Software Engineering

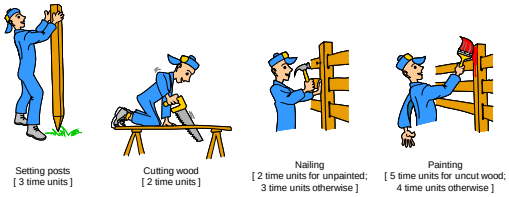
LECTURE 1: Introduction

Lecture time: 1 hr. 20 min.

Introduction: Software is Complex

- Complex ≠ complicated
- Complex = composed of many simple parts related to one another
- Complicated = not well understood, or explained

Complexity Example: Scheduling Fence Construction Tasks



Setting posts [3 time units]

Cutting wood [2 time units]

Nailing [2 time units for unpainted; 3 time units otherwise]

Painting [5 time units for uncut wood; 4 time units otherwise]

Setting posts < Nailing, Painting

Cutting < Nailing

…shortest possible completion time = ?

[⇒ "simple" problem, but hard to solve without a pen and paper]

More Complexity

Suppose today is Tuesday, November 29

What day will be on January 3?

[To answer, we need to bring the day names and the day numbers into coordination, and for that we may need again a pen and paper]

The Frog in Boiling Water

- Small problems tolerate complacency—lack of immediate penalty leads to inaction
- Negative feedback accumulates subtly and by the time it becomes painful, the problem is too big to address
- Frog in gradually heated wate analogy:
  - The problem with little things is that none of them is big enough to scare you into action, but they keep creeping up and by the time you get alarmed the problem is too difficult to handle
  - Consequently, "design smells" accumulate, "technical debt" grows, and the result is "software rot"

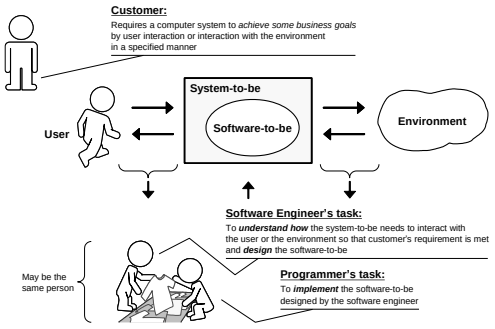
https://en.wikipedia.org/wiki/Design_smell
https://en.wikipedia.org/wiki/Technical_debt
https://en.wikipedia.org/wiki/Software_rot

The Role of Software Engg. (1)

A bridge from customer needs to programming implementation

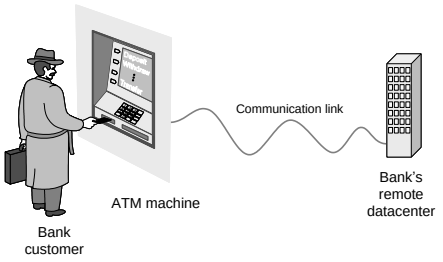
First law of software engineering
Software engineer is willing to learn the problem domain (problem cannot be solved without understanding it first)

The Role of Software Engg. (2)



Example: ATM Machine

Understanding the money-machine problem:

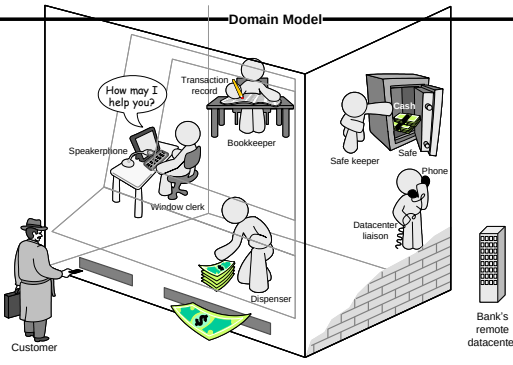


Problem-solving Strategy

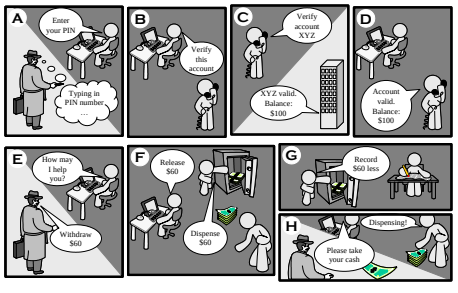
Divide-and-conquer:

- Identify logical parts of the system that each solves a part of the problem
- Easiest done with the help of a domain expert who already knows the steps in the process ("how it is currently done")
- Result:
  A Model of the Problem Domain
  (or "domain model")

How ATM Machine Might Work



Cartoon Strip: How ATM Machine Works



Software Engineering Blueprints

- Specifying software problems and solutions is like cartoon strip writing
- Unfortunately, most of us are not artists, so we will use something less exciting:
  UML symbols
- However …

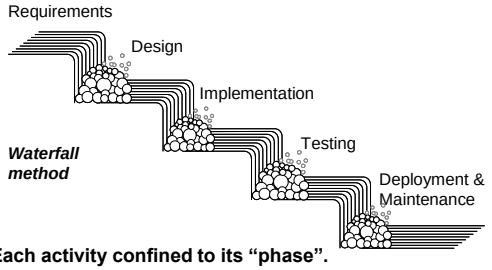
Second Law of Software Engineering

- **Software should be written for people first**
  - (Computers run software, but hardware quickly becomes outdated)
  - Useful + good software lives long
  - To nurture software, people must be able to understand it

Software Development Methods

- Method = work strategy
  - The Feynman Problem-Solving Algorithm: (i) Write down the problem (ii) think very hard, and (iii) write down the answer.
- Waterfall
  - Unidirectional, finish this step before moving to the next
- Iterative + Incremental
  - Develop increment of functionality, repeat in a feedback loop
- Agile
  - *Continuous* user feedback essential; feedback loops on several levels of granularity

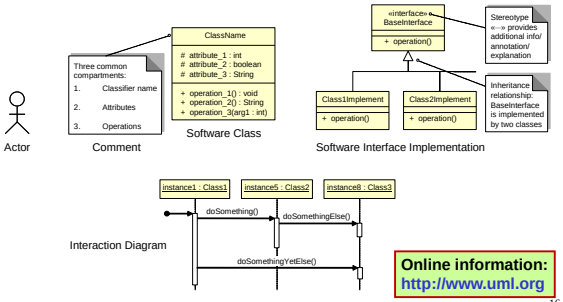
Waterfall Method



Each activity confined to its "phase".
Unidirectional, no way back;
finish this phase before moving to the next

UML – Language of Symbols

UML = Unified Modeling Language



Online information:
http://www.uml.org

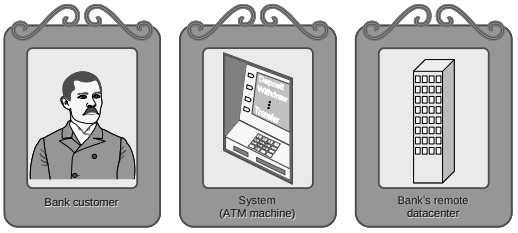
How Much Diagramming?

- Use **informal**, ad-hoc, **hand-drawn**, scruffy diagrams during early stages and within the development team
  - Hand-drawing forces economizing and leads to low emotional investment
    - Economizing focuses on the essential, most important considerations
      - Prioritize substance over the form
    - Not being invested facilitates critique and suggested modifications
  - Always take snapshot to preserve records for future
- Use **standardized**, neat, **computer-generated** diagrams when consensus reached and designs have "stabilized"
  - Standards like UML facilitate communication with broad range of stakeholders
  - But, invest effort to make neat and polished diagrams only when there is an agreement about the design, so this effort is worth doing
    - Invest in the form, only when the substance is worth such an investment

Understanding the Problem Domain

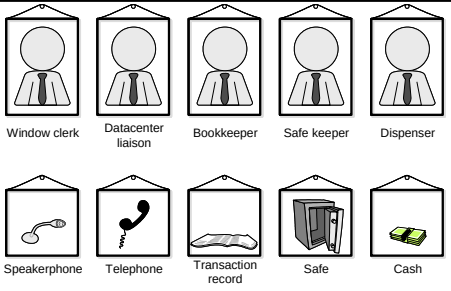
- System to be developed
- Actors
  - Agents external to the system that interact with it
- Concepts/ Objects
  - Agents working inside the system to make it function
- Use Cases
  - Scenarios for using the system

ATM: Gallery of Players



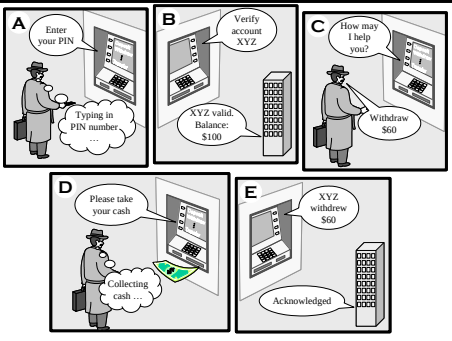
Actors (Easy to identify because they are visible!)

Gallery of Workers + Tools



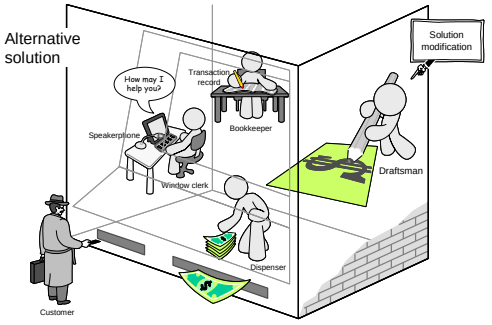
Concepts (Hard to identify because they are invisible/imaginary!)

Use Case: Withdraw Cash



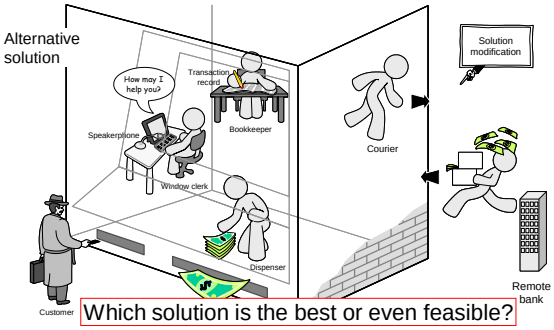
How ATM Machine Works (2)

Domain Model (2)



How ATM Machine Works (3)

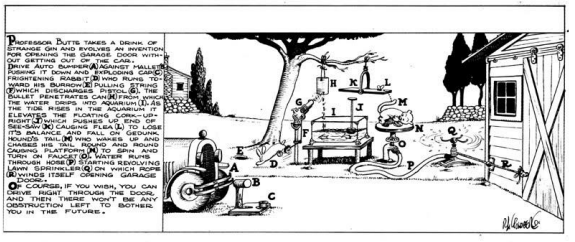
Domain Model (3)



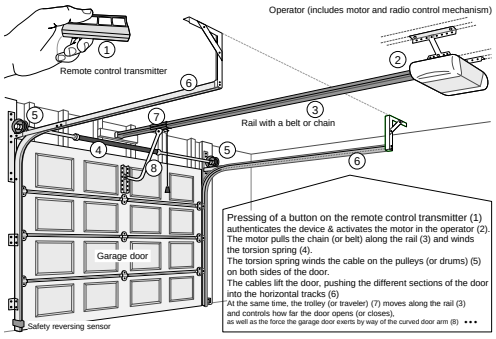
Which solution is the best or even feasible?

Rube Goldberg Design

Garage door opener



Actual Design



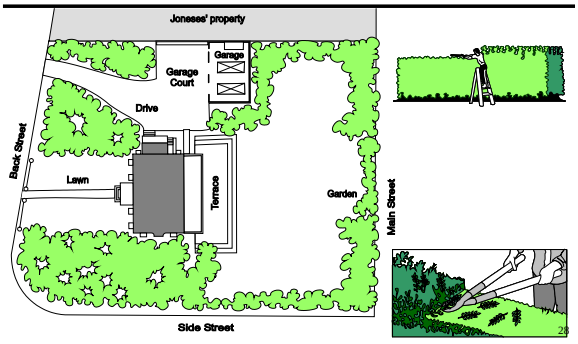
Feasibility & Quality of Designs

- Judging feasibility or quality of a design requires great deal of domain knowledge (and commonsense knowledge!)

Software Measurement

- What to measure?
  - Project (developer's work), for budgeting and scheduling
  - Product, for quality assessment

Formal hedge pruning

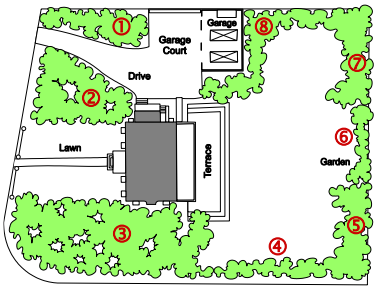


Work Estimation Strategy

1. Make initial guess for a little part of the work
2. Do a little work to find out how fast you can go
3. Make correction on your initial estimate
4. Repeat until no corrections are needed or work is completed

Sizing the Problem (1)

Step 1: Divide the problem into *small* & *similar* parts



Step 2: Estimate *relative* sizes of all parts

Size(①) = 4
Size(②) = 7
Size(③) = 10
Size(④) = 3
Size(⑤) = 4
Size(⑥) = 2
Size(⑦) = 4
Size(⑧) = 7

Sizing the Problem (2)

- Step 3: Estimate the size of the total work

$$\text{Total size} = \sum \text{points-for-section } \boldsymbol{i} \quad (i = 1..N)$$

- Step 4: Estimate speed of work (velocity)
- Step 5: Estimate the work duration

$$\text{Travel duration} = \frac{\text{Path size}}{\text{Travel velocity}}$$

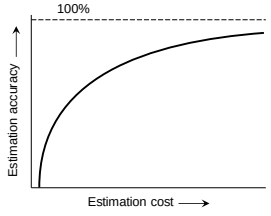
Sizing the Problem (3)

- Assumptions:
  - Relative size estimates are accurate
    - That's why parts should be small & similar-size!
- Advantages:
  - Velocity estimate may need to be adjusted (based on observed progress)
  - However, the total duration can be recomputed quickly
    - Provided that the relative size estimates of parts are accurate —accuracy easier achieved if the parts are small and similar-size

Unfortunately:

- Unlike hedges, software is mostly **invisible** and **does not exist** when project is started
  ➔ The initial estimate hugely depends on experience and imagination

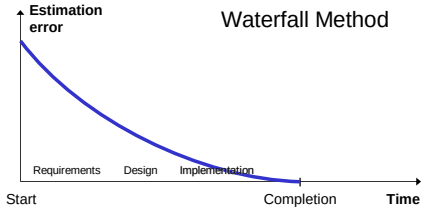
Exponential Cost of Estimation



- **Improving accuracy of estimation beyond a certain point requires huge cost and effort (known as the law of diminishing returns)**
- **In the beginning of the curve, a modest effort investment yields huge gains in accuracy**

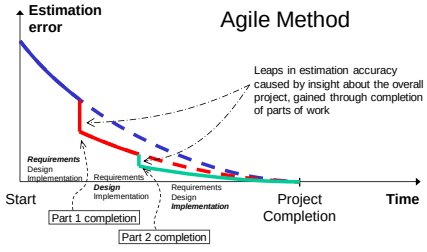
33

Estimation Error Over Time



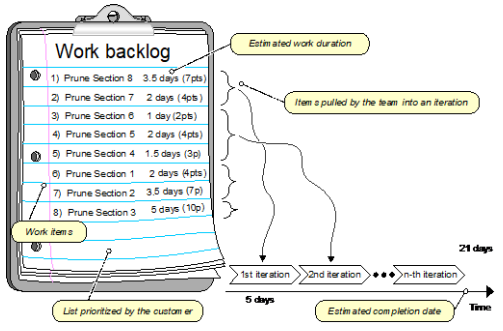
Waterfall method *cone of uncertainty* starts high and *gradually* converges to zero as the project approaches completion.

Estimation Error Over Time



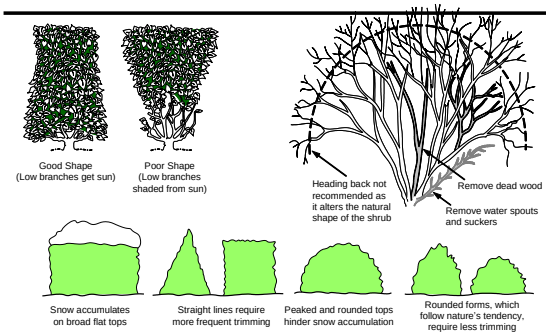
Agile method *cone of uncertainty* starts high and *in leaps* converges to zero as the project approaches completion.

Agile Project Effort Estimation



36

Measuring Quality of Work



37

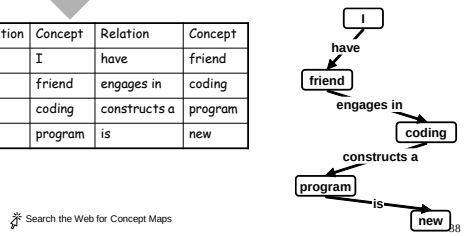
Concept Maps

Useful tool for problem domain description

SENTENCE: "My friend is coding a new program"

translated into propositions

Proposition	Concept	Relation	Concept
1.	I	have	friend
2.	friend	engages in	coding
3.	coding	constructs a	program
4.	program	is	new

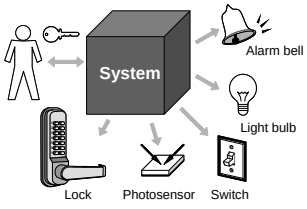


Search the Web for Concept Maps

38

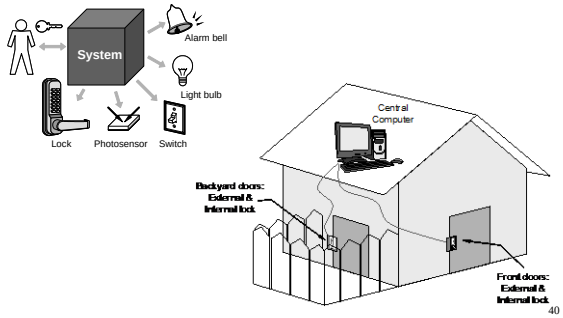
Case Study: Home Access Control

- Objective: Design an electronic system for:
  - Home access control
    - Locks and lighting operation
  - Intrusion detection and warning



39

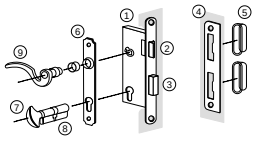
Case Study – More Details



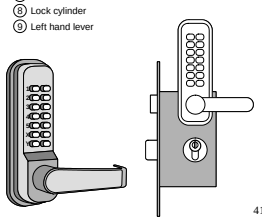
40

Know Your Problem

Mortise Lock Parts

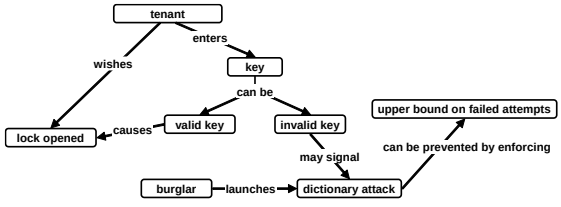


1. Lock case
2. Latch bolt
3. Dead bolt
4. Strike plate
5. Strike box
6. Protective plate
7. Thumb-turn
8. Lock cylinder
9. Left hand lever



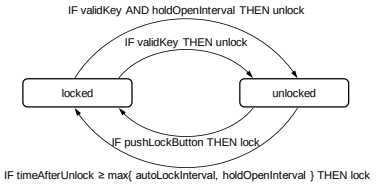
41

Concept Map for Home Access Control



42

States and Transition Rules



… what seemed a simple problem, now is becoming complex