Open Issues & Course Summary

CS3 / SEOC1

Note 18

Is software engineering with objects and components a good way of building good systems?

Software engineering, UML modelling, Components and Quality

- Software development process:
  - lifecycle models and main stages
  - process management
  - testing
  - maintenance and evolution
- Introduction to UML Diagrams:
  - use cases
  - class models
  - CRC cards
  - interaction diagrams
  - activity diagrams
  - state diagrams
  - implementation diagrams
- Reuse and components
- Dependable computer-based systems

Software Engineering with Objects and Components *(Notes 1 & 2)*

- Why are we doing this?
  - to build "good systems"
    - *what are good systems?*
    - *why do we need them?*
- Why a unified language?
- A unified language should be (and UML is?):
  - expressive,
  - easy to use,
  - unambiguous,
  - tool support,
  - widely used

SEOC cont. *(Note 6 and Using UML: chap.4)*

- Development process:
  - risk management is central
  - iteration to control risk
  - architecture-centric and component-based
- (Unified?) design methodology
  - *Pros:* dependable, assessment, standards
  - *Cons:* constraints, overheads, generality
  - Unified *modelling language* combines *pros* while avoiding *cons*
- The unified process: inception, elaboration, construction, transition
- other processes: Catalysis, OPEN, Extreme Programming, DSDM, SSADM, . . .

UML: status and issues

- History: Quest for open specification

  1989–1994 OO “method wars”

  1994–1995 three Amigos and birth of UML

  Oct 1996 feedback invited on UML 0.9

  Jan 1997 UML 1.0 submitted as RFP (Request for Proposal) to OMG (Object Management Group)

  Jun 1999 UML 1.3 released

  Sep 2000 (some) UML 2.0 RFP’s submitted

  Feb 2001 UML 1.4 draft specification released; further UML 2.0 RFP’s planned

- Open issues:
  - UML semantics
  - tool support
  - OCL (Object Constraint Language)

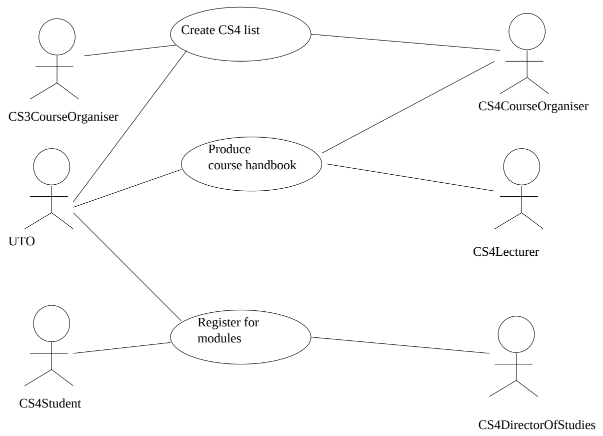
Case Study: CS4 Administration
Using UML: chap.15

At the end of the academic year, the HoD allocates teaching duties for CS4 modules. Every lecturer updates the course handbook entry for their module The CS4 coordinator updates other parts of the handbook and checks lecturers entries (all in LaTeX). The UTO produces paper versions; the CS4 coordinator produces HTML versions. The CS3 coordinator gives list of students to UTO and CS4 coordinator. Students are advised by DoS. Students provisionally register for modules with UTO. The UTO produces lists of attendees for lecturers.

Requirements Capture *(Note 3)*

- Users have different potentially conflicting views of the system.
- Users can’t express requirements clearly
  - missing information,
  - superfluous and redundant information,
  - inaccurate information...
- Users are poor at imagining what a system will be like.
- Identifying all the work needing support by the system is difficult.

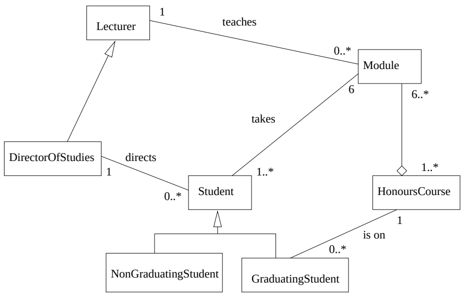
CS4: use case model
Using UML: Figure 15.1



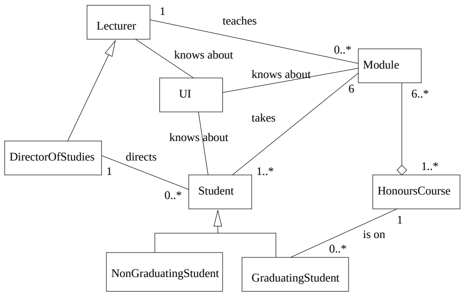
Static Structure *(Note 4)*

- desirable to build system quickly and cheaply
- desirable to make system easy to maintain and modify
- Identifying classes:
  - Data driven design
  - Responsibility driven design
  - Use case driven design
  - Design by contract
- class diagrams document: classes (attributes, operations) and associations (multiplicities, generalisations)
- system *is* some collection of objects in class model

A class model
Using UML: Figure 15.2



Another class model
Using UML: Figure 15.3



Validating the Class Model *(Note 5)*

- CRC cards: class, responsibility, collaborators
- UML interaction diagrams
- CRC cards and quality
  - too many responsibilities implies low cohesion
  - too many collaborators implies high coupling
- CRC cards used to:
  - validate class model, using role play
  - record changes
  - identify opportunities to refactor

CRC cards needed for `Produce course handbook`
Using UML: Figure 15.4

Class name: HonoursCourse	
Responsibilities	Collaborators
Keep collection of modules Generate course handbook text	Module

Class name: DirectorOfStudies	
Responsibilities	Collaborators
Provide human DoSs' interface to the system	

Class name: Module	
Responsibilities	Collaborators
Keep description of course Keep Lecturer of course	

Interactions *(Notes 7 and 8)*

- collaboration and sequence diagrams
- document how classes realise use cases
- (thus) helps validate design
- other uses: design patterns, component use, packages
- instance *versus* generic;
  procedural *versus* concurrent
- Law of Demeter
- creation and deletion of objects
- timing

Other UML Diagrams...

- Describing object behaviour
  - State diagrams *(Note 10)*
  - Activity diagrams *(Note 11)*
- Implementation diagrams *(Note 14)*
  - Component diagrams
  - Deployment diagrams

Other S/E Issues

- Testing *(Note 9)*
  - testing strategies:
    * top-down versus bottom-up
    * black-box versus glass-box
    * stress testing
  - categories (unit, integration, acceptance)
  - regression testing
  - test plans
  - OO and component issues
- Reuse and Components *(Notes 12 and 13)*:
  - Types of reuse
    * knowledge (artifact, patterns)
    * software (code, inheritance, template, component, framework)
  - success stories, pitfalls and difficulties with (component) reuse
  - reuse not free and requires management

What else did we do?

- Maintenance and Evolution *(Note 15)*:
  - accounts for significant part of project costs and developer effort
  - types: corrective, adaptive, perfective, preventive
  - ... is hard, requires management, ...
  - dealing with legacy code:
    * redevelopment, transformation (restructure, re-engineer, recapture), encapsulation
- High Dependability Engineering *(Notes 16 and 17)*
  - lots of scary stories...
  - software engineering *borrows heavily* from traditional engineering
  - but... software is significantly different:
    * focus on process
    * more complex and less “visible”
    * fails in different ways