

Software Engineering with Objects and Components



Open Issues and Course Summary

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

Software Engineering with Objects and Components

- Is software engineering with objects and components a good way of building systems?
- 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
 - State diagrams
 - Implementation diagrams
- Reuse and components
- Dependable systems

Software Engineering with Objects and Components

- 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

Software Engineering with Objects and Components

■ Development process

- Risk management is central
- Iteration to control risk
- Architecture-centric and component-based

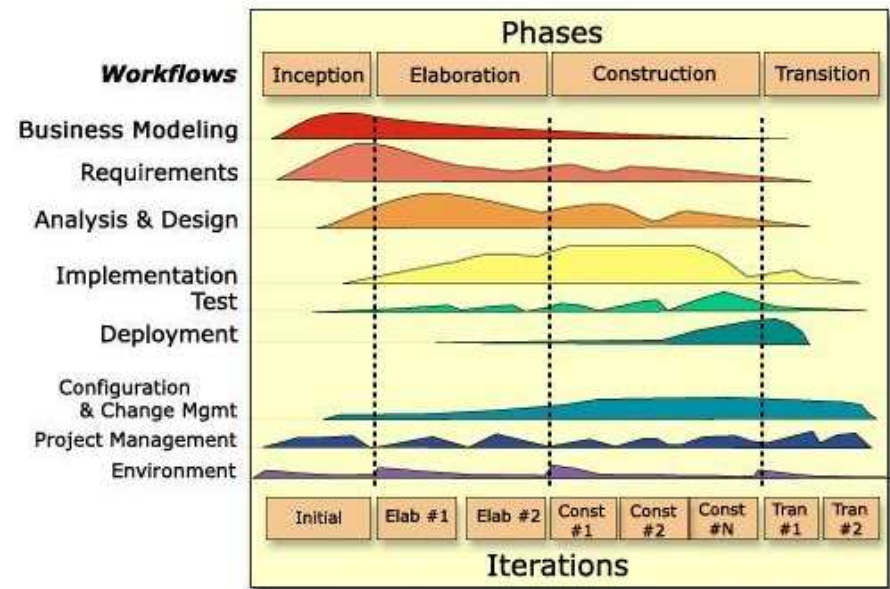
■ (Unified?) design methodology

- **Pros:** dependable, assessment, standards
- **Cons:** constraints, overheads, generality
- Unified modelling language combines pros while avoiding cons

■ The unified process

- **Inception, Elaboration, Construction, Transition**
- There are many other processes (e.g., Spiral, Extreme Programming, etc.)

(Rational) Unified Process - RUP



UML: Status and Issues

History:

- 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
- **UML 1.5**;
- Current Version: **UML 2.0**. adopted in late 2003

Open issues

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

What's new in UML 2.0

- **Nested Classifiers:** In UML, almost every model building block you work with (classes, objects, components, behaviors 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 Behavioral Modeling:** In UML 1.X, the different behavioral models were independent, but in UML 2.0, they all derive from a fundamental definition of a behavior (except for the Use Case, which is subtly different but still participates in the new organization).
- **Improved relationship between Structural and Behavioral Models:** UML 2.0 lets you designate that a behavior represented by (for example) a State Machine or Sequence Diagram is the behavior 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

- **Collaboration and sequence diagrams**
 - documents how classes realize 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



Other UML Diagrams...

- Describing object behaviour
 - State diagrams
 - Activity diagrams
- Implementation diagrams
 - Component diagrams
 - Deployment diagrams



Other Software Engineering Issues

■ 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



What else did SEOC cover?

■ Maintenance and Evolution

- Accounts for significant part of project costs and developer effort
- Types: corrective, adaptive, perfective, preventive
- ...is hard, requires management,...
- Dealing with legacy code: redevelop, transform (restructure, re-engineer, recapture), encapsulate

■ Software Quality

■ High Dependability Engineering

- Lots of scary stories...
- Software engineering borrows heavily from traditional engineering
- Although software is significantly different
 - Focus on process rather than product
 - More complex and less visible
 - Fails in different ways
 - Is far more subject to change



Lecture Notes

- Lecture Note 01 - SEOC1 Overview
- Lecture Note 02 - Requirements Engineering
- Lecture Note 03 - Use Cases
- Lecture Note 04 - Software Design
- Lecture Note 05 - Class Diagrams
- Lecture Note 06 - CRC Cards
- Lecture Note 07 - Project Management
- Lecture Note 08 - Collaboration Diagrams
- Lecture Note 09 - Sequence Diagrams
- Lecture Note 10 - Activity Diagrams
- Lecture Note 11 - Statechart Diagrams
- Lecture Note 12 - Implementation Diagrams: Component and Deployment Diagrams
- Lecture Note 13 - Software Construction
- Lecture Note 14 - Software Testing
- Lecture Note 15 - Software Maintenance and Evolution
- Lecture Note 16 - Reuse and Components
- Lecture Note 17 - Software Quality
- Lecture Note 18 - Engineering High-Dependability Systems
- Lecture Note 19 - The End



Software Engineering Any Magic/Silver Bullet?

See (in the resource web page)
a (kind of historical) list of papers on
"Software Engineering Bullets"