CS 4500
Software Development

[Sofware Construction Process]

Ferdinand Vesely
(based on notes by Matthias Felleisen)

September 6, 2019

Today

- Software development processes
- Motivation
- Overview of standard ones
- Our approach

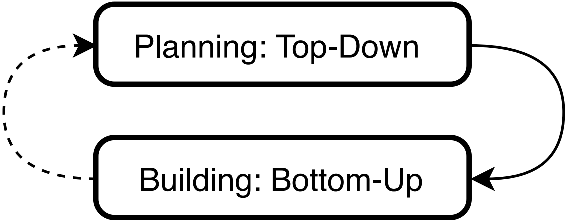
Plan Top-Down, Build Bottom-Up

Developing Systems

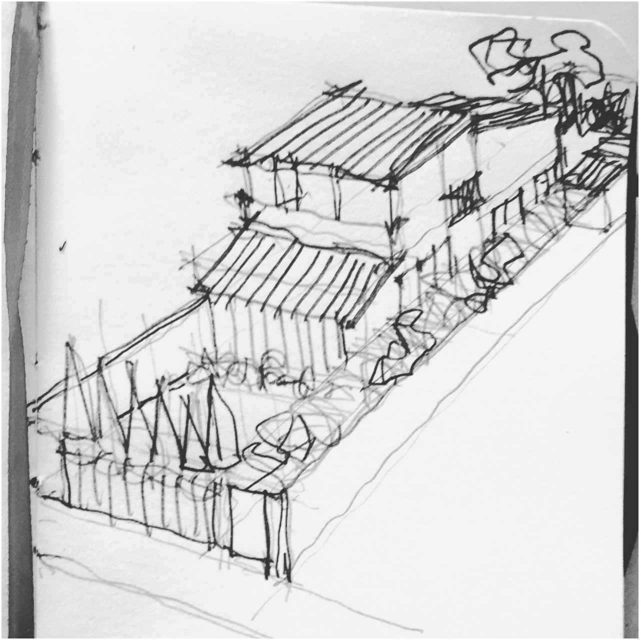
Creating a non-trivial system of cooperating pieces:

1. Identify pieces
2. Figure out how they fit together/interact
3. Devise plan of how to build the individual pieces and integrate them

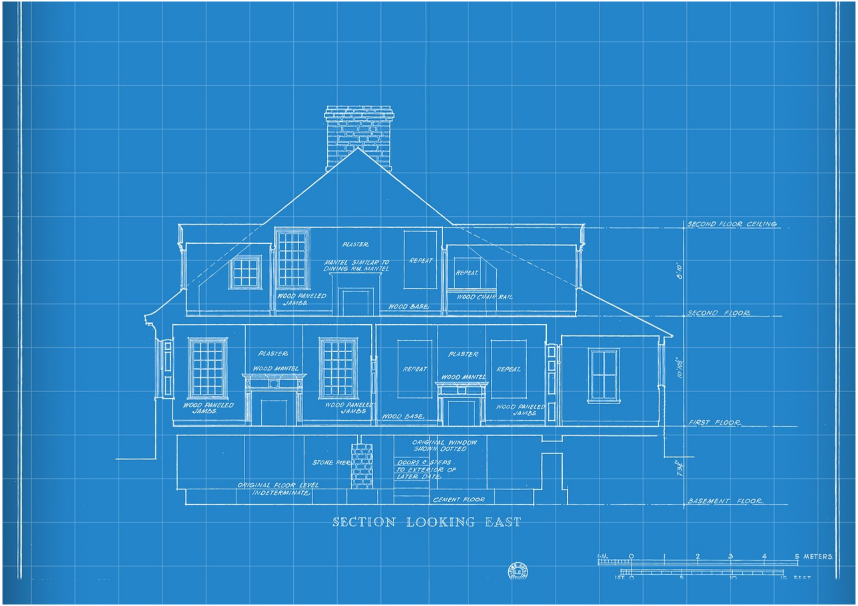
Well established, proven way to go about this all:

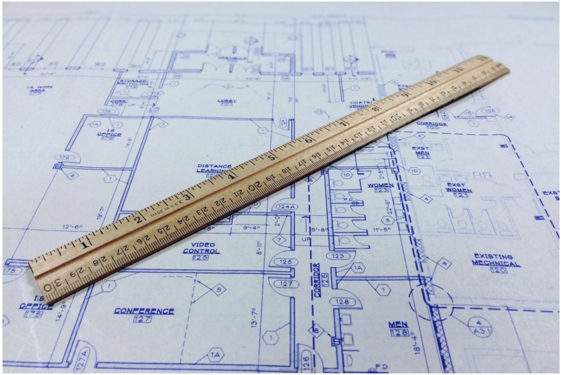


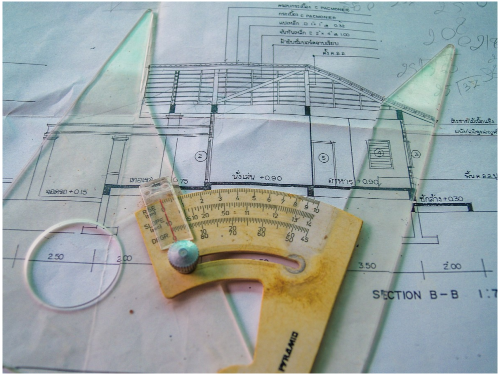
Big Picture – Sketch



Planning – Blueprints




I.S. WORK AREA
STORAGE/RECYCLING
LOBBY
COATS/VENDING
CORRIDOR
I.S. OFFICE
DISTANCE LEARNING
EXST. MEN
EXST. WOMEN
WOMEN 123
VIDEO CONTROL 125
EXISTING MECHANICAL 120
CONFERENCE 127
MEN 128
CORRIDOR 126
I.S. OFFICE 175
12'-6"
9'-4"
11'-5"
5'-8"
13'-7"
18'-4"
MADE IN U.S.A.


SECTION B-B 1:7
RISE
SLOPE
DEGR
2.50
2.00
3.50
2.00
+0.15
+0.90
+0.90
+0.30
PYRAMID

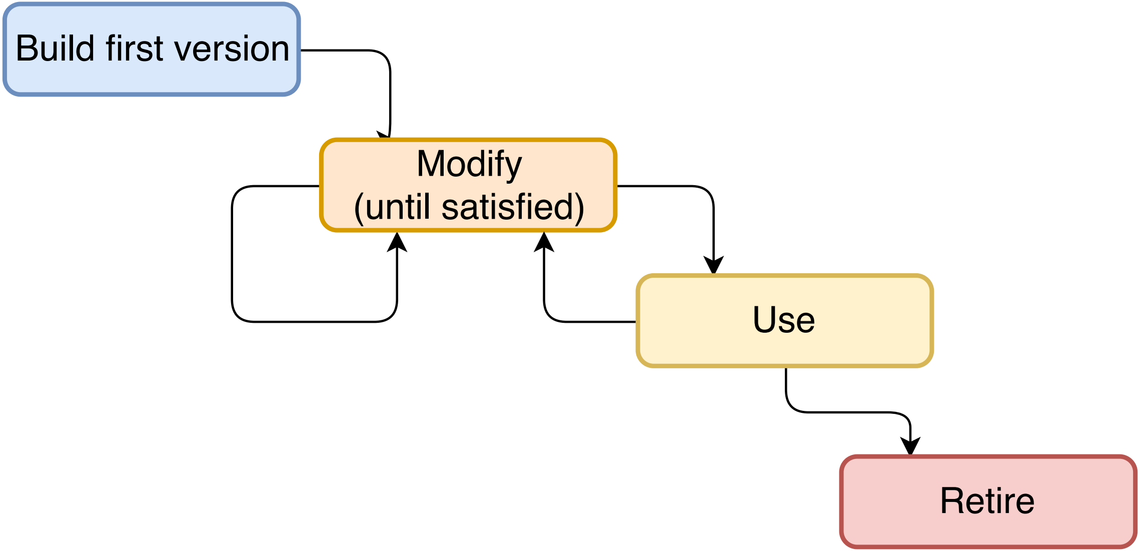
Terrasse 32.36 m²
Kuche 11.93 m²
Bad 5.58 m²
Wohnen 15.19 m²
Schlafen 14.85 m²
Zimmer 2
Zimmer 3
+2.90
2.17 m²
3.21 m²
3.90
4.15
8.40
9.00
3.14
3.53
1.94
3.67
7.74
2.00

Construction – Bottom-Up

- Still changing things on-the-fly
- Possibly changing the plans themselves

Software Process Models

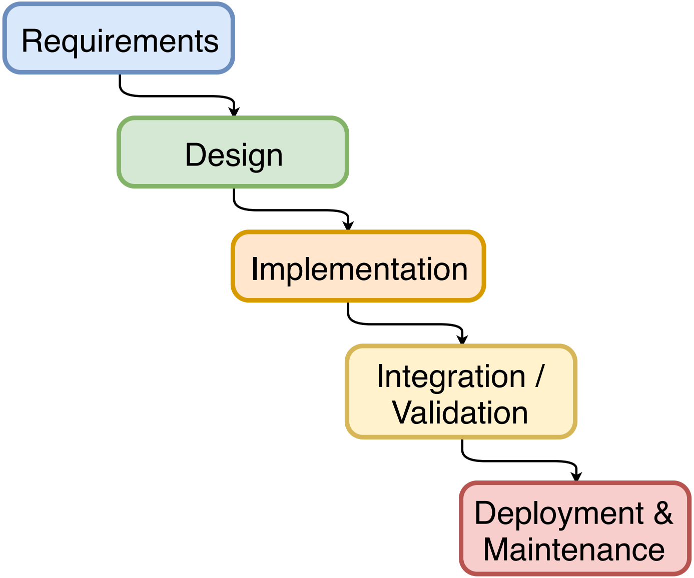
“Default”: Build and Fix



Question

If software was a house, what would the process look like?

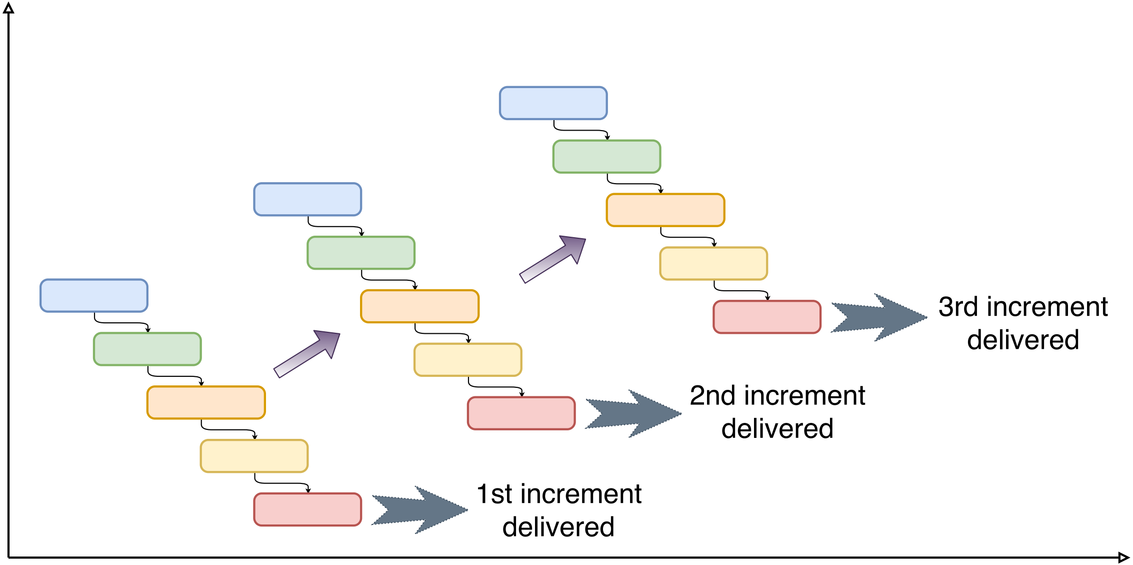
Waterfall Model



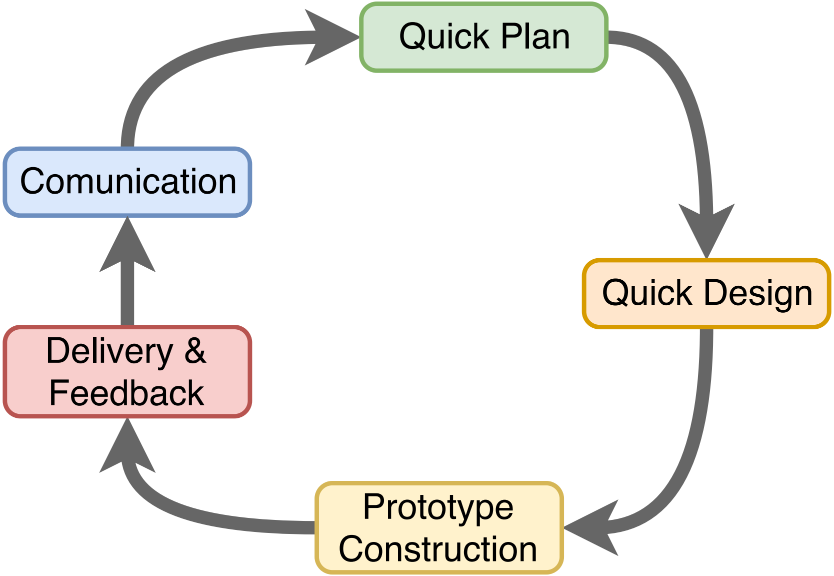
Waterfall Model

- A “Do your task and throw it over the wall” approach
- Each task – specialized team
- Once done – pass on, move on
- Minimal or no interaction between phases
- Any error can propagate downward 💣
- Customer needs to know what they want (Do they ever?)
- Does anybody actually develop software this way?

Alternative: Work in Increments



Evolutionary: Prototyping



Quick Plan

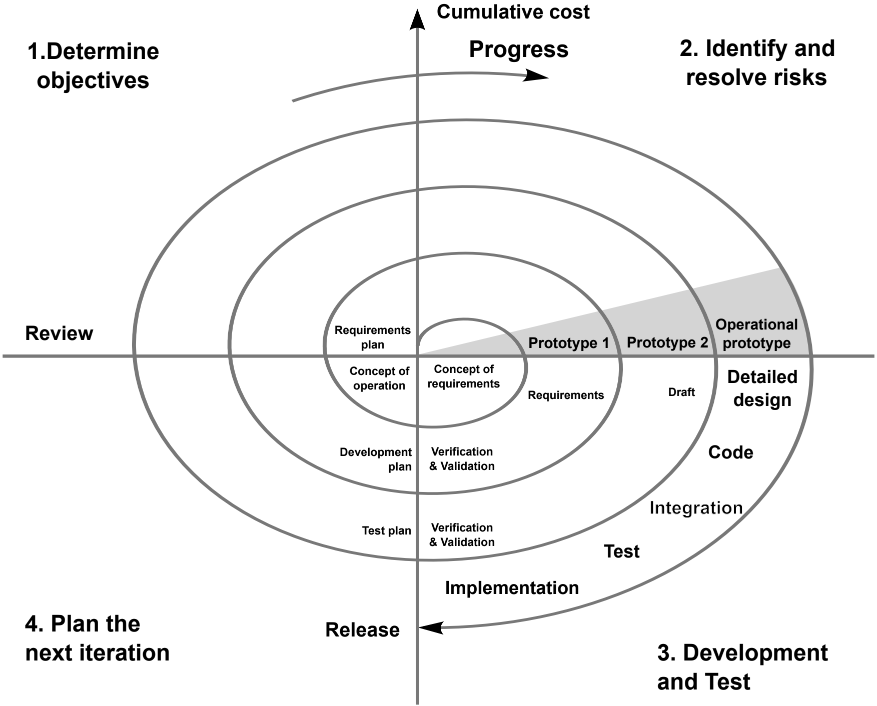
Quick Design

Prototype Construction

Delivery & Feedback

Comunication

Spiral


Cumulative cost
1.Determine objectives
Progress
2. Identify and resolve risks
Review
Requirements plan
Prototype 1
Prototype 2
Operational prototype
Concept of operation
Concept of requirements
Requirements
Draft
Detailed design
Code
Development plan
Verification & Validation
Integration
Test plan
Verification & Validation
Test
Implementation
4. Plan the next iteration
Release
3. Development and Test

Agile Approaches

Observations:

- Pervasiveness of *change*
- *Unpredictability*: of customer priorities, of development stages
- Software process: interleaving of design and construction

Need:

- Adaptability
- Incremental development

Agile

- Effective response to change (new team members, new technology, requirements)
- Effective communication
- Customer collaboration over contract negotiation
- Emphasize individuals and activities over processes and tools
- Incremental delivery – working software as rapidly as is feasible

Agility Principles

1. Priority: satisfy the customer – early & continuous delivery of valuable software.
2. Changing requirements welcome at any stage.
3. Frequent delivery of working software – weeks, months – shorter = better.
4. Daily collaboration of business people and developers.
5. Projects around motivated individuals.
6. Communication: face-to-face conversation.

Agility Principles

7. Primary measure of progress: working software.
8. Sustainable development. Maintain constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity: maximize amount of work **not done**.
11. Self-organizing teams lead to best architectures, requirements, and designs.
12. Self-reflecting teams.

See: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Agile Model Examples

- Extreme Programming
- SCRUM
- Adaptive Software Development
- ...

Our Process Principles

- We borrow elements from various approaches
- “Principal elements of development processes”

Step 1: Figure out what you want

- Start with a phrase describing the system
- Collect ideas around the phrase
  - Until any further extension produces ideas beyond the desired system
- Draw a line between
  1. Elements belonging to the system
  2. The rest
- 1 = the system, 2 = the environment

Step 2: Analyze use cases

Questions:

1. How does the environment initiate computation?
2. Where do responses go?

- Answer both
- Figure out what has to happen between answers to 1 and 2 → *use cases*
- Collect many use cases

Step 3: Identify software components and possible interactions

- Components represent knowledge and information
- Suggested by use cases
- Some components “know”, some components “need to know” → interfaces
- Information flow may need introducing additional components

Step 4: Plan a stripped-down prototype

Identify:

1. Most essential use case
2. Components needed to build a prototype realizing the use case

Components in 2 are to be built first (bottom-up), then integrated into a working prototype

Step 5: Iteratively refine the prototype

- Deal with more use cases
- Improve existing use cases
- Ensure: use cases reuse components, but do not interfere

Example: Grocery Store

Imagine:

- A small grocery store
- Wants to automate its “points of sales” and inventory management