Western

UNIVERSITY · CANADA

Topic 2

Managing the Software Process (Chapter 11)

Computer Science 2212b
Introduction to Software Engineering
Winter 2014

Jeff Shantz
jeff@csd.uwo

Agenda / Announcements

- Today
  - Finish up Topic 1
  - Start on Topic 2
  - Meet your team members near the end

11.1 – What is Project Management?

Project management encompasses all the activities needed to plan and execute a project:

- Deciding what needs to be done
- Estimating costs
- Ensuring there are suitable people to undertake the project
- Defining responsibilities
- Scheduling
- Making arrangements for the work

What is Project Management?

(continued…)

- Directing
- Being a technical leader
- Reviewing and approving decisions made by others
- Building morale and supporting staff
- Monitoring and controlling
- Co-ordinating the work with managers of other projects
- Reporting
- Continually striving to improve the process

What is Project Management?

Can you think of any recent, high-profile software projects that have gone horribly wrong (failed, delivered late, defective)?

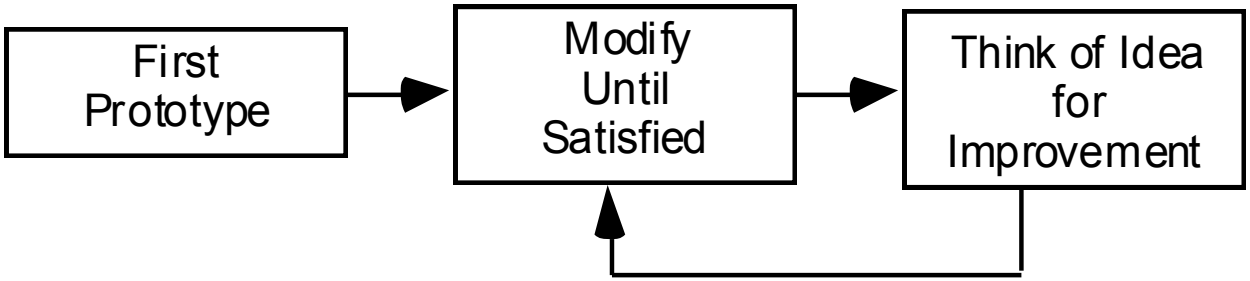
healthcare.gov

11.2 – Software Process Models

Software process models

- General approaches for organizing a project into activities.
- Help the project manager and his or her team to decide
  - What work should be done
  - In what sequence to perform the work
- The models should be seen as *aids to thinking*, not rigid prescriptions of the way to do things.
- Each project ends up with its own unique plan.

The Opportunistic Approach

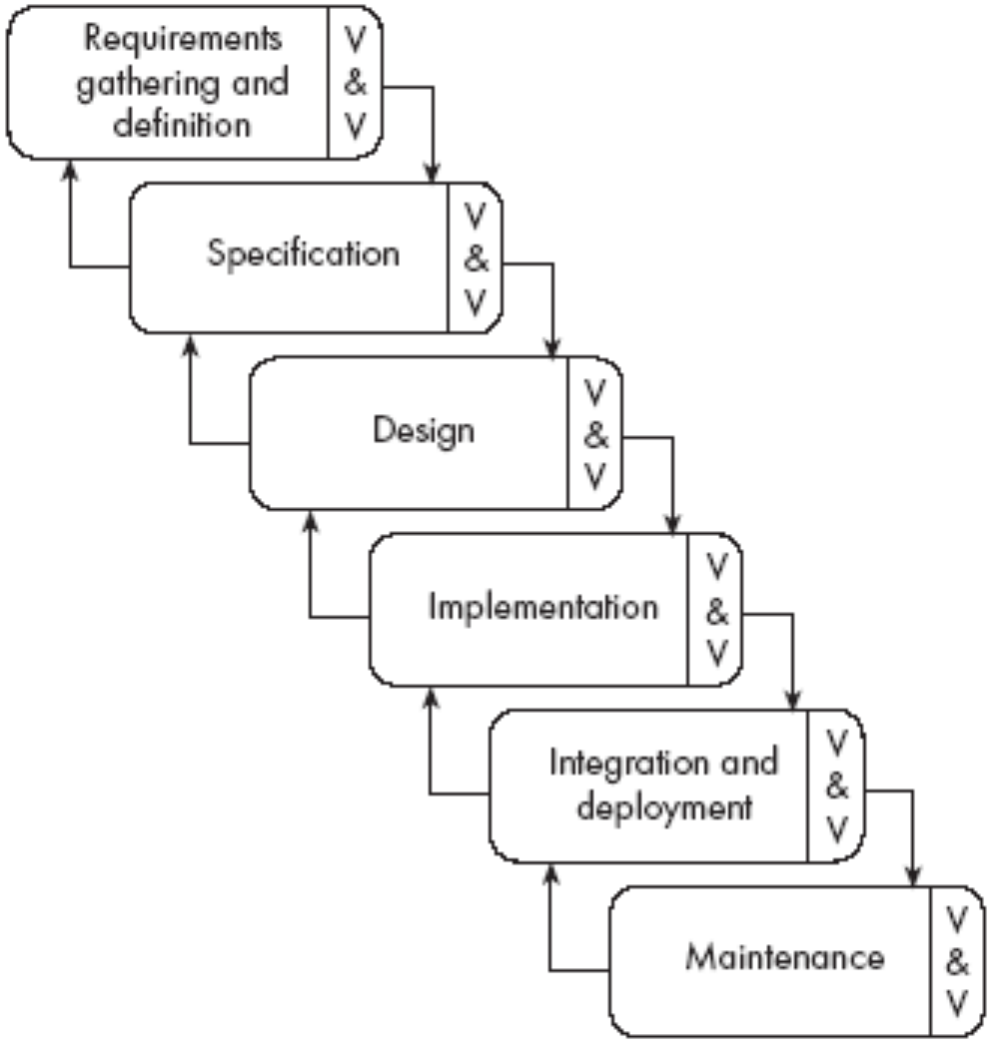


The Opportunistic Approach

… is what occurs when an organization does not follow good engineering practices.

- Does not acknowledge importance of working out the requirements and the design before implementing a system.
- Software design deteriorates faster if it is not well designed.
- No plans = nothing to aim towards
- No explicit recognition of the need for systematic testing and other forms of quality assurance.
- The above problems make the cost of developing and maintaining software very high.

The Waterfall Model



The Waterfall Model

The classic way of looking at SE that accounts for the importance of requirements, design and quality assurance.

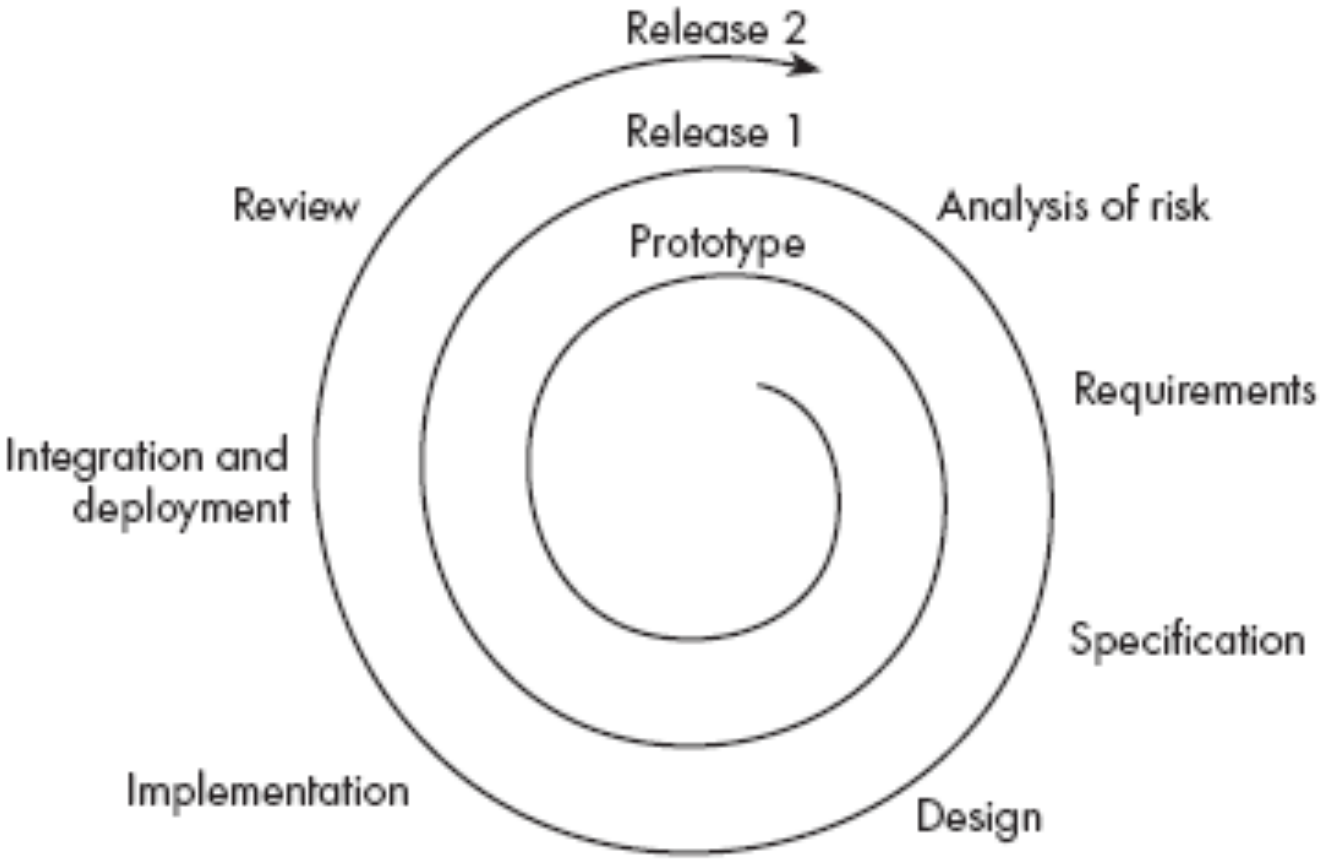
- The model suggests that software engineers should work in a series of stages.
- Before completing each stage, they should perform quality assurance (verification and validation).
- The waterfall model also recognizes, to a limited extent, that you sometimes have to step back to earlier stages.
- Part of a family of approaches known as Big Design Up Front (BDUF)

The Waterfall Model

Problems

- Implies that a given stage must be completed before moving on to the next stage.
  - No accounting for the fact that requirements constantly change.
  - Customers can not use anything until the entire system is complete.
- Makes no allowances for prototyping.
- Implies that you can get the requirements right by simply writing them down and reviewing them.
- The model implies that once the product is finished, everything else is maintenance.

The Spiral Model



The Spiral Model

It explicitly embraces prototyping and an *iterative* approach to software development.

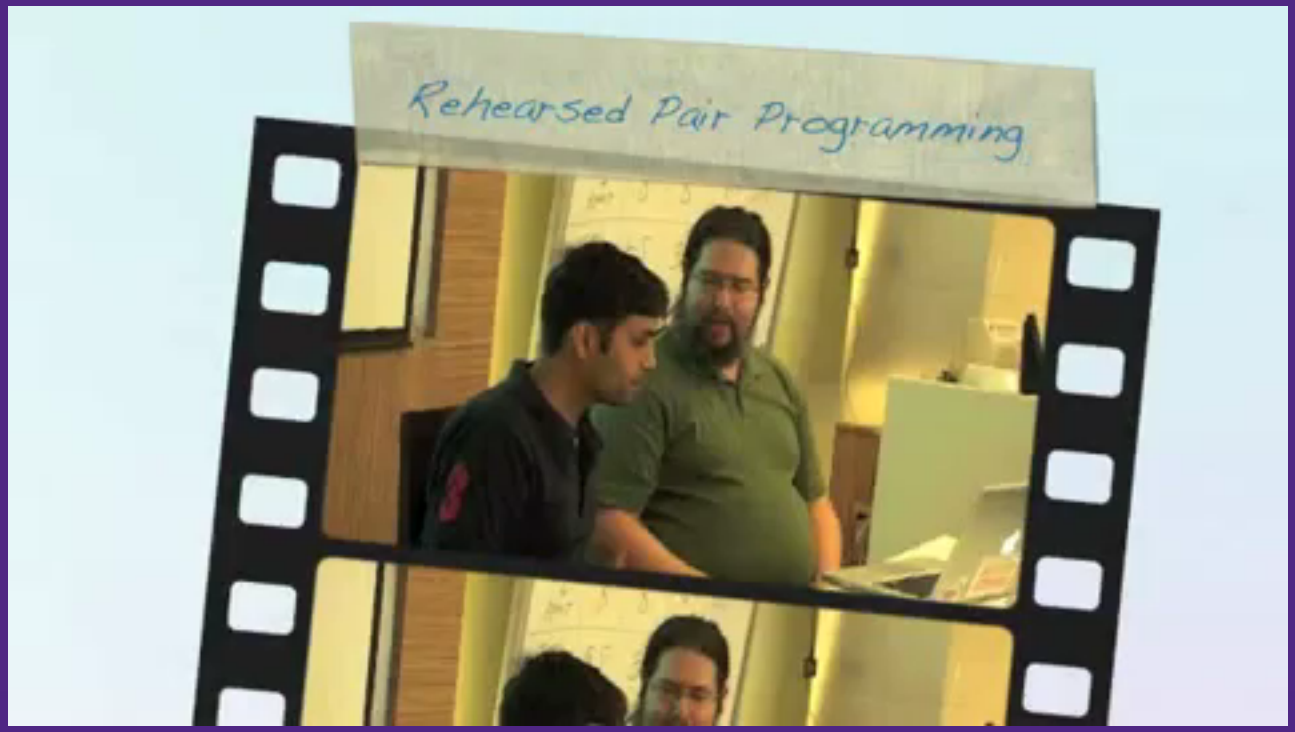
- Start by developing a small prototype.
- Followed by a mini-waterfall process, primarily to gather requirements.
- Then, the first prototype is reviewed.
- In subsequent loops, the project team performs further requirements, design, implementation and review.
- The first thing to do before embarking on each new loop is risk analysis.
- Maintenance is simply a type of on-going development.

Agile Approaches

These approaches encourage the development of particularly small iterations

- Well suited for projects that involve uncertain, changing requirements and other high risk
- The most famous agile methodology is eXtreme Programming (XP)
  - All stakeholders work closely together
  - User stories instead of requirements document
  - Small and frequent releases: 1 to 3 weeks
  - Test-driven development (TDD)
  - A large amount of refactoring is encouraged
  - Pair programming is recommended

Pair Programming



Re-engineering

Periodically project managers should set aside some time to re-engineer part or all of the system

- The extent of this work can vary considerably:
  - Cleaning up the code to make it more readable.
  - Completely replacing a layer.
  - *Re-factoring* part of the design.
- In general, the objective of a re-engineering activity is to increase maintainability.

11.3 – Cost Estimation

To estimate how much software engineering time will be required to do some work.

Elapsed time

- Difference in time from the start date to the end date of a task or project.

Development effort

- Amount of labour used in *person-months* or *person-days.*
- To convert an estimate of development effort to an amount of money:
  - You multiply it by the *weighted average cost* (*burdened* cost) of employing a software engineer for a month (or a day).

Burdened Cost Example

Average salary: $4000 / month
Development effort estimate: 7 person-months

Therefore, 7 developers required on the project, on average, at any given time

If we consider only our engineers' salaries:
7 x $4000 = $28,000

Weighted average salary: $11,000 / month

7 x $11,000 / month = $77,000

Ignoring the burdened cost leads to a severe underestimation of costs.

Cost Estimation

Suppose I give you a project and ask you to estimate how much elapsed time it will take. How might you go about estimating this?

Principles of Effective Cost Estimation

Principle 1: Divide and conquer.

- To make a better estimate, you should divide the project up into individual subsystems.
- Then divide each subsystem further into the activities that will be required to develop it.
- Next, you make a series of detailed estimates for each individual activity.
- Sum the results to arrive at the grand total estimate for the project.

Principles of Effective Cost Estimation

Principle 2: Include all activities when making estimates.

- The time required for *all* development activities must be taken into account.
- Including:
  - Prototyping
  - Design
  - Inspecting
  - Testing
  - Debugging
  - Writing user documentation
  - Deployment

Principles of Effective Cost Estimation

Principle 3: Base your estimates on past experience combined with knowledge of the current project.

- If you are developing a project that has many similarities with a past project:
  - You can expect it to take a similar amount of work.
- Base your estimates on the *personal judgement* of your experts; or
- Use *algorithmic models* developed in the software industry as a whole by analyzing a wide range of projects.
  - They take into account various aspects of a project's size and complexity, and provide formulas to compute anticipated cost.

Algorithmic Models

Allow you to systematically estimate development effort.

- Based on an estimate of some other factor that you can measure, or that is easier to estimate:
  - The number of use cases
  - The number of distinct requirements
  - The number of classes in the domain model
  - The number of widgets in the prototype user interface
  - An estimate of the number of lines of code

Algorithmic Models

A typical algorithmic model uses a formula like the following:

- COCOMO:

$$E = a + bN^c$$

- Functions Points:

$$S = W_1F_1 + W_2F_2 + W_3F_3 + \ldots$$

Principles of Effective Cost Estimation

Principle 4: Be sure to account for *differences* when extrapolating from other projects.

- Different software developers
- Different development processes and maturity levels
- Different types of customers and users
- Different schedule demands
- Different technology
- Different technical complexity of the requirements
- Different domains
- Different levels of requirement stability

Principles of Effective Cost Estimation

Principle 5: Anticipate the worst case and plan for contingencies.

- Develop the most critical use cases first
  - If the project runs into difficulty, then the critical features are more likely to have been completed
- Make three estimates:
  - Optimistic (O)
    - Imagining a everything going perfectly
  - Likely (L)
    - Allowing for typical things going wrong
  - Pessimistic
    - Accounting for everything that could go wrong

Principles of Effective Cost Estimation

Principle 6: Combine multiple independent estimates.

- Use several different techniques and compare the results.
- If there are discrepancies, analyze your calculations to discover what factors causing the differences.
- Use the Delphi technique.
  - Several individuals initially make cost estimates in private.
  - They then share their estimates to discover the discrepancies.
  - Each individual repeatedly adjusts his or her estimates until a consensus is reached.

Principles of Effective Cost Estimation

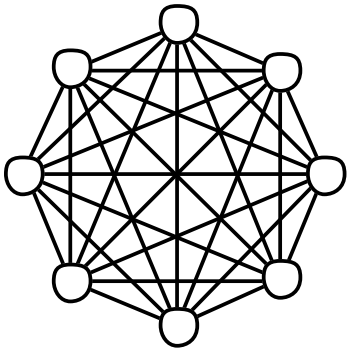
Principle 7: Revise and refine estimates as work progresses

- As you add detail.
- As the requirements change.
- As the risk management process uncovers problems.

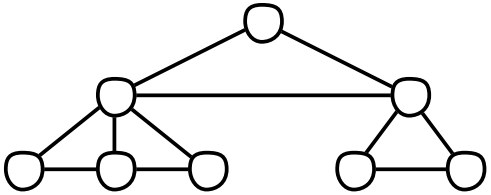
11.4 – Building Software Engineering Teams

Software engineering is a human process.

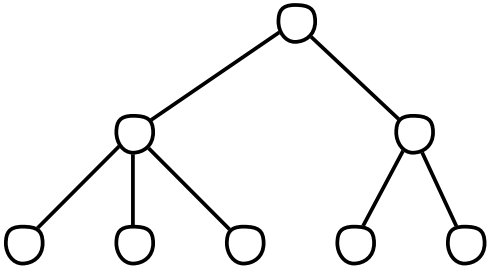
- Choosing appropriate people for a team, and assigning roles and responsibilities to the team members, is therefore an important project management skill
- Software engineering teams can be organized in many different ways



a) Egoless



b) Chief programmer

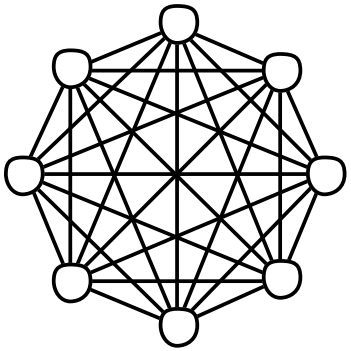


c) Strict hierarchy

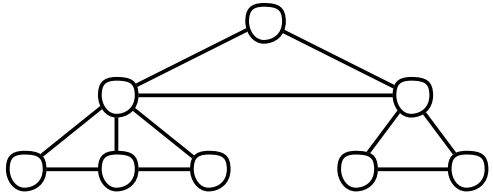
Building Software Engineering Teams

Egoless team:

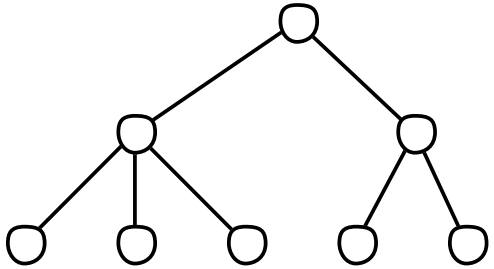
- In such a team everybody is equal, and the team works together to achieve a common goal.
- Decisions are made by consensus.
- Most suited to difficult projects with many technical challenges.



a) Egoless



b) Chief programmer

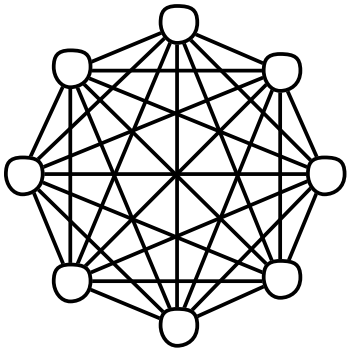


c) Strict hierarchy

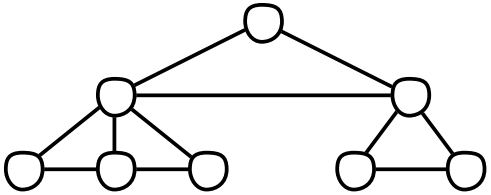
Building Software Engineering Teams

Hierarchical manager-subordinate structure:

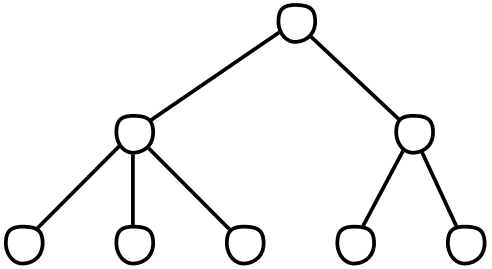
- Each individual reports to a manager and is responsible for performing the tasks delegated by that manager.
- Suitable for large projects with a strict schedule where everybody is well-trained and has a well-defined role.
- However, since everybody is only responsible for their own work, problems may go unnoticed.



a) Egoless



b) Chief programmer

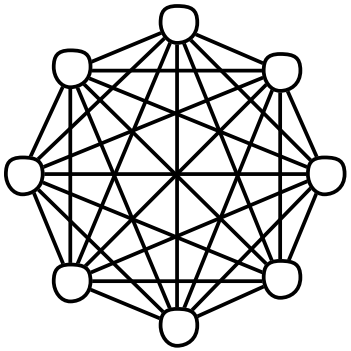


c) Strict hierarchy

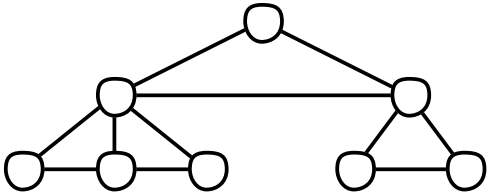
Building Software Engineering Teams

Chief programmer team:

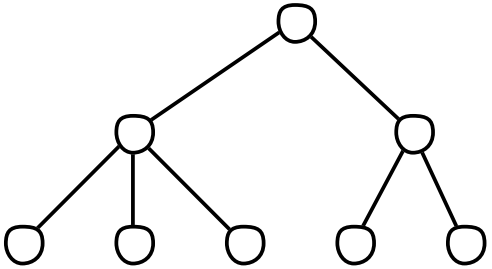
- Midway between egoless and hierarchical.
- The chief programmer leads and guides the project.
- He or she consults with, and relies on, individual specialists.



a) Egoless



b) Chief programmer



c) Strict hierarchy

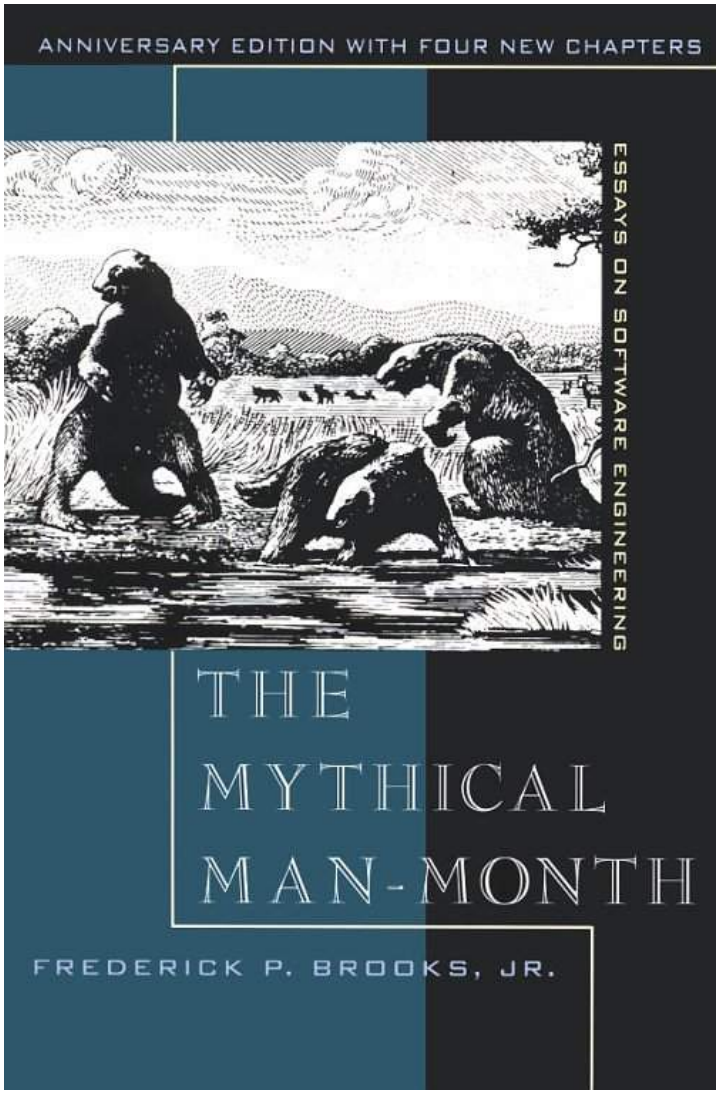
Choosing an Effective Team Size

- For a given estimated development effort, in person months, there is an optimal team size.
  - Doubling the size of a team will not halve the development time.
- Subsystems and teams should be sized such that the total amount of required knowledge and exchange of information is reduced.
- For a given project or project iteration, the number of people on a team will not be constant.

Choosing an Effective Team Size

Brooks' Law

Adding manpower to a late project makes it later.



Skills Needed on a Team

- Architect
- Project manager
- Configuration management and build specialist
- User interface specialist
- Technology specialist
- Hardware and third-party software specialist
- User documentation specialist
- Tester

11.5 – Project Scheduling and Tracking

- *Scheduling* is the process of deciding:
  - In what sequence a set of activities will be performed.
  - When they should start and be completed.
- *Tracking* is the process of determining how well you are sticking to the cost estimate and schedule.

Basic Project Management Terminology

- ***Deliverable:*** **some concrete thing to be delivered to the client or internally to the development team; e.g.**
  - Specifications reports
  - Executable program
  - Source code
- ***Task/Activity*****: something we have to do during the project; e.g.**
  - Defining user requirements
  - Coding a module
  - Doing system testing
- **Each task or activity will take some length of time**
  - Referred to as ***duration*** of task
  - Sometimes measured in days, weeks, etc.
  - Sometimes measured in person-days, person-weeks, etc.
  - ***Person-day*** = number of people X number of days
    - Example: 12 person days for writing all code could mean 1 person 12 days or 4 people 3 days
    - **Note**: not always true that a task that takes 1 programmer 12 days would take 12 programmers 1 day

Dependencies and Milestones

- **Might not be able to start a task without some other tasks having been completed; e.g.**
  - Cannot start coding without completing design
  - Cannot start system testing without completing code integration and test plan
- **If task B cannot start without A being completed:**
  - B *depends on* A
  - There is *a dependency between* A and B
- ***Milestone*: some achievement which must be made during the project; e.g.**
  - Delivering some deliverable
  - Completing some task
- **Note:** delivering a deliverable may be a milestone, but not all milestones are associated with deliverables

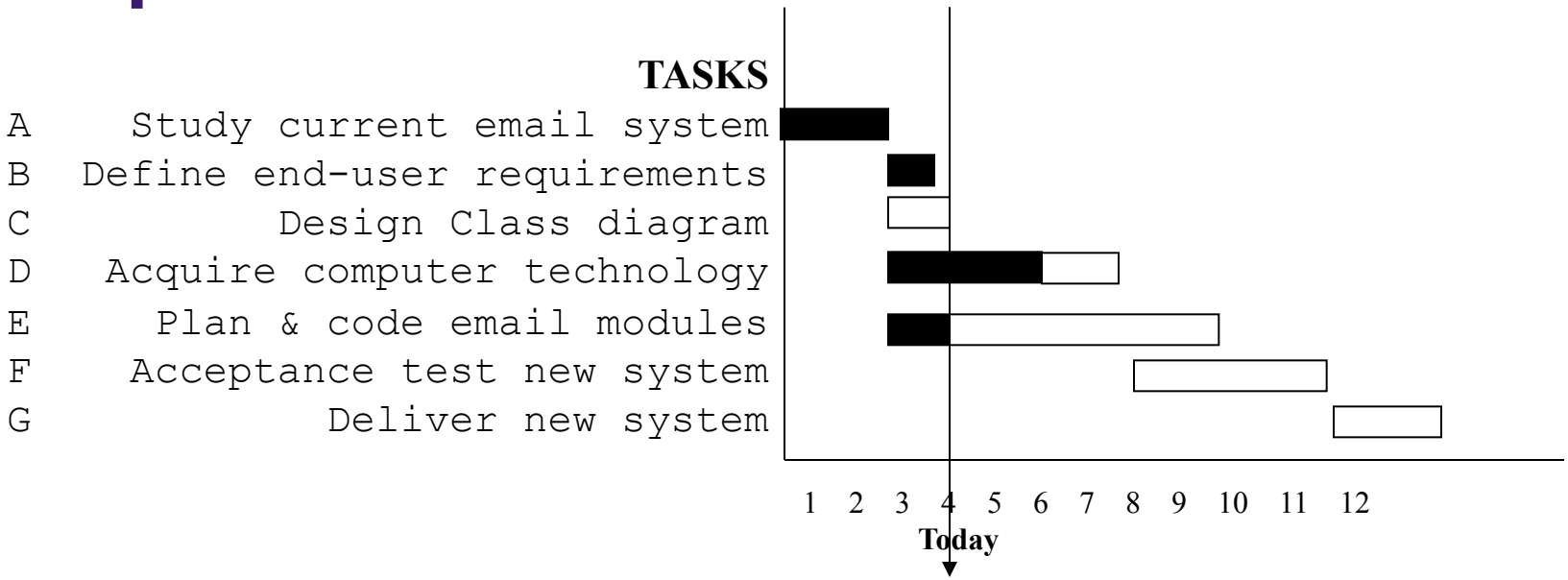
Setting and Making Deadlines

- ***Deadline* time by which milestone has to be met**
  - Some set by the client; others set by us
- **To set a deadline for completing task T, we must consider how long it will take to:**
  - Complete the tasks that task T depends on
  - Complete task T itself
- **If we miss a deadline, we say (euphemistically) "the deadline has slipped"**
  - This is virtually inevitable
- **Important tasks for project managers**
  - Monitor whether past deadlines have slipped
  - Monitor whether future deadlines are going to slip
  - Allocate or reallocate resources to help make deadlines

Gantt Charts

- **Graphical representation of a schedule**
- **Named after Henry Gantt who invented them in 1917**
- **Help to plan, coordinate, and track tasks in a project**
  - Tasks
  - Durations
  - Dependencies
- **On any given day, can see:**
  - Tasks that have been completed
  - Tasks that are in progress
  - Tasks that are late
  - Tasks that can be completed concurrently

Example Gantt Chart



Try to guess which tasks, if any, …	
should have been completed by today and aren't even started?	
have been completed?	
have been completed ahead of schedule?	
are on or ahead of schedule?	
are behind schedule?	

Building and Using a Gantt Chart

Steps for building a Gantt Chart

1. Identify the tasks to be scheduled
2. Determine the durations of each task
3. List each task down the vertical axis of chart
   - In general, list tasks to be performed first at the top and then move downward as the tasks will happen
4. Use horizontal axis for the dates
5. Determine start and finish dates for tasks
   - Consider which tasks must be completed or partially completed before the next task

Building and Using a Gantt Chart

To use the Gantt chart to report progress:

- If the task has been completed, completely shade in the bar corresponding to the task
- If the task has been partially completed, shade in the percentage of the bar that represents the percentage of the task that has been completed
- Unshaded bars represents tasks that have not been started.

Gantt Chart: Exercise

Task	Prec. Tasks	Description	Time (hrs)
A	None	Decide on date for party	1
B	A	Book bouncy castle	1
C	A	Send invitations	4
D	C	Receive replies	7
E	D	Buy toys and balloons	1
F	D	Buy food	3
G	E	Blow up balloons	2
H	F	Make food	1
I	H, G	Decorate	1
J	B	Pick up bouncy castle	1
K	J, I	Have party	1
L	K	Clean up	4
M	K	Send back bouncy castle	1
N	L	Send thank you letters	3
O	M	Donate unwanted gifts	3

Gantt Chart: Exercise

Draw the Gantt chart using the following criteria:

- label hours 0 to 30 across the horizontal axis
- Mark a review stage at hour 14 to monitor the progress
- Assume and illustrate that tasks A, B, C and D have been completed at hour 14
- State which tasks are on schedule and which tasks are behind schedule
- **Note**: if you are using MS Project and want a different unit of time, just type 2 hours (instead of 2 days). Also, if you want to have a milestone, like *Handing in Team Assignment 1*, then give it a duration of zero.

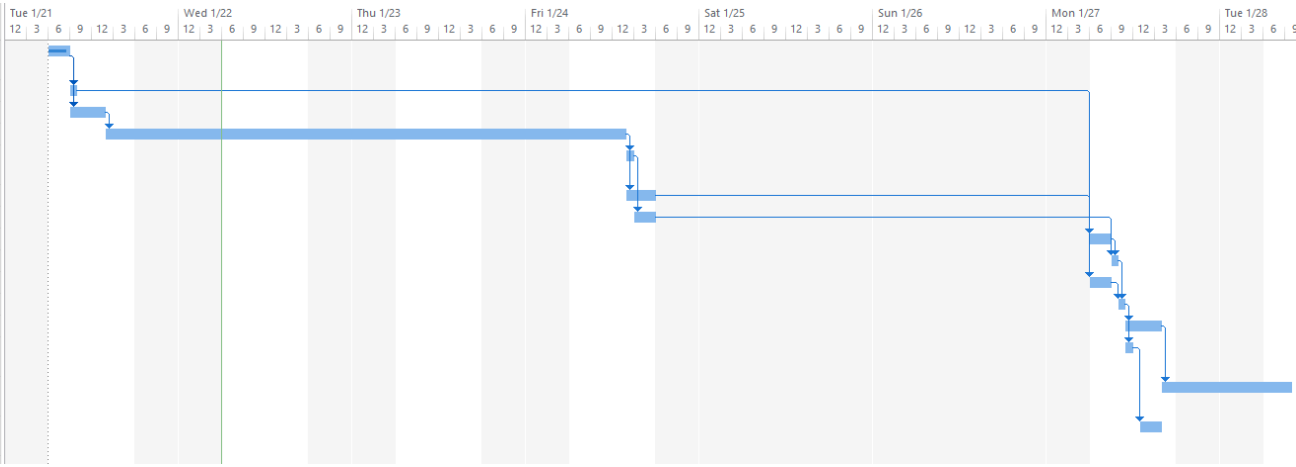
Gantt Charts

Microsoft Project

- Get this from DreamSpark (http://www.csd.uwo.ca/msdnaa)
  - Only available for Windows
  - Use a virtual machine or use different software
    - http://www.csd.uwo.ca/vmap
- **To change the unit of time, just type in the desired unit:**
  - e.g. Just type 2 hours instead of 2 days
- **To represent milestones**
  - e.g. *Handing in Team Assignment 1*
  - Give them a duration of zero

Bouncy Castle Gantt Example using Project

	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Decide on date for party	1 hr	Tue 1/21/14	Tue 1/21/14	
2		Book bouncy castle	1 hr	Tue 1/21/14	Tue 1/21/14	1
3		Send invitations	4 hrs	Tue 1/21/14	Tue 1/21/14	1
4		Receive replies	3 days	Tue 1/21/14	Fri 1/24/14	3
5		Buy toys and balloons	1 hr	Fri 1/24/14	Fri 1/24/14	4
6		Buy food	3 hrs	Fri 1/24/14	Fri 1/24/14	4
7		Blow up balloons	2 hrs	Fri 1/24/14	Fri 1/24/14	5
8		Make food	1 hr	Mon 1/27/14	Mon 1/27/14	6
9		Decorate	1 hr	Mon 1/27/14	Mon 1/27/14	8,7
10		Get bouncy castle	1 hr	Mon 1/27/14	Mon 1/27/14	2
11		Have party	1 hr	Mon 1/27/14	Mon 1/27/14	9,10
12		Clean up	4 hrs	Mon 1/27/14	Mon 1/27/14	11
13		Send back bouncy castle	1 hr	Mon 1/27/14	Mon 1/27/14	11
14		Send thank you letters	3 hrs	Mon 1/27/14	Tue 1/28/14	12
15		Donate unwanted gifts	3 hrs	Mon 1/27/14	Mon 1/27/14	13



11.7 – Difficulties/Risks in Project Management

- **Accurately estimating costs is a constant challenge**
  - *Follow the cost estimation guidelines.*
- **It is very difficult to measure progress and meet deadlines**
  - *Improve your cost estimation skills so as to account for the kinds of problems that may occur.*
  - *Develop a closer relationship with other members of the team.*
  - *Be realistic in initial requirements gathering, and follow an iterative approach.*
  - *Use Gantt charts to monitor progress.*

11.7 – Difficulties/Risks in Project Management

- **It is difficult to deal with lack of human resources or technology needed to successfully run a project**
  - *When determining the requirements and the project plan, take into consideration the resources available.*
  - *If you cannot find skilled people or suitable technology then you must limit the scope of your project.*

11.7 – Difficulties/Risks in Project Management

- **Communicating effectively in a large project is hard**
  - *Take courses in communication, both written and oral.*
  - *Learn how to run effective meetings.*
  - *Review what information everybody should have, and make sure they have it.*
  - *Make sure that project information is readily available.*
  - *Use groupware technology to help people exchange the information they need to know*

11.7 – Difficulties/Risks in Project Management

- **It is hard to obtain agreement and commitment from others**
  - *Take courses in negotiating skills and leadership.*
  - *Ensure that everybody understands*
    - *The position of everybody else.*
    - *The costs and benefits of each alternative.*
    - *The rationale behind any compromises.*
  - *Ensure that everybody's proposed responsibility is clearly expressed.*
  - *Listen to everybody's opinion, but take assertive action, when needed, to ensure progress occurs.*