Software Architecture, Process, and Management

Massimo Felici
Acknowledgements:
James Bednar, Massimo Felici, Conrad Hughes, Perdita Stevens
Welcome to SAPM!

SAPM is about *large-scale, long-term* software development projects:

**Large scale:** Anything *substantially* larger than the types developed in your university courses or projects.

**Long term:** Development time is years, not weeks.

Our focus will be on how to make such projects *successful.*
Criteria for success

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

1. The software is delivered \textit{on schedule}

2. Development costs were \textit{within budget}

3. The software \textit{meets the needs of users} (in scope and quality)
Reality of large projects

- Software crisis?

- Rates of project failure vary dramatically with samples and criteria, but none so far have suggested that a majority of large projects succeed. In fact, most projects fail in multiple ways — e.g., schedule, budget, and functionality.

The most famous collector of statistics is the Standish CHAOS group.
Resolution of Projects

CHAOS 2004
Survey Results
Resolution of Projects

- Challenged: 53%
- Succeeded: 29%
- Failed: 18%

Copyright © 2006 The Standish Group International, Inc.
### Change over time?

<table>
<thead>
<tr>
<th>Year</th>
<th>Succeeded</th>
<th>Failed</th>
<th>Challenged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>16%</td>
<td>31%</td>
<td>53%</td>
</tr>
<tr>
<td>1996</td>
<td>27%</td>
<td>40%</td>
<td>33%</td>
</tr>
<tr>
<td>1998</td>
<td>26%</td>
<td>28%</td>
<td>46%</td>
</tr>
<tr>
<td>2000</td>
<td>28%</td>
<td>23%</td>
<td>49%</td>
</tr>
<tr>
<td>2002</td>
<td>34%</td>
<td>15%</td>
<td>51%</td>
</tr>
<tr>
<td>2004</td>
<td>29%</td>
<td>18%</td>
<td>53%</td>
</tr>
</tbody>
</table>
Techniques to avoid failure

- Since at least the 1960s, many techniques have been proposed with the hope of turning software development from an art or lottery into an engineering exercise.

- This course will survey the approaches available, their limitations, and the prospects for future improvement. Many approaches fall under Software Engineering (SE), but we will consider other factors also.

- The techniques complement those learned in typical Computer Science (CS) courses, which focus on small-scale, short-lived systems.
Reasons for Success

Top Ten Reasons for Success

- 1. User Involvement
- 2. Executive Management Support
- 3. Clear Business Objectives
- 4. Optimizing Scope
- 5. Agile Process
- 6. Project Manager Expertise
- 7. Financial Management
- 8. Skilled Resources
- 9. Formal Methodology
- 10. Standard Tools and Infrastructure
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 organisation 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?
- 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?
Course topics

- **Planning and Managing Software Projects**: project management, risk management, planning, project tools, working in a team

- **Software Development Processes**: heavyweight vs. lightweight methodologies

- **Software Architecture and Design**: successful architectures for large systems

The depth of coverage of each topic varies widely to avoid overlap with other courses. Most of these issues are quite high level, not technical, and depend on how human beings behave rather than on how to manipulate bits.
Survey

- How many have previously been employed on a large project?
- On a *successful* large project?
- How many of you expect to take jobs that involve software engineering?
- How many prefer Windows? Mac? GNU/Linux? Other?
- How many students have taken SEOC?
Applying SAPM 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.

• SAPM 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.

- SAPM 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.

- Nearly any software development method will work in such a case, including “hacking and heroics”.

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

- **Inf2C**: Many Software Engineering (SE) concepts were introduced in Inf2C. Those who have not taken Inf2C may wish to review the material on the Inf2C web page as background reading.

- **SEOC**: SEOC is not a prerequisite, but we will try not to duplicate much of the material in SEOC. In particular, SAPM is not based on UML, and focuses on abstract issues involving large-scale, long-term software development, rather than specific design problems or formal notations.
UG4 assessed coursework

• For **UG4 students**, the coursework consists of **two design and management exercises**, each worth **15% of the overall mark**.

• The coursework section of the web site has a long list of writing tips; you are strongly advised to read this now and then read it again before every assignment.

**Coursework deadlines**

Assignment 1: Friday, 2pm, 18th February 2011 (week 6)

Assignment 2: Friday, 2pm, 18th March 2011 (week 10)
MSc assessed coursework

- **MSc students** do the same assignment 1 as UG4 students, with the same deadline and weighting. However, instead of assignment 2, MSc students do a report worth 25%.

- The report will be a *critical analysis and evaluation* of work in an area listed in the course syllabus. Please note that the work must be *published* (usually in a journal or conference proceedings), and must have been *peer reviewed* (which one can usually assume journals and conference proceedings are).

- **Coursework deadlines**

  Assignment 1: Friday, 2pm, 18th February 2011 (week 6)

  Report: Friday, 2pm, 18th March 2011 (week 10)
Lectures

• Why come to class?

• Lectures are the essence of this course. There is no required textbook, relatively little reading, and the slides are relatively sparse. Very few of those who rarely attend do well.

• Without attending, it is hard to know which of the topics are most important, why some approaches are better than others, or how it all fits together. These issues are the focus for the exam, and should be your focus throughout.
Summary

• Large-scale, long-term software development is extremely difficult and unpredictable

• SAPM approaches and tools can help, but are not guaranteed cures

• This course will help expose you to useful approaches and tools

• Many of the most important lessons cannot be put into an itemised list, and require experience with failures and successes