

CA314 – Object Oriented Analysis & Design - 1

File name: CA314_Section_01_Ver01

Author: W.G. Tuohey

No. of pages: 9

Table of Contents

1. Introduction.....3
 1.1 Overview.....3
 1.2 Some specific topics to be covered.....3
 1.3 Some References.....3
 1.4 OO Processes4
 1.5 Quotations & Definitions.....6
 1.5.1 Components, Classes & Objects.....6
 1.5.2 Design Patterns & Frameworks7
 1.5.3 Unified Modeling Language (UML)7
 1.5.4 Types of Reuse.....8
 1.6 A Caveat about Component-Based Software Engineering.....8

1. Introduction

1.1 Overview

- Introduction to main ideas in OO Analysis & design
- Practical experience in applying ideas
- A large element is use of UML as an aid to thinking & describing

Will also cover some other SW Engineering topics including aspects of testing.

1.2 Some specific topics to be covered

Essentials of UML:

Use cases

Class models

Interaction diagrams

State diagrams

Activity diagrams

Implementation & deployment diagrams

Examples & case studies using UML:

To show how UML is used but also the ‘processes’ it supports & helps document

Object Constraint Language (OCL)

Part of UML (non-diagram) – Will use informally

System testing via Use cases

plus aspects of verification in general

Design Patterns

Introduction only

Frameworks

Architecture ...

Components

In brief

1.3 Some References

Note: Some of the associated web-sites have examples of source code etc

- 1) “Using UML - Software Engineering with Objects & Components”, Stevens & Pooley (Addison-Wesley)

Note: We will use this book quite a bit.

- 2) “Object-Oriented Software Engineering, using UML, Patterns, and Java”, Bruegge & Dutoit (Pearson Prentice-Hall)

- 3) “Software Design, from Programming to Architecture”, Braude (Wiley)
- 4) “Component-Based Software Engineering, putting the pieces together”, Heineman & Councill (Eds) (Addison-Wesley)

+ Many Other References

1.4 OO Processes

There have been several approaches to developing OO software suggested by different authors. Approaches typically include both some kind of modeling language and a process that specifies the tasks to be done and how they should be ordered etc. The UML language is the result of some of the leading authors coming together to combine their individual languages and notations. The following diagram (Fig. 1.4-1) summarises the OO process as seen by one author – essentially, it is an overview of how *this* writer sees various elements of UML & Software Engineering Processes fitting together. We will come across variations in some of the cases and examples we will look at.

OOA&D Roadmap (from Braude, ref 3, p463)

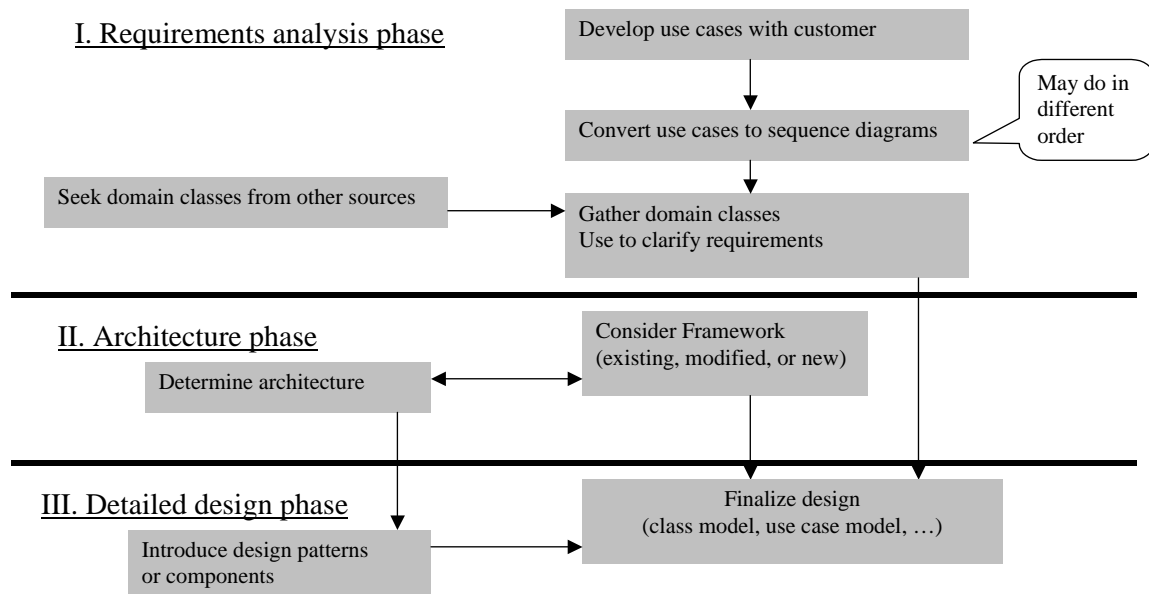


Figure 1.4-1: OOA&D Roadmap (Braude)

Note regarding architecture: It may be helpful to bear in mind the following classification of software architectures presented by Braude,

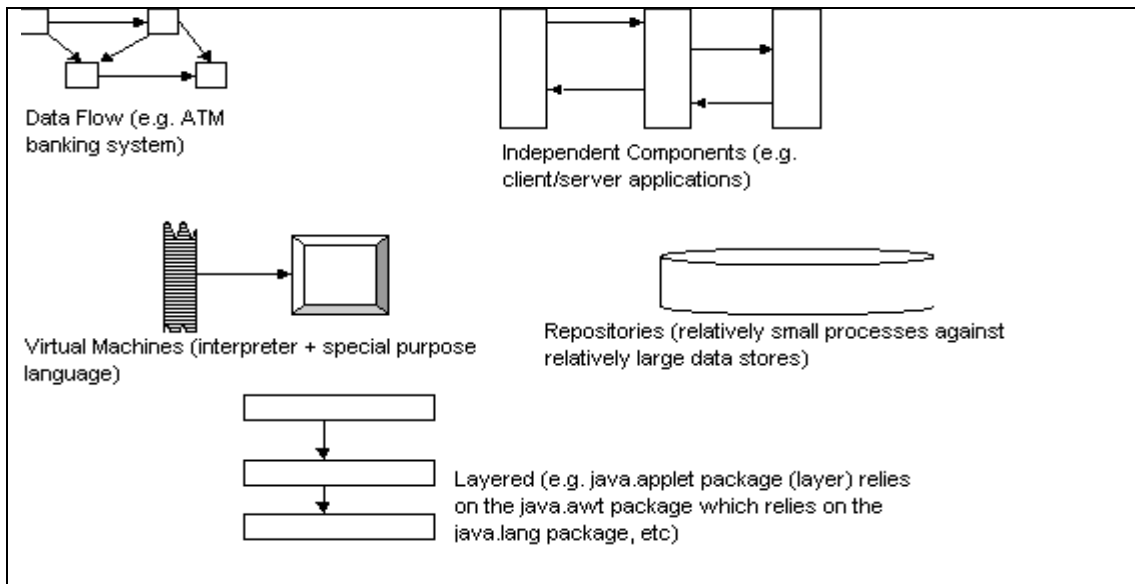


Figure 1.4-2: A Classification of Software Architectures

As another OO process example, the following two figures (Figures 1.4-3a and 1.4-3b) summarise the process suggested by Booch, later one of the three originators of UML.

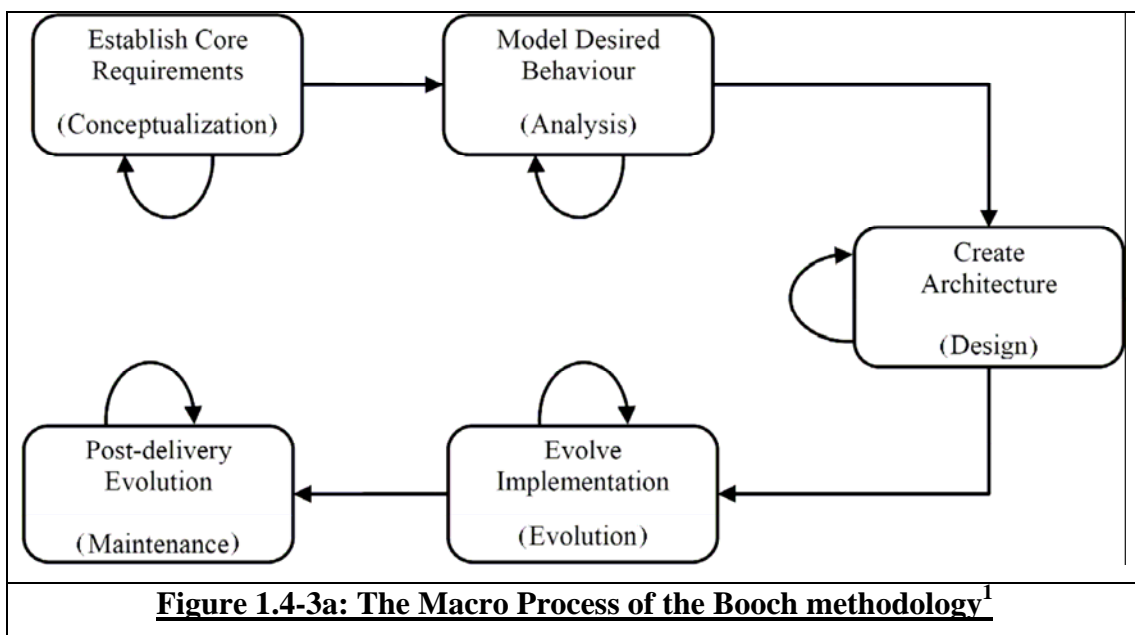
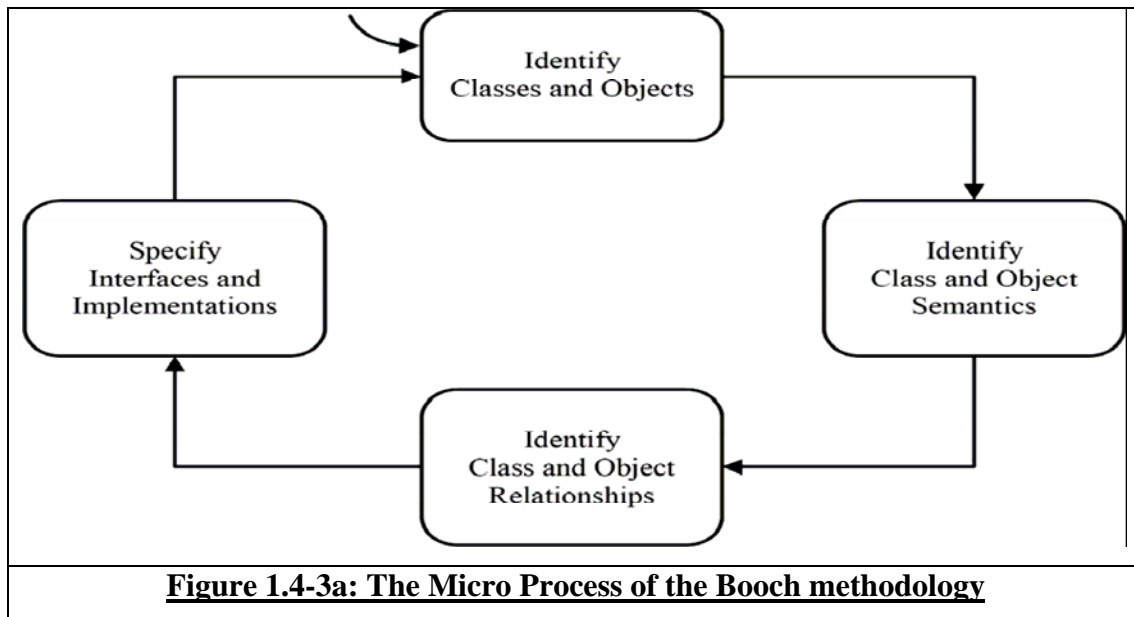


Figure 1.4-3a: The Macro Process of the Booch methodology¹

¹ As adapted by Ramsin, R, Paige, R. F. (2008), "Process-Centered Review of Object Oriented Software Development Methodologies", ACM Computing Surveys, Vol. 40, No. 1, Article 3



1.5 Quotations & Definitions

1.5.1 Components, Classes & Objects

[Ref (4), p36]:“Objects ... encapsulate state and behaviour, and have a unique identity.

The behaviour and structure of objects are defined by the classes.

A class ...

- Implements the concept of an Abstract Data Type (ADT)² and provides an abstract definition of the behaviour of its objects.
- Provides the implementation of object behaviour.
- Is used for creating objects i.e. instances of the class.

The basic principle of object-orientation is to construct programs from sets of collaborating and interacting objects.

Components are similar to classes³. Like classes, components define object behaviour and create objects [i.e.] component instances.

Both components and classes make their implemented functionality available through abstract behaviour descriptions called interfaces.

Unlike classes, the implementation of a component is generally completely hidden ...

Unlike classes, component names may not be used as type names.

...

² An ADT specifies the type of data stored, the operations performed on them, and the types of the parameters of the operations. An ADT specifies what each operation does but not how it does it.

³ At least as ‘component’ is as defined in Ref 4.

Most important distinction is that software components conform to a standard defined by a component model.

1.5.2 Design Patterns & Frameworks

(from ref 2):

A design pattern is a template solution that developers have refined over time to solve a range of recurring problems. A design pattern includes a name, a problem description, a solution, and a set of consequences.

[e.g. ref (2) presents examples,

- Abstract factory
 - Adapter (wrapping legacy code)
 - Bridge (allowing for alternate implementations)
- etc]”

Note: There’s a great deal on the internet on design patterns.

A Framework is a set of classes providing a general solution that can be refined to provide an application or a subsystem”

1.5.3 Unified Modeling Language (UML)

UML is a standard set of notations for representing models”

- “OMG Unified Modeling Language Specification”, Version 1.5 can be viewed from module CA314 web-pages. We will occasionally use material from this specification but it is definitely not necessary to be familiar with all of it. In fact, Version 2.0 is now the current version so we may refer to that if necessary – it also can be located on the internet. However for the purpose of CA314, the differences are not likely to be that important.

- The following extract from the specification highlights that UML supports different models of a system that are suitable for different viewpoints (user, customer, requirements, design, etc):

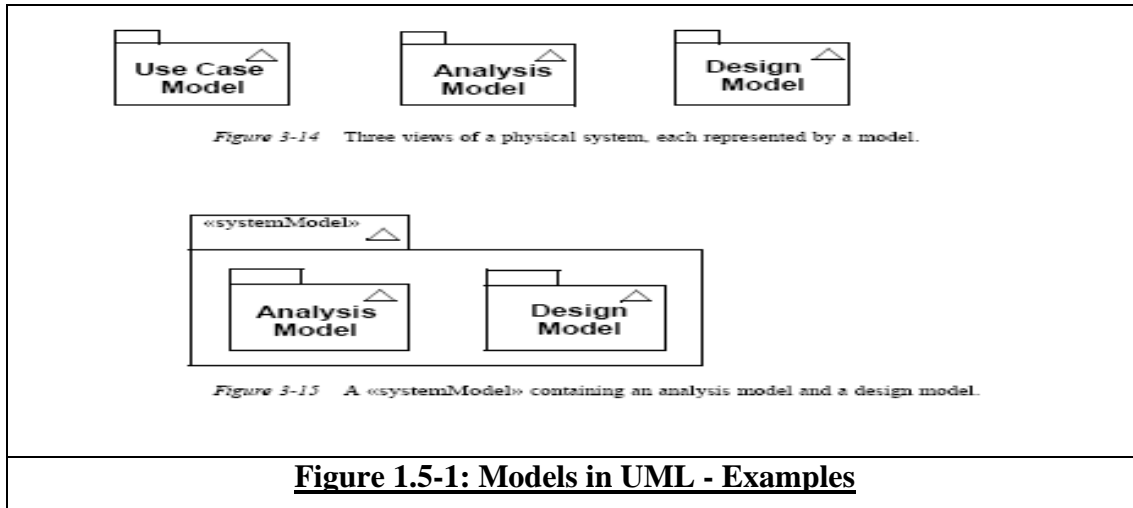


Figure 1.5-1: Models in UML - Examples

- A model captures a view of a physical system. Hence, it is an abstraction of the physical system with a certain purpose; for example, to describe behavioral aspects of the physical system to a certain category of stakeholders. A model contains all the model elements needed to represent a physical system completely according to the purpose of this particular model. ...

1.5.4 Types of Reuse⁴ (from ref 3)

- Object code
- Classes - In source code form
- Assemblies of related classes
- Patterns of class assemblies

Also,

“A straightforward kind of reuse is one in which we do not alter the element being reused. ... component technology uses this approach.”

1.6 A Caveat about Component-Based Software Engineering

(from ref 4) “For component-based software engineering to revolutionize software development, developers must be able to produce software significantly cheaper and faster [than heretofore].

At the same time, the resulting software must meet high reliability standards while being easy to maintain. ...”

Problems may well arise in verifying these objectives:

- Cannot unit test?

⁴ Remember that authors may have different views of such topics!

- Is “component” tested in every anticipated environment [In the often cited, though now rather old case of the ARIANE 5 failure, we have for example the findings

“m) The inertial reference system of Ariane 5 is essentially common to a system which is presently flying on Ariane 4. The part of the software which caused the interruption in the inertial system computers is used before launch to align the inertial reference system and, in Ariane 4, also to enable a rapid realignment of the system in case of a late hold in the countdown. This realignment function, which does not serve any purpose on Ariane 5, was nevertheless retained for commonality reasons and allowed, as in Ariane 4, to operate for approx. 40 seconds after lift-off.

n) During design of the software of the inertial reference system used for Ariane 4 and Ariane 5, a decision was taken that it was not necessary to protect the inertial system computer from being made inoperative by an excessive value of the variable related to the horizontal velocity, a protection which was provided for several other variables of the alignment software. When taking this design decision, it was not analysed or fully understood which values this particular variable might assume when the alignment software was allowed to operate after lift-off.

o) In Ariane 4 flights using the same type of inertial reference system there has been no such failure because the trajectory during the first 40 seconds of flight is such that the particular variable related to horizontal velocity cannot reach, with an adequate operational margin, a value beyond the limit present in the software.

p) Ariane 5 has a high initial acceleration and a trajectory which leads to a build-up of horizontal velocity which is five times more rapid than for Ariane 4. The higher horizontal velocity of Ariane 5 generated, within the 40-second timeframe, the excessive value which caused the inertial system computers to cease operation.”

<http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html>, **checked 28/09/2009]**

- How to maintain, upgrade software with components? [A problem is that components are bought in and source code may not be available]