Handout 15
Software Management I
With Markup

Steve Young
Contents

1. Software Engineering
2. Software Life Cycle
3. Team Organisation
4. Product Development
5. Specification and High Level Design
6. Project Management
7. Quality Control

Copies of these notes plus additional materials relating to this course can be found at:
http://mi.eng.cam.ac.uk/~sjy/teaching.html.
Software Engineering

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

Software typically evolves in 4 main stages:

**Specification** defines the functionality of the software and the constraints on its operation. This will usually involve interaction with the client in order to understand the requirements.

**Implementation** produces the software to meet the specification. This is the design and build phase.

**Validation** ensures that the software does what it is supposed to do. This will be a continuation of the testing done in the development phase, but will usually be more comprehensive and more formal. The end of this process is marked by the client agreeing contract completion.

**Evolution** adapts the software to meet changing customer needs. This includes maintenance, enhancements and re-design.

These phases constitute the software life cycle.
The Software Life Cycle

WaterFall Model

This view of the software life cycle is based on traditional engineering processes. It presents a good abstract view of what is involved but is somewhat idealistic.

Evolutionary Model

Real projects typically involve multiple iterations at each stage. Hence, the evolutionary model is a more pragmatic view of the software life cycle.
What can go wrong?

Project management\textbf{Risk high}
- poor cost estimation and scheduling
- Inadequate progress tracking
- Poor man-management

Specifications\textbf{Risk high}
- Failing to capture user requirements properly
- Specification was incomplete and/or imprecise

Design\textbf{Risk low}
- Poor choice of overall architecture
- Inconsistent module interfaces
- Code unreliable, slow, and crashes frequently.
- Confusing user interface

Testing\textbf{Risk medium}
- Inadequate

Documentation\textbf{Risk medium}
- Insufficient
- Badly written

End result is software which is delivered late, over-budget, runs too slowly, does not scale with increasing demand, unreliable and very difficult to use.
Types of Software Project

There are many types of software project:

- 1 person making software for personal use.
- 2 or 3 people writing some research code.
- 5-9 people in a small start-up developing the first version of a new product.
- 10-50 people in a small/medium company (SME) maintaining and developing a relatively mature product.
- Large software consultancy developing large bespoke software systems for clients e.g. a University financial management system.
- Microsoft, Oracle, Adobe, ...

Good practice varies across these as the emphasis and scale changes.

For the rest of this course, the focus will be on engineering development in the Start-Up → SME range of companies.
Team Organisation

**Architect:** The principal designer, defines the overall architecture, module structure and all major interfaces, usually also an expert in the associated technology.

**Project Manager:** Responsible for scheduling the work, tracking progress and ensuring that all of the process steps are properly completed.

**Lead Programmer:** Leader of a programming team. Will typically spend 30% of his/her time managing the rest of the team.

**Programmer:** Implements specific modules and often implements module test procedures.

**Tester:** Designs test and validation procedures for the completed software. Tests are based on initial specification and will focus on the overall product, rather than the individual modules.

When manpower is limited, one individual may perform multiple roles but ideally they should be distinct.
Software Product Development

In the start-up/small company scenario, the focus will be on converting an idea or concept into a working product. The requirements will be loosely known and will evolve over time. The key is rapid prototype development and frequent user testing.
Specification and High Level Design

This is the job of the Architect.

The first step is to produce a high-level “white paper” setting out the vision, the major functionality, the user experience etc. If the software includes a user interface, show mock-ups and list use cases. If there are performance targets, state them. **Gaining Concensus**

This document should be readable by all sections of the company i.e. marketing, sales, technical staff. It should be agreed by senior management before going further.

Program specification and design:

1. divide software into modules (usually compilation units)
2. describe overall function of each module
3. define/document any special algorithms required
4. each module should define one or more class interfaces
5. document public functions/methods in each class

Assign modules to teams. Work with team leaders to assign module and/or class implementations to team members. Sit-in on initial meetings to verify understanding and resolve any issues.
Project Management

GOAL: ONTIME, ON BUDGET
The role of the project manager is: unpopular w. staff but CRUCIAL

1. Identifying required tasks and their dependencies
2. Assigning staff to tasks
3. Estimating task durations
4. Tracking progress through the life-time of the project
5. Advising senior management on progress
6. Rescheduling when necessary
7. Recording task information and completion rates.

(1), (2) and (3) will be in consultation with the Architect and Lead Programmers.

In practice, (3) is the hard part. Staff will consistently underestimate the required time, accumulated historical data generated by (7) provides a very valuable “sanity check”.

Tracking progress (4) should be via weekly meetings between the Project Manager and the Team leaders.

Milestones should be set at key points

- to signal progress to higher management
- to motivate staff
Project Scheduling Tools

Given a list of tasks, predicted durations and staff assignments, project management tools will provide:

1. **Gantt charts** showing time bars, dependencies and progress.
2. **PERT diagrams** emphasising dependencies and critical paths.
3. Staff loading

**Example** Suppose that the following tasks have been identified:

<table>
<thead>
<tr>
<th>Task</th>
<th>Duration (Days)</th>
<th>Who?</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>3</td>
<td>John Smith[50%]</td>
<td>-</td>
</tr>
<tr>
<td>T2</td>
<td>15</td>
<td>John Smith[50%]</td>
<td>-</td>
</tr>
<tr>
<td>T3</td>
<td>15</td>
<td>Susan Good[100%]</td>
<td>T1 (M1)</td>
</tr>
<tr>
<td>T4</td>
<td>10</td>
<td>Ian Black[100%]</td>
<td>-</td>
</tr>
<tr>
<td>T5</td>
<td>10</td>
<td>JS[80%], SG[20%]</td>
<td>T2, T4 (M2)</td>
</tr>
</tbody>
</table>
Example: Microsoft Project

A Gantt chart shows the tasks on a calendar:

Must actively track progress (add today and color % task completion)

A PERT diagram is a form of *activity network*, which emphasises the relationships between tasks:
A Resource usage chart shows the loading on each staff member, e.g. the chart for Susan Good is

In this case, there is an overload in weeks 3/4 which needs to be resolved.
Delays and Rescheduling

Software projects are infamous for being late. There are many causes of over-runs, e.g:

- Programmers are optimists.
- Programmers are often reluctant to admit to delays.
- Sales apply pressure for an early delivery.
- Designs often need to be modified as a result of early testing.
- Debugging times are unpredictable.

It is often assumed that

\[
\text{work done} = \text{time} \times \text{people}
\]

This is not true, hence, delays are rarely soluble by adding more manpower.

The only thing you can do with a late project is to

- cut features and/or
- reschedule
The Mythical Man-month

Brooks’s Law: Adding manpower to a late software project makes it later.

- An ideal project: infinitely partitionable, with no communication required.

- An unpartitionable project: cf “3 women cannot produce a baby in 3 months”.

- A project requiring communication: A software project has complex intercommunications, and the communication pathways increase as $O(n^2)$. This extra communication can quickly overhaul any decrease due to partitioning.
Quality Control

Software development should not rely on testing/debugging to transform a poor quality product into a good one. Strict quality control must be applied throughout the development process via

1. a guide to software standards
2. source code control
3. module testing
4. the “nightly build”
5. weekly progress meetings
6. code reviews

Weekly progress meetings are essential at all levels. Typically, the Project Manager will meet with the Lead Programmers weekly, and the Lead Programmers will meet with the rest of their team at least once per week.

The principle is that nobody should be left alone for more than a week.

Progress meetings are not a waste of time - they are essential for keeping everyone on track.
Software Coding Standards

Every organisation should have a set of software coding standards defining

- essential banners e.g. license declarations
- naming conventions for constants, variables, types and functions, e.g. LINESIZE, int myvar, class TreeNode, DoSomething().
- layout conventions including indentation, layout of control structures, placing of braces, etc
- deprecated programming styles, e.g.
  - x=10, y=4;
  - if (p==0) vs if (p==NULL), etc.
- consistent use of alternative library functions, e.g.
  - printf("x=%d\n",x) vs cout << "x=" << x << '\n'
- use of comments
- etc

The aim is to ensure that all code looks as if it was written by the same person. This greatly simplifies understanding/maintenance of other people’s code amongst the team.
Source Code Control

A source code control system provides

1. a central place to store all source code
2. a historical record of what has been done over time
3. a facility to record a set of sources as a “release”
4. an ability to reconstruct a project as it was at any time in the past
5. a facility to create separate code branches and merge them later

Example: Concurrent Versions System (CVS)

CVS is very popular, and a little unusual in that it does not lock files which are checked out of the archive for editing. Instead, it defers synchronisation until files are committed i.e. checked back in.