Software Architecture, Process, and Management

Summary

Massimo Felici
Software Architecture, Process, and Management

- In this lecture we review the topics we have covered this semester, focusing on what we suggest are 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 surpass this minimum standard.
Project management

• You should know what projects are, and how to draw and interpret the basic charts used in PM: Work Breakdown Structure, Network diagram (and that it is sometimes called a PERT or CPM chart), and Gantt chart.

• You should know the basic PM terms, such as critical path, slack, crashing.
**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*)
- Debuggers (e.g. *gdb*)
- Testing (e.g. *JUnit*, *FIT*)
- Bug/issue tracking (e.g. *BugZilla*)
- Documentation generation (e.g. *JavaDoc*)
- Project management (e.g. *MS Project*)
- Integrated suites (e.g. *RUP*)
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.
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.
Design Patterns

You should know what design patterns are, how to use them, why they are useful for large teams, and the basic properties of several example patterns (e.g. Composite and Proxy).
**Architectural Patterns**

You should know what a high-level architectural pattern is, and how to use and apply several specific high-level 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. Microkernel
- **Configurable systems**: e.g. Scripted Components
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.
Methodologies

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

- The Waterfall Model

- The Unified Process (UP)

- Extreme Programming (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
Software failures

You should know about many of the typical pathological problems which occur on large projects, and have looked at a few small case studies:

- Organisation 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
Risk management

You should be able to analyse typical risks faced by particular projects and organisations, 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
Software Evolution

• **Revision control:** You should know why revision control is important, how it works, how basic operations are specified using Subversion, and how revision control works with software releases.

• **Refactoring:** You should know what refactoring is, and how to do it (alone and with testing and revision control)

• **Legacy code:** You should know the main techniques for dealing with legacy code (refactoring and encapsulating), and why it is worthwhile to keep running systems working even as they are changed
People and Groups

You should be able to discuss intelligently the role of human factors in software development, including:

- How communication limits team size, due to quadratic scaling in communication requirements
- The effect of distributed work sites
- Programmer motivations, incentives, and rewards
- Resistance to reuse, formal methods
- Effect of users with fuzzy, drifting requirements
Exam Preparation

- **Past Examination Papers** are on [http://www.inf.ed.ac.uk/teaching/exam_papers/](http://www.inf.ed.ac.uk/teaching/exam_papers/), but remember that the content of the course is slightly different every year.

- This year is largely similar to the last two years. Compared to 2005–6, there is more material on project management, change management, and legacy systems, and less emphasis on formal modelling of quality tradeoffs and programming standards.

- In general, review the lecture slides, the required reading, and the work you did in your projects, 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 SAPM 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.
Feedback

Please fill in the feedback questionnaire:

http://www.inf.ed.ac.uk/cgi-bin/ito/course_questionnaire.pl