Object Oriented Analysis and Design and Software Development Process Phases

Calin Curescu
Why object oriented?

• Because of growing complexity!
  – How do we deal with it?

1. Divide and conquer

2. Iterate and increment

3. Reuse and recycle

4. Strong cohesion and low coupling
  – Among different entities
Principles of OO

- **Abstraction**
  - Identify essential characteristics
  - 1,2,3

- **Hierarchy**
  - Order of abstractions
  - 1,2

- **Encapsulation**
  - Need to know principle
  - 4,1

- **Classification**
  - Bring similar things together
  - 3,1,4

- **Polymorphism**
  - Classify related differences
  - 3,2

- **References**
  - “Object Oriented …”
  - By UML guys
    - Booch
    - Rumbaugh
    - Jacobson
  - And many others
OO-A/D/P and Processes

- Business modeling
- Capture requirements
- Analysis
- Design
- Implementation
- Testing and validation
- Deployment
- Configuration and change
- Project management
Before and after contract

- **Inception**
  - Elucidate requirements
  - Perform a rough OOA + OOD
  - Calculate resources and costs
    - Personnel, Time, COTS

- **Elaboration, construction, transition**
  - Detailed OOA & OOD
  - Test planning and testing
  - Lots of documents
Requirements

- **Functional**
  - What the system should do!
    - Client should be able to withdraw cash
    - Students should be registered

- **Non-functional**
  - How it should behave while doing it
    - portability
    - reliability
    - performance
    - testability
    - modifiability
    - security
    - cost
    - presentation
    - reusability
    - understandability
    - acceptance criteria
    - interoperability
Requirements

- **Satisfiability**
  - Normal
  - Expected
  - Exciting

- **Criticality**
  - Mandatory
  - Desirable
  - Non-critical

- **Stability**
  - Stable
  - Non-stable

- **User types**
  - Ones that order
  - Ones that use
Requirements Specification

- Correct. There should be no factual errors.
- One interpretation only. We have seen a few examples of ambiguous statements above.
- The SRS should be complete.
  - But requirements do change. So the format should allow changes
- All requirements must be verifiable (testable).
- All requirements must be consistent and non-conflicting.

- The requirements specification should be design neutral
  - Concentrate on *what* the system should do not on *how* it should do it.
What should we do? (1)

• **Requirements**
  – Requirements specification
  – Construct use cases
    • Is any use case not described in the requirements specification?
    • Are there any conflicts between the descriptions and specification?

…Iterative

…Incremental
What should we do? (2)

- **Object oriented analysis**
  - Find objects and relations
    - Substantive
    - Brainstorm
    - Checklist
  - Delegate responsibilities of activities to objects
    - What should they do?
    - CRC cards
  - Create Class diagrams
  - Interaction diagrams
    - Sequence diagrams
    - Collaboration diagrams
What should we do? (3)

- **Object oriented design**
  - Detail class diagrams
  - Detail interaction diagrams
  - Detail statecharts and/or activity diagrams

- **Create test plans**
  - Unit tests
  - Integration tests

- **Implement and Test**

- **Note** all the activities in the last 3 slides are
  - Iterative
  - Incremental
How to find objects

- **Substantives**
  - From requirements specification
  - From use cases

- **Brainstorming**
  - Free flow of ideas

- **Checklist**
  - Roles
  - Facts
  - Situations
  - Interactions
  - Concepts
  - Information
  - Controllers
  - Outer entities

- **Afterwards: classify, group, eliminate, relate them**
  - Design patterns are useful here
• A standard (handy) way for describing classes
• Small cards are used
  – Should not support too much text
  – A4 / 4

• Class name
• Responsibilities
• Collaborations
• Back side can be used for data attributes
• Can only show that errors exist, not that they don’t exist

• **Unit test**
  • **Module test**
    – Whitebox
    – Blackbox
    – Regression

• **Integration test**
  – Top down
  – Bottom up

• **System test**
  – Against requirements specification

• **Validation & Verification**
Example: Course management

• **Requirements**
  – Students should be able to register pairwise
    • Students data should be available in the system
  – A course should have different examination moments
    • Every moment has a status out of a grade scale
    • Every such scale should have a “passed” value
  – Assistants should be able to set grades
    • Data about assistants should be available in the system
  – Examiner + Assistants should be able to get a list of
    • Students that have successfully finished
    • Students that have not finished
    • All students
**Use cases**

- **Register to lab**
- **Set grade**
  - **Get list of passed**
  - **Get list of students**
- **Get list of not passed**

**Objects**
- **Student**
- **Examiner**
- **Assistant**

**Relationships**
- **Student** includes **Register to lab**
- **Student** includes **Set grade**
- **Examiner** includes **Get list of passed**
- **Assistant** includes **Get list of students**
Find objects

- Student
- Assistant
- Examiner
- Lab group
- Lab moment
- Grade scale
- Result list
- Course

- Result register
- Person register,
- Student register
A initial class diagram

- Define relations
- Set multiplicity

- Problems:
  - Grade for group or individual?
  - How do I link moment and grade?
  - Did we find all objects?
More objects to add

- A result links a student a grade and a moment
  - Grade follows even if moved to another group
  - If a group member resigns – the other has the grade
Do an assessment

- Do we still need some objects?
- Do we have not necessary ones?
- Do we still have some new relationships?
- Do we have some not necessary relationships?

- Remember
  - Association
  - Generalisation
  - Aggregation
  - Composition
Specialisations of person
Define responsibilities for objects

- **Good objects do one (or a few) things well**
  - Otherwise we have to split the object
  - Remember divide and conquer
- **It can be a more complex task**
  - That needs several operations
  - Needs collaboration with other objects
- **Classes can have attributes and operations**
  - And they represent what?
  - Remember
    - low coupling among objects
    - Strong cohesion within object
Some rules of the thumb

• **The expert**
  – If a class has the knowledge to do something then let it do it
  – But not too many things

• **Who creates whom? – B creates A if**
  – B contains A
  – B is responsible for storing A
  – Uses A (strongly coupled)
  – B has the initialisation data for A
  – Factory? Anyone?

• **Who deals with system functions?**
  – The objects used until now are descriptive (contain data)
  – Should we use “Course” to get the lab registration, set grade, write lists – what if a course disappears?
  – Or should we use a controller?
    • Remember design patterns?
    • E.g. “StudentHandler”
After adding some controller objects
• Look at the requirements
• Look at the use cases
  – “assistant sets the grade for a student at a certain moment”
    • Handled by assistant handler
  – Who should manipulate the grade?
    • Should the assistant handler create it and send to student?
    • Should the student create it?
Cohesion and coupling again

- For each communication with a new class the coupling increases
  - Create, return, have as argument, call method…
- Internal data should be private
  - Use interfaces to provide only a part of the object
  - Or use private and protected modifiers
- A class should not have several strong relationships with two other classes
  - Increases coupling