study
2. Identify and describe the use cases
3. Application domain modeling
4. Define classes/ objects involving in the use cases
5. Modeling interactions in the use cases
6. Behavior modeling
7. Create the user interface prototype
8. Design the system architecture
9. Detailed design
10. Implem entation
Inception
Elaboration
Construction and **Transition**

Main steps of RUP (2)

1. **Preliminary study**
   - Give an overview of the software system (functions, performance, technology, ...) and the software development project (scope, goals, feasibility, etc.)
   - Make conclusion: Whether to develop or give up the project?
2. **Identify and describe the use cases**
   - Understand user needs and identify the use cases
   - Each use case must be specified (described) in the form of a scenario and/or a sequence diagram
3. **Application domain modeling**
   - Provide class diagrams that reflect all concepts and businesses
   - <u>The classes here are domain classes</u> (not design classes)

Main steps of RUP (3)

4. **Define classes/ objects involving in the use cases**
   - For each use case, define entity classes, control classes, boundary classes
5. **Modeling interactions in the use cases**
   - Objects interact by exchanging messages
   - Create use case scenarios: Sequence diagrams, Communication diagrams
6. **Behavior modeling**
   - Control objects have the ability to react to events coming from outside
   - Use state machine diagrams to describe the behavior of control objects

Main steps of RUP (4)

7. **Create the user interface prototype**
    - Use graphical user interface (GUI) design tools to create (design) interfaces prototype early, making the system’s modeling and implementation easier
8. **Design the system architecture**
    - Design the system’s overall architecture
    - Divide into sub-systems
    - Use component diagrams to describe physical components
    - Use deployment diagrams to describe the arrangement and deployment of the system's executable components to the hardware and infrastructure platform

Main steps of RUP (5)

9. **Detailed design**
   - Detailed design for classes, associations, properties, methods
   - Determine the system’s implementation solution
10. **Implementation**
    - Programming and testing
    - The system is approved (i.e., accepted) on the use cases

Support tools (1)

- Support for system development programming (Integrated Development Environment – IDE)
  - Write source codes, compile
  - Debug, test
  - Create interfaces prototype
- Support for system modeling
  - Produce, transform, modify models and diagrams
  - Check the syntax of models
  - Store and manage versions of models
  - Test and evaluate models
  - Simulate and execute models
  - Generate models from existing source codes (i.e., reverse engineering)

Support tools (2)

- Support the management of system development process and project
  - Guide and support, specify the work and deliverables (i.e., outputs) at each stage
  - Support iterative process
  - Support teamwork
  - Assist in project management, planning and monitoring