Université de Paris

Génie Logiciel
Initiation to UML

Sylvain Lobry

14/10/2022

Introduction

Before we start…

- https://www.wooclap.com/L3GL6

Cost estimation

COCOMO

- by 1000 lines of codes (KLOC): Cost = $\alpha \times KLOC^{\beta} + \gamma$ with
  - α: marginal cost per 1000 LOC (KLOC)
  - γ: fixed cost of a project
  - β: scale factor

Cost estimation

COCOMO

- Example on prices of yoghurt
- 1 yoghurt = 1 euro (marginal cost, α)
- Scale factor: the more you buy, the less expensive each unit is: β = 0.95
- fixed cost: distributor = γ = 0.20 euros

- 4 yoghurts : price = $\alpha \times Y^{\beta} + \gamma$ = 1 * 4 ^ 0.95 + 0.2 = 3.93 euros
  - price per yoghurt = 3.93 / 4 = 0.98 euros

- 12 yoghurts: price = $\alpha \times Y^{\beta} + \gamma$ = 1 * 12 ^ 0.95 + 0.2 = 10.80 euros
  - price per yoghurt = 10.80 / 12 = 0.90 euros

Cost estimation

COCOMO

- by 1000 lines of codes (KLOC): Cost = $\alpha \times KLOC^{\beta} + \gamma$ with
  - α: marginal cost per 1000 LOC (KLOC)
  - γ: fixed cost of a project
  - β: scale factor
- Parameters proposed by Boehm in the COCOMO (COnstructive COst MOdel) method (1981)
- For a simple project:
  - α = 2.4
  - β = 1.05
  - γ = 0
  - Cost in man months -> to be multiplied by the average monthly cost of an employee

Cost estimation

COCOMO

- For a project of 3000 lines of code: $2.4 \times 3^{1.05} + 0 = 7.6$ person.months
- For a salry of 5000 euros / months: 7.6 x 5000 = 38 000 euros

Planification

- Gantt, PERT, WBS can be considered as views on the **same** plan
- Different usages:
  - Gantt for visualization
  - WBS to understand the semantic relationships between tasks
  - PERT to analyze the temporal relationships between the tasks and compute the critical path

Administration du cours

Semaine prochaine

- Cette semaine: TD en autonomie
- La semaine prochaine: **cours en Amphi Binet** (noté sur ADE)

Initiation to UML

Introduction to UML

- Graphical modeling language
- Objectives:
  - Providing a description of a software
  - Allowing the visualization of the different aspects of a software
  - Analyzing the software
  - Allow for communication inside and outside a project; with technical and non technical people
  - Verification of completeness, consistency and correctness
- General purpose modeling language
  - Process independent
  - Can be used to represent information on the **structure**, the **behaviour**, or the **interaction**

Initiation to UML

Views

- A model is composed of several views
- A view describes a system from different perspectives.
- Example of views:
  - Structural view: gives information on the structure of the model
  - Behavioral view: gives information on the behavior of the model
  - Interaction view: gives information on the way different parts of the model behave **with respect to each other**

Initiation to UML

Types of diagrams

- UML defines 13 diagrams in 3 categories which can define a system according to different points of view
- Structure diagrams
  - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram
- Behavior diagrams
  - Use Case Diagram Activity Diagram and State Machine Diagram
- Interaction diagrams
  - Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram

Initiation to UML

Reminders on object-based approach

- UML is based on an object-based approach

- Definition of an **object**: An object is an entity referenced by an identifier. It is often tangible.
- An object has a set of attributes (structure) and methods (behaviour)

Initiation to UML

Reminders on object-based approach

- Definition of a **class**: set of similar objects (i.e. having the same attributes and the same methods).
- An object from a class is an *instance of this class*.

- Defintion of the **abstraction**: principle of selecting the relevant properties of an object for a given problem
- Important aspect of UML: the real object is simplified by its abstraction to only keep what is relevant to the modelization.

Initiation to UML

Reminders on object-based approach

- Definition of **encapsulation**: to hide some attributes or methods to other objects. Note that this in an abstraction.

- **Specialization**: a new class A can be created as a subclass of another class B, in which case class A specializes the class B.
- **Generalization** is the opposite (superclass B is a generalization of subclass A).

- **Inheritance**: the fact that a subclass gets the behaviour and the structure of the superclass
- This is a **consequence** of specialization

Initiation to UML

Reminders on object-based approach

- **Abstract** and **concrete** classes: abstract classes are classes that do not have instances (e.g. Mammal). Concrete classes do (e.g. Human).
- Abstract classes allow for class hierarchies and to group attributes and methods. They should have subclasses.

- **Polymorphism**: behavior from objects of a same class (in general abstract) can be different as they are instances of different subclasses.

Initiation to UML

Reminders on object-based approach

- **Composition**: complex objects can be composed of other objects.
- It is defined at the class level, but we only compose actual instances
- It can be:
  - a strong relationship: components cannot be shared; destruction of the composed object implies destruction of the components
  - a weak relationship (a.k.a. aggregation): components can be shared

Initiation to UML

What is a software model?

- Formalized as a document
- Not just diagramms!
- The document should state:

1. Practical information (authors, date, version)
2. Context of the project
3. Introduction to the model (choices, which views, discussion)
4. The diagrams, centered around use cases

Initiation to UML

Conclusion

- UML allows to model software
- UML is a standard
    - People thought about it
    - Allows for good communication
    - Strong community
    - Evolution
    - Will not disappear tomorrow
- UML is hard to master (and we will not master it in this class)