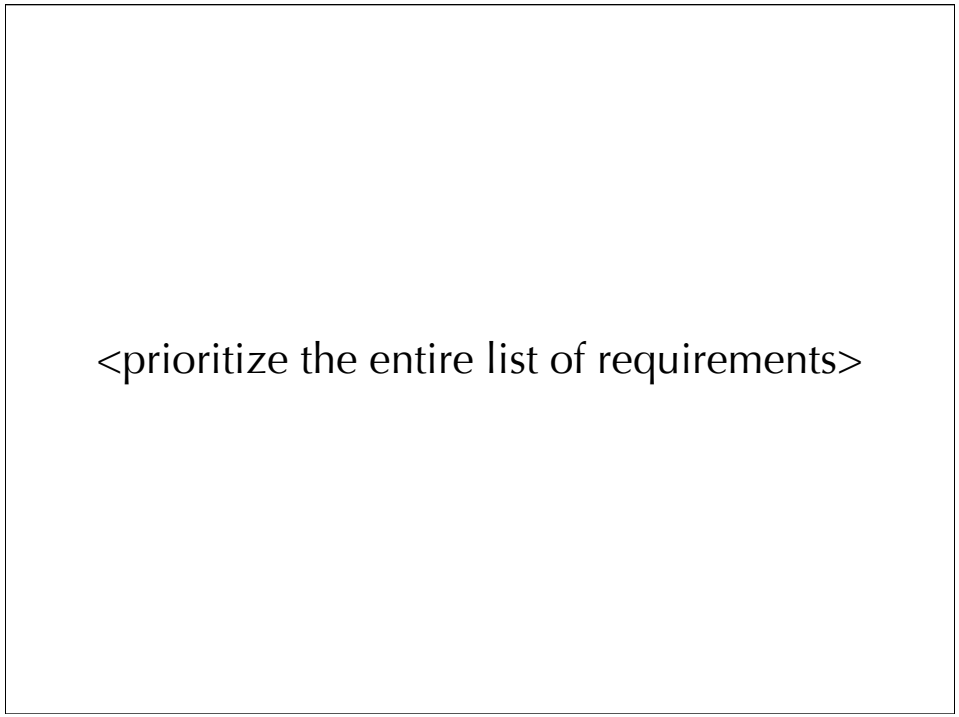
The CS department would like to develop and use an open-source authoring system.

Work in groups of 2 or 3 to estimate how long it would take to implement each of these desired features.

"Implement" means write and test -- at the end, it works.

Give either absolute estimates ("2 hours", "3 days")
or relative estimates (1 to 5 "points", where 1 is easiest and 5 is hardest).



Pick one team with a complete set of estimates.
Total the estimates. Divide by four. Prioritize.
Select stories for first iteration. Discuss trade-offs.

Notes:

- Ordinarily, the list of stories is created in a collaboration between client-user and developer.

- I had planned to use the requirements for on-line schedule book you created last time,
  with my estimates as developer and your prioritizing as client, but... taken.

... from last time:

What about the
requirements
and a
functional specification?

Prioritize based on value and cost (time).
Implement features. Deliver working system. Get feedback. Repeat.

The specification is
a list of requirements.

The requirements are often called “stories” of typical uses.
The goal is to be less formal than what is usually intended by “requirements”.

The stories should be small. The team should be able to implement each story quickly
— in a few hours, a day, a few days, a week, ...

... from last time: *short iterations*

"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale."

Agile approaches value implementation and delivery on very short cycles.

Stories that are done.
Stories being implemented in the current iteration.
Stories to implement in future iterations.

Stories are selected for the next iteration by the client-user
based on their value to the client and their cost, as estimated by the developers.

The project plan is
a partitioned set
of stories.

Stories that are done.
Stories being implemented in the current iteration.
Stories to implement in future iterations.

Short iterations, continuous integration, and continuous delivery give everyone involved access to **continuous feedback**, which can be used to “do better”.

feedback
to the client

Working software lets the client see the effects of its choices.
They recognize that:

- There are new stories they want to have implemented.
- Some of the stories already on the list should be changed.
- Some stories on the list can be deleted.

“Requirements” sounds fixed, but in most context the features of the system are in constant negotiation.

The list of stories changes over time.
Do a conscious update after each iteration, and allow changes even during an iteration.

The goal is a more desirable end product.

feedback to the developers

The process of implementing the system teaches the developers how to implement the system.

The developers make choices on a small scale and have frequent opportunities to change.

They also learn how long it takes to implement stories.
Which stories took longer than expected? Less time than expected? Why?

Use this feedback to update the estimates on remaining stories
and give more accurate estimates for new stories.

With feedback, **developers can improve the quality of their estimates over time**.

Change happens.

The two responses...

Traditional approach on requirements: **lock the door**.

- Advantage: The client and the developer alike know what systems will be built.
  (... if both have been honest and the developer estimates correctly ...)
- Risk: You build **the wrong system**.
  The original (old) spec is inaccurate **and you know it**.

Agile approach on requirements: **keep the door open**.

- Advantage: The client can shape the system as development proceeds.
  (... if both are honest about constraints ...)
- Risk: The process never converges, leading **time overruns** and **cost overruns**.
  The evolving (unsettled) spec is bloated **and you know it**.

... from last time:

"Welcome changing requirements, even late in development."

The goal is to harness change to the **customer's advantage**.

There are four variables that characterize anyway software project and resulting system.
Choosing to improved the value of one requires giving something else up.
It is a matter of trade-off.

Cost is more generally resources: people, tools, overtime, ...

“Projects under stress often squeeze quality because no one can see it happening (until they release crap too late).”
-- Ron Jeffries, http://www.c2.com/cgi/wiki?FourVariables

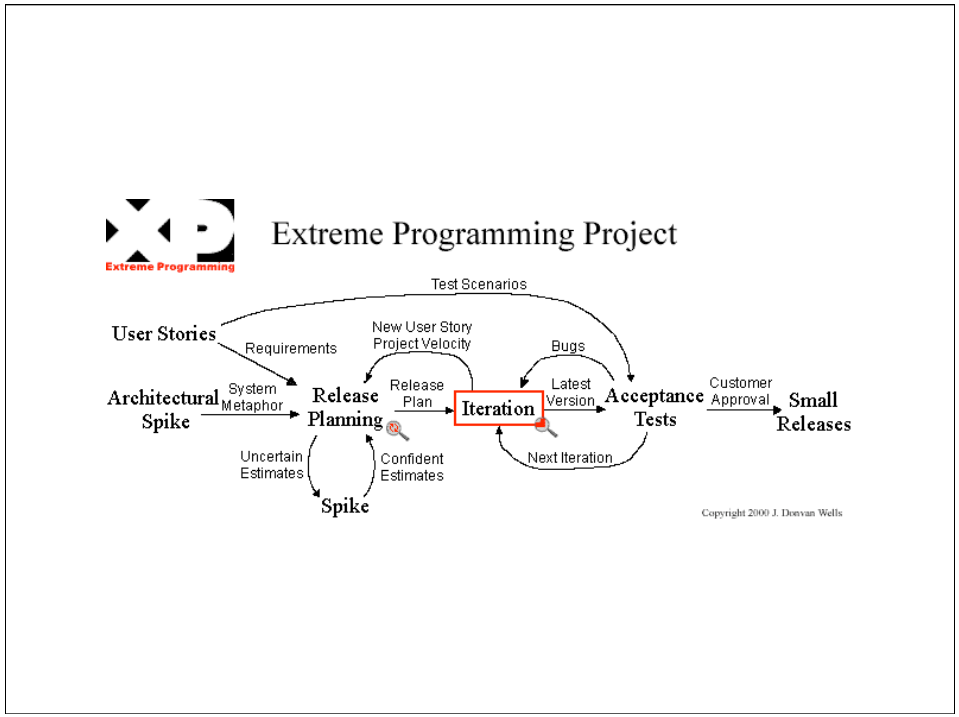
Users don’t want software of low quality.
Developers don’t want to create software of low quality.
This is the one variable whose value we should generally hold steady.

(Still there may be trade-offs available!
Can you think of software quality you would be willing to surrender?)

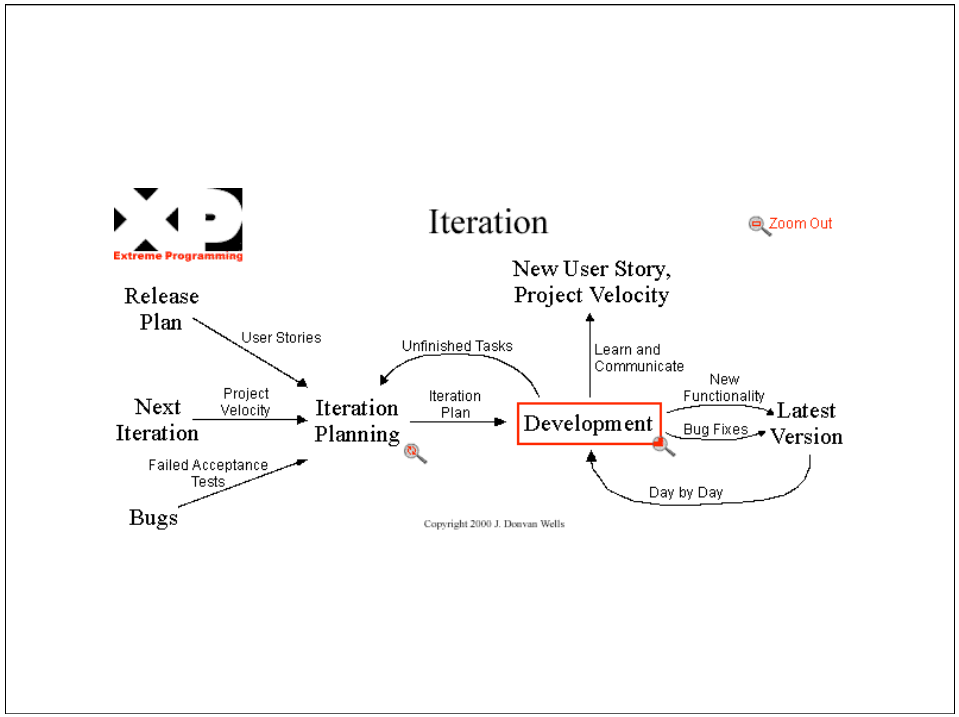
In many contexts, cost is relatively fixed or is related directly to time.

If we think of time and cost as proportional, then this is where the trade-off usually lies. **Adding features takes time**. Software developers cannot perform magic.

Negotiation of the system's features requires both client-users and developers to be honest:
The client-user must be honest in trading scope for time and vice versa.
The developer must be honest in trading scope for time and in estimating time.



Courtesy of http://www.extremeprogramming.org/

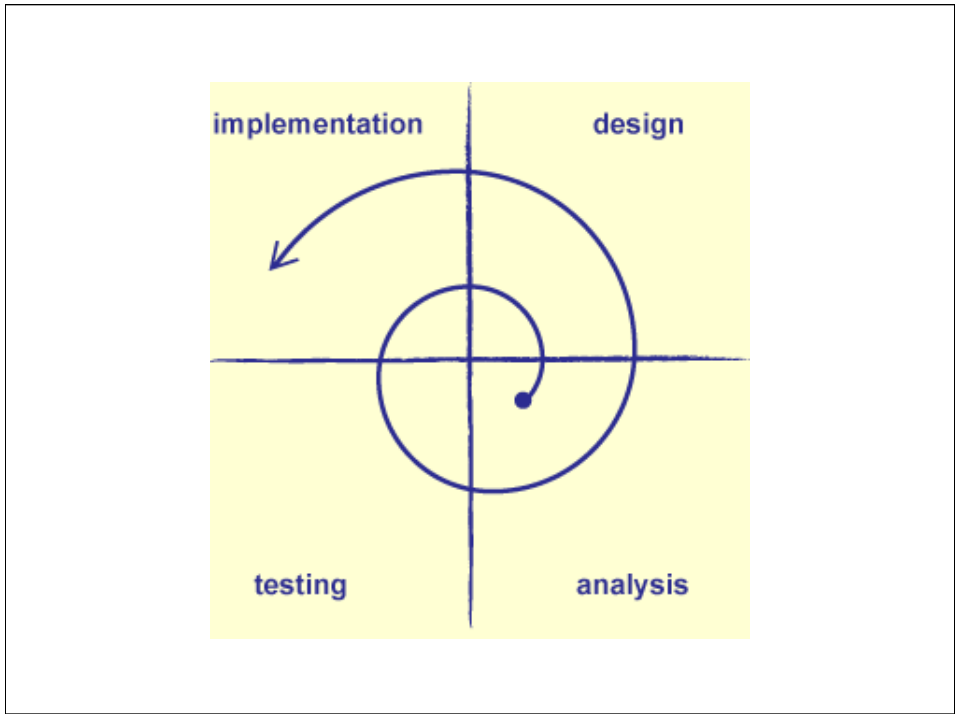


Courtesy of http://www.extremeprogramming.org/

agile
software
development

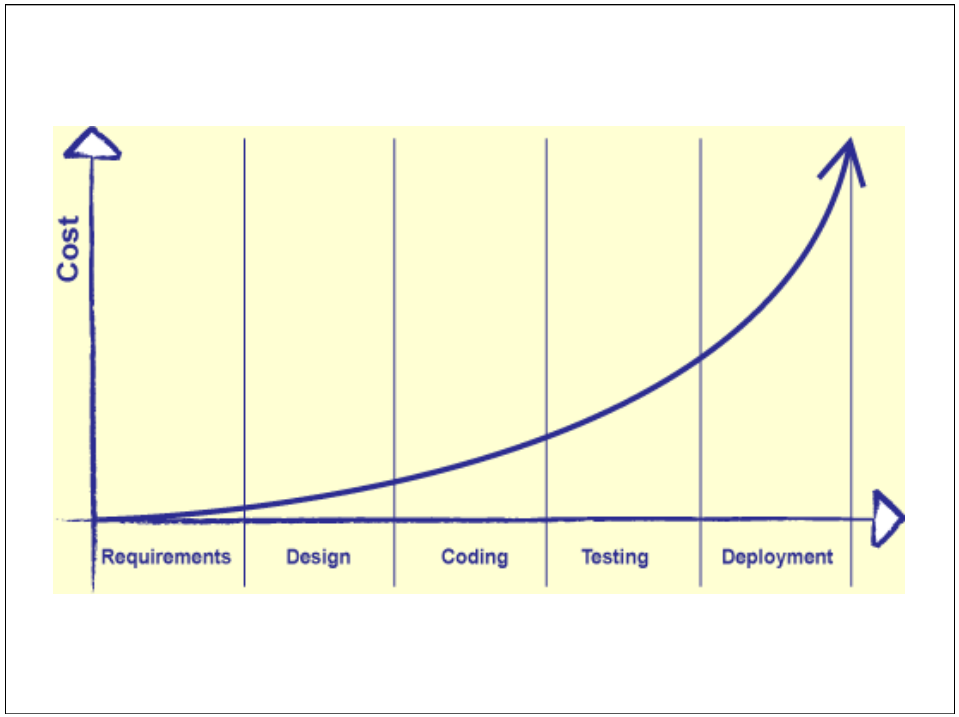
The last two sessions is a brief introduction to agile development.
We will see more of its influence as we move through the course.

The idea:



... is to use the traditional iterative software development model.
How quickly do you spiral around? **As quickly as you meaningfully can**.
Feedback is good.

But...



But... what about Boehm's Curve:
Change costs more the farther you go in the product lifecycle.

Yes! So do requirements gathering and design **more often**, not slower.

The sometimes unstated assumption: **Feedback is good**.

On to the project...

Watch for project phase 1. It will come out on the website today or early tomorrow.
Your first job: meet with clients to gather requirements.

You will have a choice of a more traditional sequence or a more agile style of development.
I will work out the details with each team.