Chapter 1: Object-Oriented Analysis and Design (OOA/D)

An Introduction to Object-Oriented Analysis and Design

PART I: INTRODUCTION

Dr. 邱 明
Software School of Xiamen University
mingqiu@xmu.edu.cn
Fall, 2009

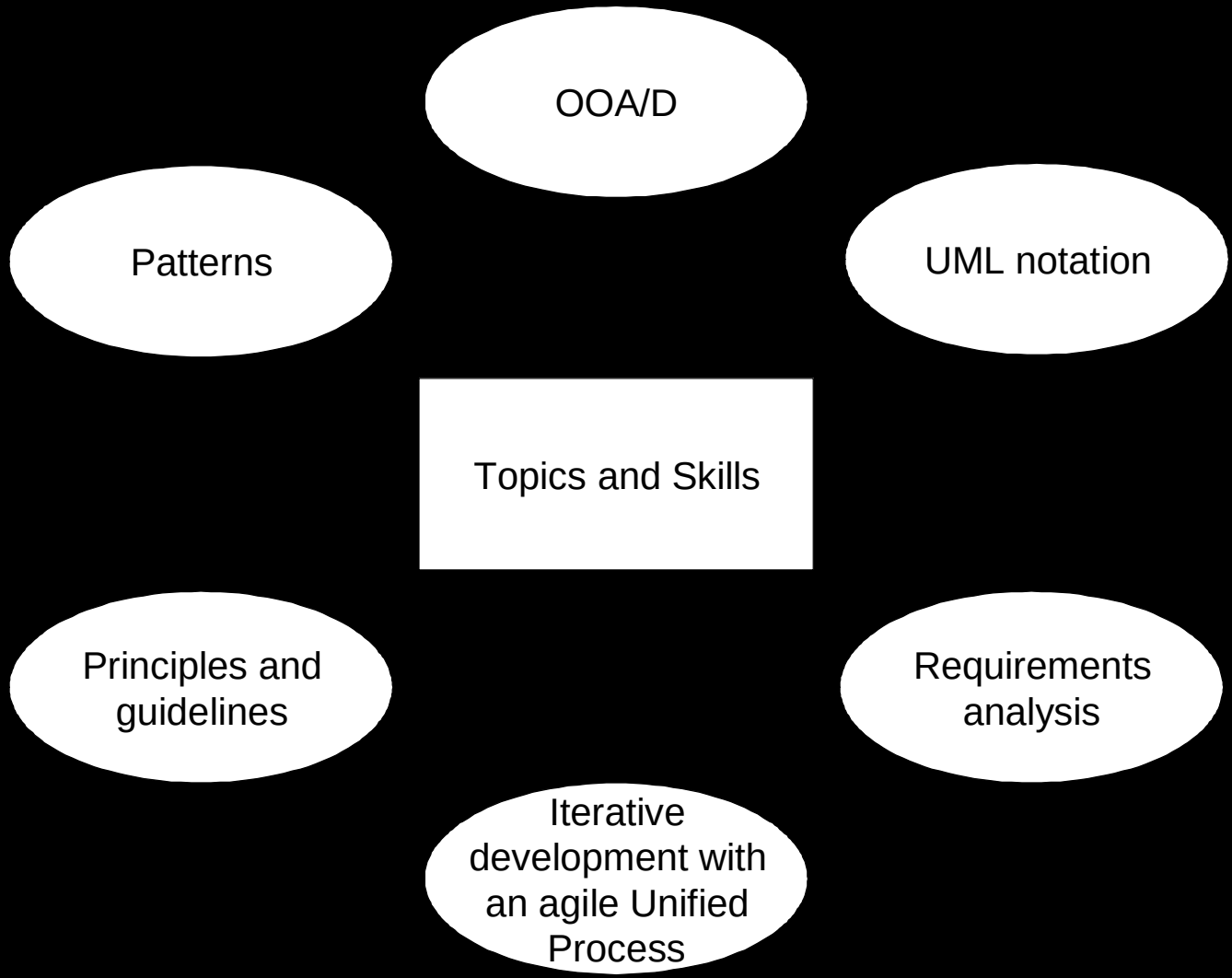
Objective

- Describe the book goals and scope.
- Define object-oriented analysis and design (OOA/D).
- Illustrate a brief OOA/D example.
- Overview UML and visual agile modeling

1.1. What Will You Learn? Is it Useful?

- Owning a hammer doesn't make one an architect.
- What does it mean to have a good object design?
- How to create a well-designed, robust and maintainable software using OO technologies and languages

1.1. What Will You Learn? Is it Useful?



1.1. What Will You Learn? Is it Useful?

This book helps you:

- Apply principles and patterns to create better object designs.
- Iteratively follow a set of common activities in analysis and design.
- Create frequently used diagrams in the UML notation

1.2. The Most Important Learning Goal?

- How to assign responsibilities to software objects
  - Nine fundamental principles in object design and responsibility assignment -- GRASP

1.3. What is Analysis and Design?

Analysis

- Emphasize an investigation of the problem and requirements

Design

- Emphasizes a conceptual solution that fulfills the requirements

1.4. What is Object-Oriented Analysis and Design?

Analysis

- Emphasize finding and describing the object(concept) in problem domain

Design

- Emphasizes defining software objects and how they collaborate to fulfill the requirements

1.4. What is Object-Oriented Analysis and Design?

domain concept

Plane
tailNumber

visualization of domain concept

representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

1.5. A Short Example – Dice game

1.5. A Short Example – Dice game

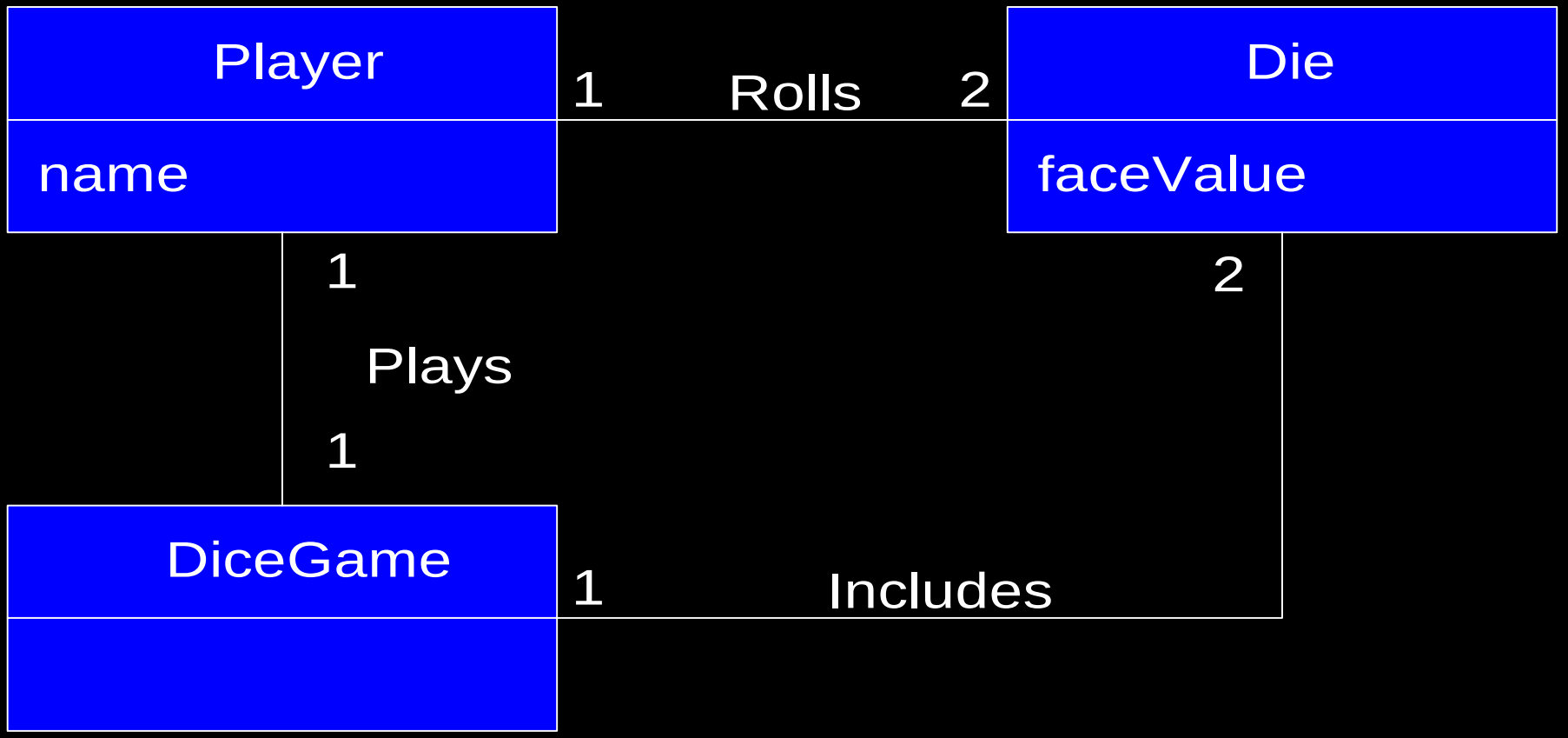
A bird's-eye view of a few key steps and diagrams

- Define Use Cases
  - Write stories or scenarios of how people use the application

> Play a Dice Game: Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.

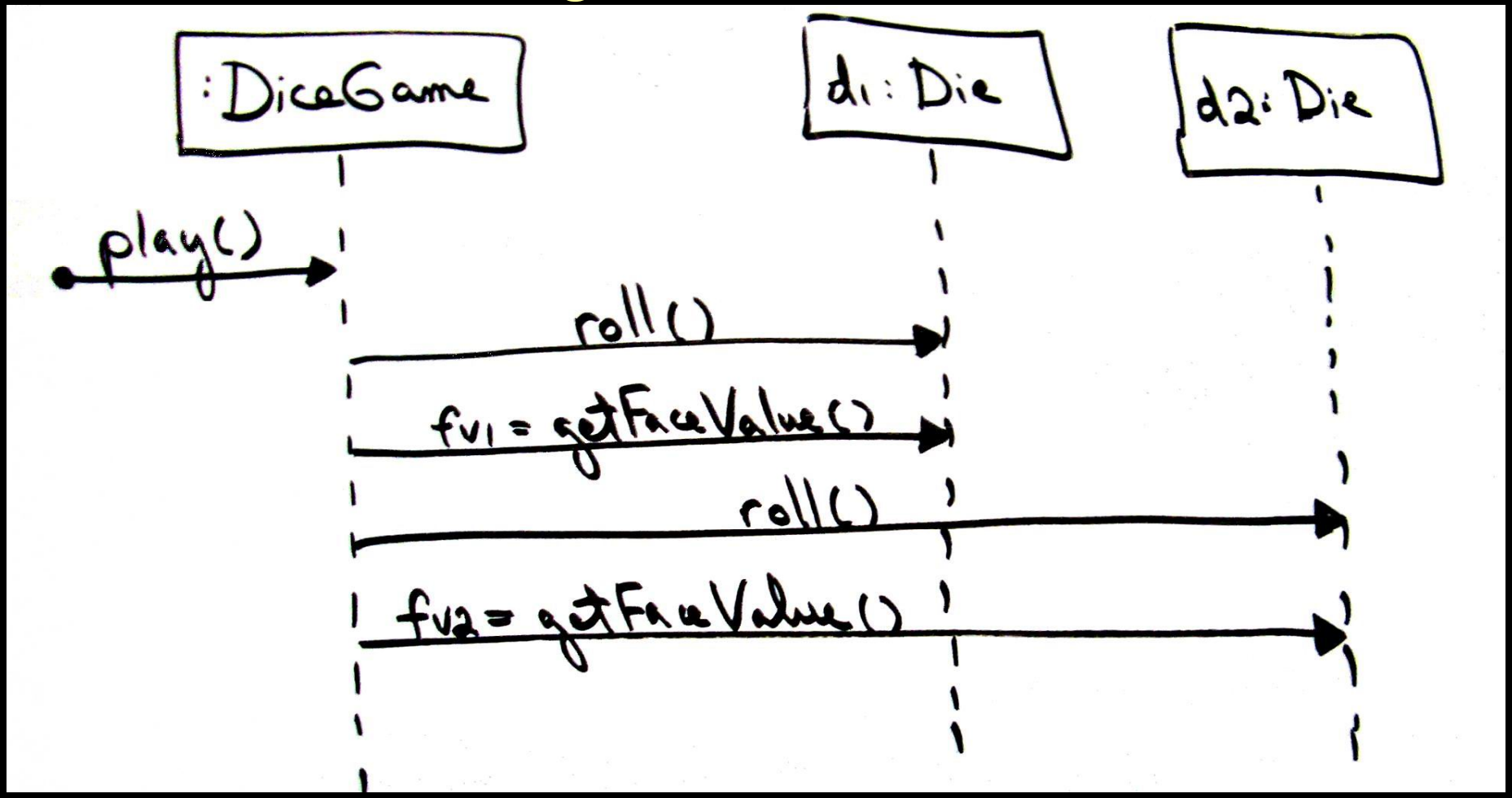
1.5. A Short Example – Dice game

- Define a Domain Model

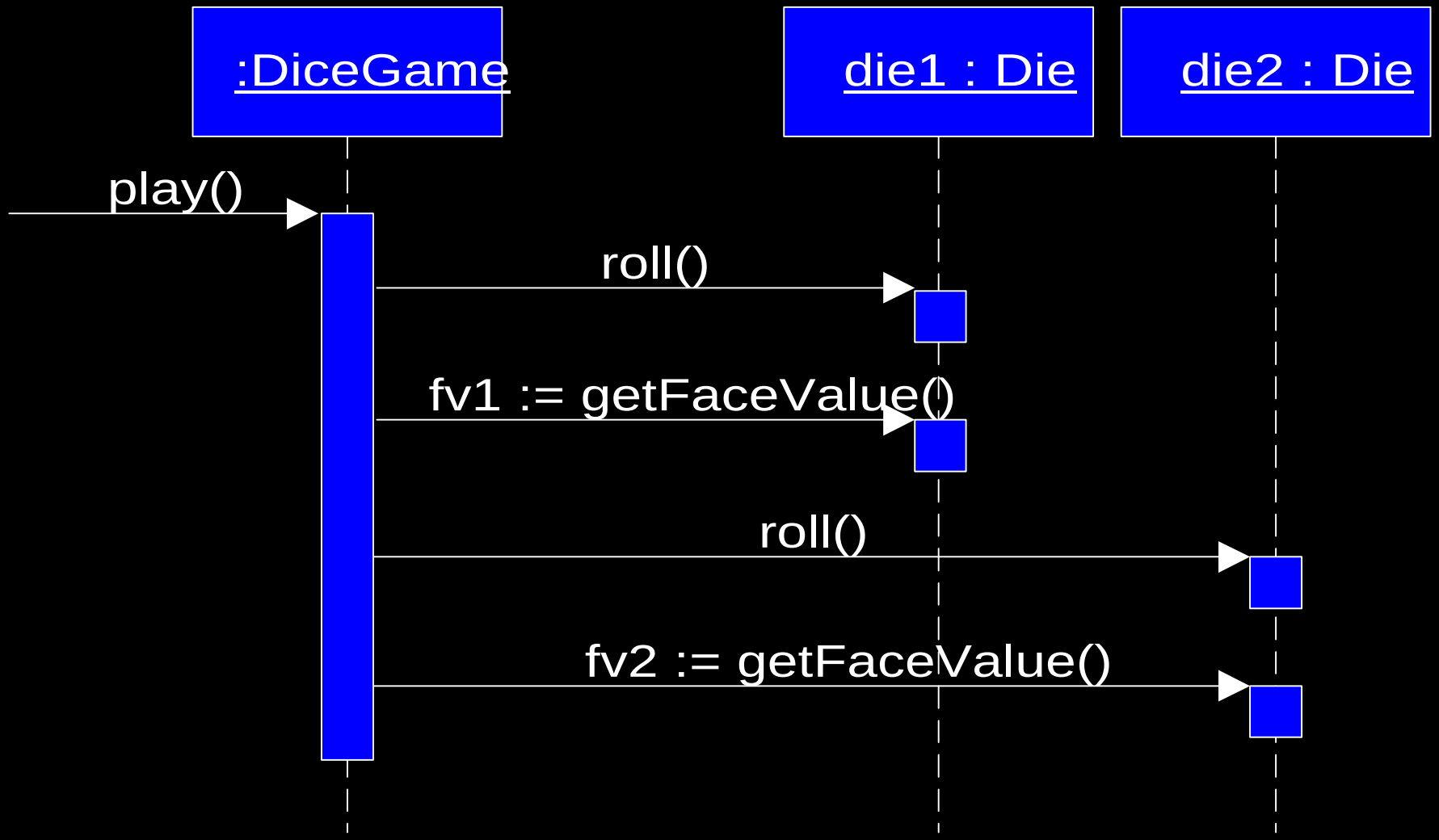


1.5. A Short Example – Dice game

- Assign Object Responsibilities and Draw Interaction Diagrams

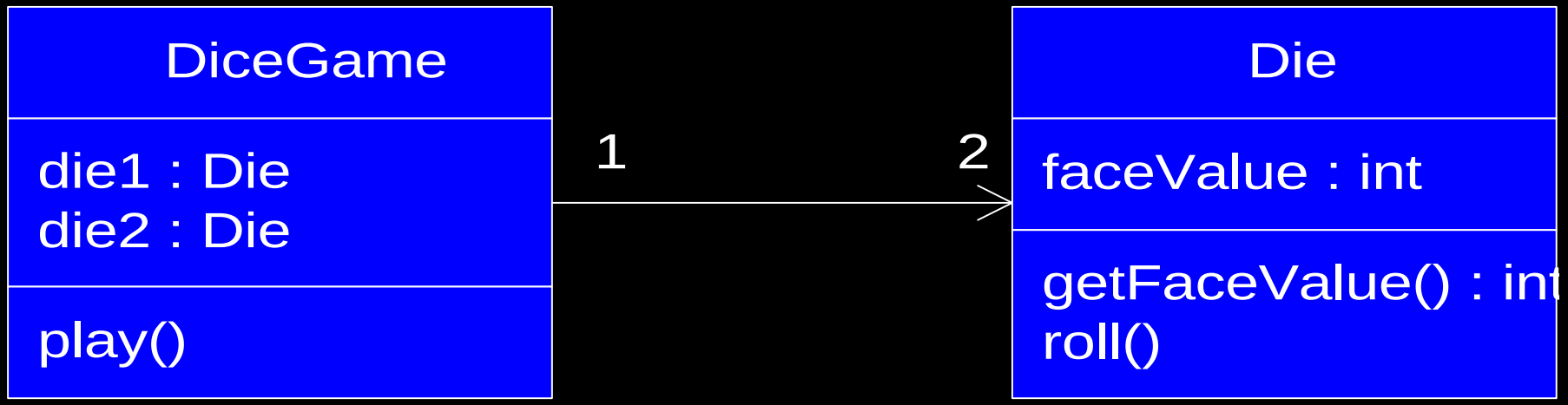


1.5. A Short Example – Dice game



1.5. A Short Example – Dice game

- Define Design Class Diagrams



Lower Representational Gap (LRG)

1.6 What is the UML?

The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems

- Three Ways to Apply UML
  - UML as sketch
  - UML as blueprint (Reverse Engineering/ Code Generation)
  - UML as programming language (MDA)

1.6 What is the UML?

UML and "Silver Bullet" Thinking

- What is "Silver Bullet“
  - From "No Silver Bullet" by Dr. Frederick Brooks, in his classic book “Mythical Man-Month” (20th anniversary edition).
- UML is not a “Silver Bullet”.
  - A person without good OO design and programming skills who draws UML is just drawing bad designs

1.6 What is the UML?

Three Perspectives to Apply UML

- Conceptual perspective
  - Diagrams are interpreted as describing things in a situation of the real world
- Specification (software) perspective
  - Diagrams describe software abstractions or components with specifications and interfaces (no implementation)
- Implementation (software) perspective
  - Diagrams describe software implementations in a particular technology (such as Java, C#).

1.6 What is the UML?

Conceptual Perspective
(domain model)

Raw UML class diagram notation used to visualize real-world concepts.

Specification or
Implementation
Perspective
(design class diagram)

Raw UML class diagram notation used to visualize software elements.

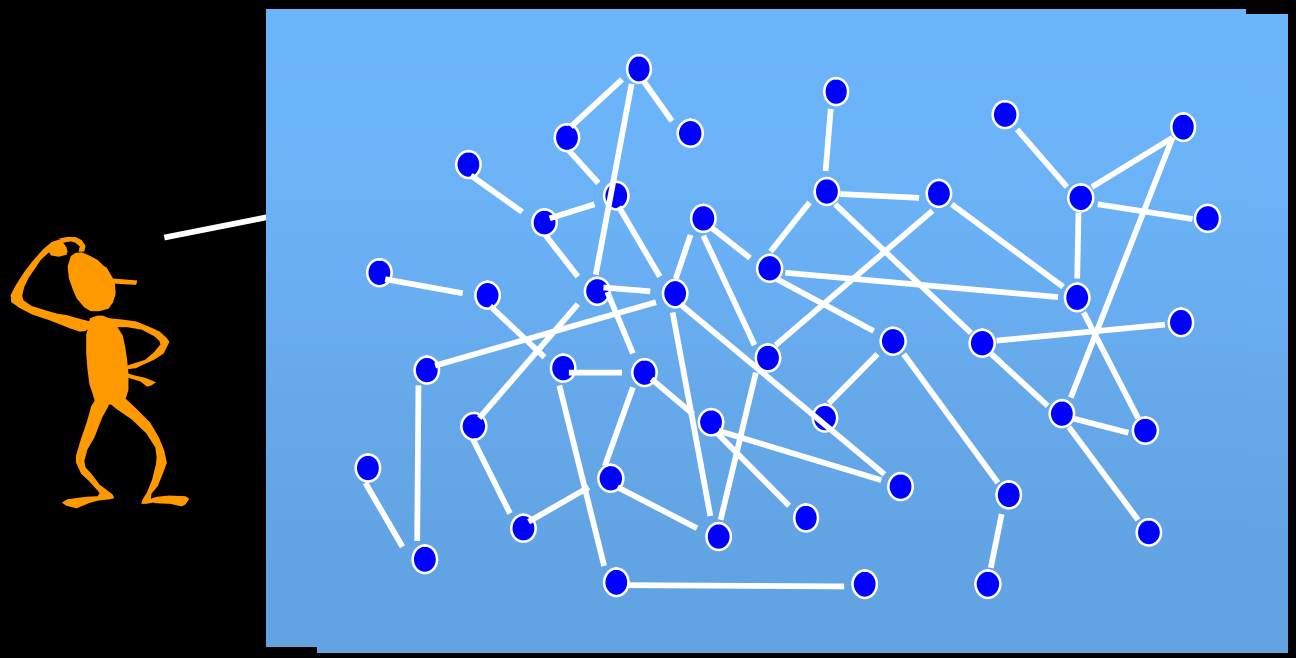
1.6 What is the UML?

Meaning of "Class" in Different Perspectives

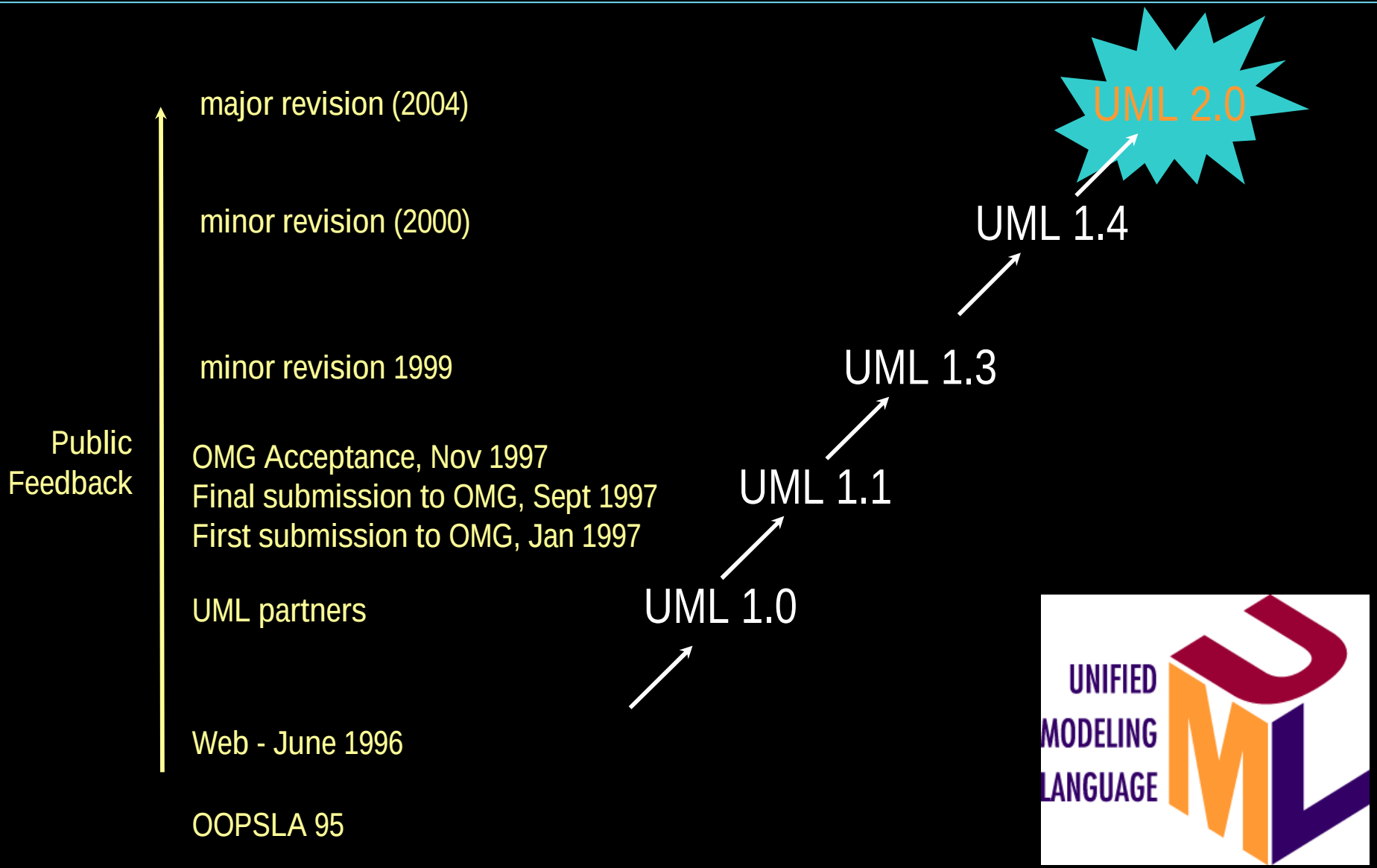
- Conceptual class
  - real-world concept or thing in UP domain model
- Software class
  - specification or implementation perspective of a software component

1.7. Visual Modeling is a Good Thing

Diagrams help to explore more of the big picture and relationships between analysis or software elements



1.8. History


major revision (2004)
minor revision (2000)
minor revision 1999
Public Feedback
OMG Acceptance, Nov 1997
Final submission to OMG, Sept 1997
First submission to OMG, Jan 1997
UML partners
Web - June 1996
OOPSLA 95
UML 1.0
UML 1.1
UML 1.3
UML 1.4
UML 2.0
UNIFIED MODELING LANGUAGE

Review

- Define object-oriented analysis and design (OOA/D).
- Illustrate a brief OOA/D example.
- Overview UML and visual agile modeling

Question & Answer

Chapter 2. Iterative, Evolutionary, and Agile

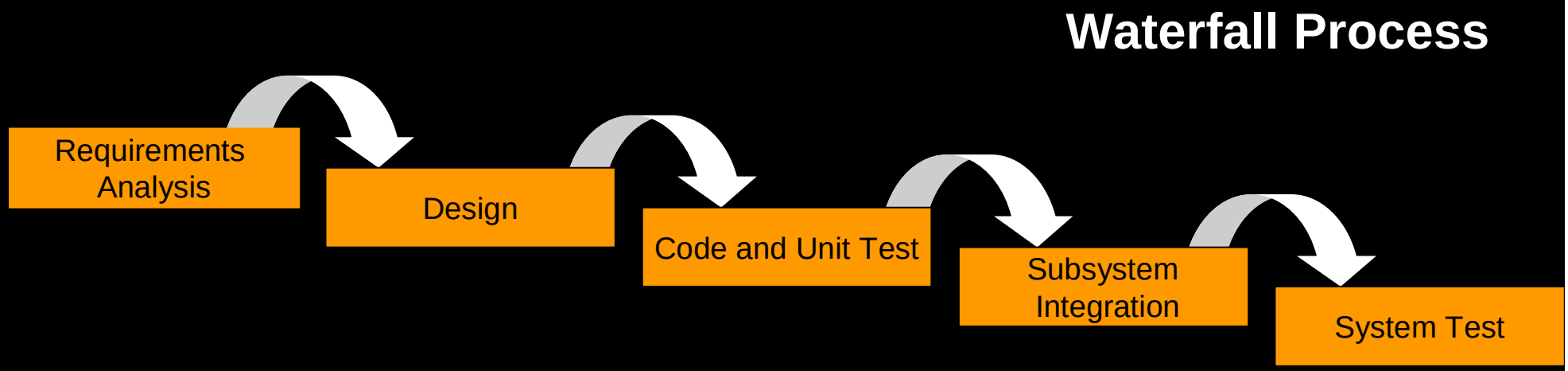
Objective

- Provide motivation for the content and order of the book.
- Define an iterative and agile method.
- Define fundamental concepts in the Unified Process.

2.1. What is the UP?

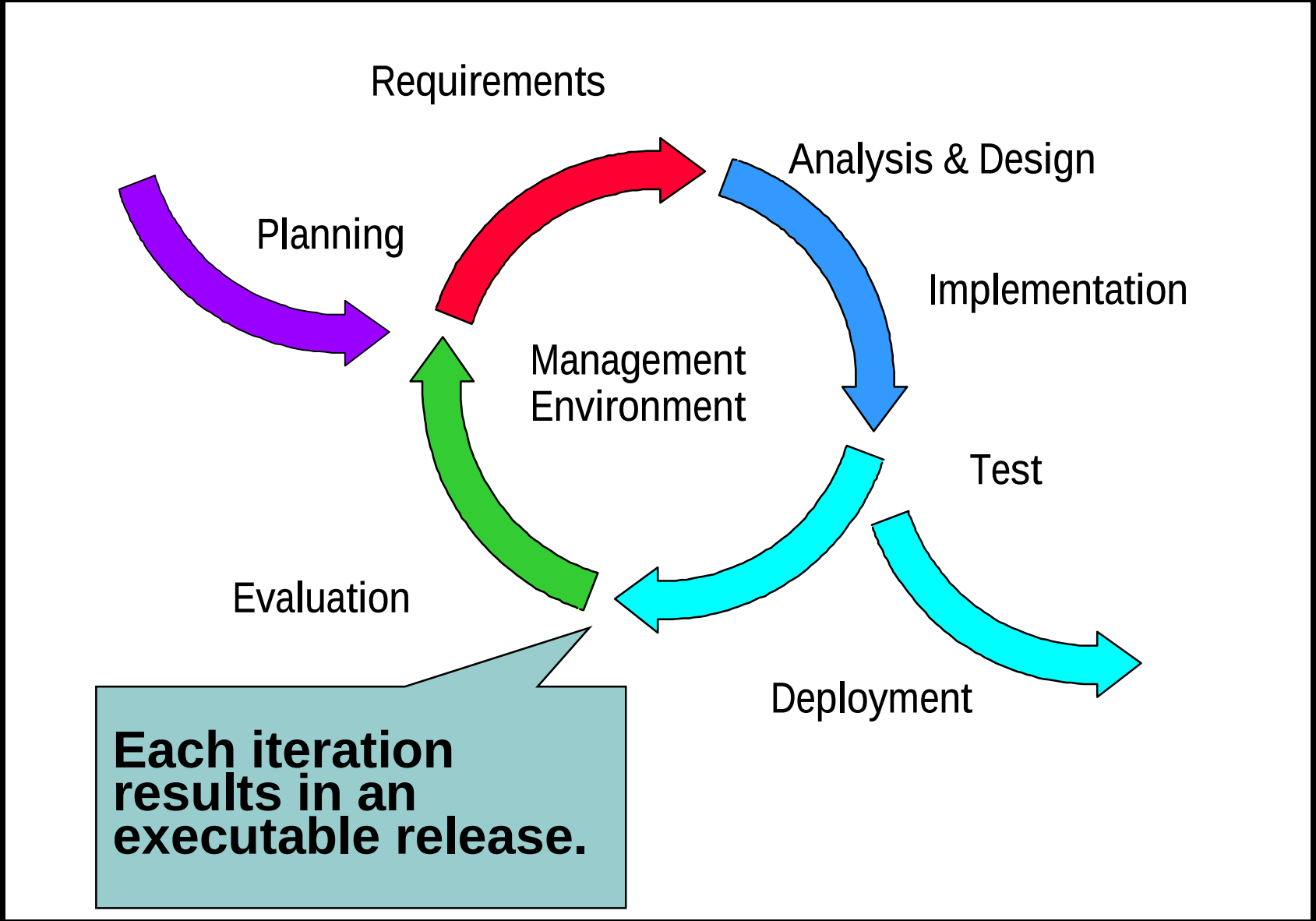
- Software Development Process
  - describes an approach to building, deploying, and possibly maintaining software

Waterfall Process



- Heavy documentation
- Frustrate designers and programmers
- Frequently results in major unplanned iterations
- Delays confirmation of critical risk resolution
- Delays and aggregates integration and testing

Iterative Development Produces Executables



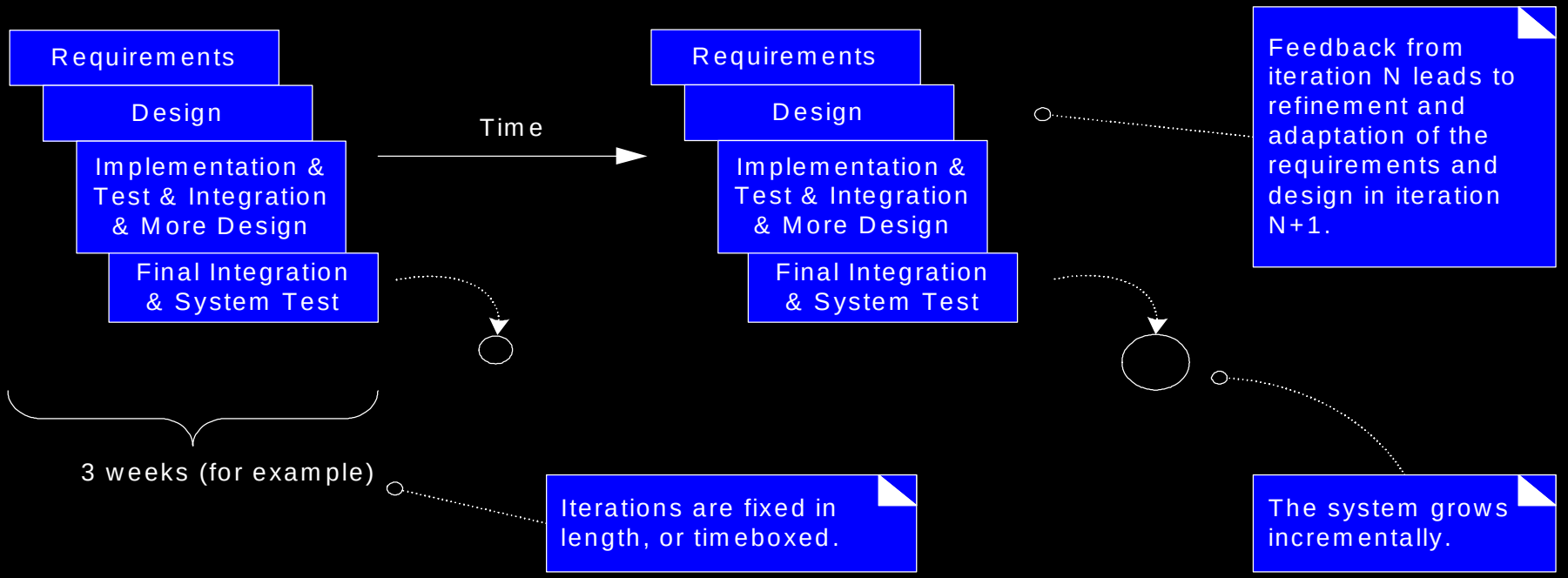
2.1. What is the UP?

- Unified Process (UP)
  - a popular iterative software development process for building OO systems
- UP encourages including practices from other iterative methods
  - Extreme Programming (XP)
  - Scrum
  - Test-driven Development
  - Refactoring
  - Continuous integration

2.2. What is Iterative and Evolutionary Development?

- Iterations
  - Development is organized into a series of short, fixed-length (3 weeks) mini-projects
- Evolution
  - Successive enlargement and refinement of the system through multiple iterations

2.2. What is Iterative and Evolutionary Development?



- Neither a rush to code, nor a long drawn-out design step
- An executable but incomplete system is got in each iteration

2.2. What is Iterative and Evolutionary Development?

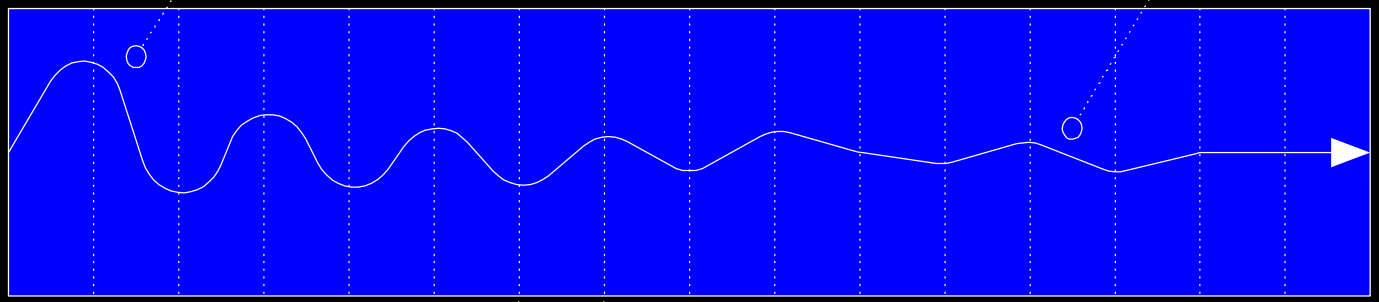
How to Handle Change on an Iterative Project?

- Embracing change
  - Balance the need to agree upon and stabilize a set of requirement with the reality of changing requirements
- Early feedback can help to clarify user requirements.
- Early testing help to prove if the partial design and implementation are on the right path.

2.2. What is Iterative and Evolutionary Development?

Early iterations are farther from the "true path" of the system. Via feedback and adaptation, the system converges towards the most appropriate requirements and design.

In late iterations, a significant change requirements is rare, but can occur. S late changes may give an organizatio competitive business advantage.

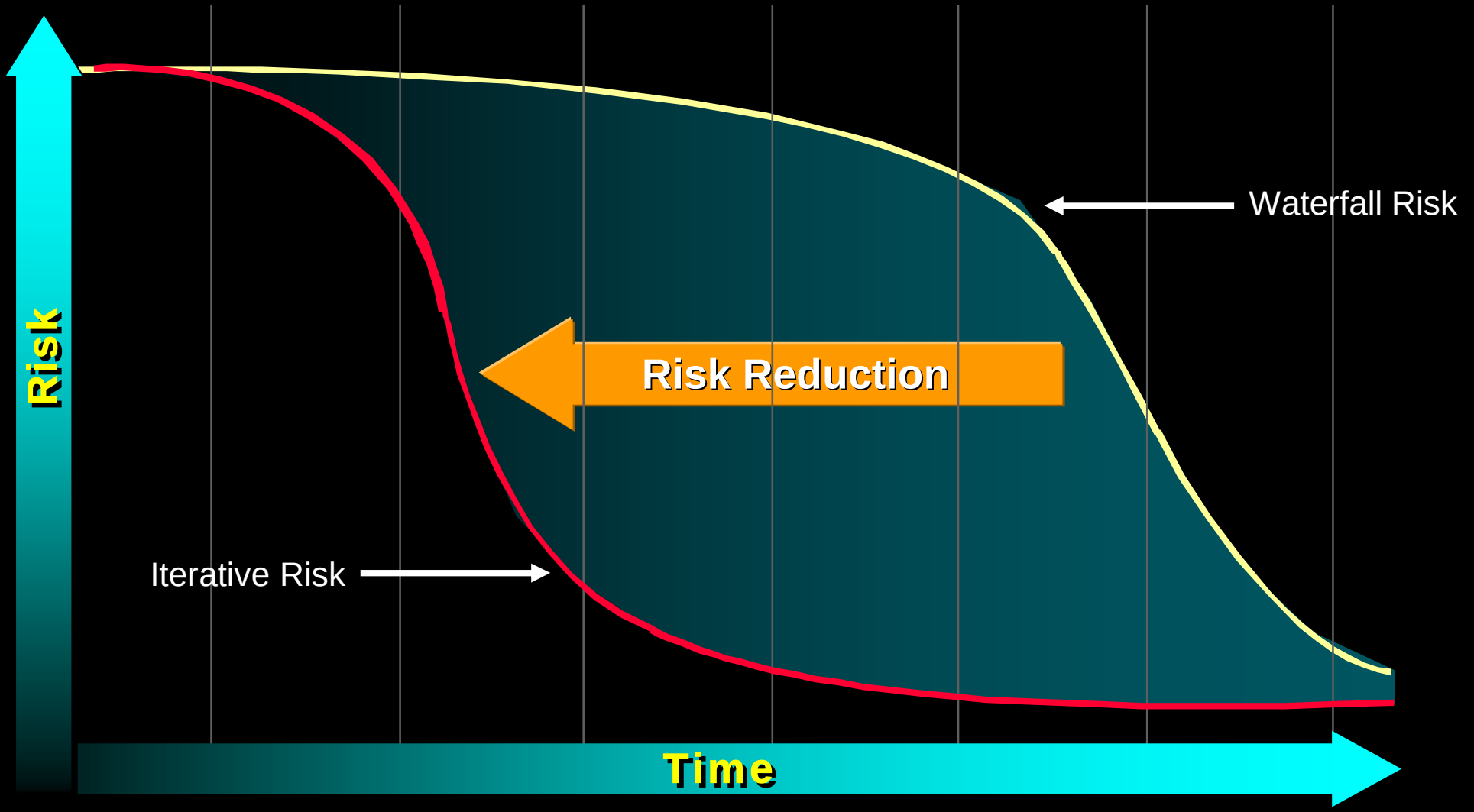


one iteration of design, implement, integrate, and test

2.2. What is Iterative and Evolutionary Development?

- Benefits to Iterative Development
  - Less project failure, better productivity, and lower defect rates
  - Early rather than late mitigation of high risks
  - Early visible progress
  - Early feedback, user engagement, and adaption
  - Managed complexity
  - The learning within an iteration can be methodically used to improve the development process

2.2. What is Iterative and Evolutionary Development?



2.2. What is Iterative and Evolutionary Development?

How long should an iteration be?

2-6 weeks

- Long iterations subvert the core motivation for iterative development and increase project risk
- Short iterations make it difficult to get meaningful throughput and feedback

2.3. What About the Waterfall Lifecycle

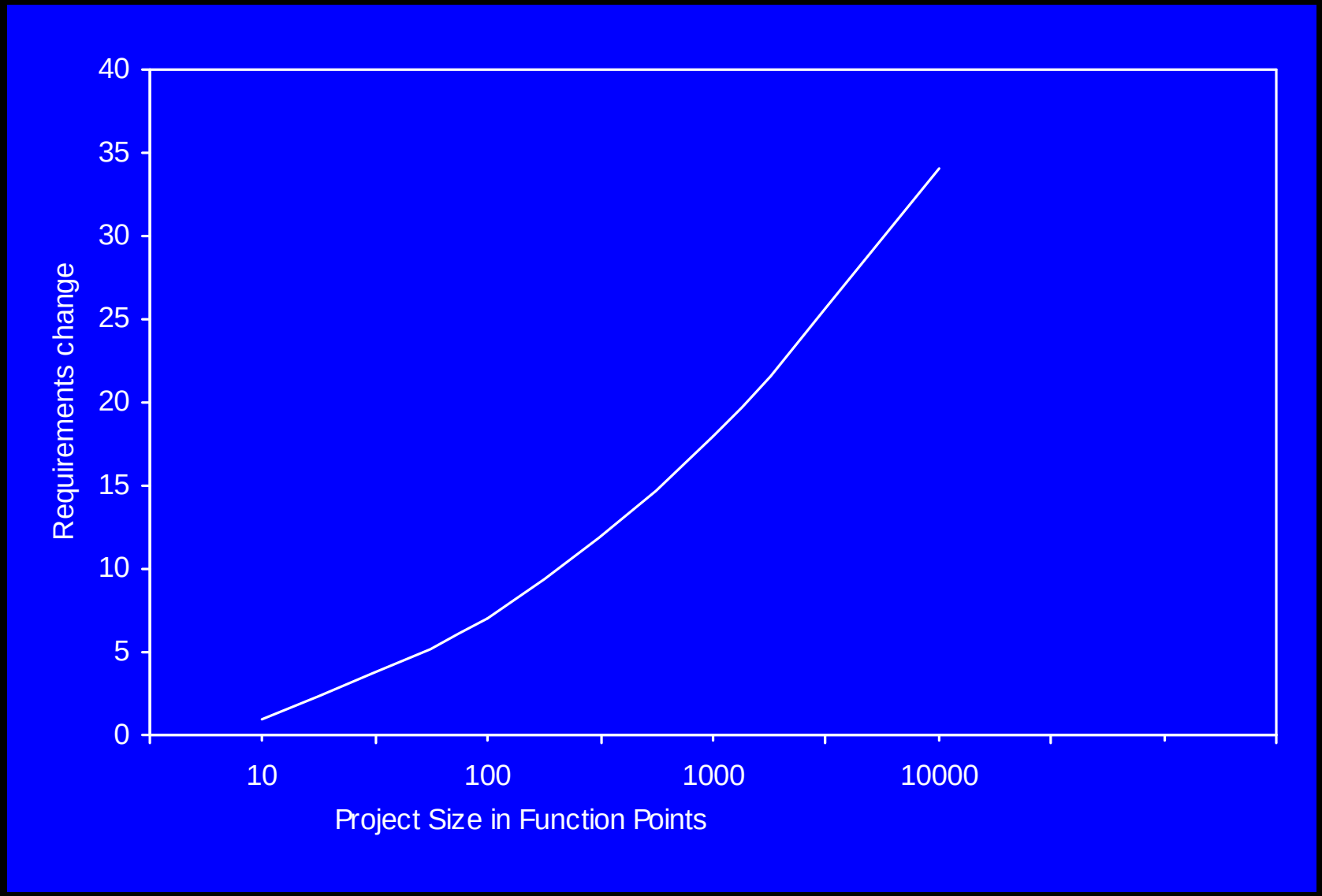
- Don’t Let Waterfall Thinking Invade an Iterative or UP Project
  - Let’s write all the use cases before starting to program
  - Let’s do many detailed OO models in UML before starting to program

2.3. What About the Waterfall Lifecycle

Why is the Waterfall so Failure-Prone

- False assumption of Waterfall process
  - Specifications are predictable and stable and can be correctly defined at the start, with low change rates.
- Software is not usually a domain of mass manufacturing. It is a domain of a new product development.
- Change is the constant on software projects.

2.3. What About the Waterfall Lifecycle



2.3. What About the Waterfall Lifecycle

- Feedback and adaptation are key ingredients for success.
  - Feedback from early development,
    - programmers trying to read specifications, and client demos to refine the requirements.
  - Feedback from tests and developers
    - to refine the design or models.
  - Feedback from the progress of the team tackling early features
    - to refine the schedule and estimates.
  - Feedback from the client and marketplace
    - to re-prioritize the features to tackle in the next iteration.

2.4. How to do Iterative and Evolutionary A/D

Assume 20 Iterations

1. 2 days (before iteration 1)
   - One morning
     - Do high level requirement analysis
     - Pick 10% use cases(UCs), which are architecturally significant and of high business value/risk.
   - 1.5 days
     - Do intensive detailed analysis of the 10% UCs

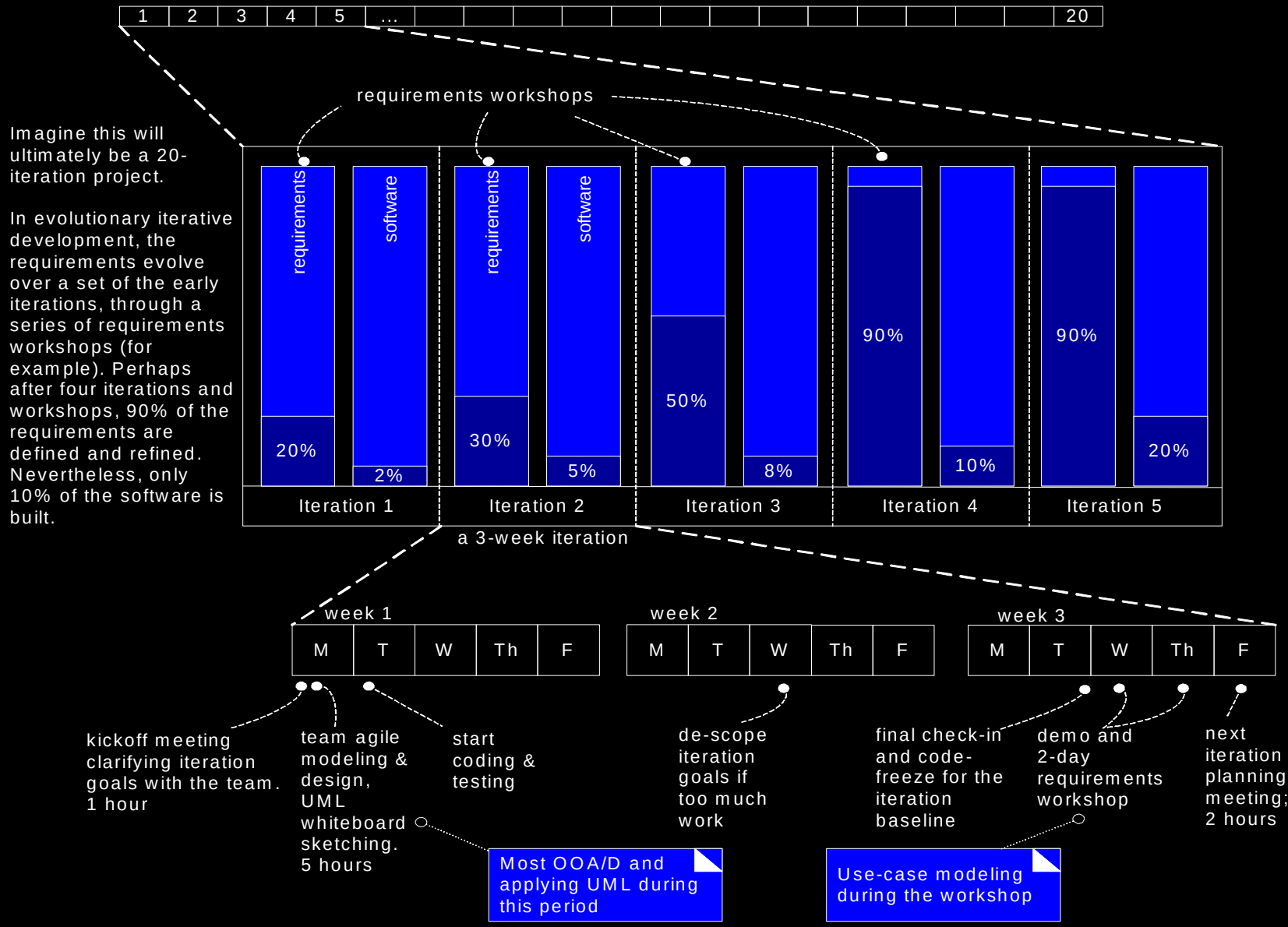
2.4. How to do Iterative and Evolutionary A/D

2. Hold an iteration planning meeting
   - A subset of 10% UCs are chosen to design, build and test.
3. Do iteration 1 (3 – 4 weeks)

   .....

- After a few iterations of early exploratory development, team can more reliably answer "what, how much, when."

2.4. How to do Iterative and Evolutionary A/D



2.5. What is Risk-Driven and Client-Driven Iterative Planning

- UP combine the risk-driven and client-driven iterative planning.
  - Identify and drive down the highest risks
  - Build visible features that client cares most about

2.6. What are Agile Methods and Attitudes

- Short timeboxed iteration with evolutionary refinement of plans, requirements and design.
- Employ adaptive planning, promote incremental delivery, and include other values and practices that encourage agility rapid and flexible response to change.
  - Common Project Workroom
  - Self-organizing Teams
  - Programming in Pairs
  - Test-driven Development

2.7. What is Agile Modeling

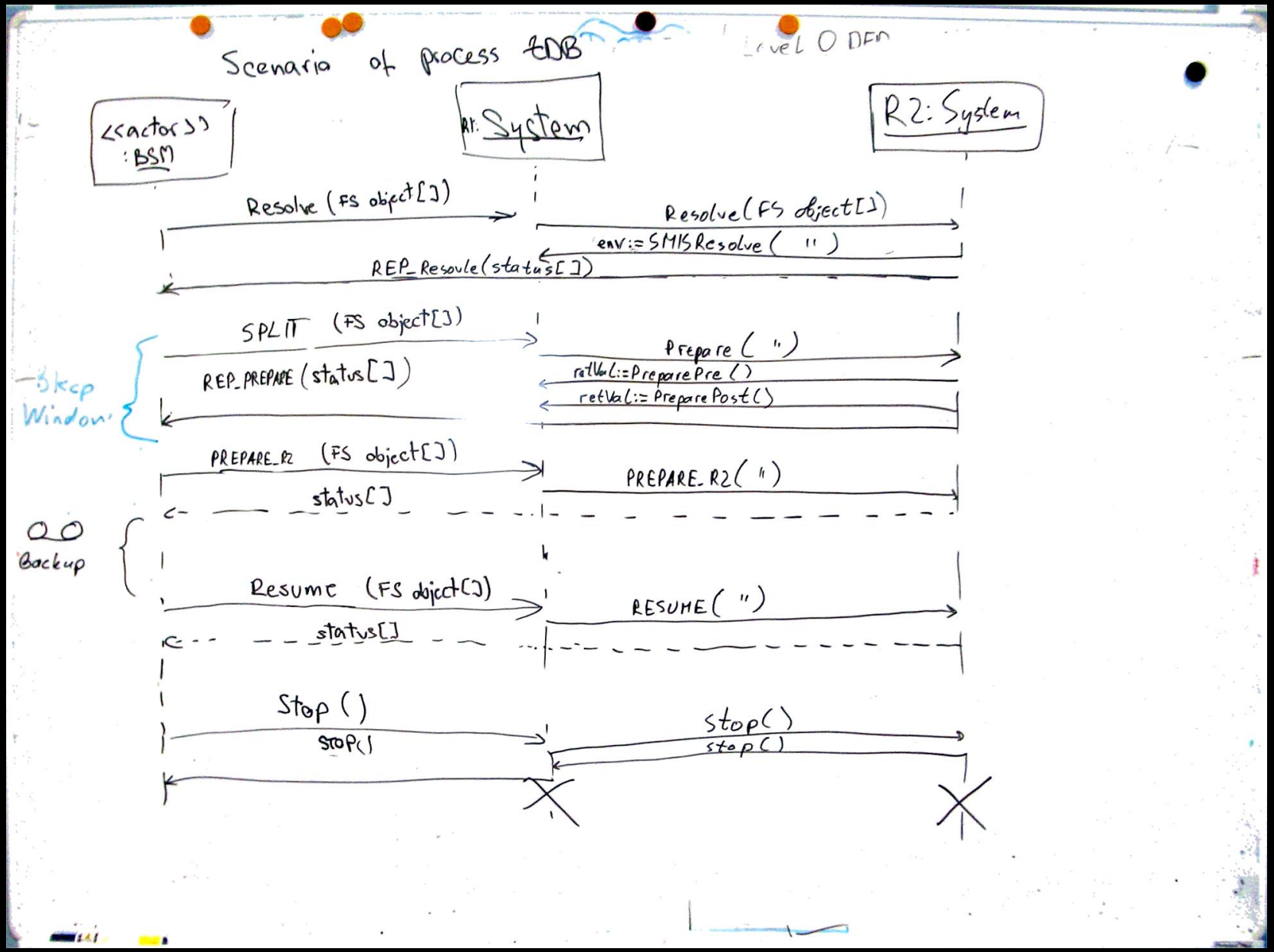
The purpose of modeling is primarily to understand, not to document

- Doing UML is
  - not for a designer to create many UML diagrams that are handed off to a programmer
  - but rather to quickly explore alternative and the path to a good OO design

2.7. What is Agile Modeling

- Practices and Values of Agile Modeling
  - Adopting an agile method does not mean avoiding any modeling
  - The purpose of modeling and models is to support understanding and communication, not documentation
  - Don’t model or apply the UML to all software design
  - Use the simplest tool possible.
  - Don’t model alone.
  - Create models in parallel.
  - Use “ good enough ” simple notation.
  - Know that all models will be inaccurate.
  - Developers should do the OO design modeling.

2.7. What is Agile Modeling



2.8. What is an Agile UP?

- Prefer a small set of UP activities and artifacts
  - All UP artifacts are optional
  - Avoid creating them unless they add values
  - Focus on early programming, not early documenting
- Requirement and designs are not completed before implementation
- Apply UML with agile modeling practices

2.8. What is an Agile UP?

- No detailed plan for the entire project
  - Phase Plan – estimates the project major milestone
  - Iteration Plan – plans with greater detail one iteration in advance.

2.9. Are There Other Critical UP Practices

- Central idea in UP
  - Short timeboxed iteration
  - Evolutionary
  - Adaptive development
- Other practices
  - Tackle high-risk and high-value issue early
  - Continuously engage users for evaluation, feedback and requirement
  - Build a cohesive, core architecture early
  - Continuously verify quality

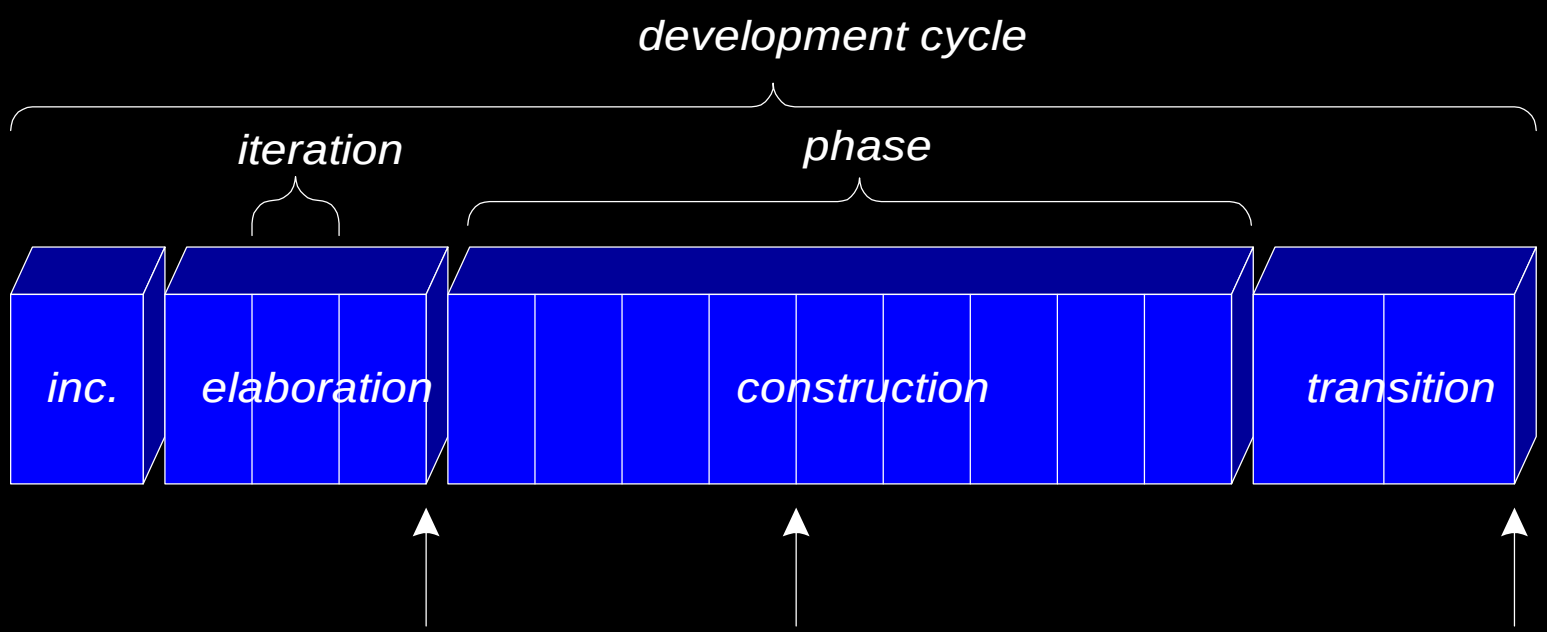
2.10. What are the UP Phases

- Inception
  - approximate vision
  - business case
  - scope
  - vague estimates.
- Elaboration
  - refined vision
  - iterative implementation of the core architecture
  - resolution of high risks
  - identification of most requirements and scope
  - more realistic estimates.

2.10. What are the UP Phases

- Construction
  - iterative implementation of the remaining lower risk and easier elements
  - preparation for deployment.
- Transition
  - beta tests
  - deployment

2.10. What are the UP Phases



milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is release for production use

2.11. What are the UP Disciplines?

- Discipline
  - a set of activities (and related artifacts) in one subject area

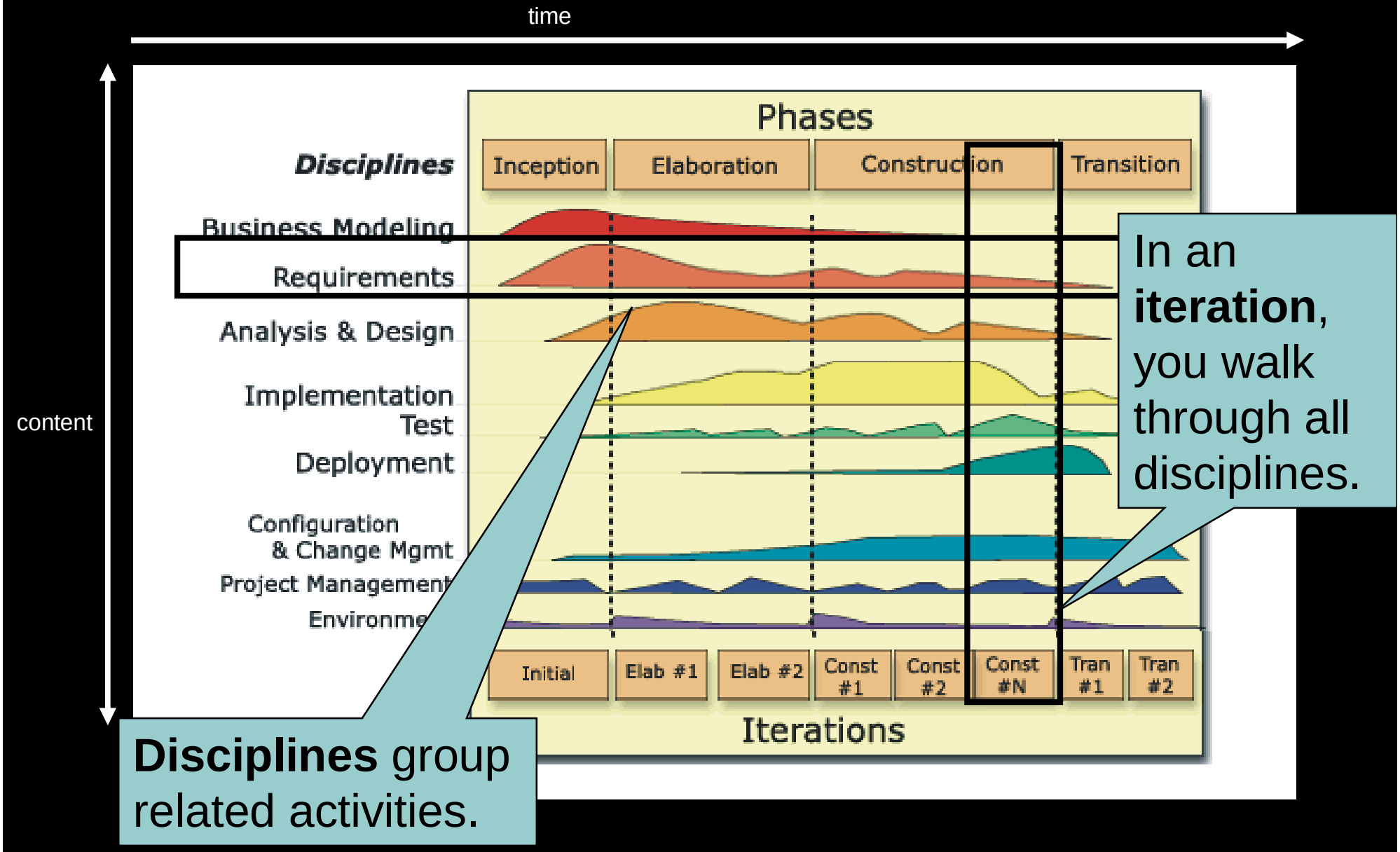
Disciplines in UP

- Business Modeling
  - The Domain Model artifact
    - visualize noteworthy concepts in the application domain.

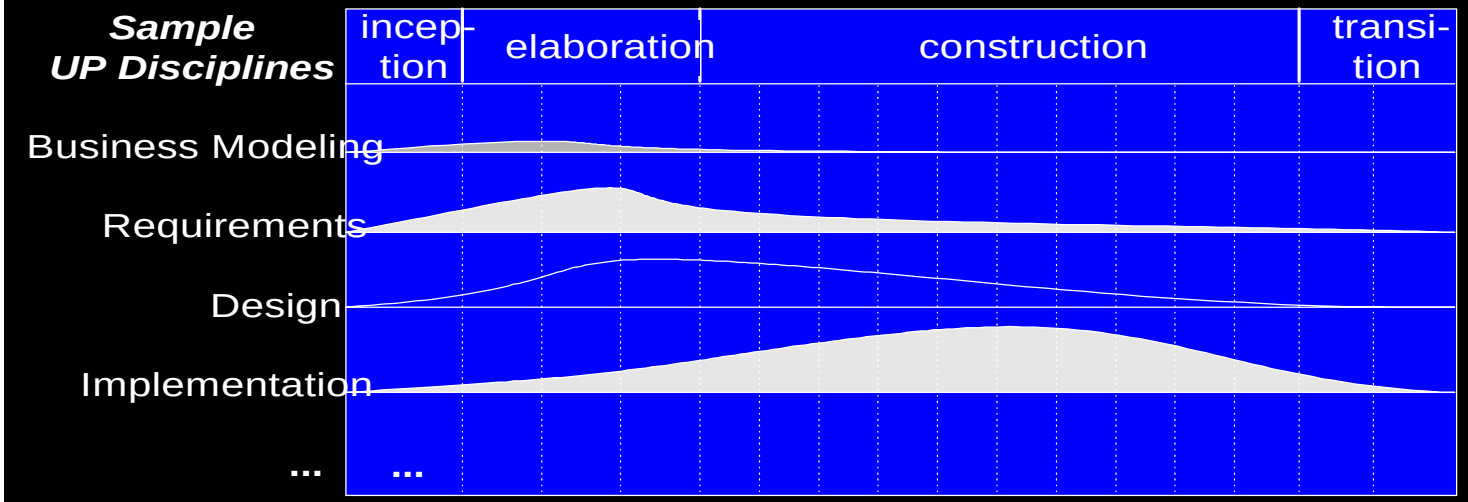
2.11. What are the UP Disciplines?

- Requirements
  - The Use-Case Model and Supplementary Specification artifacts
    - capture functional and non-functional requirements.
- Design
  - The Design Model artifact
    - design the software objects.

2.11. What are the UP Disciplines?



2.11. What are the UP Disciplines?



The relative effort in disciplines shifts across the phases.

This example is suggestive, not litera

- One iteration work goes on in most or all disciplines.
- Relative effort across these disciplines changes over time.

2.11. What are the UP Disciplines?

How is the Book Structure Influenced by UP Phases and Disciplines?

- The inception phase chapters
  - introduce the basics of requirements analysis
- Iteration 1
  - introduces fundamental OOA/D and assignment of responsibilities to objects.
- Iteration 2
  - focuses on object design, especially on introducing some high-use "design patterns."
- Iteration 3
  - introduces a variety of subjects, such as architectural analysis and framework design.

2.12. How to Customize the Process?

- Everything is optional
  - Set of possible artifacts can be viewed like a set of medicine in a pharmacy.
- Development Case
  - Choise of practices and artifacts

2.12. How to Customize the Process?

Discipline	Practice	Artifact	Incep.	Elab.	Const.	Trans.
		Iteration	I1	E1..En	C1..Cn	T1..T2
Business Modeling	agile modeling req. workshop	Domain Model		s		
Requirements	req. workshop vision box exercise dot voting	Use-Case Model	s	r		
		Vision	s	r		
		Supplementary Specification	s	r		
		Glossary	s	r		
Design	agile modeling test-driven dev.	Design Model		s	r	
		SW Architecture Document		s		
		Data Model		s	r	
Implementation	test-driven dev. pair programming continuous integration coding standards	…				
Project Management	agile PM daily Scrum meeting	…				
…						

2.13. You Didn't Understand Iterative Development

- Try to define most of the requirements before starting design or implementation.
- Spend days or weeks in UML modeling before programming
- Think that
  - inception = requirements
  - elaboration = design
  - construction = implementation
- Think that the purpose of elaboration is to fully and carefully define models

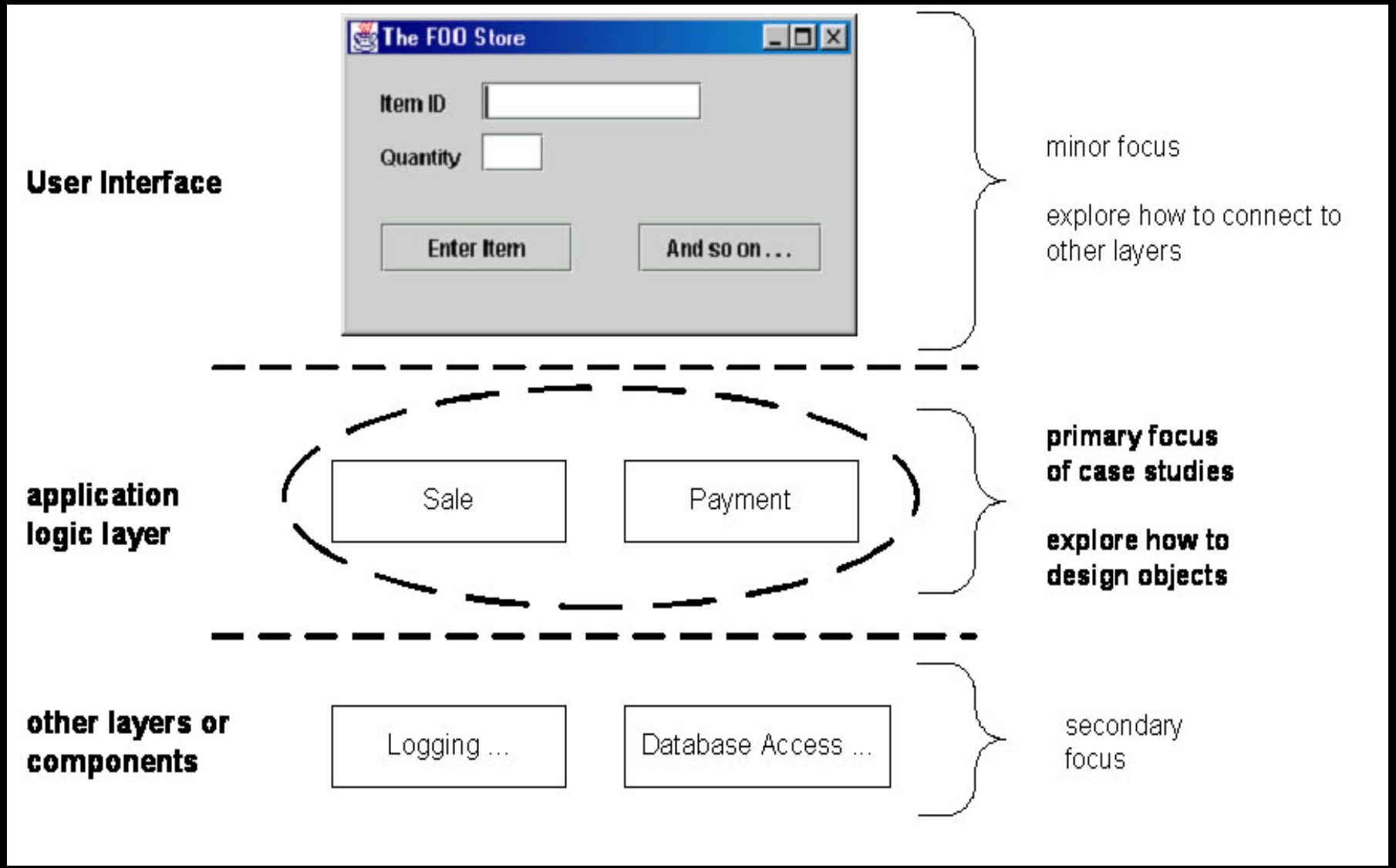
2.13. You Didn't Understand Iterative Development

- Believe that a suitable iteration length is three months long
- Think that adopting the UP means
  - do many of the possible activities
  - create many documents
  - with many steps to be followed
- Try to plan a project in detail from start to finish
  - try to speculatively predict all the iterations

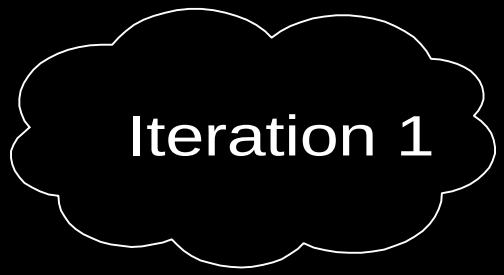
Question & Answer

Chapter 3. Case Studies

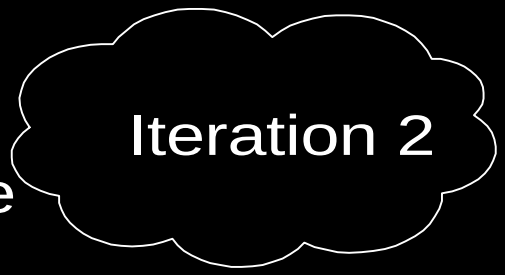
3.1. What is and isn't Covered in the Case Studies?



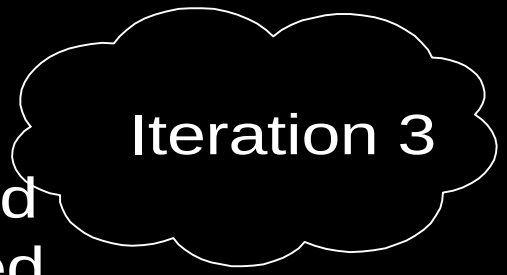
3.2. Case Study Strategy



Introduces just those analysis and design skills related to iteration one.



Additional analysis and design skills introduced.



Likewise.

3.3. Case One: The NextGen POS System

- POS system is a computerized application to record sales and handle payments,

It includes

- computer
- bar code scanner
- software to run the system.

It interfaces

- third-party tax calculator
- inventory control.

3.3. Case One: The NextGen POS System

- POS systems must
  - be fault-tolerant
  - support multiple and varied client-side terminals and interfaces
  - support a unique set of logic to execute at certain predictable points in scenarios

3.4. Case Two: The Monopoly Game System

- Software versions of Monopoly
  - support rich client and Web UIs
  - run as a simulation