9. Project Management

Prof. O. Nierstrasz
Roadmap

- Risk management
- Scoping and estimation
- Planning and scheduling
- Dealing with delays
- Staffing, directing, teamwork
Literature

Sources


Recommended Reading

> *The Mythical Man-Month*, F. Brooks, Addison-Wesley, 1975
> *Succeeding with Objects: Decision Frameworks for Project Management*, A. Goldberg and K. Rubin, Addison-Wesley, 1995
Roadmap

- Risk management
- Scoping and estimation
- Planning and scheduling
- Dealing with delays
- Staffing, directing, teamwork
Almost all software products are obtained via *projects*.(as opposed to manufactured products)

**Project Concern** = *Deliver on time and within budget*

Achieve Interdependent & Conflicting Goals

Limited Resources

*The Project Team is the primary Resource!*
What is Project Management?

**Project Management** = *Plan the work and work the plan*

**Management Functions**

- **Planning**: Estimate and schedule resources
- **Organization**: Who does what
- **Staffing**: Recruiting and motivating personnel
- **Directing**: Ensure team acts as a whole
- **Monitoring (Controlling)**: Detect plan deviations + corrective actions
Risk Management

*If you don’t actively attack risks, they will actively attack you.*

— Tom Gilb

**Project risks**
— budget, schedule, resources, size, personnel, morale ...

**Technical risks**
— implementation technology, verification, maintenance ...

**Business risks**
— market, sales, management, commitment ...
Management must:

> **identify** risks as early as possible
> **assess** whether risks are acceptable
> take appropriate action to *mitigate and manage* risks  
  — e.g., training, prototyping, iteration, ...
> **monitor** risks throughout the project
## Risk Management Techniques

<table>
<thead>
<tr>
<th>Risk Items</th>
<th>Risk Management Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personnel <em>shortfalls</em></td>
<td>Staffing with top talent; <em>team building</em>; cross-training; pre-scheduling key people</td>
</tr>
<tr>
<td><em>Unrealistic schedules</em> and budgets</td>
<td>Detailed multi-source cost &amp; schedule estimation; <em>incremental development</em>; reuse; re-scoping</td>
</tr>
<tr>
<td>Developing the <em>wrong</em> software functions</td>
<td>User-surveys; <em>prototyping</em>; early users’s manuals</td>
</tr>
</tbody>
</table>
Risk Management Techniques …

<table>
<thead>
<tr>
<th>Risk Items</th>
<th>Risk Management Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuing stream of requirements changes</td>
<td>High change threshold; information hiding; incremental development</td>
</tr>
<tr>
<td>Real time performance shortfalls</td>
<td>Simulation; benchmarking; modeling; prototyping; instrumentation; tuning</td>
</tr>
<tr>
<td>Straining computer science capabilities</td>
<td>Technical analysis; cost-benefit analysis; prototyping; reference checking</td>
</tr>
</tbody>
</table>
Roadmap

- Risk management
- **Scoping and estimation**
- Planning and scheduling
- Dealing with delays
- Staffing, directing, teamwork
Focus on Scope

For decades, programmers have been whining, “The customers can’t tell us what they want. When we give them what they say they want, they don’t like it.” Get over it. This is an absolute truth of software development. The requirements are never clear at first. Customers can never tell you exactly what they want.

— Kent Beck
Myth: Scope and Objectives

**Myth**

“A general statement of objectives is enough to start coding.”

**Reality**

*Poor up-front definition is the major cause of project failure.*
Scope and Objectives

In order to plan, you must set clear *scope & objectives*

> **Objectives** identify the *general goals* of the project, not how they will be achieved.

> **Scope** identifies the *primary functions* that the software is to accomplish, and bounds these functions in a quantitative manner.

Goals must be *realistic and measurable*

— Constraints, performance, reliability must be explicitly stated
— Customer must set *priorities*
# Estimation Strategies

*These strategies are simple but risky:*

| **Expert judgement** | Consult experts and compare estimates  
☞ *cheap, but unreliable* |
|-----------------------|----------------------------------------------------------------------------------|
| **Estimation by analogy** | Compare with other projects in the same application domain  
☞ *limited applicability* |
| **Parkinson's Law** | Work expands to fill the time available  
☞ *pessimistic management strategy* |
| **Pricing to win** | You do what you can with the budget available  
☞ *requires trust between parties* |
Estimation Techniques

“Decomposition” and “Algorithmic cost modeling” are used together

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>Estimate costs for components + integration</th>
<th>top-down or bottom-up estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic cost</td>
<td>Exploit database of historical facts to map size on costs</td>
<td>requires correlation data</td>
</tr>
<tr>
<td>modeling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Measurement-based Estimation

A. Measure
Develop a system model and measure its size

B. Estimate
Determine the effort with respect to an empirical database of measurements from similar projects

C. Interpret
Adapt the effort with respect to a specific Development Project Plan

More on this later (Metrics lecture) …
Estimation and Commitment

Example: The XP process

1. a. Customers write stories and 
b. Programmers estimate stories 

— else ask the customers to split/rewrite stories

2. Programmers measure the team load factor, the ratio of ideal programming time to the calendar

3. Customers sort stories by priority

4. Programmers sort stories by risk

5. a. Customers pick date, programmers calculate budget, customers pick stories adding up to that number, or 
b. Customers pick stories, programmers calculate date 
(customers complain, programmers ask to reduce scope, customers complain some more but reduce scope anyway) …
Roadmap

- Risk management
- Scoping and estimation
- **Planning and scheduling**
- Dealing with delays
- Staffing, directing, teamwork
Good planning depends largely on project manager’s intuition and experience!

> Split project into *tasks*.
  — Tasks into subtasks etc.

> For each task, *estimate* the time.
  — Define tasks small enough for reliable estimation.

> Significant tasks should end with a *milestone*.
  — *Milestone* = A *verifiable* goal that must be met after task completion
  — Clear unambiguous milestones are a necessity! (“80% coding finished” is a meaningless statement)
  — *Monitor progress* via milestones
Planning and Scheduling ...

> Define *dependencies* between project tasks
  — Total time depends on longest (= critical) path in activity graph
  — Minimize task dependencies to avoid delays
> Organize tasks *concurrently* to make optimal use of workforce

Planning is *iterative*
⇒ *monitor and revise* schedules during the project!
Myth: Deliverables and Milestones

**Myth**

“The only deliverable for a successful project is the working program.”

**Reality**

Documentation of all aspects of software development are needed to ensure maintainability.
Deliverables and Milestones

Project **deliverables** are results that are delivered to the customer.

- E.g.:
  - initial requirements document
  - UI prototype
  - architecture specification

- Milestones and deliverables help to *monitor progress*
  - Should be scheduled roughly every 2-3 weeks

**NB:** *Deliverables must evolve as the project progresses!*
### Example: Task Durations and Dependencies

<table>
<thead>
<tr>
<th>Task</th>
<th>Duration (days)</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>15</td>
<td>T1</td>
</tr>
<tr>
<td>T4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>10</td>
<td>T2, T4</td>
</tr>
<tr>
<td>T6</td>
<td>5</td>
<td>T1, T2</td>
</tr>
<tr>
<td>T7</td>
<td>20</td>
<td>T1</td>
</tr>
<tr>
<td>T8</td>
<td>25</td>
<td>T4</td>
</tr>
<tr>
<td>T9</td>
<td>15</td>
<td>T3, T6</td>
</tr>
<tr>
<td>T10</td>
<td>15</td>
<td>T5, T7</td>
</tr>
<tr>
<td>T11</td>
<td>7</td>
<td>T9</td>
</tr>
<tr>
<td>T12</td>
<td>10</td>
<td>T11</td>
</tr>
</tbody>
</table>

What is the minimum total duration of this project?
Pert Chart: Activity Network
Gantt Chart: Activity Timeline

Planned completion

Latest completion
Gantt Chart: Staff Allocation
Roadmap

- Risk management
- Scoping and estimation
- Planning and scheduling
- **Dealing with delays**
- Staffing, directing, teamwork
Myth: Delays

**Myth**

“If we get behind schedule, we can add more programmers and catch up.”

**Reality**

Adding more people typically slows a project down.
Scheduling problems

> Estimating the difficulty of problems and the cost of developing a solution is hard
> Productivity is not proportional to the number of people working on a task
> Adding people to a late project makes it later due to communication overhead
> The unexpected always happens. Always allow contingency in planning
> Cutting back in testing and reviewing is a recipe for disaster
> Working overnight? Only short term benefits!
Planning under uncertainty

> State clearly *what you know and don’t know*
> State clearly *what you will do* to eliminate unknowns
> Make sure that *all early milestones* can be met
> Plan to *replan*
Spot potential delays as soon as possible

... then you have more time to recover

How to spot?

- Earned value analysis
  — planned time is the project budget
  — time of a completed task is credited to the project budget
Dealing with Delays ...

How to recover?

A combination of following 3 actions

> Adding senior staff for well-specified tasks
  — outside critical path to avoid communication overhead

> Prioritize requirements and deliver incrementally
  — deliver most important functionality on time
  — testing remains a priority (even if customer disagrees)

> Extend the deadline
Gantt Chart: Slip Line

**Visualize slippage**

- Shade time line = portion of task completed
- Draw a slip line at current date, connecting endpoints of the shaded areas
  - bending to the right = ahead of schedule
  - to the left = behind schedule

| J | F | M | A | M | J | J | A | S | O | N | D | J | F | M | A | M | J | J | A |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

1. Start
2. Design
3. Implementation
   - 3.1. build scanner
   - 3.2. build parser
   - 3.3. build code generator
4. Integrate & Test
5. Write manual
6. Reviewing
7. Finish

ahead of schedule
behind
**Timeline Chart**

*Visualise slippage evolution*

- downward lines represent planned completion time as they vary in current time
- bullets at the end of a line represent completed tasks
### Slip Line vs. Timeline

<table>
<thead>
<tr>
<th></th>
<th>Slip Line</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitors</td>
<td>current slip status of project tasks</td>
<td>monitors how the slip status of project tasks evolves</td>
</tr>
<tr>
<td></td>
<td>&gt; many tasks</td>
<td>&gt; few tasks</td>
</tr>
<tr>
<td></td>
<td>&gt; only for 1 point in time</td>
<td>&gt; complete time scale</td>
</tr>
<tr>
<td></td>
<td>— include a few slip lines from the past to illustrate evolution</td>
<td>— crossing lines quickly clutter the figure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— colours can be used to show more tasks</td>
</tr>
</tbody>
</table>
Roadmap

- Risk management
- Scoping and estimation
- Planning and scheduling
- Dealing with delays
- **Staffing, directing, teamwork**
Software Teams

Team organisation

> Teams should be relatively small (< 8 members)
  — minimize communication overhead
  — team quality standard can be developed
  — members can work closely together
  — programs are regarded as team property (“egoless programming”)
  — continuity can be maintained if members leave

> Break big projects down into multiple smaller projects

> Small teams may be organised in an informal, democratic way

> Chief programmer teams try to make the most effective use of skills and experience
Chief Programmer Teams (Example)

> Consist of a kernel of specialists helped by others as required
  > — chief programmer takes full responsibility for design, programming, testing and installation of system
  > — backup programmer keeps track of CP’s work and develops test cases
  > — librarian manages all information
  > — others may include: project administrator, toolsmith, documentation editor, language/system expert, tester, and support programmers …

> Reportedly successful but problems are:
  > — Can be difficult to find talented chief programmers
  > — Might disrupt normal organizational structures
  > — May be de-motivating for those who are not chief programmers
Directing Teams

Managers serve their team

> Managers ensure that team has the necessary information and resources

“The manager’s function is not to make people work, it is to make it possible for people to work”
— Tom DeMarco

Responsibility demands authority

> Managers must delegate
   — Trust your own people and they will trust you.
Directing Teams ...

**Managers manage**
>
> Managers cannot perform tasks on the *critical path*
> — Especially difficult for technical managers!

**Developers control deadlines**
>
> A manager cannot meet a deadline to which the developers have not agreed
What you should know!

> How can prototyping help to reduce risk in a project?
> What are milestones, and why are they important?
> What can you learn from an activity network? An activity timeline?
> What’s the difference between the 0/100; the 50/50 and the milestone technique for calculating the earned value.
> Why should programming teams have no more than about 8 members?
Can you answer these questions?

> What will happen if the developers, not the customers, set the project priorities?
> What is a good way to measure the size of a project (based on requirements alone)?
> When should you sign a contract with the customer?
> Would you consider bending slip lines as a good sign or a bad sign? Why?
> How would you select and organize the perfect software development team?
License

http://creativecommons.org/licenses/by-sa/3.0/

Attribution-ShareAlike 3.0 Unported

You are free:
  to Share — to copy, distribute and transmit the work
  to Remix — to adapt the work

Under the following conditions:
  Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

Any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.