SEOC2 Overview

In this lecture we work through the topics of the module and identify what I consider to be the most important points to remember.

These follow the sequence of groups of lectures.

Lecture slides are on the SEOC2 Web page.

You also can find a guide to background reading on the SEOC2 Web page.

In this lecture I will recommend reading, although the lecture slides, along with the experience gained from your practical, give all of the basic material required to prepare for your exam.

Architectures

You know several basic forms of architecture for different categories of system:

High level decompositions : *e.g.* Layers

Distributed systems : *e.g.* Broker

Interactive systems : *e.g.*
Model-view-controller

Adaptable systems : *e.g.* Reflection

Book: Buschmann *et.al.* 1996, *A System of Patterns*, Chapter 2
Web: `hillside.net/patterns/` is one of many patterns sites

Methodologies

You know the essentials of two contrasting forms of development methodology:

The Unified Process : A highly controlled design method consisting of waterfall iterations within phases.

Extreme Programming : A more adaptive style of design relies on tight design cycles and configuration management.

Book: Jacobson, Booch and Rumbaugh 1998 *The Unified Software Development Process*, Chapter 1
Web:
`www.rational.com/products/rup/index.jsp` describes the UP according to Rational Software
Web: `www.extremeprogramming.org/` gives an introduction to XP

Open source

You know several ways in which successful open-source development efforts have been structured:

Benevolent dictatorship : *e.g.* Linux

Open committee : *e.g.* Apache

Ring-fenced committee : *e.g.* Mozilla

Web: `www.opensource.org/` is the Open source initiative's site

Measurement

You know the sorts of things to include in a software measurement plan. In particular, you know:

Some key issues to address : *e.g.* Growth measures.

Means of identifying issues : *e.g.* Risk assessments.

Limitations of measurement : *e.g.* Incremental design means measuring incomplete functions.

Basic estimators : *e.g.* Plot of staff months against number of lines of source code.

Book: Humphrey 1995 *A Discipline for Software Engineering* Chapter 4
Web:
`irb.cs.uni-magdeburg.de/sw-eng/us/index.shtml`
is the Software Measurement Laboratory in Denmark

Software size

You know several methods for estimating software size:

Consensus methods : *e.g.* Delphi

Population data methods : *e.g.* Fuzzy

Standard component methods : *e.g.* Component estimating

Function based methods : *e.g.* Function point analysis

Book: Humphrey 1995 *A Discipline for Software Engineering* Chapter 5
Web: `sern.ucalgary.ca/courses/seng/621/W97/couprie/size.htm` gives a summary of size estimation

Risk reduction patterns

You know how different aspects of projects can create different risks to project success and, for each aspect, you know ways of reducing the risk:

Knowledge inadequacies : *e.g.* Prototype

Teaming : *e.g.* Holistic diversity

Productivity : *e.g.* Gold rush

Ownership : *e.g.* Owner per deliverable

Distractions : *e.g.* Team per task

Training : *e.g.* Day care

Web:
`members.aol.com/acockburn/riskcata/riskbook.htm`
summarises Cockburn's risk patterns

Verification and validation

You know several techniques for V & V:

Testing : *e.g.* Black/white box testing

Proof of correctness : *e.g.* Temporal logic

You also know how V & V extends beyond individual models:

- Integration testing
- System testing
- Regression testing

Book: Sommerville 1996 *Software Engineering* Chapters 22, 23 and 24

Web: `www.ivv.nasa.gov` is NASA's Independent Verification and Validation Facility

Standards

You know the basics of software standards:

Why they are useful : *e.g.* Repeatability of process

Their legal implications : *e.g.* Fault attribution

The levels at which standards apply : *e.g.* Principles v element standards.

Key organisations producing standards : *e.g.* IEEE, ISO

Examples of standards in key areas : *e.g.* Systems engineering standards

Economics of quality

You know some of the factors involved in balancing quality against cost:

The means of quality control : *e.g.* inspection v testing

Quality over lifecycles : *e.g.* cumulative quality improvement

Key quality parameters : *e.g.* defect injection v defect reduction

algorithmic cost models : *e.g.* COCOMO

Book: Sommerville 1996 *Software Engineering* Chapter 29

Web: `directory.google.com/Top/Computers/Software/Software_Engineering/Quality_Assurance/` is a Google index on quality assurance

Automated software engineering

You know several ways in which to introduce automation into software design:

Automating component-based design : *e.g.* Web site synthesis

Linking components to rationale : *e.g.* Safety shutdown system design

Automating traceability : *e.g.* Augmented System Dynamics models

Connecting to established lifecycles : *e.g.* BSDM

Book: Robertson & Agusti 1999 *Software Blueprints* Chapters 4 and 5
Web: `ase.arc.nasa.gov/` is NASA's Automated Software Engineering Group

Software failures

You know about the sorts of pathological problems which can occur on large and small projects:

Documented failures in large projects : *e.g.* Deadline fixation

Misconceptions leading to failure : *e.g.* Timing of benefits

Web: `www.scs.carleton.ca/~beau/PM/Standish-Report.html` is a summary of the 1995 Standish Group report

Web: `catless.ncl.ac.uk/Risks/` is the Risks Digest