Software Engineering with Objects and Components 2

Dr. James A. Bednar

jbednar@inf.ed.ac.uk

http://homepages.inf.ed.ac.uk/jbednar

Welcome to SEOC2!

SEOC2 is about large-scale, long-term software development projects.

For the purposes of this course, a software project will be considered successful if:

- The software is delivered on time
- Development costs were within budget
- The software meets the needs of users

Reality of large projects

I do not personally know of *any* large-scale commercial software project that is a success by those criteria.

Instead, failure is ubiquitous (Standish Report, 1994):

- 91% of projects at large companies failed
- 30% of projects at large companies were eventually cancelled

Rates vary dramatically for other samples and criteria, but none so far have suggested that a majority of large projects succeed. In fact, most projects fail in multiple ways (schedule, budget, *and* features).

Reducing the risk of failure

Since at least the 1960s, a variety of software engineering techniques have been designed to reduce the risk of failure, albeit with only modest success so far.

This course will survey the approaches available, what their limitations are, and the prospects for future improvement.

The approaches complement those learned in typical CS courses, which focus on small-scale, short-lived systems. Most of the issues are quite high level, not technical, and depend on how human beings behave rather than on how to manipulate bits.

Links to other fields

Psychology Programmer motivation, users’ needs

Sociology Programmer group dynamics

Business Marketplaces, business models, business practices, management

Economics Command-and-control development vs. free market

Philosophy Software as a model/representation of reality

Managing creativity

Underlying problem: Programming is a highly creative process, with an enormous space of flexibility

- Can that flexibility be controlled, measured, predicted without stifling creativity?
- How does one manage artists? (Herding cats...)
- Would a large organization be able to manage the writing of a good epic novel, on schedule and to budget?

Dealing with flexibility

Another underlying problem: because of the large space of flexibility, it is possible to spend an infinite amount of time on most of the subtasks of software development

- How do we ensure we are solving the right problem? I.e., how can we tell whether the project will meet the needs of users?
- How do we decide how well to solve each subproblem?
- How do we rank non-functional requirements like reliability, scalability, security, elegance?

My background

- Primarily academic
- Commercial experience: National Instruments, Shell
- Primarily UNIX, rarely Windows
- Largest project managed to date:
  - Research simulator developed over eight years
  - 35 KLOC, much of which should never have been written
- Current project: 3 year grant, managing 5 part-time programmers

Applying SE in the real world

Every job is different. Different companies, projects will:

- use vastly different tools, platforms, development processes
- have different cultures, expectations, environments

General problem-solving and management skills transfer, but those are (nearly?) impossible to teach

Typical real-world scenario 1

When you arrive at a new job at a company, one likely possibility is to find a vague development process, loose or no standards, and a mishmash of incompatible tools.

You will need to be able to know when SE tools/approaches should be brought in, and candidate tools to use.

SEOC2 will help expose you to these tools and approaches.

Typical real-world scenario 2

You may instead get a job at a company with a well-defined development process and standards, and a common set of tools.
(Of course, the tools and processes are likely to be obsolete because they have been in use for many years.)

You will need to be able to work within their system, quickly learning the concepts specific to their setup.

SEOC2 will give you general experience with typical approaches, so that your specific situation can be understood more easily.

Typical real-world scenario 3

A third common employment situation is to be a member of a small team, perhaps just yourself, charged with building a web site or other small project.

Many different software development methods will work in such a case, including “hacking and heroics”.

However, following approaches surveyed in SEOC2 will help the system you create to be maintainable over the long term, not just for the month it was originally written.

What SEOC2 will not give you

- A recipe for success

  (impossible, despite many claims)

- A plug-and-play set of SE tools (quickly outdated)

- Knowledge of all areas of SE

- A magic ability to predict schedule, budget, and the needs of users

Relation to other courses

CS2: Most of the major SE concepts were introduced in CS2. These will be reviewed briefly in SEOC2, but those who have forgotten or have not taken CS2 should study the CS2 material on the course web page.

SEOC1: SEOC1 is not a prerequisite, but we will try not to duplicate material in SEOC1. In particular, this class is not based on UML, and focuses on abstract issues involving large-scale, long-term software development, rather than specific design problems.

Major topics

The syllabus lists the set of SE topics from which lectures and assignments will be drawn. The depth of coverage of each topic varies widely, in part to avoid overlap with other courses. Major topics include:

Project management: Software development processes, mitigating project risks

Design: Successful architectures for large systems

Techniques and tools: Refactoring, unit testing, bug tracking, regression testing, configuration management

People issues: How programming teams work

Assessed coursework

Previous semesters used a single large design practical. Because of licensing and other issues the assessed coursework this semester will consist of smaller exercises and reports. Due dates will be posted on the web page very soon, so please check the web site at least weekly.

In place of some of the exercises, Master’s-level students do a literature review term paper on a topic related to large-scale, long-term software development.

Summary

- Large-scale, long-term software development is extremely difficult and unpredictable
- SE approaches and tools can help, but are not guaranteed cures
- SEOC2 will help expose you to useful approaches and tools
- Many of the most important lessons cannot be put into an itemized list, and require experience with failures and successes