Semester Summary

Dr. James A. Bednar
jbednar@inf.ed.ac.uk
http://homepages.inf.ed.ac.uk/jbednar

SEOC2 Overview

In this lecture we review the topics we have covered this semester, focusing on what I consider the most important points to remember.

The lecture slides on each topic, coupled with the required readings as distributed in class or listed at the end of some lectures, contain all of the basic material required to prepare for the exam.

The background readings listed on the course web page, plus experience gained during the practical assignments, will help you achieve excellent, not just satisfactory, results.

Design Patterns

You should know what a design pattern is, how to use them, why they are useful for large teams, and several example patterns (e.g. Composite and Proxy).

Book: Gamma et al. 1995

Web: Search for “design patterns”, etc.

Architectural Patterns

You should know what a high-level architectural pattern is, and how to use and apply several high-level architectural patterns suitable for different types of systems:

High level decompositions: e.g. Layers

Distributed systems: e.g. Broker

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

Adaptable systems: e.g. Reflection

Configurable systems: e.g. Scripted Components

Book: Buschmann *et al.* 1996, *A System of Patterns*, Chapter 2

Scripting Reusable Components

You should know why reuse is difficult and rare, some properties that make some languages more suitable for building components and others for gluing them together, and how the Scripted Components pattern facilitates component reuse.

Article: Ousterhout 1998

Web: `http://www.doc.ic.ac.uk/~np2/patterns/scripting/scripting.html`

Methodologies (1)

You should know the essentials of at least four development methodologies, including their strengths, disadvantages, and basic tenets:

The Waterfall Model
The Unified Process (UP)
Extreme Programming (XP)
The Cleanroom Process

Only basic knowledge is expected for the Waterfall and Cleanroom processes; we covered the other two in some detail.

Methodologies (2)

Book: Jacobson, Booch and Rumbaugh 1998 *The Unified Software Development Process*, Chapter 1

Web:
`www-306.ibm.com/software/awdtools/rup`
The UP according to IBM/Rational Software

Web: `www.extremeprogramming.org` gives an introduction to XP

Open source

You should know the assumptions behind open-source and closed-source approaches, the advantages and disadvantages of each, and 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/`: OSI site

Web: `http://www.catb.org/~esr/writings/cathedral-bazaar/`

Article: Mockus et al. 2002

Measurement

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

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

Means of identifying issues: e.g. Risk assessments

What to measure for various issues: e.g. Number of components

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 2002 chapter 4

Estimating size and effort

You should 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

And how COCOMO can be used to estimate effort, given the size.

Book: Humphrey 2002 chapter 5

Web: `sunset.usc.edu/research/COCOMOII`: COCOMO site

Verification and validation

You should know the difference between verification and validation, why both are important, and the basics, pros, and cons for several techniques for V & V, e.g. black/clear box testing and formal proofs of correctness. You also know the different levels at which V & V is applied, and some ways to do tests at each level:

- Unit tests
- Integration testing
- System testing
- Regression testing

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

SW Project Management Tools

You should know several categories of useful tools, and have some familiarity with at least one suitable tool in each category, particularly the first two:

Build control (e.g. **make**)
Revision control (e.g. **CVS**)
Unit/regression testing (e.g. **JUnit**)
Bug/issue tracking (e.g. **GNATS**)
Documentation generation (e.g. **JavaDoc**)
Integrated suites (e.g. **RUP**)

Web: See tools.html on the course web page

Risk reduction patterns

You should be able to analyze some risks faced by particular projects and organizations, including how to reduce them and how to tell when too much correction has been applied:

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`: Cockburn's risk patterns

Economics of quality

You should know some of the factors involved in developing high-quality software:

Tradeoffs: cost/benefit, feature/bug

Differences between achieving quality through inspection or through testing

How to model quality improvements

Book: Sommerville 1996 *Software Engineering* Chapter 29

Standards

You should have a basic familiarity with software standards:

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

Their legal implications: e.g. Fault attribution

Key organizations producing standards: e.g. IEEE, ISO

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

You will not need to memorize any individual standards.

Book: Moore 1998 *Software engineering Standards*

Software failures

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

Organization problems: e.g. Poor reporting structures
Management problems: e.g. Political pressures
Problems conducting the project at each phase: e.g. being technology focused in the initial phase

Book: Flowers 1996

Web: `www.cs.nmt.edu/~cs328/reading/Standish.pdf`: Summary of the 1995 Standish Group report

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

Exam preparation

Old exams are on inf.ed.ac.uk, but remember that the content of the course is slightly different every year.

Compared to recent years, I put less emphasis on formal methods, automation/synthesis, and agents. Compared to much older versions of the course (2000 and earlier), this year had no prerequisite for knowing details of UML beyond use case diagrams.

In general, just review the lecture slides and required reading, following up with your own web exploration or other books wherever your interests take you.

Summary

- Large-scale, long-term software development is extremely difficult and unpredictable
- In SEOC2 you have been exposed to some useful approaches and tools
- These approaches and tools can help, but are not guaranteed cures
- Always be on the lookout for risks and indications that your project is headed for failure, so that you can address the issues or abort the project when appropriate.
- Good luck beating the odds!

References

Flowers, S. (1996). *Software Failure: Management Failure: Amazing Stories and Cautionary Tales*. Reading, MA: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

Humphrey, W. S. (2002). *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley.

Mockus, A., Fielding, R., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM*

Transactions on Software Engineering and Methodology, *11* (3), 309–346.

Ousterhout, J. K. (1998). Scripting: Higher level programming for the 21st century. *Computer*, *31* (3), 23–30.