Software Engineering with Objects and Components 1 Overview

Corin Gurr
Human Communication Research Centre
Room 2/4R5, 2 Buccleuch Place
Tel: 0131 650 4448
Email: C.Gurr@ed.ac.uk

Note 1

Administration

Course Web Page please read it it is the main way of making information on the course available:
`http://www.dcs.ed.ac.uk/teaching/cs3/seoc1/module-page.html`

Assessment: Coursework counts 25% towards the overall assessment. Remaining 75% is derived from the degree examination.

Coursework: Tutorial groups collaborate on a single project throughout the term. Assessed work consists of two deliverables for this project, each worth 50% of the coursework mark.

Software: We use Java and Argo/UML on the course. Details of how to use these will be posted on the web page.

Main Themes

Software Engineering *is concerned with processes, techniques and tools which enable us to build "good" systems*

Object-Orientation *is a (paradigm, methodology, technique, process, suite of design and programming languages and tools) with which we may build good systems*

Components *are "units of reuse and replacement"*

1. Software Dependability
2. Software Engineering Process
3. OO and Component Based Development
4. Unified Modelling Language (UML)
5. Group Development Project

London Ambulance Service: computerised despatch system *What happened?*

- 7am, Monday 26 Oct. 1992, system goes live
- system quickly overloaded, logged calls appear to get "lost"
- callers held in call-queueing for up to 30 minutes
- ambulance allocation system fails to recognise certain roads (despatch staff revert to maps)
- by Monday night: system swamped, new calls overwrite existing calls
- system generates exception messages; these subsequently also swamp staff and system
- exception messages now clog up system, at one point all manually deleted
- claims made later of up to 20–30 deaths

LAS: world's largest ambulance service

- 600 square miles, 7 million residents (far more during working hours)
- typical day: 5000 patients, 2-2,500 calls (1,600 A & E)
- over 300 A & E ambulances, 500,000 patient journeys per year
- over 400 Patient Transport ambulances, 1.3 million patient journeys per year
- from receiving call to despatching ambulance – less than 3 minutes

LASCAD: Development history

1987 First computerisation project commences, £3 million budget

1990 Project abandoned, cost estimated at £7.5 million

1990-1991 New senior management team appointed: requirements completed Feb. 1991; specification July 1991; partial system goes live Jan. 1992; piecemeal implementation across LAS divisions Jan–Sept 1992

Oct. 1992 System changeover

- **26 Oct.** system goes live
- **27 Oct.** system closed down
- **28 Oct** reverts to semi-manual operation

Nov. 1993 system crashes, fallback routines fail to operate; system closed down; reverts to entirely manual operation.

LAS despatch system: causes of failure

- ***Every mistake in the book.***
- Primary causes:
  - system design: idealised world view
  - management ethos
  - procurement process
  - development timetable
- Secondary causes
  - inexperience of suppliers
  - inadequate testing
  - poor quality assurance
  - poor training
  - inadequate project management

What about coding?

- R. Lutz, 1994: cause and effect analysis of Voyager and Galileo mission software
  - over 40,000 SLOC
  - 387 software faults
  - roughly half safety related
- interface faults (roughly 25%)
  - safety-related – primarily intra-team communication errors
  - non-safety related – even distribution of causes
- functional faults (roughly 75%)
  - safety-related – primarily problems in requirements recognition (understanding)
  - non-safety related – requirements implementation errors
- **Moral:** we can get the coding right *when we're careful*, the greatest problems lie elsewhere

The need for software engineering

- Write code $\checkmark$ – Right code?
- Greatest problems lie elsewhere in development & maintenance process
  - intangible, hard to determine what's "good"
  - process requires "good" management
  - communication amongst large, diverse teams is problematic
- software engineering, OO (including UML), and components attempt to address this
- Do they succeed? If so, how well?
- Recommended Reading: Flowers, Stephen. "Software failure: management failure". John Wiley and Sons, 1996.

What is an Object?

A Thing: objects represent physical and conceptual things that appear in the system being modelled. To implement all of the kinds of systems we want to we need conceptual things.

Has State: Usually objects have attributes that can change throughout the lifetime of an object. E.g. the attribute *weight* of a *patient* object in a medical information system.

Has Behaviour: Understands some set of messages that can be sent to it and collaborates with other objects by sending them messages.

Has Identity: Is more than just the collection of attributes. You can have two non-equal objects with identical attribute values. Objects can often self refer (e.g. by sending themselves messages).

Classes

- a *Class* defines a family of objects that all take similar roles in a system.
- In Java every object is a member of a class
- Corresponding to every kind of message understood by all objects of the class the class defines a method of responding to the message on the basis of the selector and list of arguments.
- The class determines the attributes of the system.

Components and Reuse

- Modularisation required to manage large bodies of code
  - OO invented to support this
  - ... but was not quite the right level
  - Components are "higher-level" modules
- Reuse most clearly illustrates need for, and success of, modularisation
- (e.g.) NASA SEL (Software Engineering Laboratory)
  - 1992–1995, developed library of reusable components, leading to:
  - routinely 75% or higher reuse
  - development time reduced by 90%
    * 58,000 hours for application development,
    * recently, with reuse, reduced to approx. 6,000

Fundamental Issues/Assumptions

- Why object oriented software engineering?

  For natural, intuitive, paradigm leads to good design, widely used

  Against not proven! (not provable?)

- Why any development methodology?

  Pros dependable, assessment, standardisation

  Cons stifles innovation & creativity, overheads, too general

Group Development Project

- Tutorial/Practical work
- Collaborative specification and design
- (Human) communication in design process
- Illustrates commercial software engineering projects
- Permits (subjective) assessment of:
  Is software engineering with objects and components a good way of building good systems?

General software engineering books

Reference *Software Design* by David Budgen, Addison-Wesley 1993, ISBN 0-201-54403-2

Reference *Software Engineering - A practitioner's approach* by Roger S. Pressman, European edition adapted by Darrel Ince, McGraw-Hill, ISBN 0-07-707936-1

Reference *Software Engineering, 5th Edition* by Ian Sommerville, Addison-Wesley, ISBN 0-201-56529-3

Object oriented methods

Purchase *Using UML*, by Perdita Stevens and Rob Pooley, Addison-Wesley, ISBN 0-201-36067-5

Reference *Object Oriented Analysis and Design with Applications, 2nd Edition* by Grady Booch, Benjamin-Cummings, ISBN 0-8053-5340-2,

Object oriented methods (cont.)

Reference *Object Oriented Modelling and Design* by Rumbaugh, Blaha, Premerlani, Eddy and Lorenson, Prentice-Hall, 0-13-630054-5

Reference *Object Oriented Software Engineering: A use case approach* by Ivar Jacobsen, Addison-Wesley 1994

Reference *UML Distilled* by Martin Fowler, Addison-Wesley, 1997

Background *Object-Oriented Software Construction* by Bertrand Meyer, Prentice-Hall, ISBN 0-13-629049-3

Background *Object Oriented Programming* by Coad and Nicola, Yourdon Press

Background *Principles of Object oriented Software Development* by Anton Eliëns, Addison-Wesley, ISBN 0-201-62444-3