Software Engineering with Objects and Components

Open Issues and Course Summary

Massimo Felici

Software Engineering with Objects and Components

- Software development process
  - Lifecycle models and main stages
  - Process management
  - Testing
  - Maintenance and Evolution
- Introduction to UML Diagrams
  - Use cases
  - Class diagrams
  - CRC cards
  - Interaction diagrams
  - Activity and Statechart diagrams
- Reuse and components

Software Engineering with Objects and Components

- Is software engineering with objects and components a good way of building systems?
- 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 supported
- Widely used

Development Processes

- Development process
  - Architecture-centric and component-based
  - Iteration to control risk
  - Risk management is central
- (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
  - There are many other processes (e.g., Spiral, Extreme Programming, etc.)

UML

- 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 RFPs submitted
- Feb 2001 UML 1.4 draft specification released
- UML 1.5;
- Current Version: UML 2.0. adopted in late 2003

UML

- UML semantics
- Tool support
- OCL (Object Constraint Language)

UML

- **Nested Classifiers:** In UML, almost every model building block you work with (classes, objects, components, behaviours such as activities and state machines, and more) is a classifier. In UML 2.0, you can nest a set of classes inside the component that manages them, or embed a behavior (such as a state machine) inside the class or component that implements it.
- **Improved Behavioural Modeling:** In UML 1.X, the different behavioural models were independent, but in UML 2.0, they all derive from a fundamental definition of a behaviour (except for the Use Case, which is subtly different but still participates in the new organisation).
- **Improved relationship between Structural and Behavioural Models:** UML 2.0 lets you designate that a behaviour represented by (for example) a State Machine or Sequence Diagram is the behaviour of a class or a component.

Requirements Capture

- Users have different potentially conflicting views of the system
- Users usually fail to 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

Static Structures

- 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

Validating the Class Model

- CRC Cards: class, responsibility and 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

Interactions

- Sequence and Communication diagrams – documents how classes realise use cases, thus, help to validate design
- Other uses: design patterns, component use, packages
- Instance versus generic
- Procedural versus concurrent
- Law of Demeter
- Creation and deletion of objects
- timing

UML

- Describing object behaviour
  - Activity diagrams
  - State diagrams
- Implementation diagrams
  - Package Diagrams
  - Composite Structures
  - Component Diagrams
  - Deployment Diagrams

Testing

- 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

- Type of reuse: Knowledge (artefacts, patterns), software (code, inheritance, template, component, framework)
- success stories, pitfalls and difficulties with (component) reuse
- Reuse not free and requires management

Software Engineering with Objects and Components

- Lecture Notes
  - 14 Lecture Notes
  - Industry Presentation and Collaboration
- Practicals
  - Requirements gathering, UML Design and Java Implementation
  - Group project
  - Different teams in each tutorial group
  - Tutorials
- Resources
  - References complementing and extending lecture notes
  - Main Tools: Eclipse and UML2 plugin

