Lecture 24 — Software Lifecycle Models & Tactics

Jeff Zarnett & Patrick Lam
jzarnett@uwaterloo.ca & p.lam@ece.uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 14, 2015

Part I

Software Lifecycle Models

Life Without Models

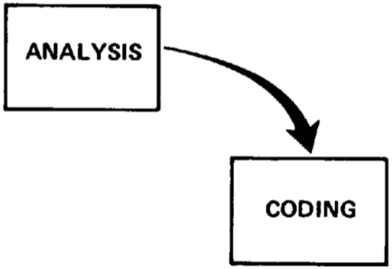
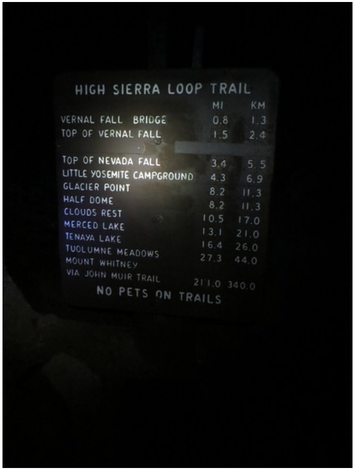


Figure 1. Implementation steps to deliver a small computer program for internal operations.

from: Winston W. Royce. “Managing the Development of Large Software Systems”, Proceedings IEEE WESCON, 1970.

Deathmarches and Fiascoes

Software project management is hard.



HIGH SIERRA LOOP TRAIL

	MI	KM
VERNAL FALL BRIDGE	0.8	1.3
TOP OF VERNAL FALL	1.5	2.4
TOP OF NEVADA FALL	3.4	5.5
LITTLE YOSEMITE CAMPGROUND	4.3	6.9
GLACIER POINT	8.2	11.3
HALF DOME	8.2	11.3
CLOUDS REST	10.5	17.0
MERCED LAKE	13.1	21.0
TENAYA LAKE	16.4	26.0
TUOLUMNE MEADOWS	27.3	44.0
MOUNT WHITNEY VIA JOHN MUIR TRAIL	211.0	340.0

NO PETS ON TRAILS

Software development lifecycle models try to avoid deathmarches and fiascoes.

Death Marches

Some amount of pressure in a project is normal.

When there is so much pressure that success is impossible, it's a *Death March*.

Some possible causes:

- Naive optimism
- Organizational politics
- Trying to build a huge project all at once
- Managerial incompetence

Iterations

http://commons.wikimedia.org/wiki/File:Cat_investigates_washing_machine_2003-07-03.png

Design is iterative.

Lifecycle models help organize the iterations.

Software Design: Like Engineering Design

Both attempt to build the best possible design given:

- sets of project requirements,
- project constraints, and
- criteria for evaluating design success.

Main difference: deploy software immediately;
result of engineering design dispatched to manufacturing.

Note: engineering design process can improve your use of software lifecycle models.

Steps in Software Design Process

- Problem Definition
- Requirements Development
- Project Planning
- High-Level Design
- Detailed Design
- Coding and Debugging
- Integration Testing
- System Testing
- Corrective Maintenance

How can we combine and iterate them?

Some Software Lifecycle Models

- Waterfall
- Concurrent Engineering
- V-Model
- Spiral

Other models are similar to the ones we’ll talk about.

Impact of Models

If you follow a model:

- maybe good things will happen.

If you follow a model poorly, potential recipe for disaster:

- poorly designed and implemented software;
- many bug-fixing design iterations.

If the project is simple enough, you might avert disaster.

Waterfall Model: The Ideal

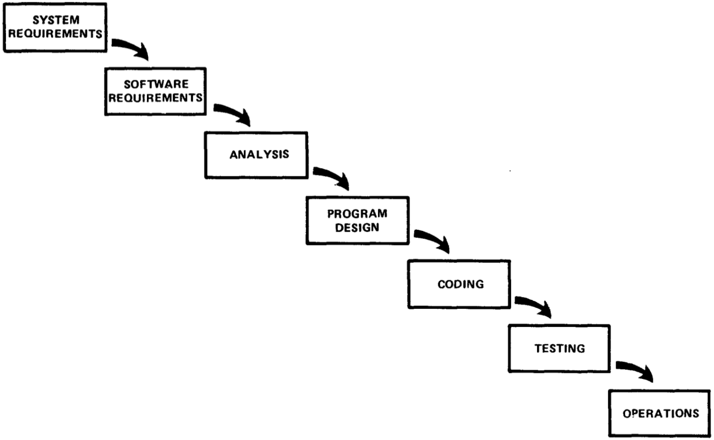


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Waterfall Model

Highly sequential: stages do not overlap.
Project moves onto the next stage following reviews.

Advantages:

1. fixes customer requirements early (hopefully the right requirements);
2. could identify problems early in the design process, when changes are less expensive.

Disadvantages:

1. working blind—don't see any software until the end of the implementation stage (a big deal!);
2. changes late in development imply wasted work.

Waterfall Model: Dealing with Change

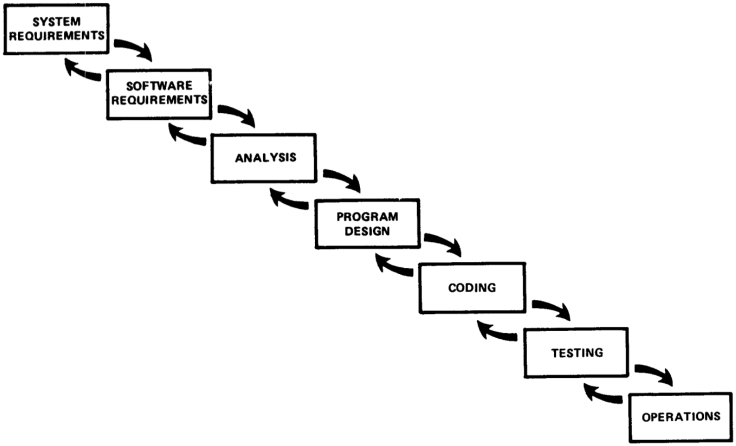


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

Waterfall Model: More Likely Scenario

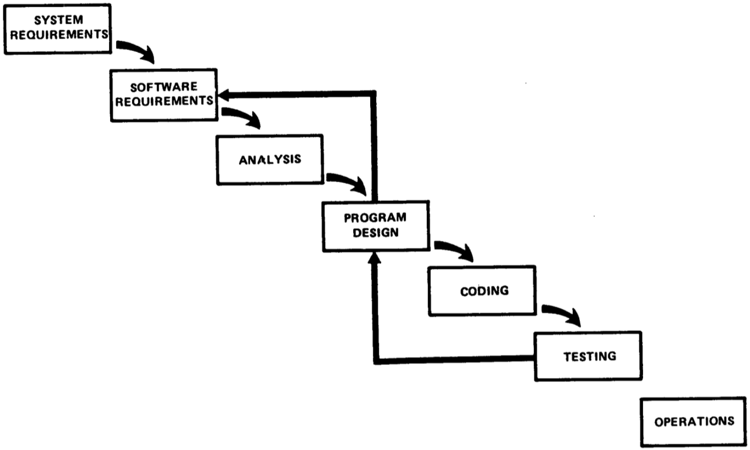


Figure 4. Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps.

Waterfall Model

It's a strawman.

Harper's Weekly, September 22, 1900, p. 881.

No one seriously advocates this model.

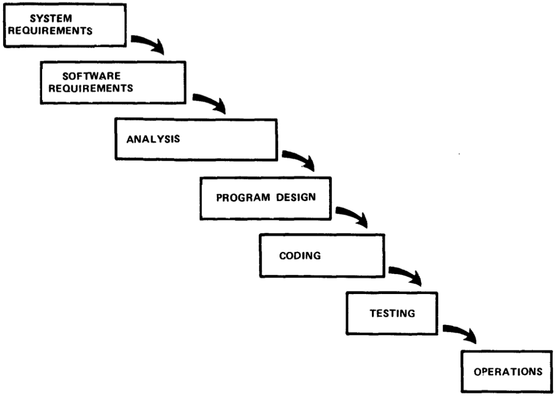
Concurrent Engineering

Also known as sashimi model:

Wikimedia commons, credit Suguri_F

Concurrent Engineering: a More Realistic Waterfall

Don’t wait on the previous stage to finish:
start the next stage as soon as possible.
(hence, sashimi).



Key idea: Why wait?
Using a product is a good way to refine it.

Concurrent Engineering: Advantages and Disadvantages

Advantages:

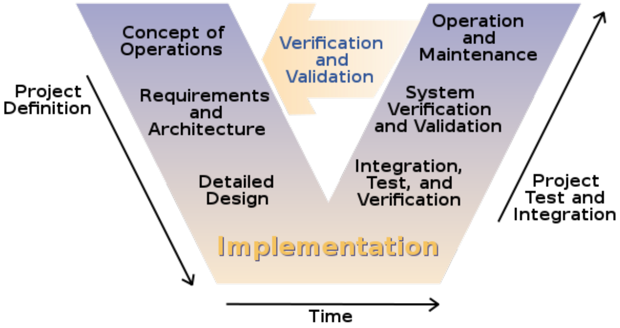
1. because you don’t need to write down every last (irrelevant) detail, might need less documentation;
2. projects need not be subdivided into smaller projects;
3. testing and use may reveal problems earlier.

Disadvantages:

1. milestones may be more ambiguous;
2. progress is more difficult to track:
   how done is stage *x*, anyway?;
3. poor communication more likely to lead to disaster.

The V-Model

Take Waterfall and link the stages horizontally.


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

Key idea: Make links explicit

V-Model Advantages and Disadvantages

Advantages:

1. Another attempt at a more realistic description of Waterfall
2. Otherwise the same as the any other variant of Waterfall

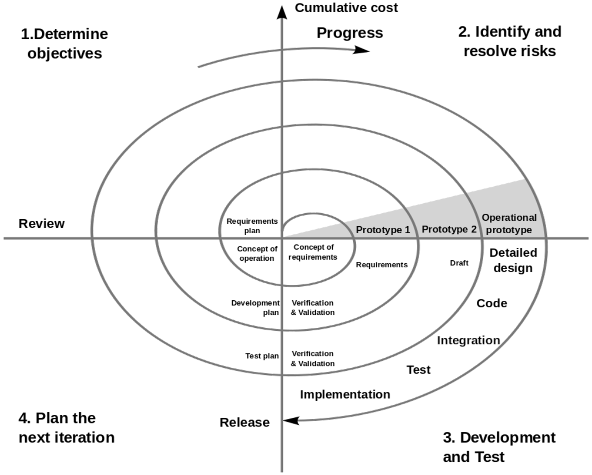
Disadvantages:

1. the same as any other variant of Waterfall

Spiral Model

Iterate the waterfall.

Spiral Model: Diagram



Spiral Model: Explanation

Iterate through stages, in order, until you get to a satisfactory solution.

Projects split into smaller sub-projects;
each iteration corresponds to a smaller project.

Iterate many times.

Not all stages require equal effort:
testing often harder than coding.

Risk-oriented model;
each sub-project addresses one or more risks (riskiest first),
until all of the major risks have been addressed.

Spiral Model: Advantages and Disadvantages

Advantage:

- addresses the biggest risks first,
  when changes are least expensive;
- progress visible to customer & management.

Disadvantage:

- some projects don't have clearly identifiable sub-projects with verifiable milestones.

Summary

In industry: accepted that Waterfall is obsolete

...if not outright harmful.

Iterations are the accepted method for development.

Now: we'll look at iterations in more detail.

Part II

Software Lifecycle Tactics

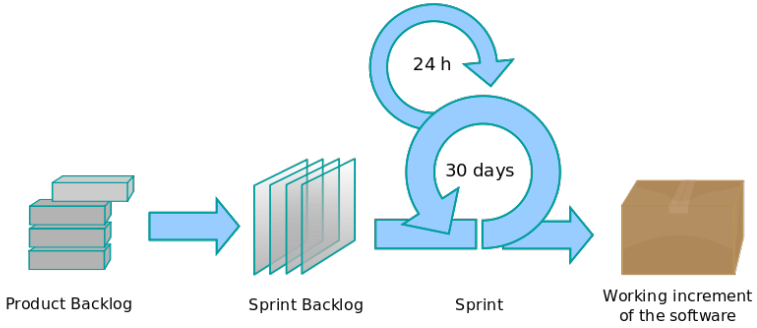
Some Software Lifecycle Tactics

- Cowboy Coding
- Scrum
- Test-Driven Development
- Behaviour-Driven Development
- Kanban

Scrum Model

Short iterations.

Scrum Model: Diagram



Scrum Model: Explanation

Break the work down into a series of short iterations.
Usually around 30 days.

Strictly defined iterations (in terms of features and time).

Daily meetings to co-ordinate.

Collect feedback for the next sprint.

Scrum Model: Advantages and Disadvantages

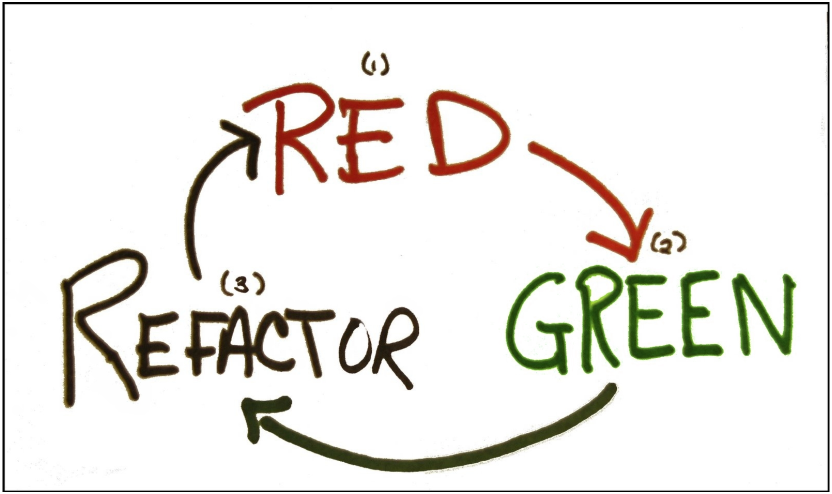
Advantages:

1. Short iterations mean lots of opportunities for input and feedback.
2. Daily meetings mean lots of co-ordination between team members.
3. It encourages breaking the software down into manageable units.

Disadvantages:

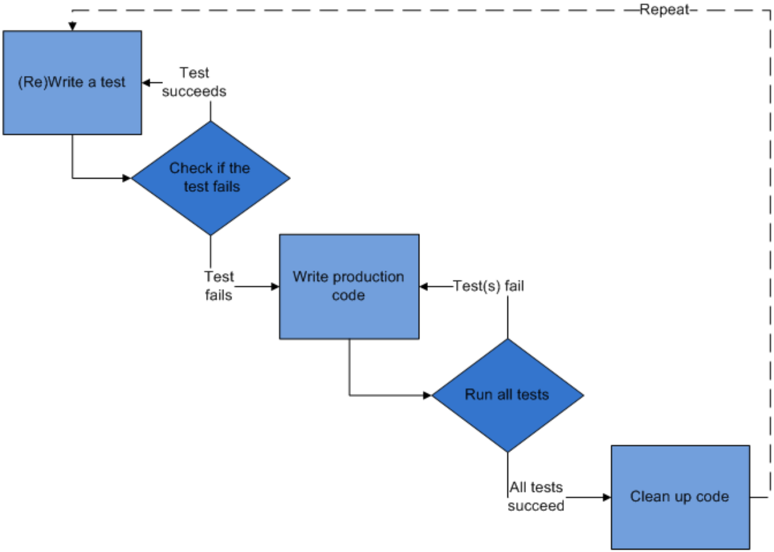
1. It does not scale well to large teams.
2. Daily meetings can result in excessive overhead.
3. At the sprint deadline, development ends whether the code is finished or not!

Test-Driven Development



Write the tests first.

TDD: Diagram


Repeat
(Re)Write a test
Test succeeds
Check if the test fails
Test fails
Write production code
Test(s) fail
Run all tests
All tests succeed
Clean up code

TDD: Explanation

Write the tests first, then the code.

Create a test for a new feature. It should fail.

Develop the code until the test passes.

Then, refactor (clean up) the code

More time spent on testing, with the goal of saving time in the long run.

TDD: Advantages

Advantages:

1. This model emphasizes testing in a way that other models do not.
   When time is short and the product needs to be released, testing is usually cut... TDD prevents this.
2. More code will be covered by the tests.
3. It encourages breaking the software down into testable units.

TDD: Disadvantages

Disadvantages:

1. An error in the code may be undetected because of a similar error in the unit test.
2. Not everything is testable.
3. TDD focuses only on unit tests, and this is not the only kind of testing.
4. Redundant or inflexible tests.
5. Tendency to disable broken tests or implement a quick change to “fix” the test.
6. Passing tests are not the same thing as functioning, useful software.

About Extreme Programming (XP)

Another software lifecycle model, but an outlier.
Most resembles spiral model, but scaled down & more agile.

Agile: Take "good" parts of good programming practice

(e.g. reviews, testing)

and "crank up all the knobs to 10".

Leave everything else behind.

XP is one of several agile methodologies:
all attempt to be less bureaucratic than the traditional "heavyweight" methodologies.

XP Values

XP comes with a set of values:

- Communication
- Simplicity
- Feedback
- Courage
- Respect

XP: Basic Activities

Four basic activities:

- coding;
- testing;
- listening; and
- designing.

XP: Coding

The code is central.
(not requirements docs, specifications)

XP: try to get working code out as soon as possible.
(even code with limited scope).

Programmers produce code in pairs.

Code runs, but also serves as main communication and experimentation medium.

XP: Testing

XP advocates test-driven development, as we've seen:

- first, write the test;
- make sure test fails;
- implement simplest possible solution;
- make sure test passes.

Code must always pass all of the unit tests.

Also, acceptance tests (more below).

XP: Listening

General problem:

Is the code doing the right thing?

XP solution: Acceptance tests,
created by on-site customer.

Also: developers must listen to business people
and vice-versa.

XP: Designing

No big up-front design.

Create a design incrementally by constantly re-factoring code as written (more later).

XP: Advantages

Can help avoid getting caught in bureaucratic tarpits;

When you have a good team, XP should deliver good results:
- get simpler designs which solve the appropriate problems;
- respond well to changes in requirements.

XP: Disadvantages and Controversies

Per Kent Beck:
XP works best when one uses all of the practices together.

Some of the practices can work alone,
like test-driven development.

Others may not work as well in isolation.
("... ring of poisonous snakes, daisy-chained together.")

XP tends to work best with smaller-sized groups
($<$ 12 members).
Lack of up-front design and requirements specifications can be worrisome.

Kanban

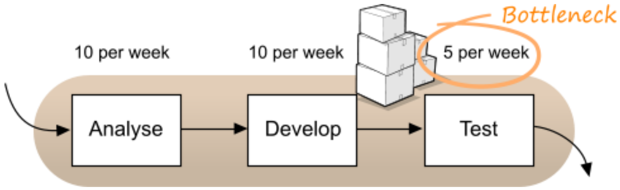
Kanban – Japanese word for “Visual Card”

Comes from Toyota’s car manufacturing process.

Support non-centralized production control:
work is pulled, not pushed.

Kanban: Identify Bottlenecks

Bottleneck: Stage of production that limits rate of output.



Temptation: cut corners to reduce backlog.

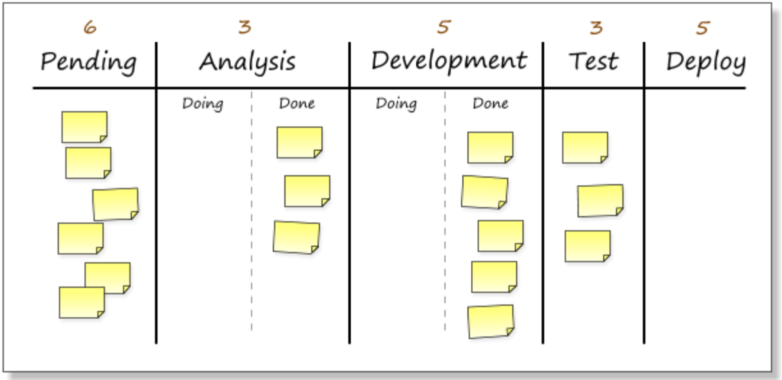
Kanban: Cards and Board

Physical or virtual cards represented on a board that has categories.

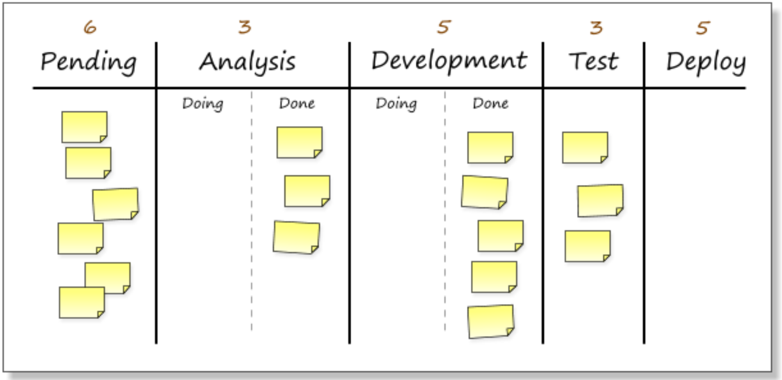
A card represents a unit of work.

Cards cannot advance until there is space farther along.

Kanban Board Example

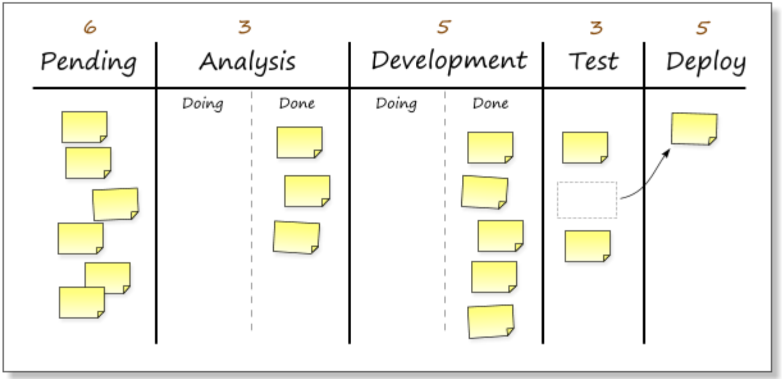


Kanban Board Example



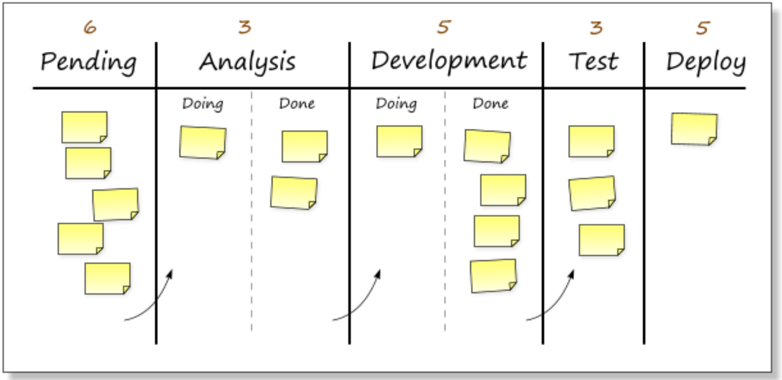
Backlog at the “Test” stage. How to address this?

Kanban Board Example



Testers finish a feature and it moves to “deploy”...

Kanban Board Example



... and now the other stages may advance.

Kanban: Advantages & Disadvangages

Advantages:

- Make bottlenecks visible.
- Prevent overloading of any stage.
- Estimation is not present.

Disadvantages:

- Stoppage in one part of the process can stop many others
- Estimation can be valuable.
- No commitment of features to versions.

Part III

Choosing a Model

Design and Planning

Traditionally:

- solicit requirements;
- make a design;
- analyze the design (with calculus);
- stamp and sign the plan.

Someone else builds the plan.

People tried this for software as well:

- solicit requirements;
- make UML diagrams;

and hire code monkeys to implement the design.

This works poorly.

The Role of Prototyping

The management question, therefore, is not whether *to build a pilot system and throw it away. You* will *do that. [...] Hence* plan to throw one away; you will, anyhow.

Frederick Brooks, *The Mythical Man-Month*, 1975.

The biggest reason the waterfall is a straw-man:

- You can never build a system without a prototype, because:
- You never really know what you need until you see it.
- Also, needs change over time.

Solution: iteration.
Different models have different iteration strategies.

Agile Lifecycle Models

Key point: handling change on-demand. Incorporate change into fast iteration process.

This requires suitable developers.

- Agile models are always collaborative.
  Hence, allegedly better at dealing with:
  1. changes to team over time; and
  2. differences between developer experience levels.

Choosing a Lifecycle Model

As we've said, we need to control the pace of iteration:

- Large teams, diverse stakeholder groups: slower iterations;
- Small teams, uncertain requirements, complex technologies: faster iterations.

Questions to Think About

- Do you understand customer requirements?
- Will you need to make major architectural changes?
- How reliable does the system need to be?
- How much future expansion and growth do you foresee?
- How risky is the project?

More Questions to Think About

- Is the schedule heavily-constrained?
- What is going to change during development?
- How much do customers need visible progress?
- How much does management need visible progress?
- What experience does the design team bring?

Software Process Improvement

Engineer the software development process:

- first, figure out what you mean to be doing—
  document your software process.
- next, figure out what you're actually doing—
  ensure that you're following existing process.
- finally, identify areas of potential improvement, and implement them.

Continuous process improvement: constant review. Beyond the scope of an individual project, but rather across projects. Allegedly: yield dramatic increases in development capability.