Engineering Design w/Embedded Systems

Lecture 21, 23—Software Lifecycle Models

Patrick Lam
University of Waterloo

March 7 & 11, 2013

Life Without Models

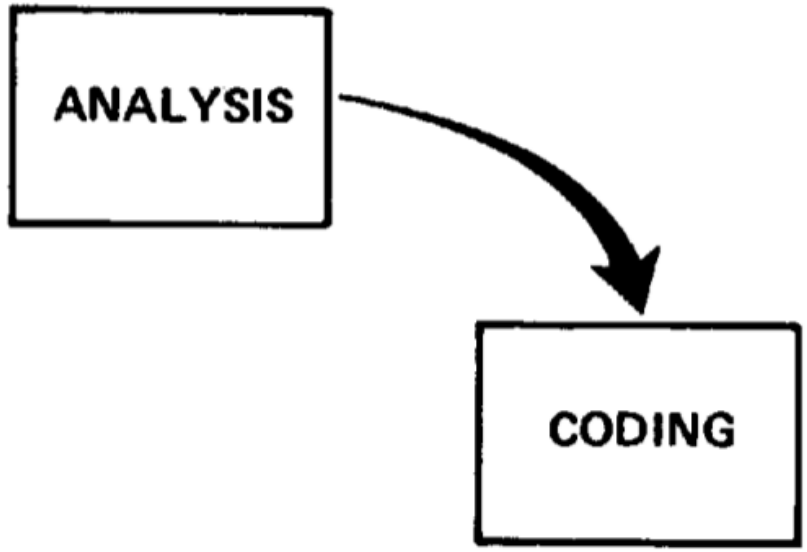
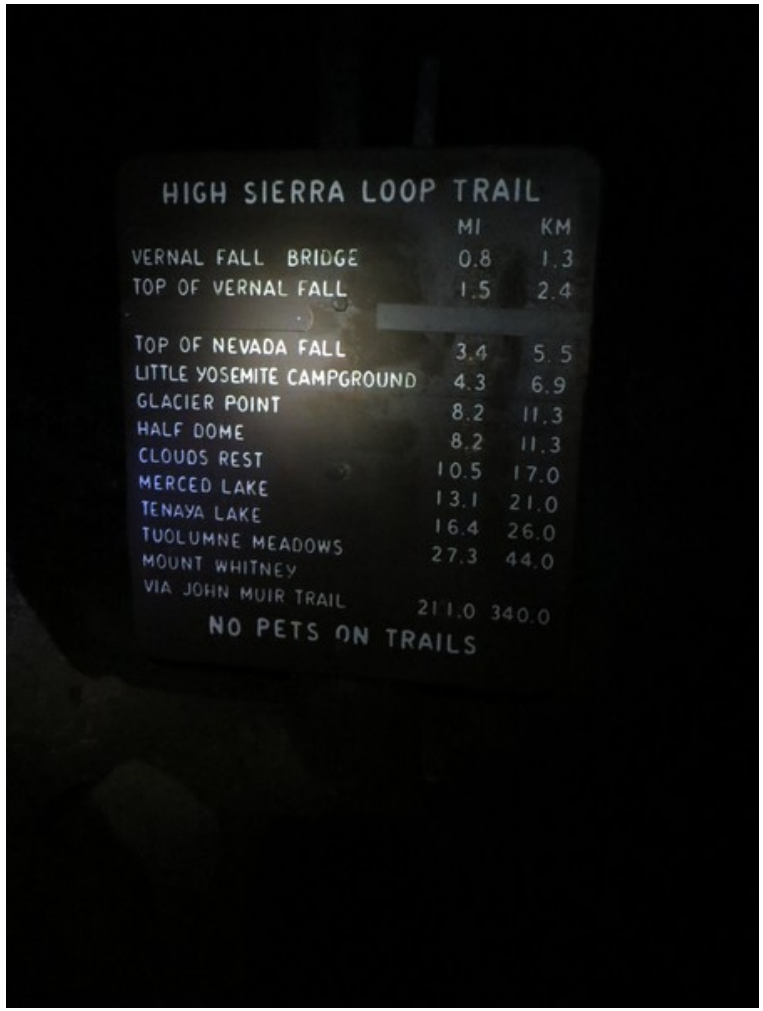


Figure 1. Implementation steps to deliver a small computer program for internal operations.

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

Deathmarches and Fiascoes

Software project management is hard.



Software development lifecycle models try to avoid deathmarches and fiascoes.

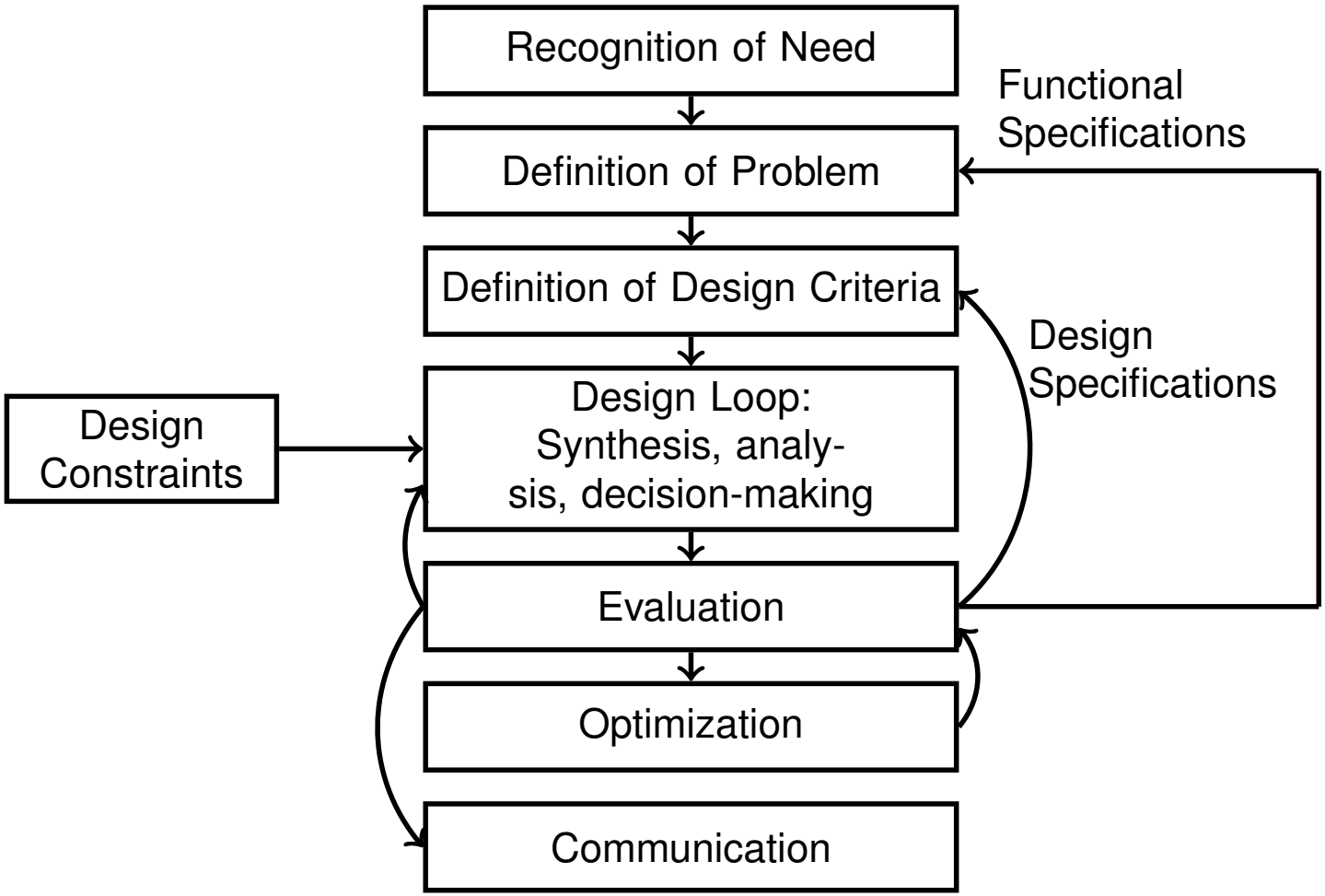
Iterations

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

Design is iterative.

Lifecycle models help organize the iterations.

Recall the Engineering Design Process



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?

Four Representative Software Lifecycle Models

- Waterfall
- Spiral
- Concurrent Engineering
- Extreme Programming

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

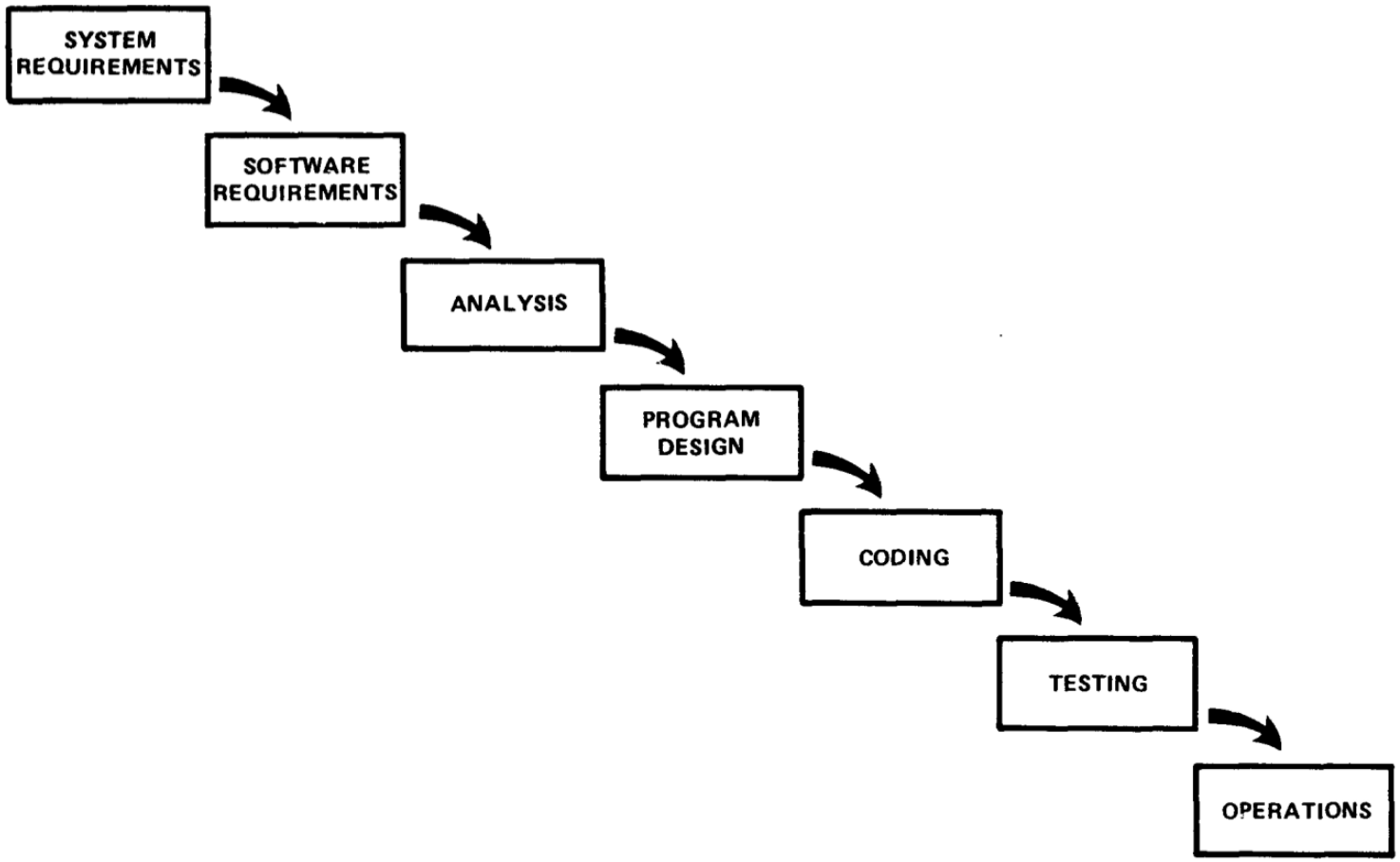


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

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

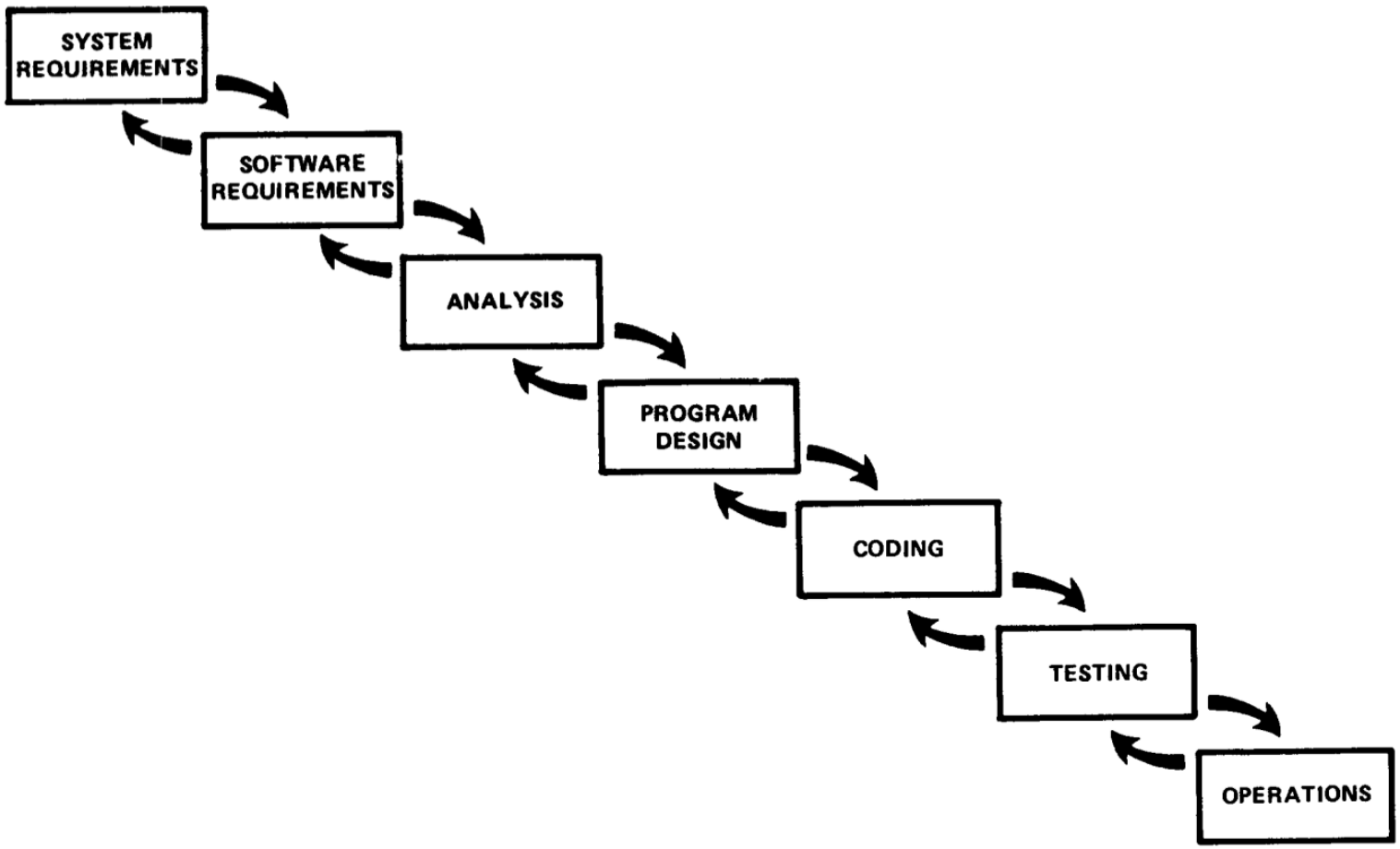


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

Waterfall Model: More Likely Scenario

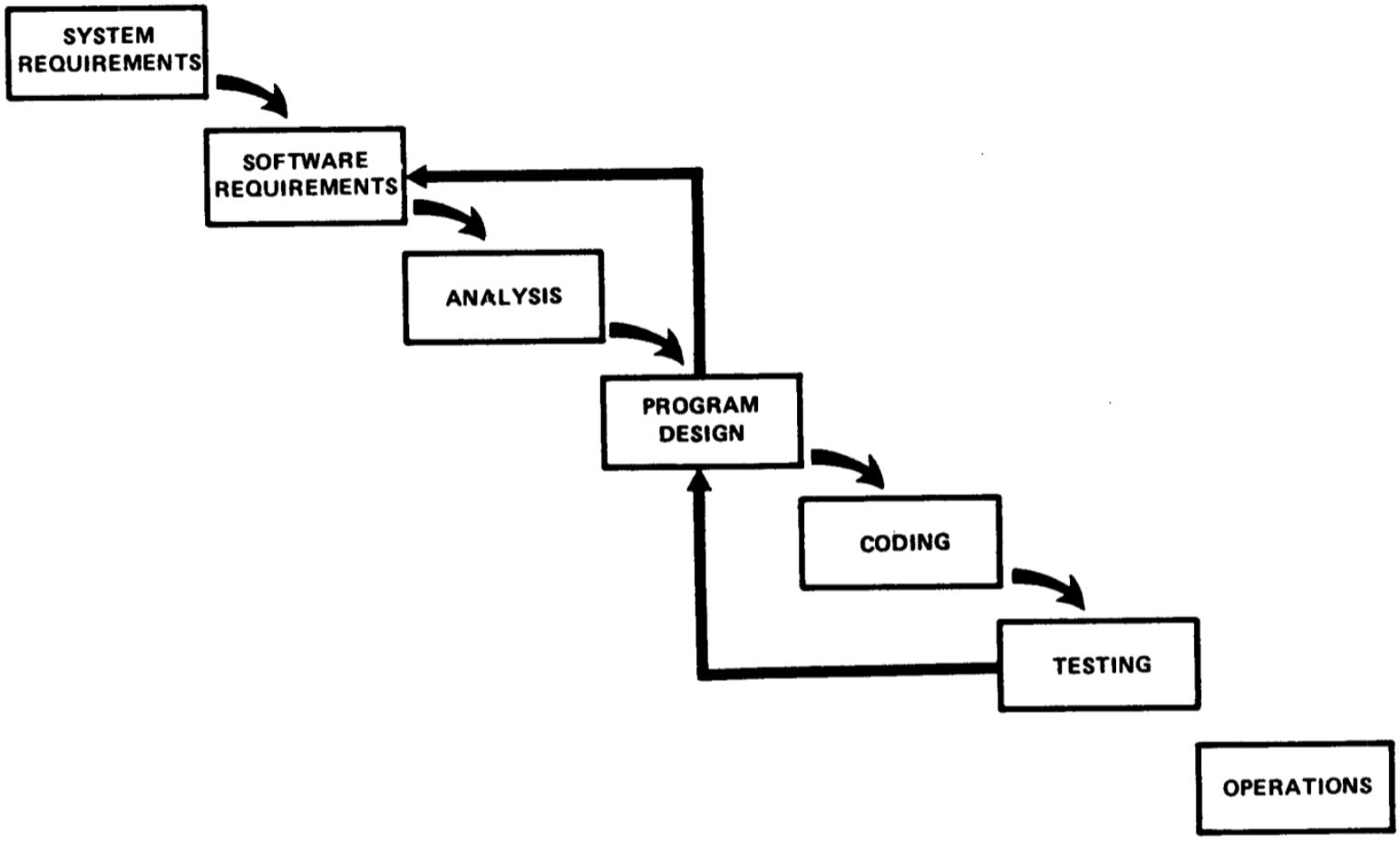


Figure 4. Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps.

Waterfall Model

It's a strawman.

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

No one seriously advocates this model.

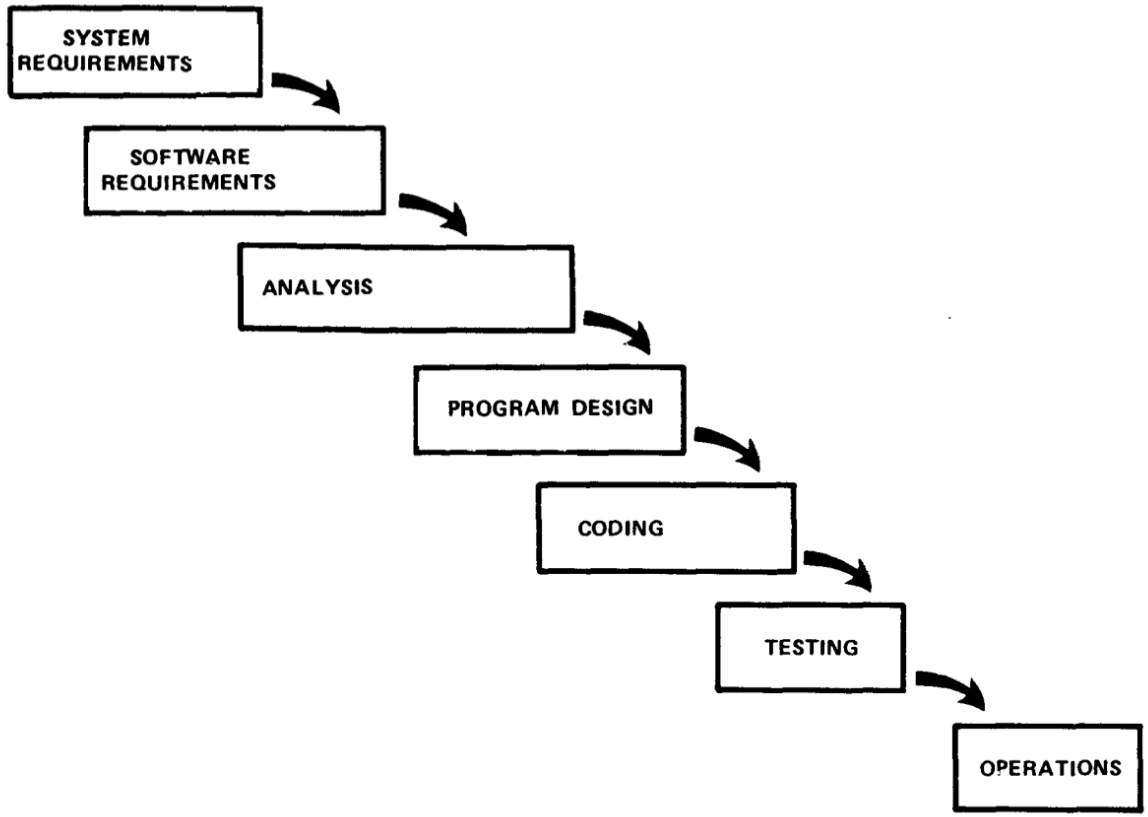
Concurrent Engineering

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.

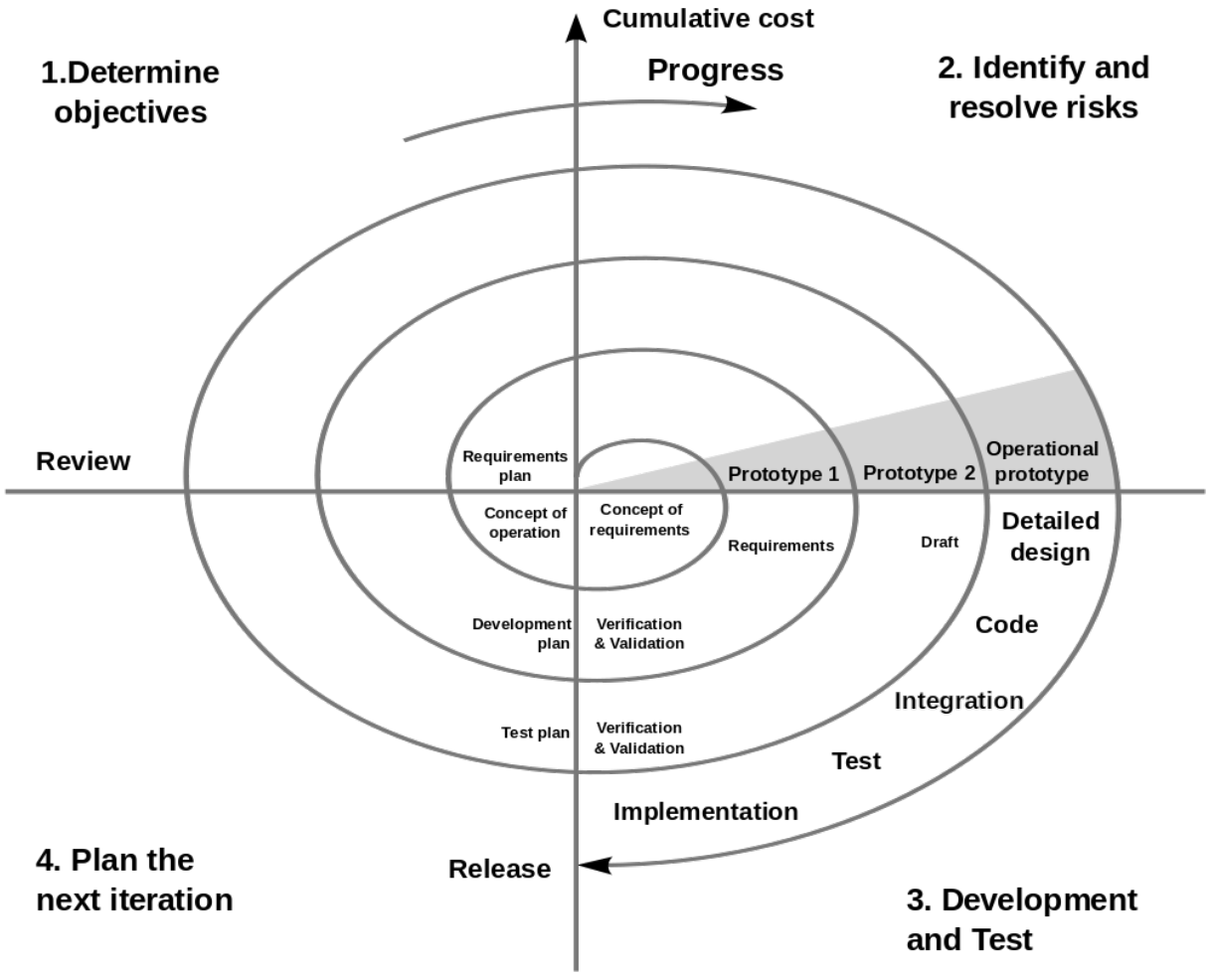
Disadvantages:

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.

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.

Extreme Programming

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.