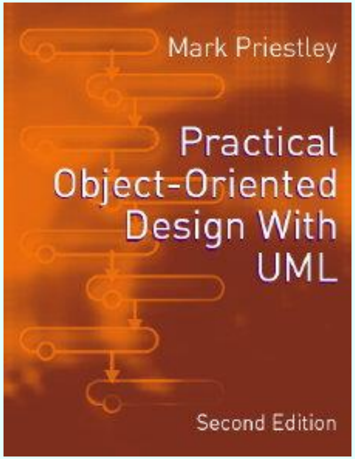
PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



Mark Priestley

Practical Object-Oriented Design With UML

Second Edition

Chapter 3: **Software Development Processes**

01.25

Software Development Processes

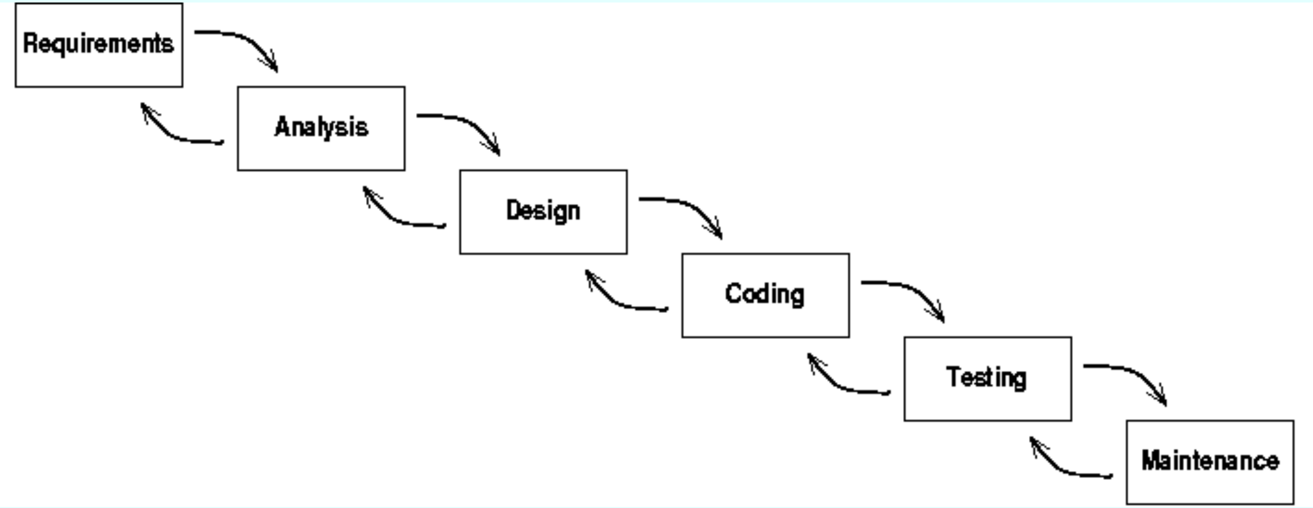
- A methodology includes:
  - a notation
  - a process
- UML has emerged as a standard *notation* for OOD
- There is much less agreement about *process*

02.28. - 04.03

Waterfall Model

พัฒนา SW เป็นขั้นเป็นตอน. ถ้ายังไม่เสร็จห้ามทำอย่างอื่นก่อน.

- Development modelled as a series of stages



Waterfall Lifecycle

- Each stage represents a different activity
- Two ‘waterfall’ assumptions
  - the stages take place in sequence
  - each activity is completed before the next stage starts
- Most versions allow some degree of feedback between stages

Risk and the Waterfall

- Software development is unpredictable
- Only testing can validate success of system
- The waterfall model postpones testing until the end of the lifecycle
- This creates *high risk:*
  - we don't find out that the system is failing until the bulk of the development has been carried out

Requirements and the waterfall

- Requirements always change
  - complexity of many software systems
  - systems fit into complex environments
  - requirement change following installation
  - flexibility of software makes change seem easy
- Waterfall model fixes requirements at start
- Inflexible: likely to fail if requirements change

04.20.

พัฒนา SW ไปพร้อมๆกัน เช่น เขียนcode พร้อม Design.

Evolutionary Process Models

ลักษณะ- ของการแก้แล้วแก้ อีกจนเสร็จ.

ข้อเสีย - ไม่มีแหล่งอ้างอิง
- SW จะเละ.

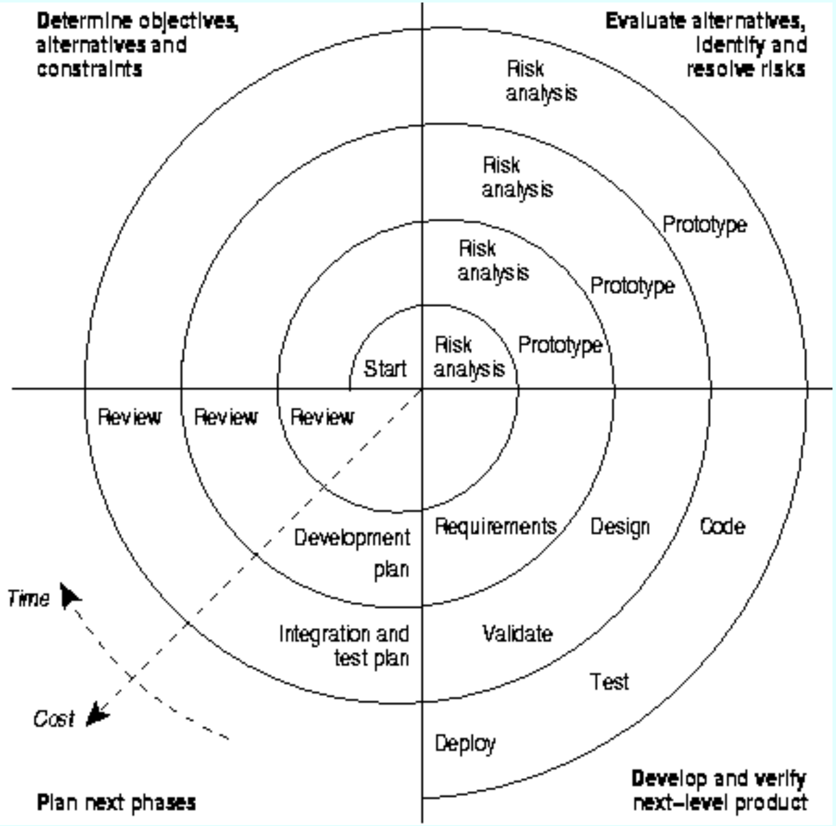
- A response to the inflexibility of the waterfall model
- Extensive use of executable prototypes
- System evolves toward final form
  - changes inspired by user feedback on prototypes
- Weak for management purposes
  - no development plan or project milestones

↓ 07.00

Spiral Model

- ปัทธระรางแบบจำลองมาๆเทรต่อน.

- An attempt to formalize the benefits of evolutionary styles without the management shortcomings



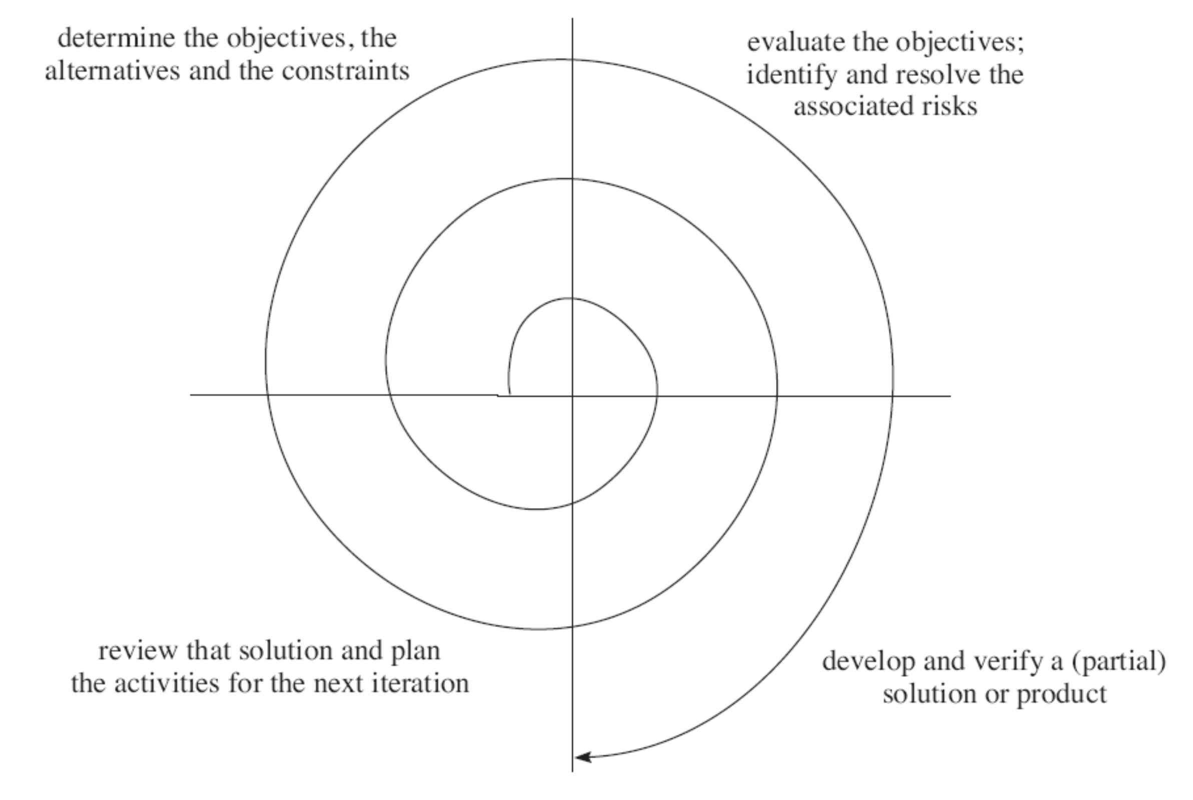
Characteristics of spiral model

- Project *iterates* through development activities (repeated turns of the spiral)
- 4 quadrants represent major activities
- Each iteration addresses the highest risk for the project
- Model does not prescribe a fixed process
  - detailed sequence of activities can differ depending on nature of project

Iterative and Incremental

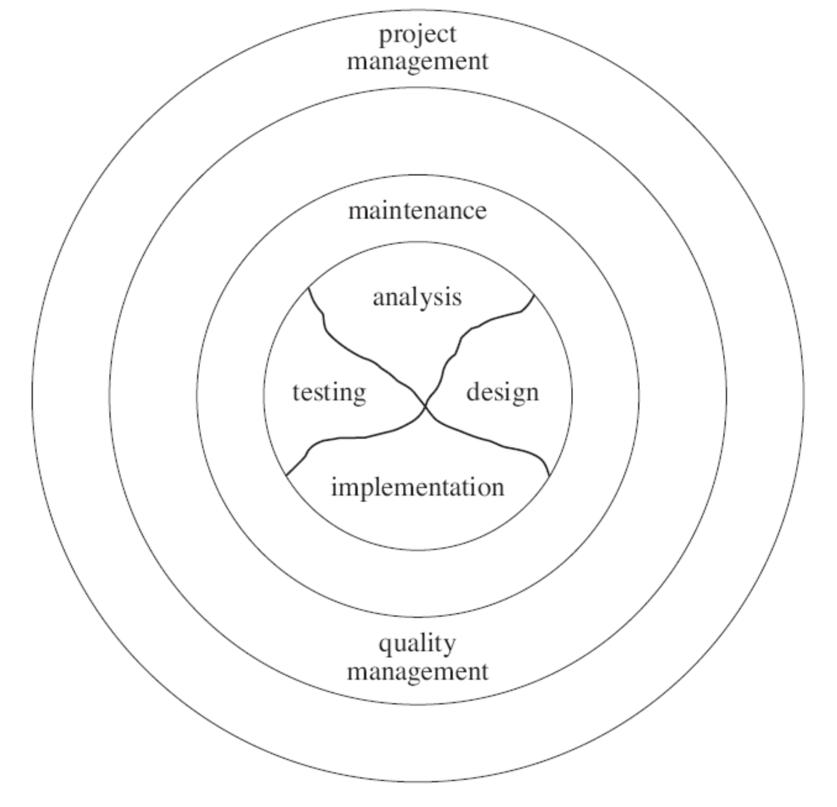
- Responses to waterfall model identify two characteristics of better process models:
  - **Incremental**: don't plan to carry out all project in one go, but spread over a number of increments
  - **Iterative**: various development activities get carried out repeatedly during development.
- Current process models are all iterative and incremental

Four Quadrants


determine the objectives, the alternatives and the constraints
evaluate the objectives; identify and resolve the associated risks
review that solution and plan the activities for the next iteration
develop and verify a (partial) solution or product

A spiral process to deal with risk

Seven Technical Activities



Relating Quadrants to Activities

- The analysis, design, implementation and testing activities all fit into the 'develop and verify a (partial) solution or product' segment.
- The analysis activity also overlaps the evaluation quadrant where requirements and risks are analyzed.
- The testing activity overlaps the 'review and plan' quadrant.
- The maintenance, project management and quality management activities (including configuration management) operate in all four quadrants.

09.40

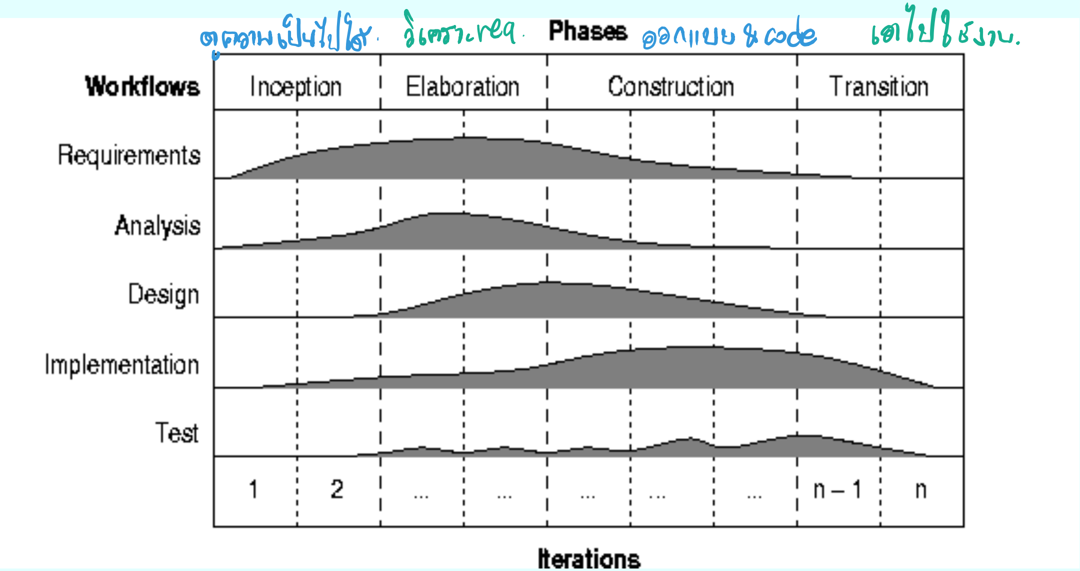
Unified Process (UP)

- Current state-of-the-art methodology
- Initially developed by designers of UML
- Structures project as a number of *phases*
- Each phase contains several *iterations* → มีกระบวนการ ที่เป็นชิ้น เป็นตอนแยกกันว่า.
- Different *workflows* (activities) are performed in each iteration (notice the traditional set of activities)
- The Unified Process takes risk analysis as a central concept. Uses iteration as a means of controlling risk.

10.35 - ๒๒.๒๐. กระบวนการพัฒนา SW. ใช้ร่วมกับ UML.

Unified Process Outline

ดูความเป็นไปได้. วิเคราะห์ req. ออกแบบ & code เอาไปใช้งาน.



- The balance between workflows is different in different phases

UML and the Unified Process

- UML is a language
- It defines diagrams which represent aspects of software systems
- UML's diagrams can be used in conjunction with many different processes (or even in the absence of a formal process)
- Because of their history, there is a close fit between UML and the UP

Use cases and the Unified Process

- UP is use case driven, meaning:
- Systematic use is made of use cases various stages in the design process, realization of use case.
- Tests can be systematically derived from use cases to provide acceptance tests for the system.

Use cases and the Unified Process

Use cases are a good way to structure requirements, but there are issues not addressed.

- Classes that do not interact with the system but are still important.
- Component based development, reuse, architectures also need to be considered.

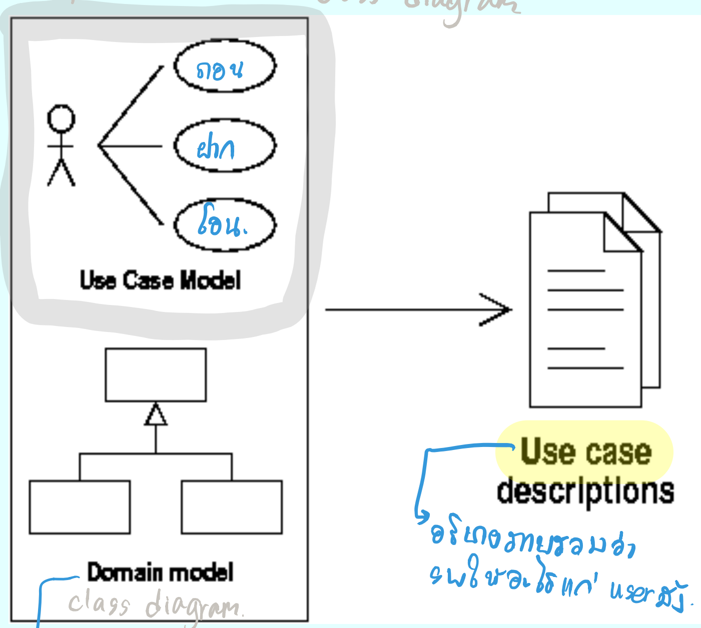
- When is UCDD the idea is to keep focus on use cases throughout a development, not just in requirements capture. Keeps attention on the requirements.
- No single viewpoint (RDD or UCD) suited to all project.

23.05

Requirements and Analysis

- UP starts with *use cases* describing how users interact with the system
- A *domain model* records facts about real world entities
- UML *use case* and *class diagrams* document these

→ เกิดการกระโดดไป class diagram



class diagram.

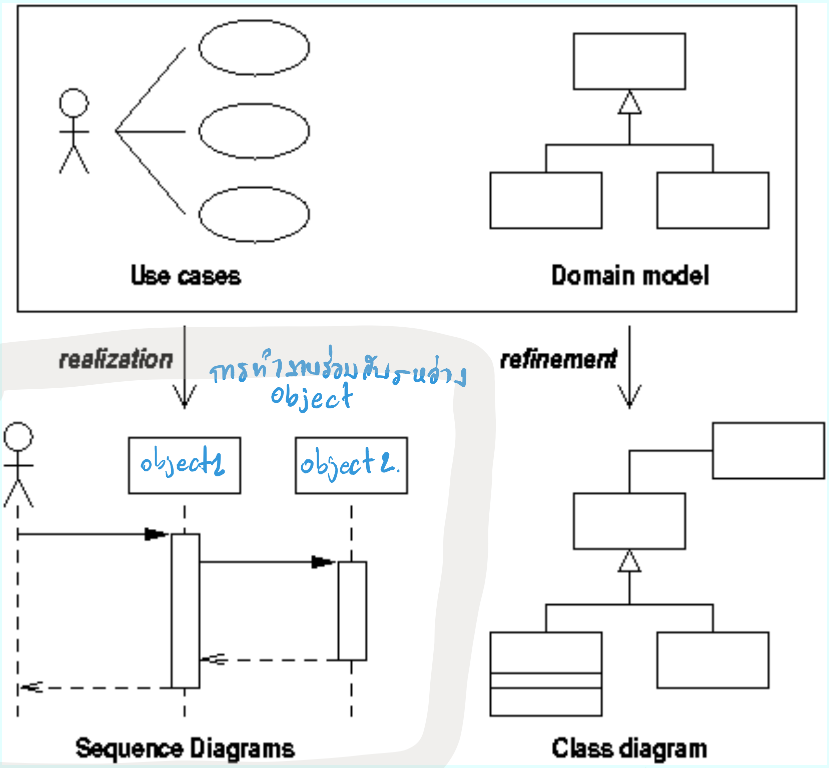
→ เล่าเรื่องเกี่ยวกับที่เราวิเคราะห์อยู่
บ.

→ อธิบายภาพรวมว่า
จะเกิดอะไรแก่ user บ้าง.

↓ 27.40 - 33.46

Realization and Refinement

วิเคราะห์



ออกแบบ.

วิเคราะห์

↳ เอาไปทำ Class diagram.

33.46. - 36.15

Realization and refinement

- Use case *realizations* indicate how the functionality will be supported by the system
- Realizations are documented in UML *interaction diagrams*
- This causes the domain model to be *refined* into a more implementation-oriented class diagram

31.45

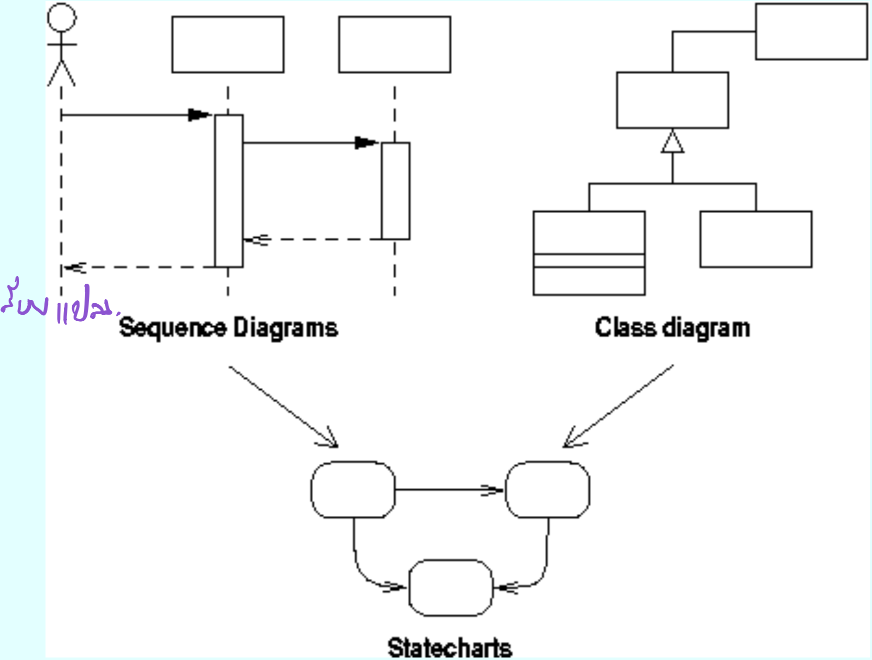
Specifying behaviour

↳ พฤติกรรมการทำงานของ object.

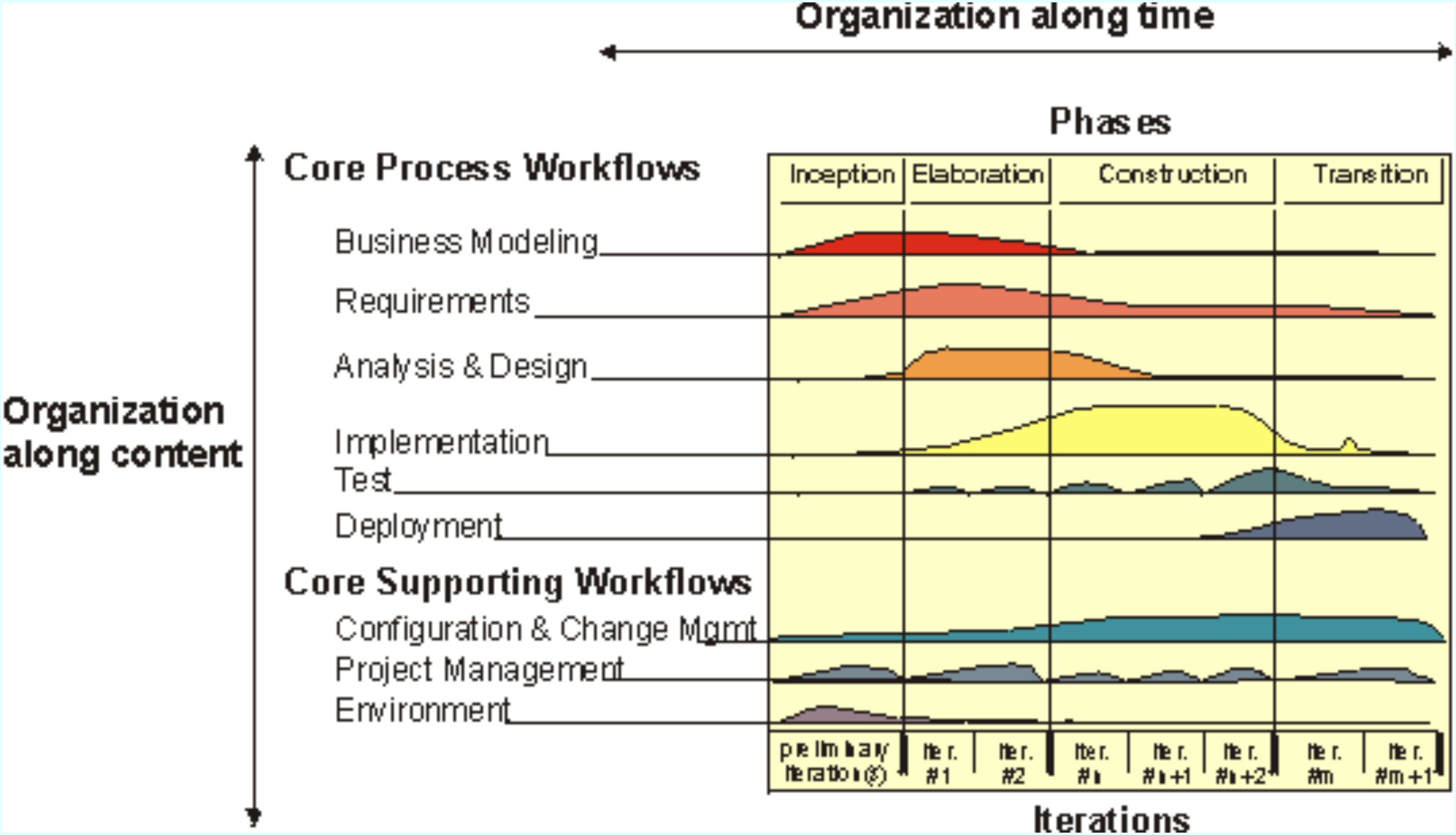
- *Statecharts* specify the behaviour of classes

↳ state. → เมื่อเกิดการเปลี่ยนแปลง.

object → object



Rational Unified Process



Unified Process Phase 1

- **Inception--**The good idea: specifying the end-product vision and its business case, defining the scope of the project. This phase has several goals:
  - To describe the initial requirements
  - To develop and justify the business case for the system
  - To determine the scope of the system
  - To identify the people, organizations, and external systems that will interact with the system
  - To develop an initial risk assessment, schedule, and estimate for the system
  - To develop an initial tailoring of the Unified Process to meet the exact needs of a particular organization or system.

Unified Process Phase 2

- **Elaboration--**Planning the necessary activities and required resources; specifying the features and designing the architecture. Identify significant risks. This phase has several goals:
  - To produce a proven, architectural baseline for the system
  - To evolve the requirements model to the "80% completion point"
  - To develop a coarse-grained project plan for the entire Construction phase
  - To ensure that the critical tools, processes, standards, and guidelines have been put in place for the Construction phase
  - To understand and eliminate the high-priority risks of the project needs

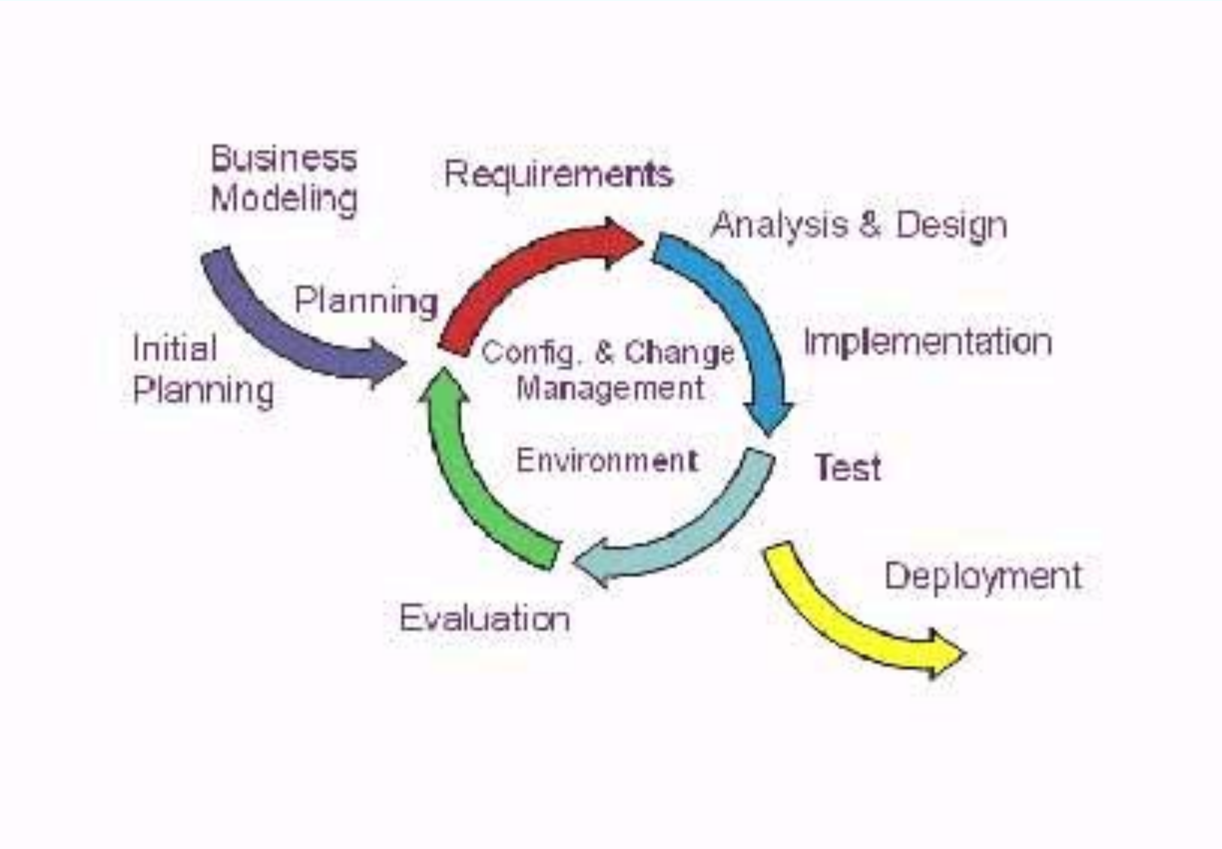
Unified Process Phase 3

- **Construction--**Building the product and evolving the vision, the architecture, and the plans until the product--the completed vision--is ready for transfer to its users' community. Main iterative spiral is placed here. This phase has several goals:
  - To describe the remaining requirements
  - To flesh out the design of the system
  - To ensure that the system meets the needs of its users and fits into the organization's overall system portfolio
  - To complete component development and testing, including both the software product and its documentation
  - To minimize development costs by optimizing resources
  - To achieve adequate quality as rapidly as possible
  - To develop useful versions of the system

Unified Process Phase 4

- **Transition--**Making the transition from the product to its user's community, which includes: manufacturing, delivering, training, supporting, maintaining the product until the users are satisfied. Final product baseline (also known as a production baseline) of the system. This phase has several goals:
  - Training materials for the system
  - Documentation, including user manuals, support documentation, and operations documentation.
  - The phase is concluded with the Produce Release milestone . To pass this milestone you must show that the users are satisfied with the system and that show that the actual expenditures versus the planned expenditures are still acceptable.

The iterative aspects of the workflows in UP.



↓ 38.55 -40.50

Extreme Programming (XP)

Tool หรือ ภาษาที่ พัฒนา รพได้เร็ว. อัปเดตแก้ไขได้เร็ว.

- XP is the modern inheritor of the evolutionary style of process model
- Key practices:
  - code review
  - automated testing
  - design refactoring (restructuring)
  - continuous integration

Design Flexibility

- XP stresses need to handle change
- Recommends *refactoring* as development progresses
- Assumes existence of technology that makes this feasible and cheap
- *Round-trip engineering*: use of reverse engineering and code generation tools to keep design and code in step

Localization

- Localization is the process of placing informational items (data and functions) in close physical proximity to each other.
- In a modeling context localization requires a mechanism for precisely defining the boundaries of the "area" into which the items are being gathered (e.g. classes, tables or functions).
- There needs to be a rationale as to why we group things together. An example of the general idea of localization is concept of cohesion within an module or class.

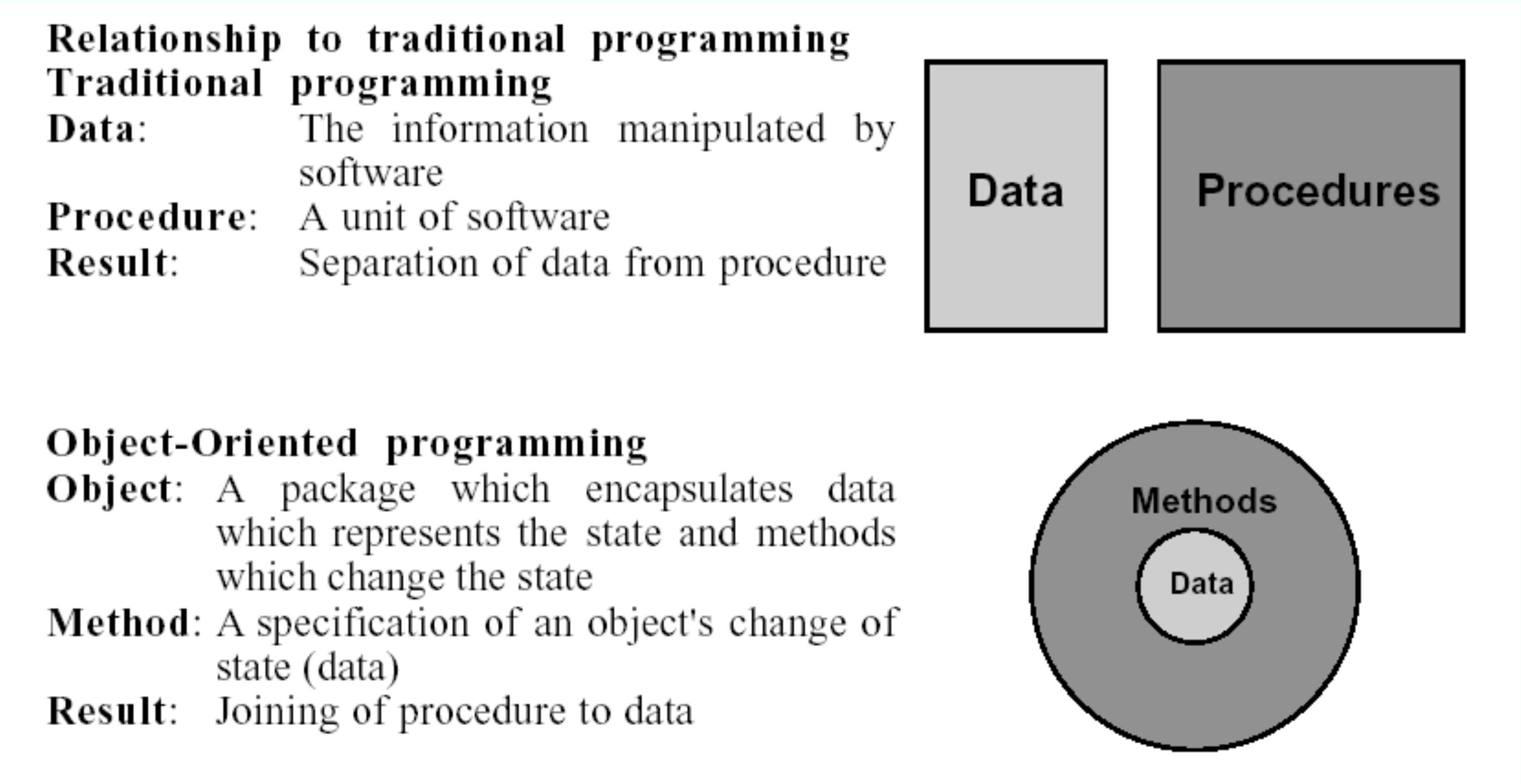
Localization

- Different development approaches require different localization schemes:
  - Functional decomposition approaches localize information around functions (or operations).
  - Data-driven approaches localize information around data.
  - Object-oriented approaches localize information around objects (or classes).

Localization

- The responsibility driven (RDD) approach, described earlier in the course, does not use functional view of the system as a basis for the creation of an object-oriented architecture for that system.
- Objects and functions do not map to each other on a one-to-one basis, and the architecture of an object-oriented system is significantly different from the architecture of a functionally decomposed system.
- Localization schemes have an effect on software architecture.

Localization



Relationship to traditional programming

Traditional programming

Data: The information manipulated by software

Procedure: A unit of software

Result: Separation of data from procedure

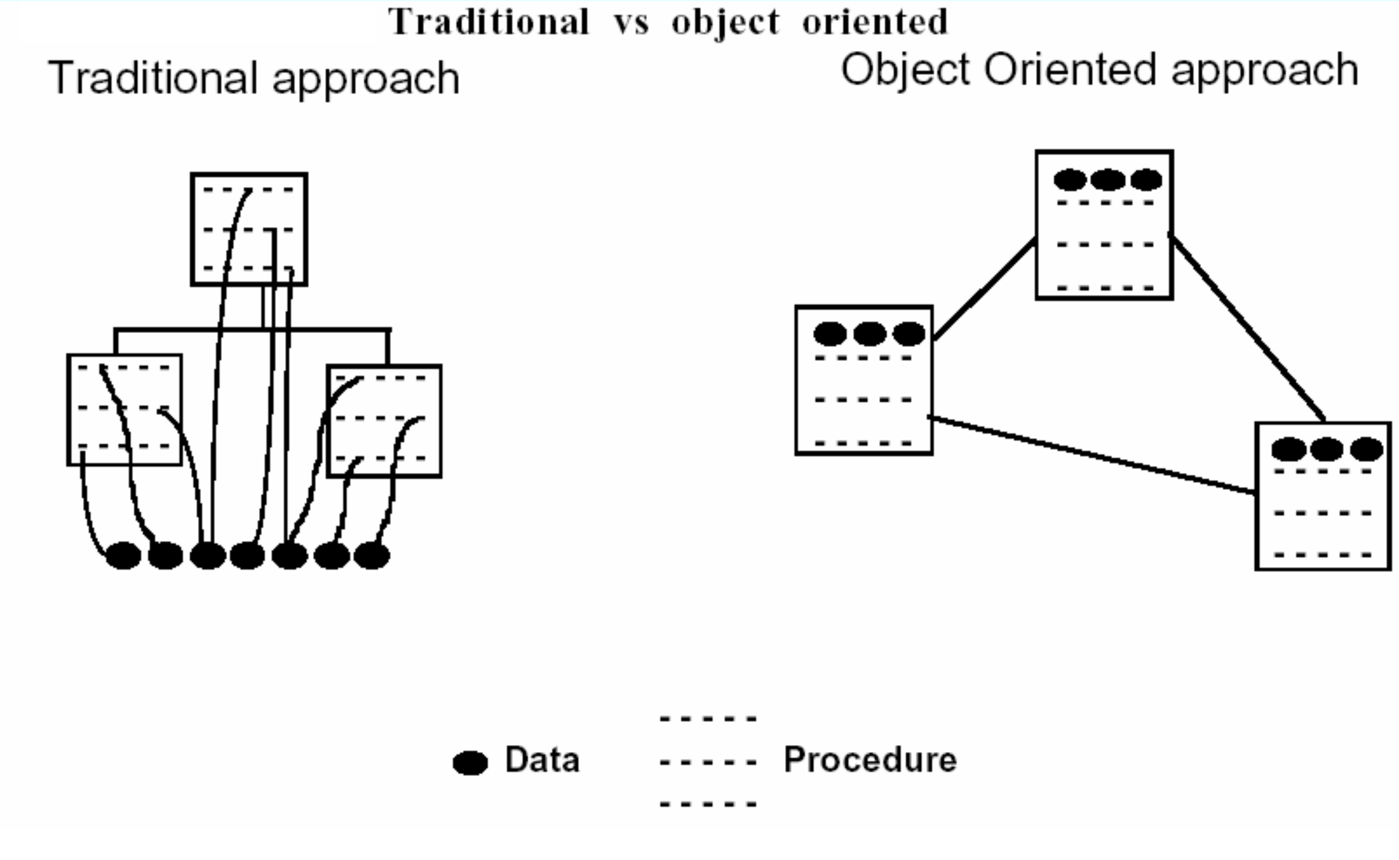
Object-Oriented programming

Object: A package which encapsulates data which represents the state and methods which change the state

Method: A specification of an object's change of state (data)

Result: Joining of procedure to data

Localization



Conclusions

- Software development involves a number of typical activities:
  - requirements specification
  - analysis
  - design
  - implementation
  - testing
- It is usually not feasible to perform these in strict sequence

Current best practice

- Use case driven
  - process driven by user requirements
- Management of risk
  - iterative development phases
  - incremental delivery of functionality
- Flexibility
  - refactoring
  - round-trip engineering

Recall RDD

- Responsibility-Driven Design is a set of tools for thinking about systems. It emphasises the concepts of Class, Responsibility and Collaboration. Objects behave by knowing, doing and deciding. Behaviour first. Data second . Class, responsibilities, collaborators are central concepts:
- **Class** A class describes the behaviour of a set of objects of the same kind. Identifies class on card essential when acting out scenarios.

Recall RDD

- **Responsibilities** are the knowledge that a class maintains and services that it provides. When acting out scenarios analyst must be able to develop or know the current responsibilities of a class. By discussing a problem in terms of responsibilities we increase the level of **abstraction**. This permits greater independence between objects, a critical factor in solving complex problems. The entire collection of responsibilities associated with an object is often described by the term **interface** or **protocol**.

Recall RDD

- **Collaborators:** A collaborator is a class whose services are needed to fulfil a responsibility. Collaborators must be related to the responsibilities that they help fulfil (1:M). Collaborators may help fulfil responsibilities for several classes. Collaborations only exist to fulfil responsibilities. Collaborations are modelled as one way communications from initiator class to collaborator , the response is a message answer.
- The above ideas can be combined using CRC cards.
- Using RDD produces a different type of architecture based more on the class diagram than the use case diagram.

Architecture and RDD

- Using RDD produces a different type of architecture based more on the class diagram than the use case diagram.
- RDD can facilitate component based reuse
- More of a ‘system view’ rather than a user view.

Abstraction

Abstraction Type	What It Does	Conformance and Justification
Operation	• Specifies what an operation achieves in terms of its effect on the object executing it rather than how it works	• Does the sequence of statements in code have the specified net effect?
Model	• Defines the state of an object (or component) as a smaller and simpler set of attributes than the actual variables or fields used in the design; or simpler than some other model that presents a more detailed view.	• How would you compute each abstract attribute from the data stored in the implementation, from the more detailed attributes?
Action	• Describes a complex protocol of interaction between objects as a single action, again characterized by the effect it has on the participants.	• What sequences of detailed actions will realize the effect of the abstract action? Use state charts, sequence or activity diagrams.
Object	• Treats an entire group of objects (such as a component or subsystem or corporation) as if it were a single one, characterizing its behaviour with a type.	• How do the constituent objects (and their actions) respond to the abstract object (and its actions)?