



MIT 813: OOAD

Session I

Object-Oriented Analysis and Design

MMUST

Kelvin K. Omieno

Disclaimer

- Slides come from a variety of sources:
 - Craig Larman-developed slides; author of this classic textbook.
 - Dr. Constantinos Constantinides, University of London
 - Slides from the University of Pittsburg
 - Slides from many of my existing slides on these same topics
 - New slides from sources unknown...

Thinking in Objects and UML - I

- The Unified Modeling Language (UML) is a standard diagramming notation; sometimes referred to as a blueprint.
- It is NOT OOA/OOD or a method
- **Only a notation for capturing objects and the relationships** among objects (dependency; inheritance; realizes; aggregates, . .)
- UML is language-independent
- Analysis and design provide software “**blueprints**” captured in UML.
- Blueprints serve as a tool for thought and as a form of communication with others.

Thinking in Objects and UML – 2

- But it is far more essential to ‘think’ in terms of **objects** as providing ‘**services**’ and accommodating ‘**responsibilities**.’
- **Discuss:** What is meant by ‘services?’ How indicated?
 - How might you think these ‘services’ impact the design of classes?
 - How might a client access these services?
 - What is encapsulation? How does it relate to reusability? Self-governance? Design?
- **Discuss:** What is meant by ‘responsibilities?’
 - Encapsulation of data and services?

Thinking in Terms of Objects and UML - 3

- **Object-Oriented Analysis (Overview)**

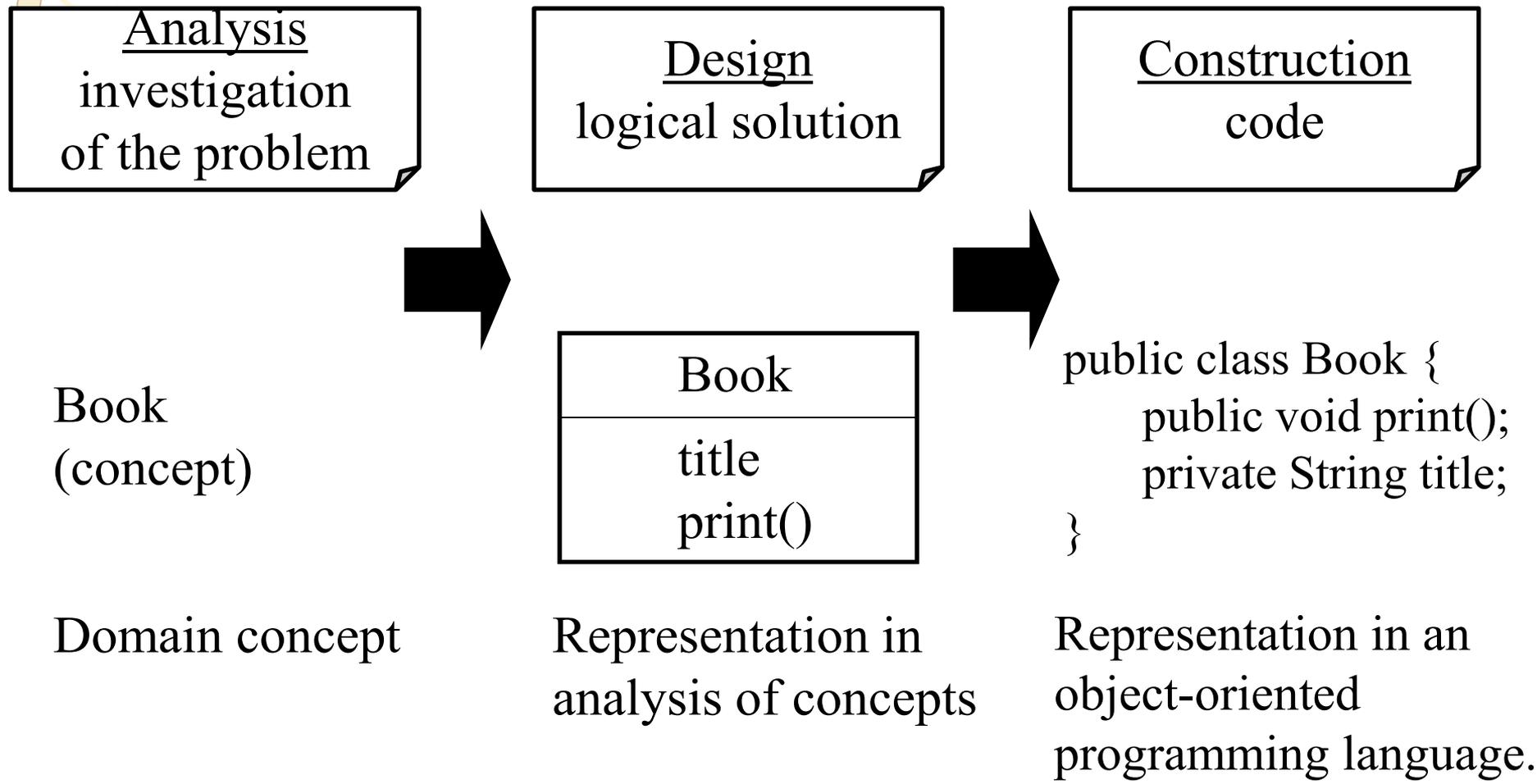
- An **investigation** of the **problem** (rather than how a solution is defined)
- During OO analysis, there is an emphasis on finding and describing the objects (or concepts) in the **problem domain**.
- For example, concepts in a Library Information System include *Book*, and *Library*.
- High level views found in the application domain.
- Oftentimes called **domain objects; entities**.

Thinking in Terms of Objects and UML - 4

- **Object-Oriented Design**
 - Emphasizes a conceptual **solution** that fulfills the requirements.
 - Need to define **software objects** and how they **collaborate** to meet the requirements.
 - For example, in the Library Information System, a *Book* software object may have a *title* attribute and a *getChapter* method.
 - What are the methods needed to process the attributes?
- Designs are **implemented** in a programming language.
 - In the example, we will have a *Book* class in Java.

Thinking in Terms of Objects and UML – 5

From Design to Implementation



Thinking in Objects and UML-6

- Then too, there are sets of **proven design solutions** to problems that are considered ‘best practices.’
 - Certain ‘groupings’ of classes with specific responsibilities / interfaces.
 - These provide specific solutions to specific problems.
 - Called **Design Patterns**
- We will discuss (much later) these standard patterns and how to **apply** them to develop solutions to common design problems.

Thinking in Objects and UML-7

- Of course, design (solution to requirements) 'assume' a robust **requirements analysis** has taken place.
- **Use Cases** are often used to capture **stories** of requirements and are often views as 'constituting' the **functional** requirements, but NOT the software quality factors (non-functional requirements).
- Use Cases are **not specifically designed** to be object-oriented, but rather are meant to capture how an application will be used.
- Many methods for capturing requirements.
- We will concentrate on Use Cases (ahead).

Basic Terms: Iterative, Evolutionary, and Agile

I. Introduction

- **Iterative** - the entire project will be composed of min-projects and will iterate the same activities again and again (but on different part of the project AND with different emphases) until completion.
- **Evolutionary (or incremental)** - the software grows by increments (to be opposed to the traditional, and somewhat old-fashioned, Waterfall model of software development).
- **Agile** - we will use a light approach to software development rather than a very rigid one (which may be needed for a safety-critical system for example)
- This kind of approach seems better at treating software development as a **problem solving activity**; also the use of objects makes it amenable.

Our Approach:

- We need a Requirements Analysis approach with OOA/OOD need to be practiced in a framework of a development process.
- We will adopt an agile approach (light weight, flexible) in the context of the Unified Process, which can be used as a sample iterative development process.
 - Within this process, the principles can be discussed.
- Please note that there are several other contexts that may be used, such as Scrum, XP, Feature-Driven Development, Lean Development, Crystal Methods and others...and we will look at a few of these.

Why the Unified Process:

- The Unified Process is a popular iterative software development process.
- Iterative and evolutionary development involves relatively early programming and testing of a partial system, in repeated cycles.
- It typically also means that development starts before the exact software requirements have been specified in detail;
- Feedback (based on measurement) is used to clarify, correct and improve the evolving specification:
- This is in complete contrast to what we usually mean by engineering!



University of
Choice

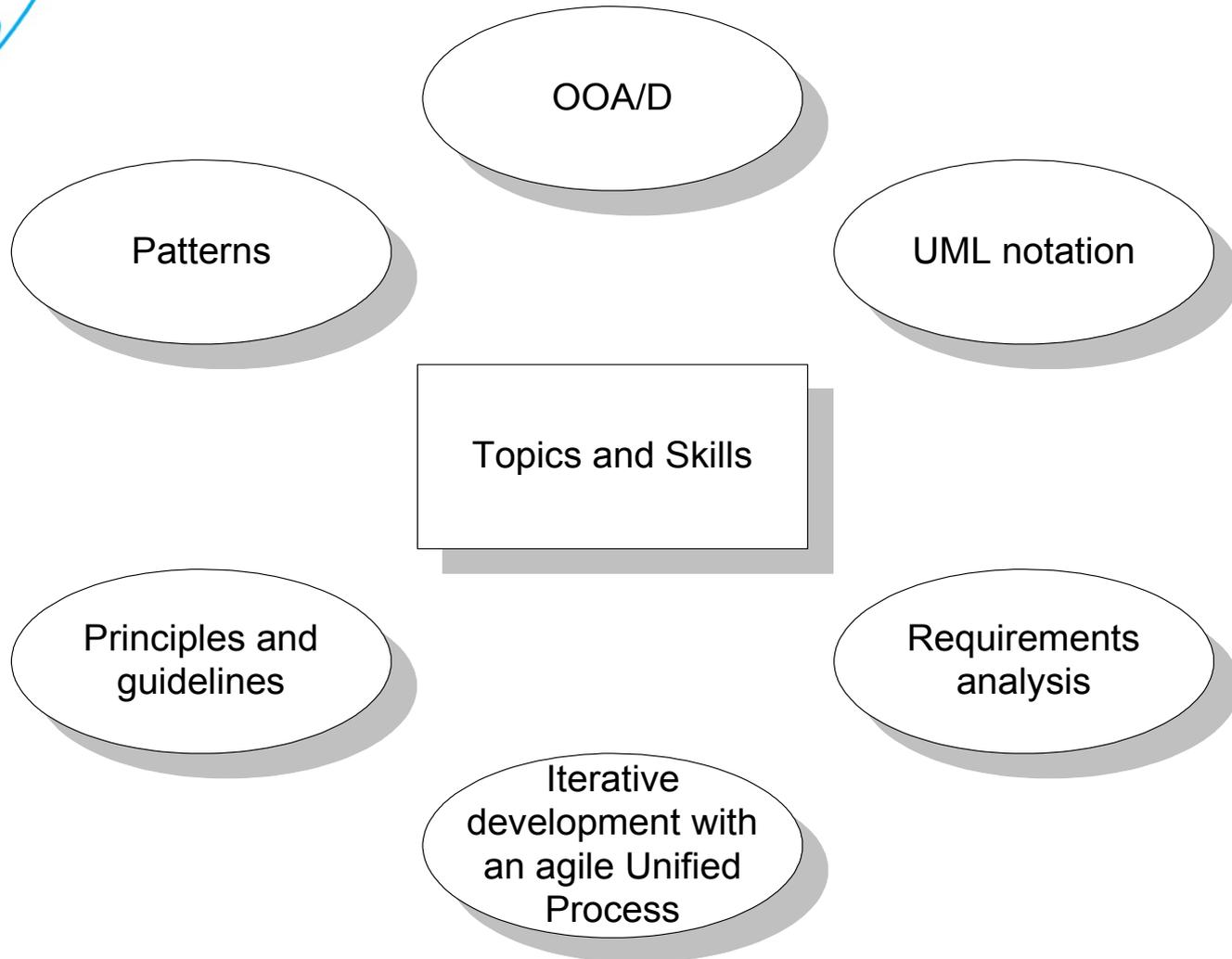
www.mmust.ac.ke

ISO 9001:2008 CERTIFIED

2. What is the Unified Process?

- The UP is very flexible and open and can include other practices from other methods such as Extreme Programming (XP) or Scrum for example.
 - e.g. XP's test-driven development, **refactoring** can fit within a UP project; So can Scrum's **daily meeting**.
 - Being **pragmatic** in adapting a particular process to your needs is an important skill : all projects are different.

We will be studying all of the topics found in Fig. 1.1





University of
Choice

www.mmust.ac.ke

ISO 9001:2008 CERTIFIED

The Rush to Code

- Critical ability to develop is to think in terms of objects and to artfully **assign responsibilities to software objects**.
- Talk at great length in COP 3538 about encapsulation and assigning methods to objects where the data is defined...
- One cannot design a **solution** if the **requirements** are not understood.
- One cannot **implement** the design if the **design** is faulty.
- If I could only stop my students....😊



University of
Choice

www.mmust.ac.ke

ISO 9001:2008 CERTIFIED

The Rush to Code

- Analysis: - investigate the **problem** and the **requirements**.
 - What is needed? Required functions? Investigate domain objects.
 - Problem Domain
 - The **Whats** of a system.
 - **Do the right thing (analysis)**
- Design:
 - Conceptual solution that meets requirements.
 - Not an implementation
 - E.g. Describe a database schema and software objects.
 - Avoid the CRUD activities and commonly understood functionality.
 - The Solution Domain
 - The '**Hows**' of the system
 - **Do the thing right (design)**

What is Object-Oriented Analysis and Design

- OOA: we find and describe **business objects** or concepts in the **problem domain**
- OOD: we define how these **software objects collaborate** to meet the requirements.
 - Attributes and methods.
- OOP: Implementation: we implement the design objects in, say, Java, C++, C#, etc.



University of
Choice

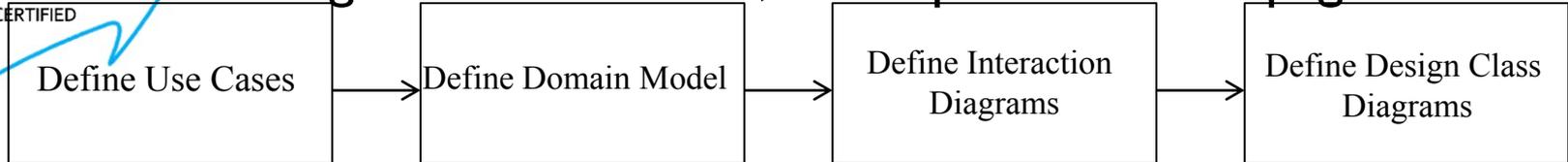
www.mmust.ac.ke

ISO 9001:2008 CERTIFIED

Homework Assignment #1

due: 3rd June start of class. Hardcopy please.

- Using the model below, develop a two-three page discussion



outlining the four activities listed and present the major features of each.

A short definition and example of a domain model, interaction diagram, and class diagram is sufficient, but be prepared to discuss each of these.

Also, have a general idea about use cases – what they are designed to do and what they are not designed to do.