Object-Oriented Software Development

Jürgen Börstler
jubo@cs.umu.se
http://www.cs.umu.se/~jubo

http://www.cs.umu.se/kurser/TDBC31

Goals of the Course

- Working in teams
- Introduction to object-oriented development
  - Iterative and incremental development processes
  - Object-oriented techniques
  - Work product orientation
- Introduction to real-life project issues
- Usage of industrial-strength tools

- This is NO programming course
  - We do not teach any (OO) programming
  - We assume you have sufficient programming skills

Literature

- Lecture overheads
- Various documents and resources linked to course web page

Project Issues

- Teamwork (5-7 students)
- Complete life-cycle coverage
- "Real" (external) customers
- Subcontractors
- Fuzzy and/or volatile requirements
- Many deliverables
- Several project presentations
- Formal team meetings
- Formal reporting routines
Project Examples

- Games
- Simulations
- Editors
- Project planning
- Trouble Reporting System
- Video Manipulation
- ...

- See course home page for current proposals
- Proposals will be presented in next lecture

Team Roles and Organisation

- Each team has an (external) customer and a supervisor
  - Customer ≠ supervisor
- Each team contracts another team for implementing a well-defined part of its project
  - Formal subcontract
  - Pairwise subcontracting is not allowed
- Each team evaluates another team’s prototype
  - Feedback for development of final product
  - Pairwise evaluation is not allowed

Teamwork

- Team members take on specialist roles (e.g. team manager, requirements specialist, ...)
- All team members must engage (at least) in OOA&D (in addition to role specific responsibilities)
- Individual diaries for project tracking
- Regular (formal) meetings to co-ordinate work
  - Team internal
  - Team managers with supervisors
- Team decisions must be followed by all team members
- All team members are finally responsible for the overall project outcome

Specialist Roles

- Team Manager
- Requirements Specialist
- User Interface Specialist
- Design Specialist
- Code Production Specialist
- Quality Assurance Specialist
- Documentation Specialist
- Tools Specialist
- Webmaster
- ...

- Please feel free to add further roles as needed
- We recommend to have at least two individuals for each role
Specialist Responsibilities

- Research your job
- Identify your tasks and specific responsibilities
- Estimate the time to complete your tasks
- Monitor the completion of your tasks
- Track your effort (diary)
- Inform team members about the status of your work
- Keep yourself informed about project progress
- Maintain a list of problems / items to discuss

- Be pro-active not re-active
  - Keep yourself informed on what is going on in the project
  - Show up in team meetings

Team Building

- Students fill in a questionnaire
  - Personal data
  - Skills
  - Preferences
- Match and mix student skills and preferences to build equal teams

- You need convincing arguments to change teams

- OOPS! Some students will arrive late
- Teams should maintain “job openings”

Effective Teamwork

- Create clear goals
  - Understand each other’s expectations
- Go for small wins
  - Set attainable and concrete short term goals
- Build mutual trust
  - Listen to each other
  - Show respect
  - Be fair and objective
- Ensure mutual accountability
  - No finger pointing
- Help each other

See web resources for more information.

Project Documentation

- Up-to-date, web-based project workbook
- Weekly reports
- Individual diaries
- Deliverables with firm deadlines
  - Team description
  - Project management
  - Requirements document
  - (G)UI design
  - Analysis/Design document
- Subcontract
- Prototype user manual
- Prototype evaluation
- Final report

- OOPS! Some deadlines might be negotiated
Presentations

- Team presentation, project plan, requirements, GUI mock-up, iteration plan
- Project progress, OOA&D, prototype demonstration, revised iteration plan
- Project summary, product presentation and demonstration, reflections, planned vs. actual work

- All presentations should include actual project data

→ See course web pages for details

Grading

- Individual grades (U, 3, 4, 5)

  Credit system:
  - Each team starts with 0 credits
  - For each “performance” a team earns credits (quality x importance)
    - Quality on scale 0..6
    - Importance on scale 1..15
  - For each deadline missed a team loses credits
  - Credits earned (adjusted by #team members) can be “freely” distributed among the team members
  - Grade is determined by the final number of credits

- In case of problems/failure:
  - Extra time for teams to complete projects
  - Extra assignments for individuals

Course Evaluation

- Very positive, in particular in recent years

- Criticism from earlier evaluations (pre 2002)
  - Workload too high
  - Too little calendar time
  - Unclear grading system
  - Too much focus on deliverables
  - Lectures not synchronised with deliverables
Major Changes Since 2002

- 15 ECTS instead of 7.5
- Several iterations (currently 3)
- Very detailed grading “rules”
- More formal progress reporting
  - Team managers meet weekly with supervisors
  - Explicit requirements for weekly reports
  - Individual diaries
- Subcontracting with teams from another course
- Questionnaire to support team formation
- Peer evaluations for early team trouble-shooting
- Signature blocks required on deliverables
- New textbook and UML tool (VP-UML)

Latest Changes

- 50%-rule for credit distribution
- Subcontracting within same course
- Web-based diary submission
- Only small changes in contents since fall 2001, since students were quite satisfied

Teams Must Take Initiative

- Teams are responsible for organising their work
- Schedule and organise meetings (with team, customer and supervisor)
- Contact customers to gather project information
- Keep all stakeholders informed about the project
- Learn about necessary methods/tools/environments
- …
- Your customers (and supervisors) are busy people
  - Make appointments in good time

Obligatory Attendance

- **Lectures:** At least 2 team members per team

- **Presentations:**
  - *Your own team presents:* All team members of your team must attend
  - *Another team presents:* At least 3 team members of your team must attend

- **Weekly Management Meetings:** At least one team member per team (typically the team managers)

- **Work Load:** On average you are expected to work about 20h/week (about 15 person months per team)
Tools: General

- The usage of an approved UML tool is obligatory
- You can use other tools/languages/builders environments etc. as you like, if they seem appropriate for your team and/or project

**HOWEVER**

- You get support only for the tools and environments we provide on our lab machines
- Problems due to the usage of “non-standard” tools/environments are solely your’s
- Make sure your projects do not depend on such tools/environments

Tools: Recommendations

- UML diagramming
  - VP-UML (local license server)
  - Many free, but restricted, versions of commercial tools
- Project Management
  - MS Project
  - Planner (Open Source)
- Version control
  - CVS
- Team collaboration
  - Shared workspaces for distributed teams (e.g. a wiki)
- Misc
  - Project workbook template
  - Documentation from old courses

Contents

- Introduction
- Object-Oriented Software Development
- Project Management
- Requirements Gathering
- (G)UI Design
- Object-Oriented Analysis and Design
- Advanced Topics in OOA&D
- Implementation and Testing
- References

Object-Oriented Software Development

- What is Object-Oriented Development
- Object-Oriented vs. Traditional Development
- An Object-Oriented Development Framework
- Phases, Activities, and Work Products
What is Object-Oriented Development

“Object-oriented software construction is the software development method which bases the architecture of any software system on modules deduced from the types of objects it manipulates (rather than the function or functions that the system is intended to ensure).”

[Meyer 97]

◆ A very brief history
  □ 1966: Object-oriented programming [Simula (Dahl and Nygaard)]
  □ 1982: Object-oriented design [with Ada (Booch)]
  □ 1988: Object-oriented analysis [OORA (Coad), OOSA (Shlaer and Mellor)]
  □ 1997: Unification of notations [UML V1.0 (Booch, Jacobson, Rumbaugh)]
  □ 1999: Unification of processes [RUP (Kruchten, Rational)]
  □ 2003: UML 2.0 [OMG]

Changing Views

Traditional view
  ➔ Operations and data are separate entities
  ➔ Data is globally visible

OO view
  ➔ Operations + data = objects

 OO and the Semantic Gap

Semantic Gap

Object
Encapsulation
Message Passing
Classes
Inheritance
Polymorphism and Dynamic Binding
(Multiple Inheritance)

See for example [Booch 94] or [Meyer 97] for details.

Important OO Concepts
OO Philosophy

- OO programs are systems of communicating objects
- Objects have an internal state, behaviour, and identity
- Classes are templates for the creation of objects of the same type
- Similarities can be expressed by inheritance
- Binding depends on the dynamic type of objects

- Readability
- Extensibility
- Maintainability

The Context of (Object-Oriented) Development

Object-Oriented vs. Traditional Development

- Unified approach for analysis, design, and implementation
- Concepts in problem domain and solution are closer
- More “natural” concepts
- Promotes encapsulation, extensibility, and reuse

- Fuzzy borderline between analysis, design, and implementation
- More difficult to manage
- More complex component interrelationships
- Transition problems

Object-Oriented Software Development

- What is Object-Oriented Development
- Object-Oriented vs. Traditional Development
- An Object-Oriented Development Framework
- Phases, Activities, and Work Products
An Object-Oriented Development Framework

- Work product oriented and workbook centred
  - Focus on the production of work products
  - Hold and manage them in a central depository
- Iterative and incremental
  - Divide the system into increments
  - Evolve the system increment-wise
  - Iteratively rework previous increments
- Scenario-driven
  - Model externally visible system behaviour by scenarios/use cases
  - Enforce traceability through work products

Iterative and Incremental Development 1

- Waterfall model
  - Strongly based on phases
  - Stable documents
  - No feedback/rework
- Problems
  - Incomplete requirements
  - Rework is necessary
  - No early prototypes
- Solution
  - Develop a solution for growing subsets of requirements (⇒ increments)
  - Rework existing solutions (⇒ iterations)

Pre-iteration phase
Gather initial requirements, plan project, start analysis

Iterations
Planning, Analysis, Design, Coding, Testing, Assessment

⇒ Plan increment [P | A | D | C | T | A]
⇒ Execute plan [P | A | D | C | T | A]
⇒ Assess quality [P | A | D | C | T | A]

Post-iteration phase
System test, package, and ship

The Project Workbook

A project workbook is a “logical document containing all the work products of a project.”

A work product is a “concrete result of a planned project-related activity such as analysis or project management. Work products include items delivered to the customer and items used purely internally within a project.”

[OOTC 97, p 25]

See
http://<course homepage>/Workbook/wb_template.html
for a project workbook template, or
examples from former groups
for actual examples.
Iterative and Incremental Development

Requirements

Increment 1
Increment 2
Increment 3

Iterations/
increments

Time

[++]

Increment Purpose Activities

1 Get project started
Develop all planned work products for the Requirements Gathering and Project management phases. Review Analysis work products with customers. Concurrently, analysts, designers, and implementers develop guidelines for their phases. Designers can also start the System Architecture, Target Environment, and Subsystem Model work products.

2 Familiarise team with process and environment
Start with a few Use Cases and develop all planned work products for the Analysis, (G)UI, Design, and Implementation phases.

3..n-1 Complete project development
Taking a few Use Cases at a time, develop all work products for each phase through Implementation. As new work products are developed, old ones may need to be extended or amended.

n Package, test, and deliver the product
Implement the Physical Packing Plan and conduct the Installation, System, and Acceptance test plans.

Modelling and Software Development

- Models make use of
  - Abstraction
  - Separation of concerns
  - Familiar and natural concepts

- Models are tools to
  - Understand a problem
  - Cope with complexity
  - Simplify problem solving
  - Enable traceability

Models Support Traceability

Real world

Software system

Model

Development

© by jubo@cs.umu.se

OOSD Copyright © by jubo@cs.umu.se
Examples of Models

◆ In the real world
  □ Maps
  □ Floor plans
  □ Diagrams
  □ ...
◆ In the computing sciences
  □ (Programming) Languages
  □ (Database) Conceptual models
  □ Graphs (for various purposes)
  □ ...

→ Choose the right model for the right purpose

Models for (OO) Software Development

◆ Use cases
◆ Scenarios
◆ Class diagrams
◆ Object interaction diagrams
◆ State machines
◆ Formal software specifications
◆ ...

A Use Case for a Course Registration System

Register for courses
Description:
This use case is initiated by the student. It provides the capability to create, review, modify, and delete a course schedule for a specialized semester. All required billing information is sent to the Billing System.

Actors:
Student, Billing System.

Notes:
A student can register for at most 4 courses each semester.
...

A Use Case (cont.)

Register for courses
Main Flow of Events:
1. The student identifies himself/ herself.
2. The system verifies student identity.
3. The student selects a valid semester.
4. The student creates, reviews, or changes a schedule.
5. The system sends billing information to the Billing System.
6. The system sends billing information to the Billing System.
A Class Diagram

Person (abstract)
- name
- p_number

Registrar
- registerStudent()
- addCourse()

Student
- credits

Professor
- research_area

An Object Interaction Diagram

Registrar
- add a course

Registrar
- registerStudent()
- addCourse()

Course Maintenance Form
- enter id
- verify id
- create course
- enter course info
- submit
- create
- close form
- save
- close form

OO Development

Requirements Gathering

Class Descriptions
- Object-Oriented Analysis & Design

Coding
- Implementation
- Test Case Development
- Test Cases

Use Case Based OO Development

Requirements Gathering

Class Descriptions
- Object-Oriented Analysis & Design

Coding
- Implementation
- Test Case Development
- Test Cases