Software Engineering

Intro to SE : Complexity

何明昕 HE Mingxin, Max

Send your email to c.max@yeah.net with a subject like: SE-id-*Andy: On What …*

Download from c.program@yeah.net

/文件中心/网盘/SoftwareEngineering2024

Exercise 01: Review & Preview (in 2 weeks)

- Read 1.1-1.2 and preview 1.3-1.4 of TextBook.
- Read ch1 of “Beginning Software Engineering”
- Read “Essence of Software Engineering”
- Preview ch1 of “Design Science Methodology for IS & SE”
- Preview ch1 of “A Philosophy of Software Design”

Send an email *in a week to* c.max@yeah.net with a subject like:

SE-id-Adam: Register for SE Course,

simply describe your expectation for the course and report your online learning context if you are learning online.

Introduction: Software is Complex

Complex (错综复杂的) ≠ complicated (复杂难懂的)

Complex = composed of many simple parts related to one another

Complicated = not well understood, or explained

Software Engineering is Complex and Complicated

1) 软件规模增长带来要素数量增长，熵增长必然导致混乱（热力学第三定律）
2) 准确定义需求很难（用户通常不知道要什么，只知道不要什么），需求蔓延是软件项目失败的首要原因
3) 不同的设计可以实现同样的功能(系统结构/行为/功能的关系），判断一个设计是否是好的设计很难（Conway's law: 软件的结构倾向于与开发团队的组织架构具相似性)

After explaining to a student through various lessons and examples that:

$$\lim_{x \to 8} \frac{1}{x-8} = \infty$$

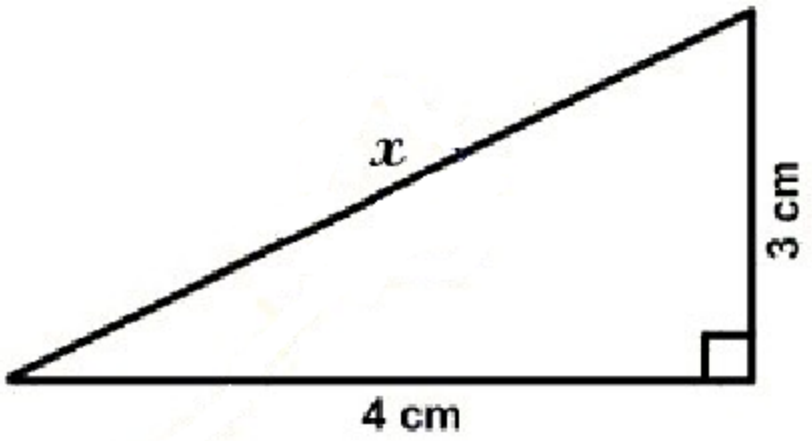
I gave a different example.

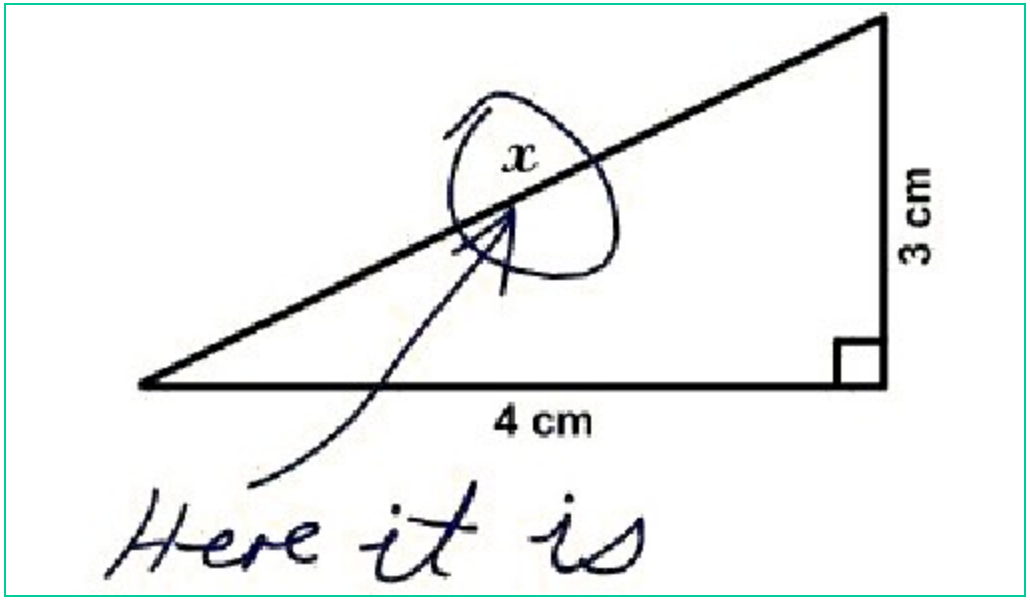
$$\lim_{x \to 5} \frac{1}{x-5} = \ ?$$

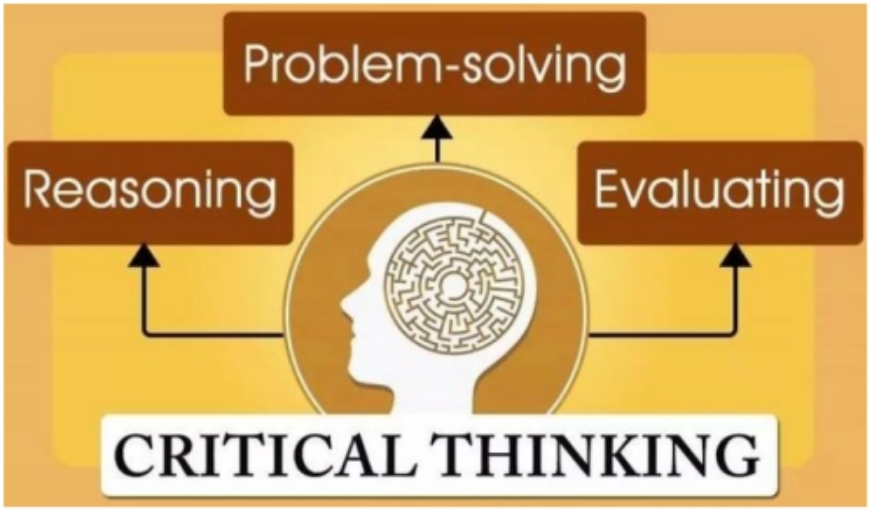
This was the result:

$$\lim_{x \to 5} \frac{1}{x-5} = \ 5$$

3. Find x.







- 批判性思维
  关键性思考
  建设性思考

- Critical Observing:
  建设性观察

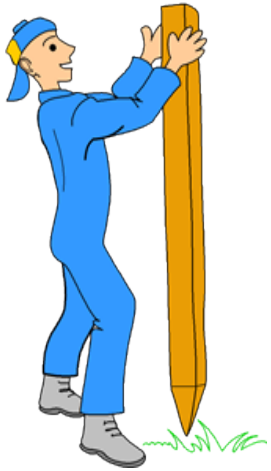
- 系统化认知问题 与
  构建解决方案的能力

Critical thinking is
a mental process involving the evaluation of arguments or statements to determine whether they are valid, accurate, or logical.

It involves the ability to think analytically, identify and assess arguments, draw conclusions, and make decisions.

Critical thinking is essential in many areas, such as problem-solving, decision-making, creativity, and research.

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 = ?

[$\Rightarrow$ “simple” problem, but hard to solve without a pen and paper]

More Complexity

We know that
Today is Wednesday, February 28

What day will be on July 1?

(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.)

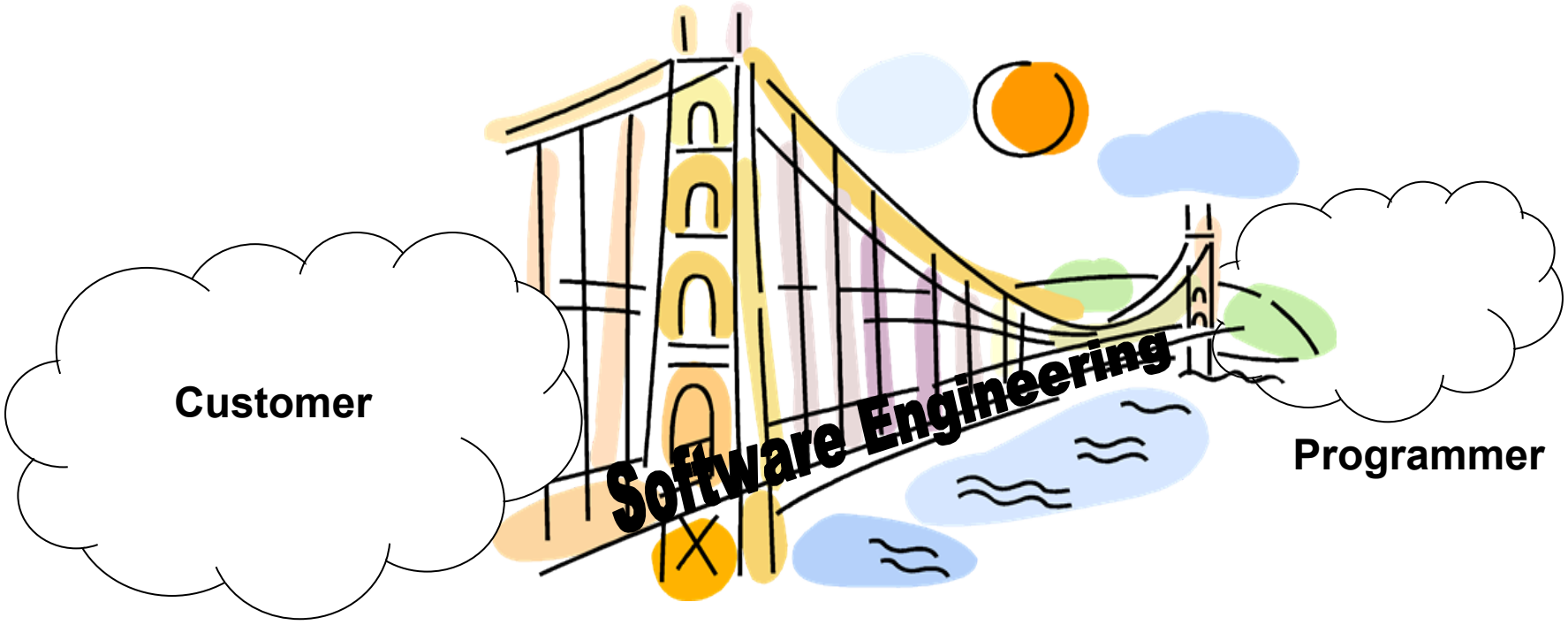
The Frog in Boiling Water

- Small problems tolerate complacency(自满)—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 water 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(设计 怪味)" accumulate, "technical debt (技术债务)" grows, and the result is "software rot(软件腐烂)"

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)

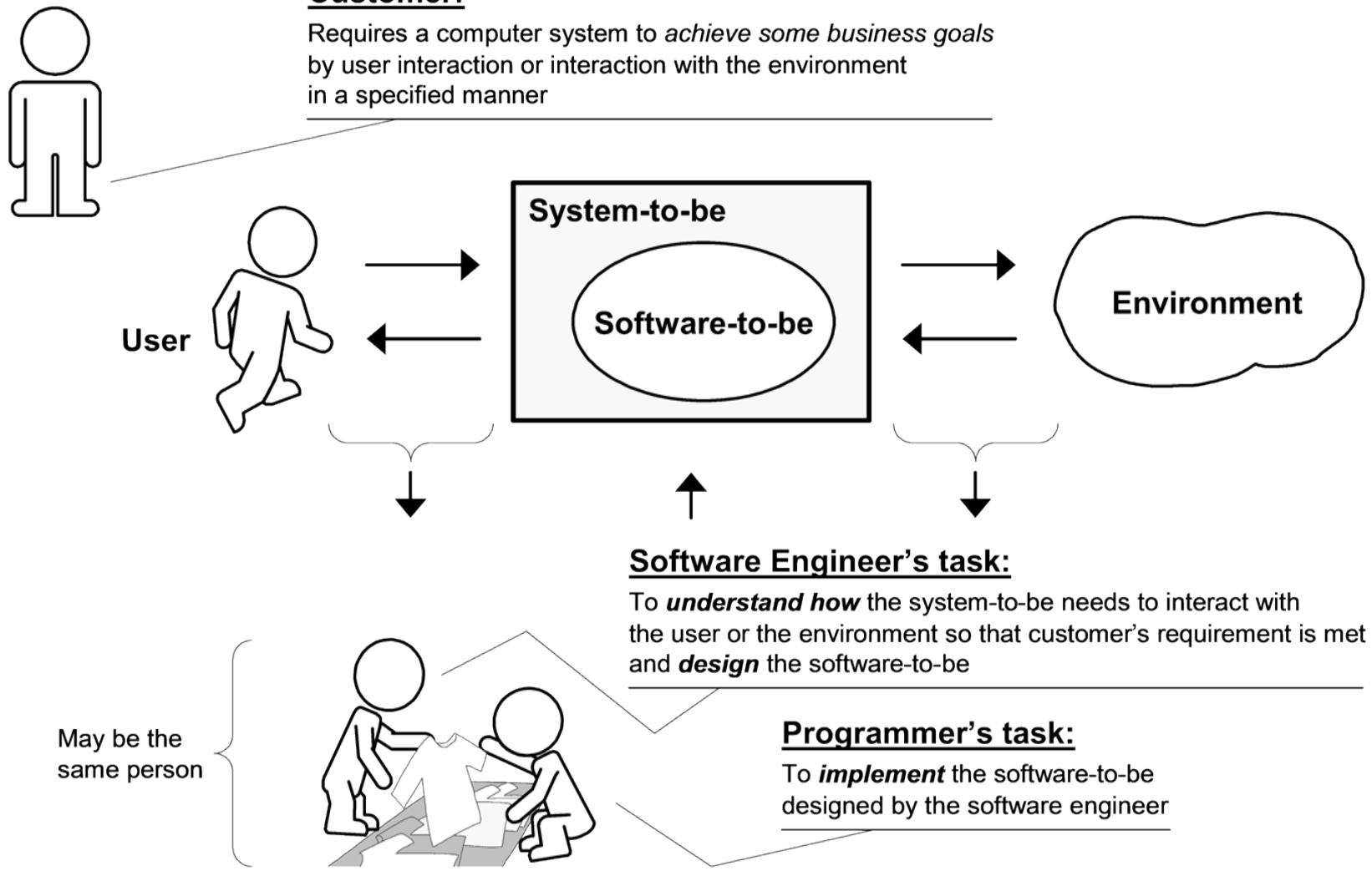
A bridge from customer needs to programming implementation



First law of software engineering

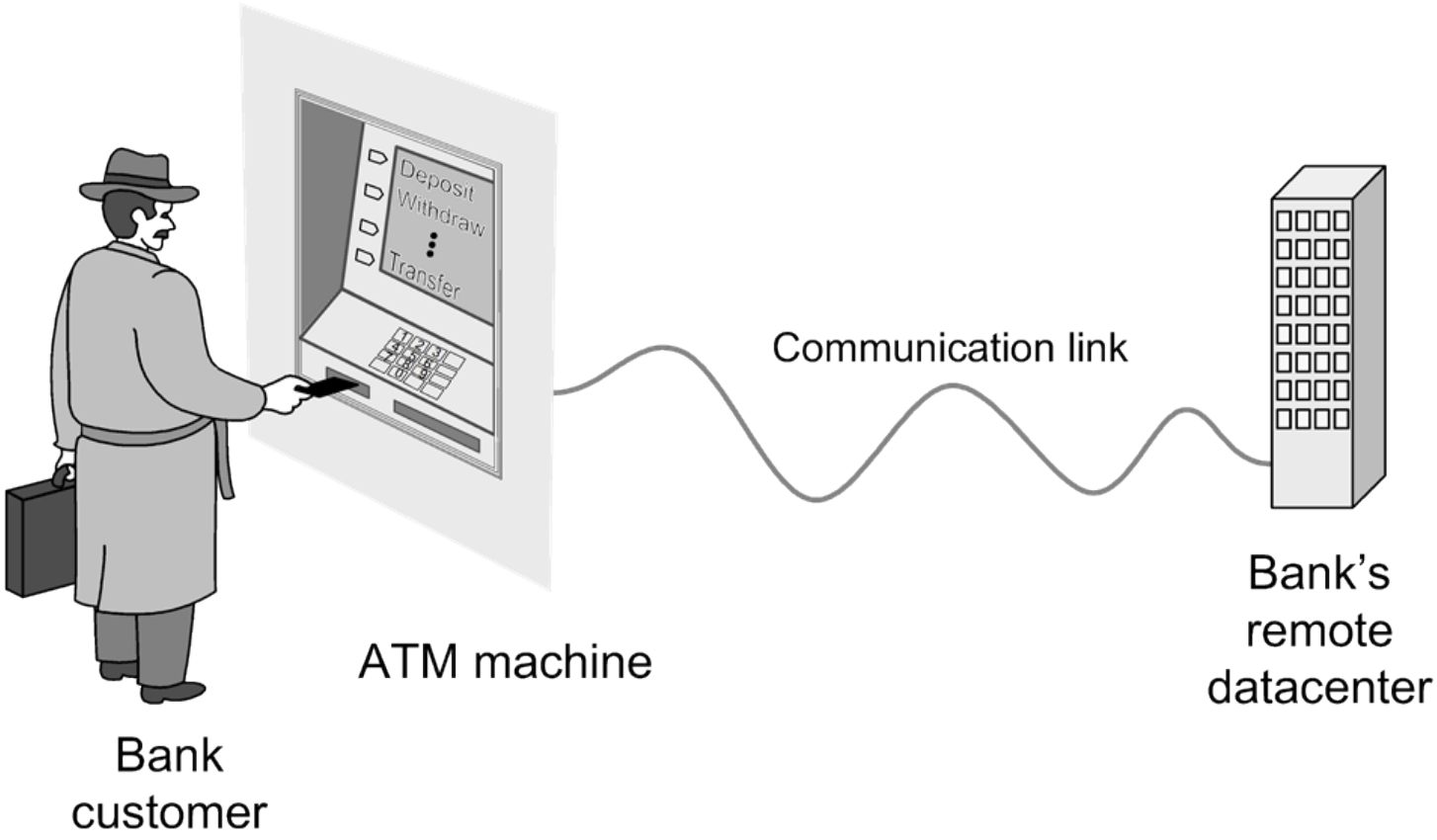
Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

The Role of Software Engg. (2)



Example: ATM Machine

Understanding the money-machine problem:



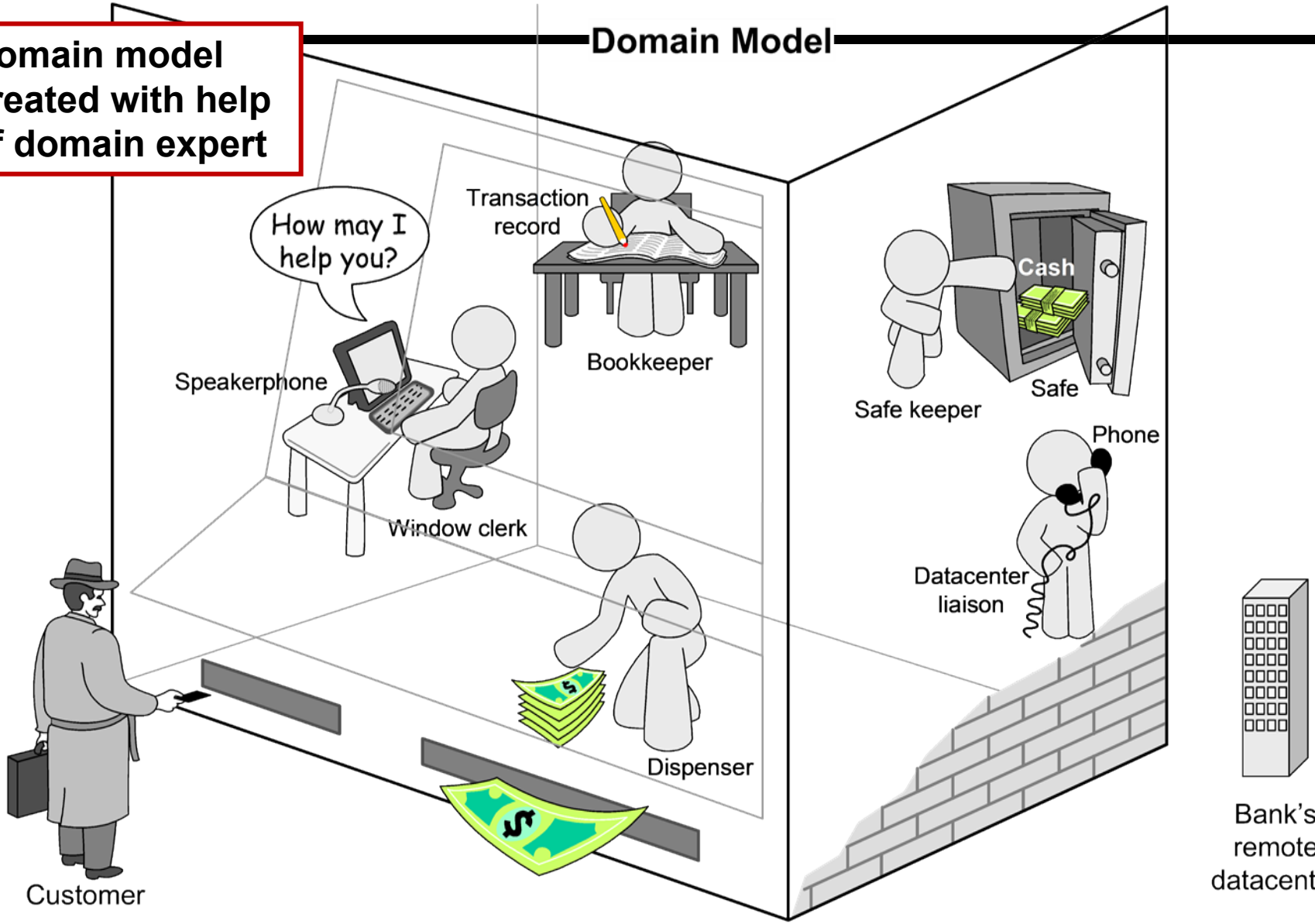
Problem-solving Strategy

Divide-and-conquer:

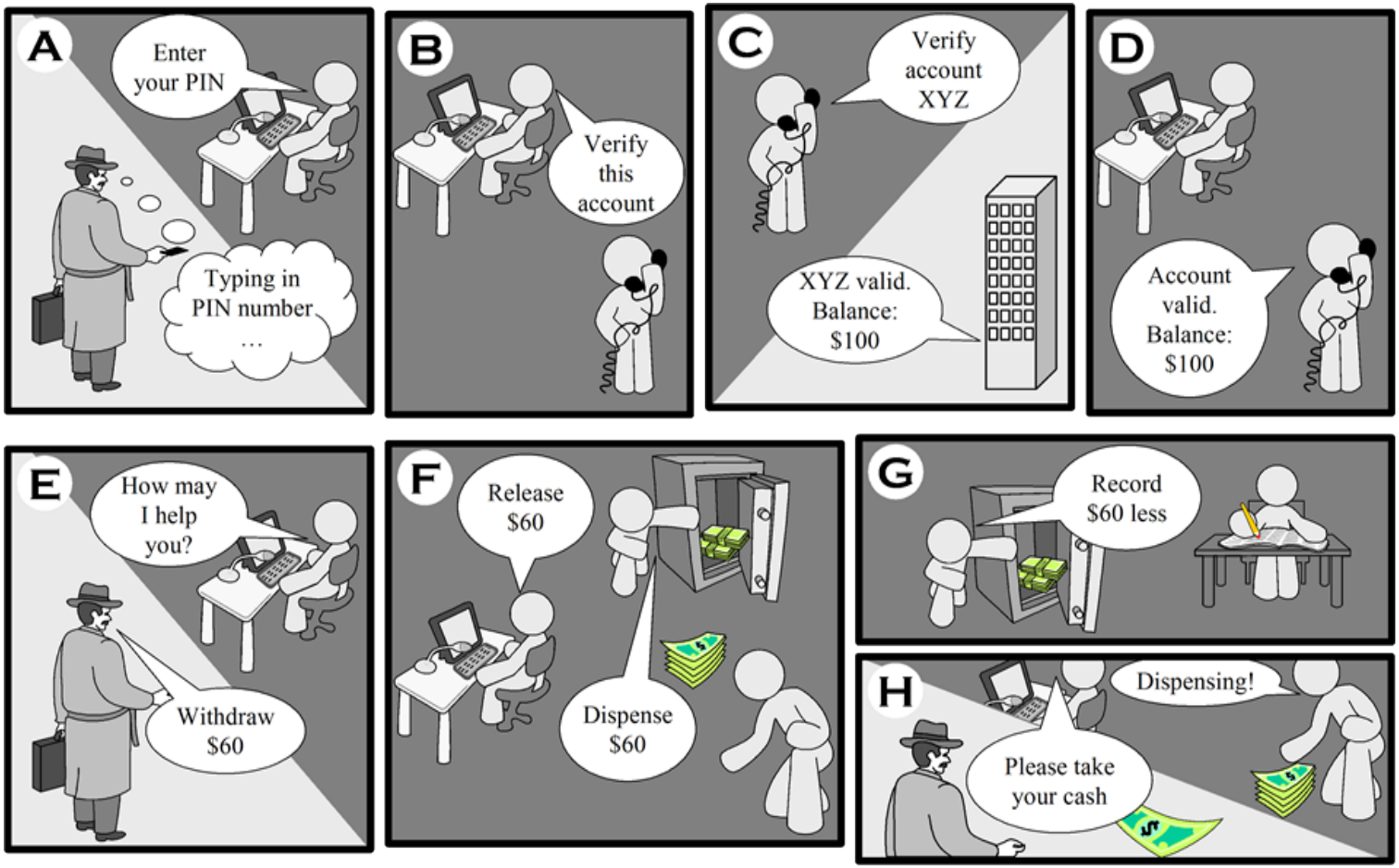
- Identify logical parts of the system that each solves a part of the problem
- Easiest done with the help of a domain expert who already knows the steps in the process ("how it is currently done")
- Result:
A Model of the Problem Domain
(or "domain model")

How ATM Machine Might Work

Domain model created with help of domain expert



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 …

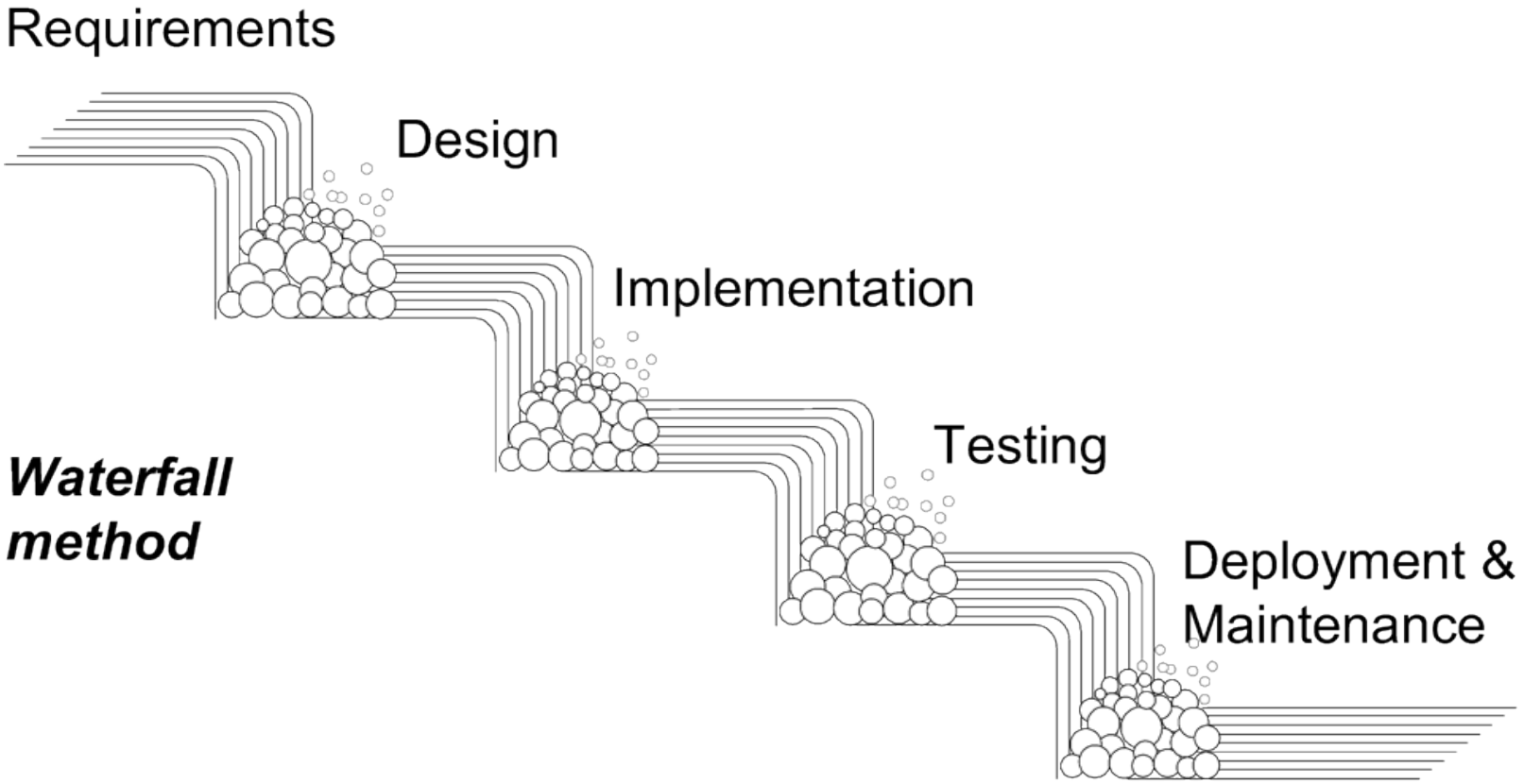
Second Law of Software Engineering

- 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
  - User feedback essential; feedback loops on several levels of granularity

Waterfall Method

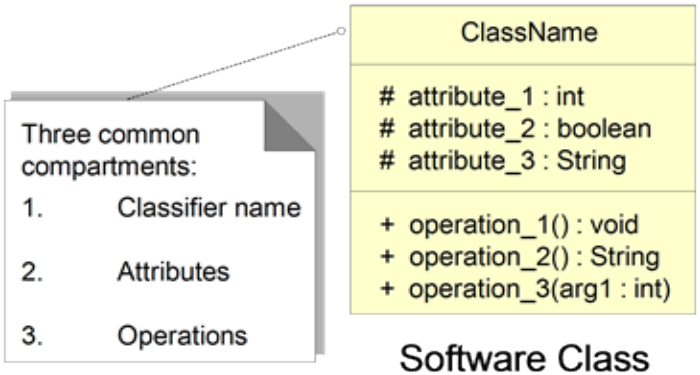


Unidirectional, no way back
finish this step before moving to the next

UML – Language of Symbols

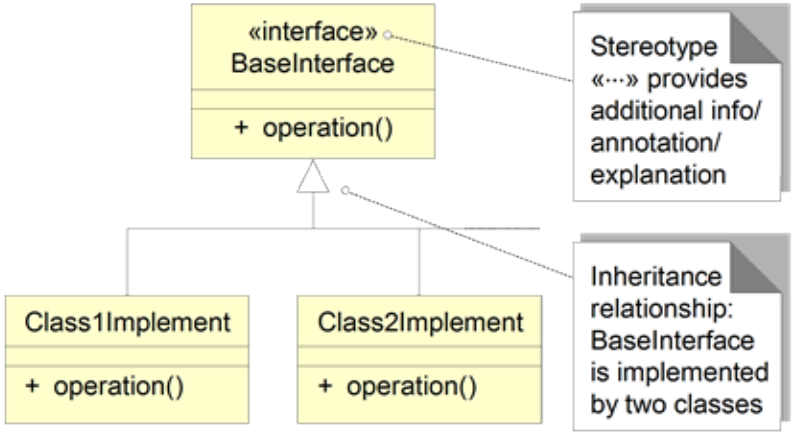
UML = Unified Modeling Language

Actor

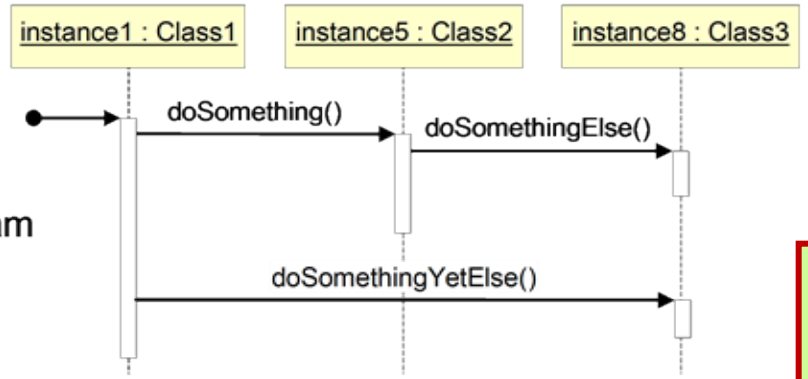


Comment

Software Class



Software Interface Implementation



Interaction Diagram

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

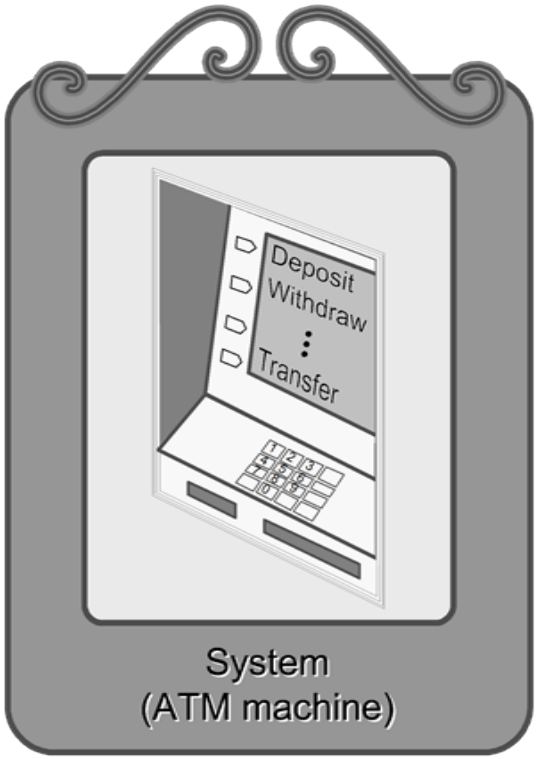
Understanding the Problem Domain

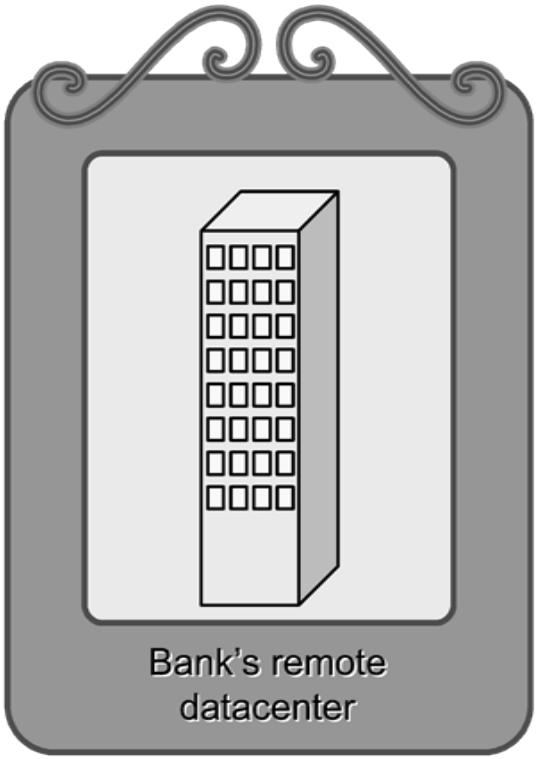
- System to be developed
- Actors
  - Agents external to the system
- Concepts/ Objects
  - Agents working inside the system
- Use Cases
  - Scenarios for using the system

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

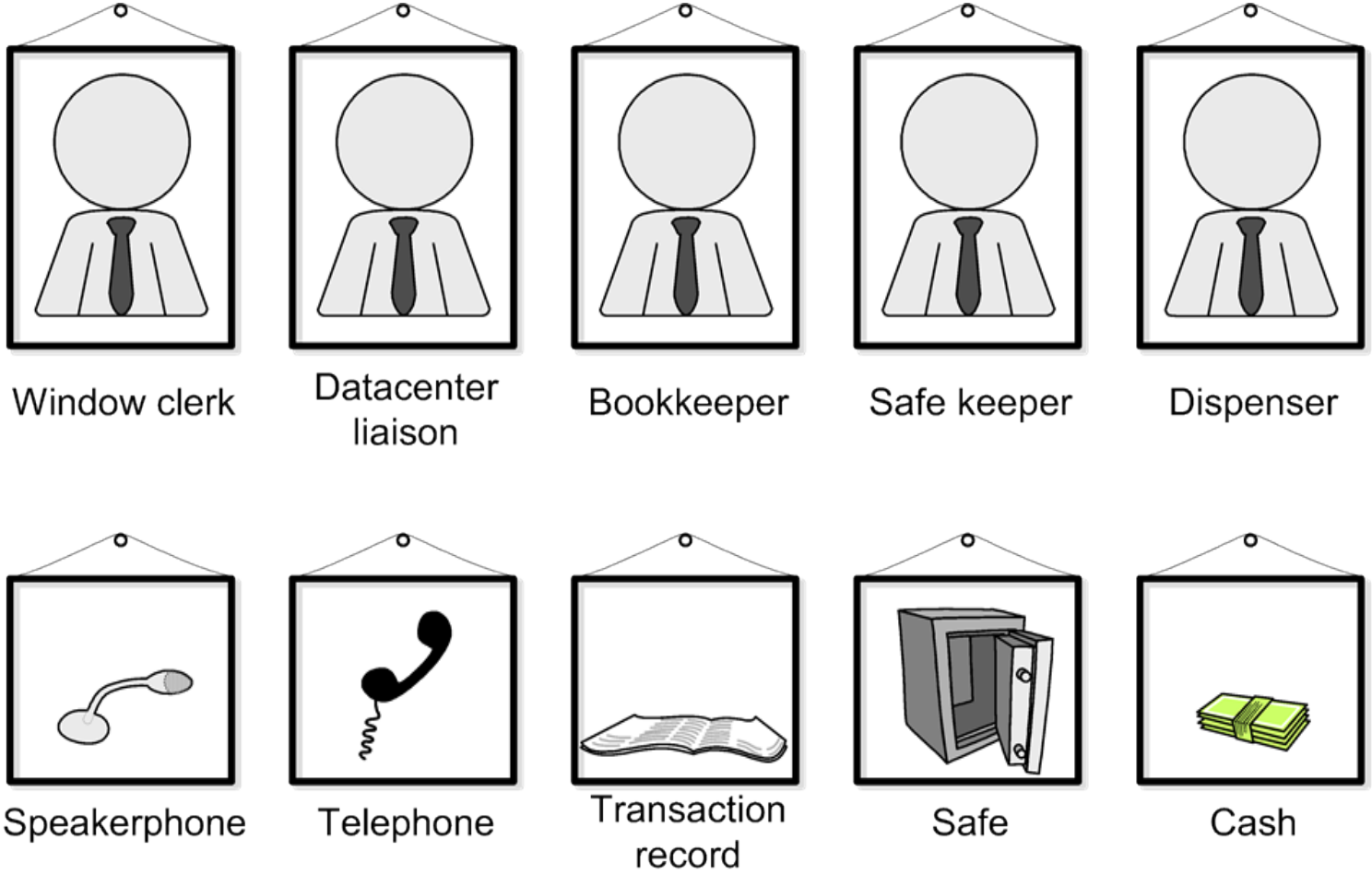
ATM: Gallery of Players





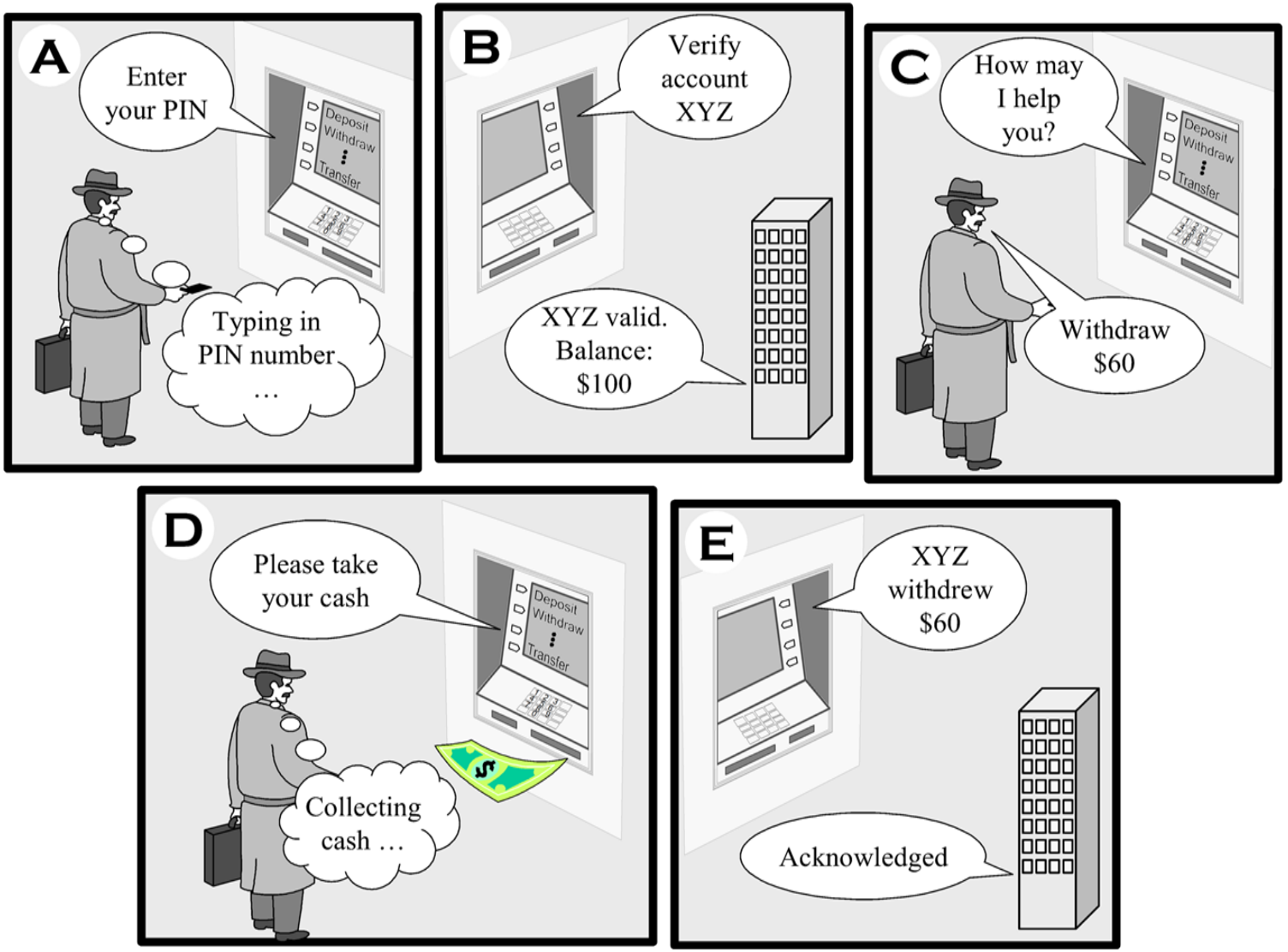
Actors (Easy to identify because they are visible!)

Gallery of Workers + Things



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

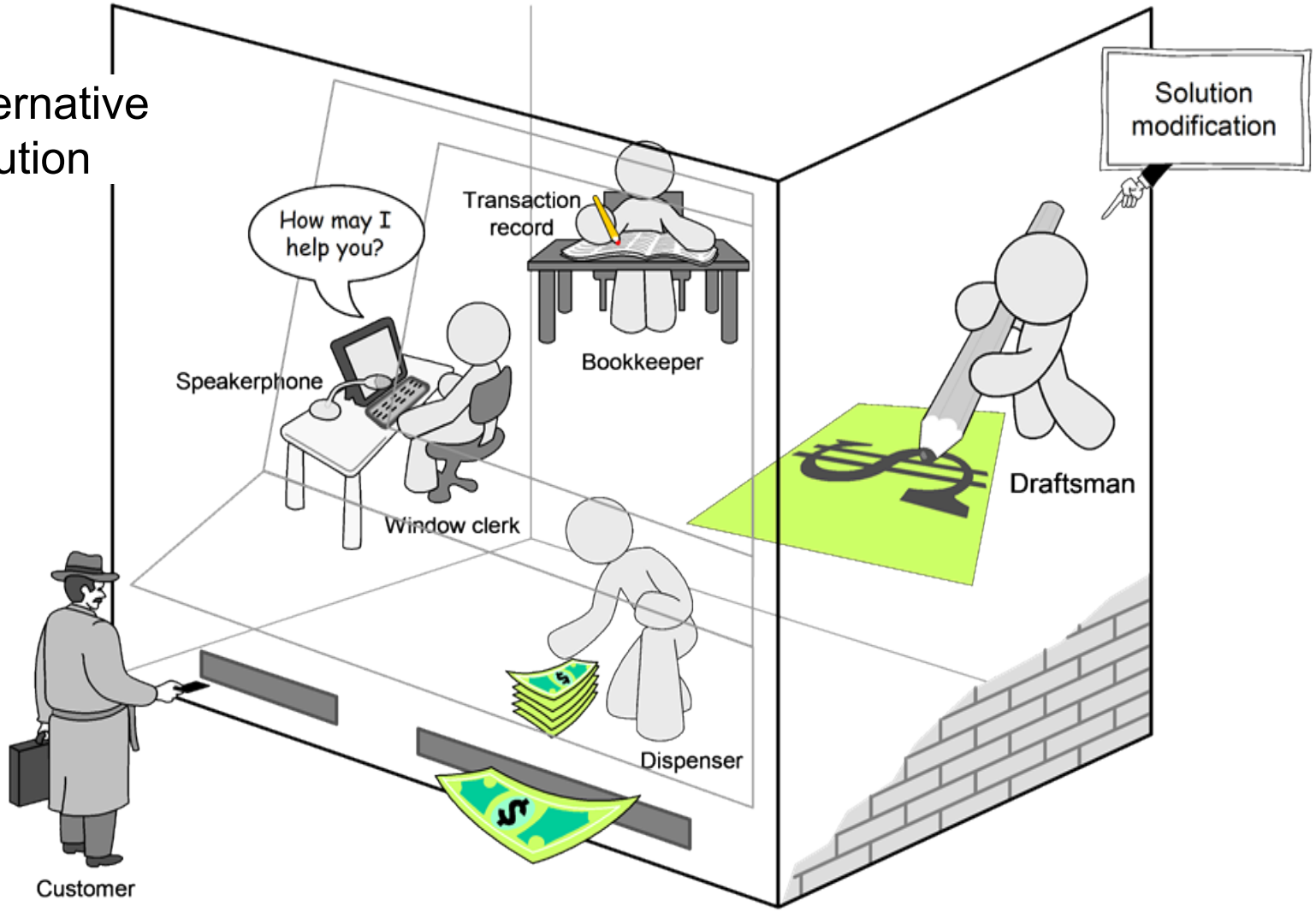
Use Case: Withdraw Cash



How ATM Machine Works (2)

Domain Model (2)

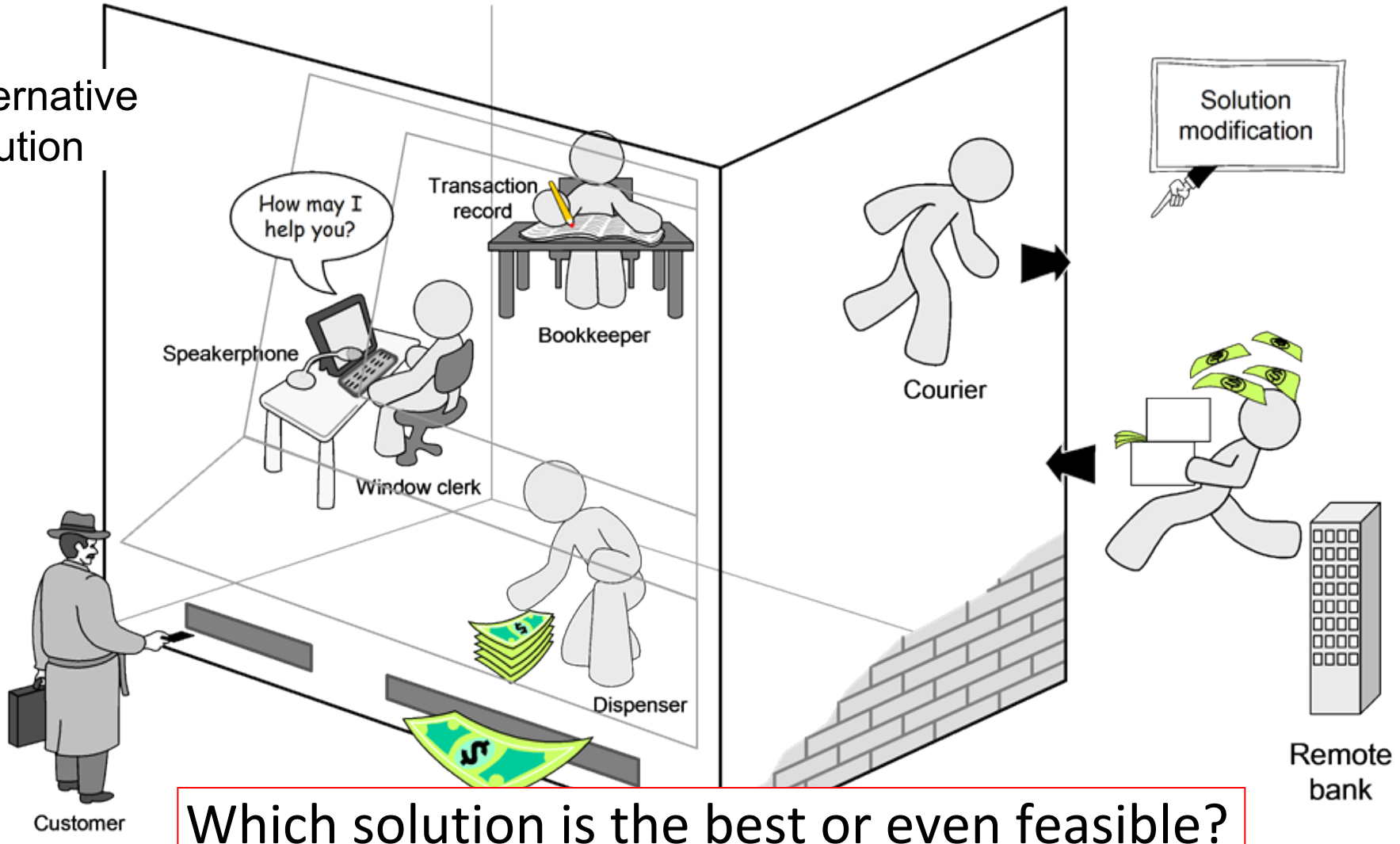
Alternative solution



How ATM Machine Works (3)

Domain Model (3)

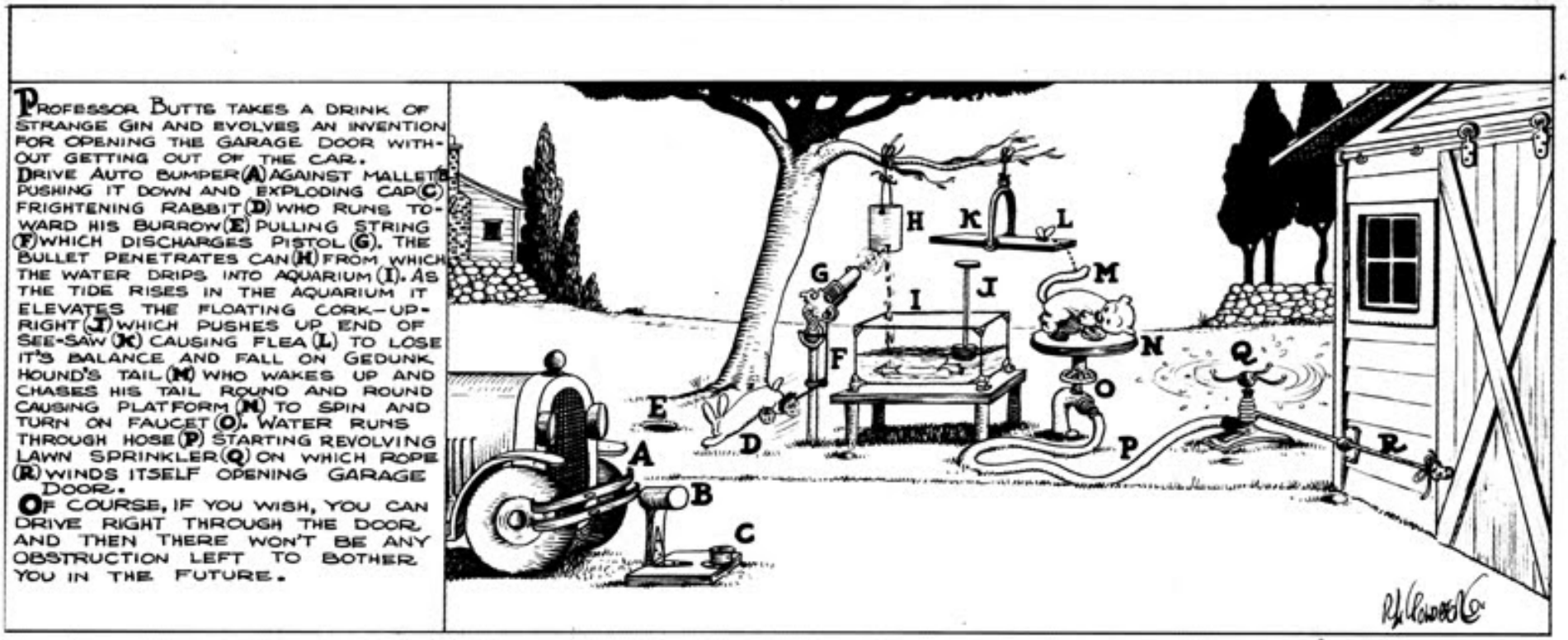
Alternative solution



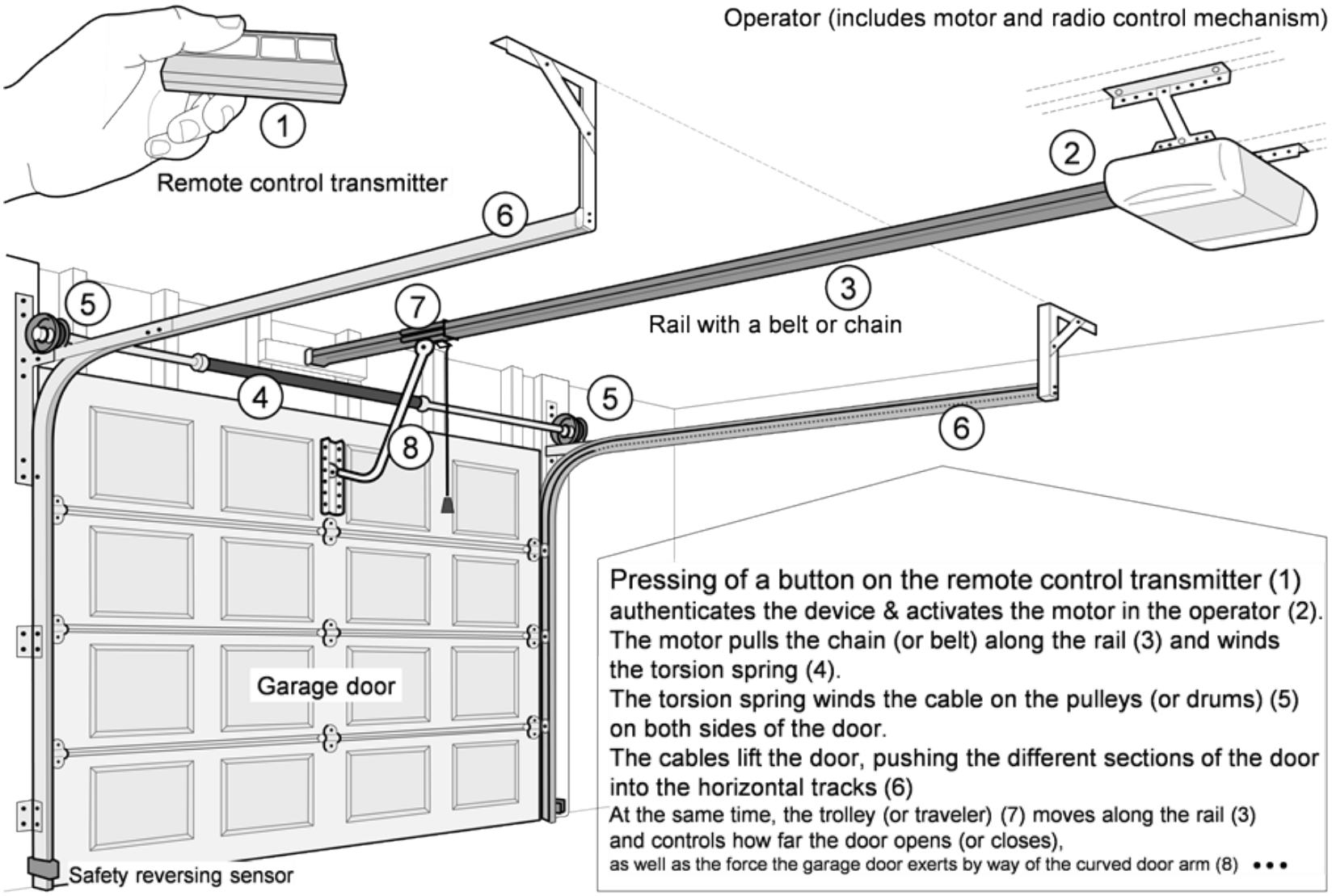
Which solution is the best or even feasible?

Rube Goldberg Design

Garage door opener



Actual Design



Pressing of a button on the remote control transmitter (1) authenticates the device & activates the motor in the operator (2).
The motor pulls the chain (or belt) along the rail (3) and winds the torsion spring (4).
The torsion spring winds the cable on the pulleys (or drums) (5) on both sides of the door.
The cables lift the door, pushing the different sections of the door into the horizontal tracks (6)
At the same time, the trolley (or traveler) (7) moves along the rail (3) and controls how far the door opens (or closes),
as well as the force the garage door exerts by way of the curved door arm (8) • • •

Feasibility & Quality of Designs

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

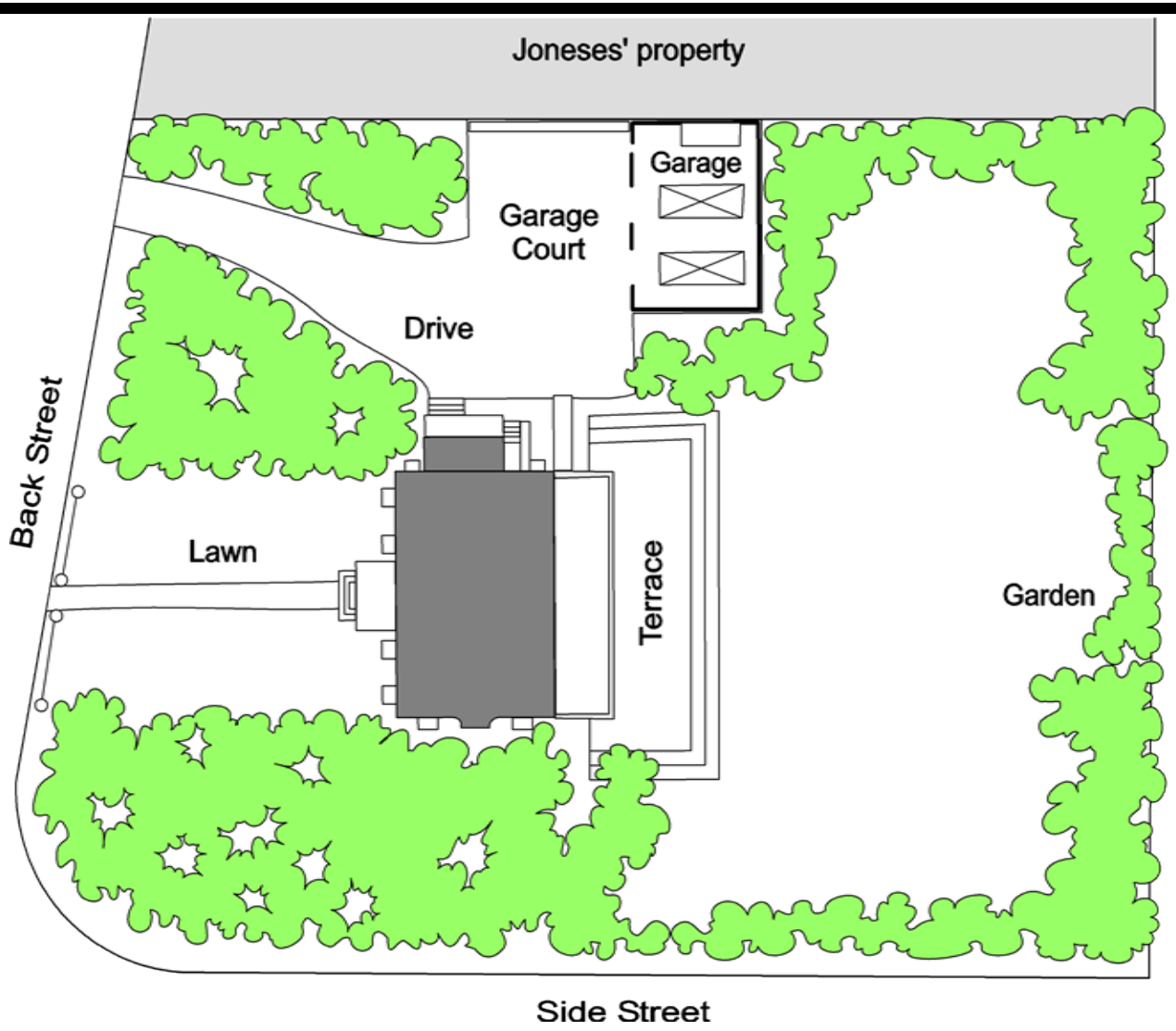
Software Measurement

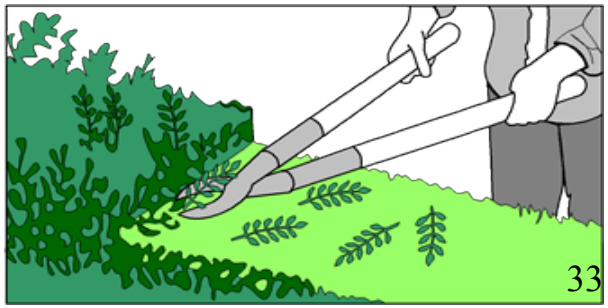
- What to measure?
  - Project (developer's work),

    for budgeting and scheduling
  - Product,

    for quality assessment

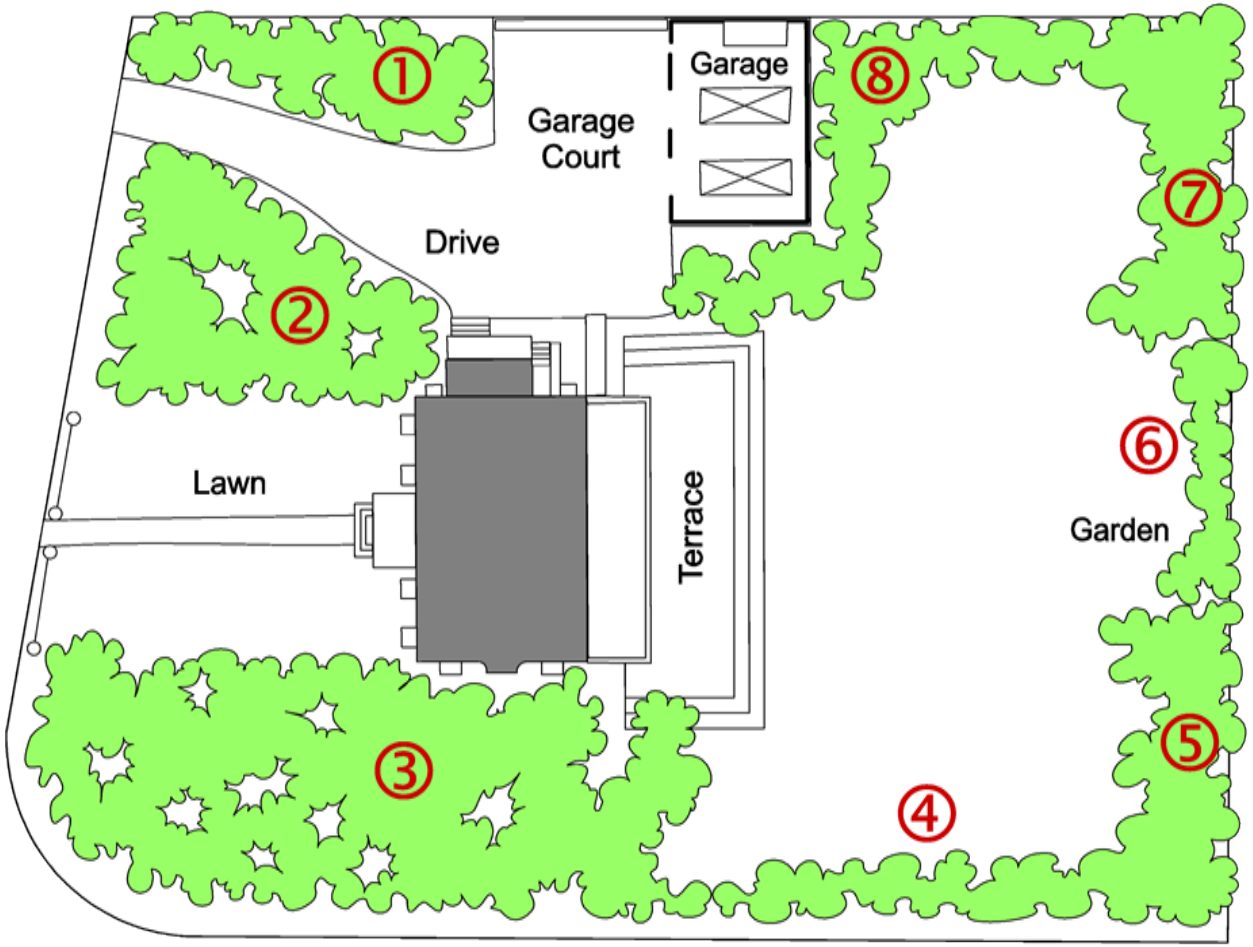
Formal hedge pruning





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)

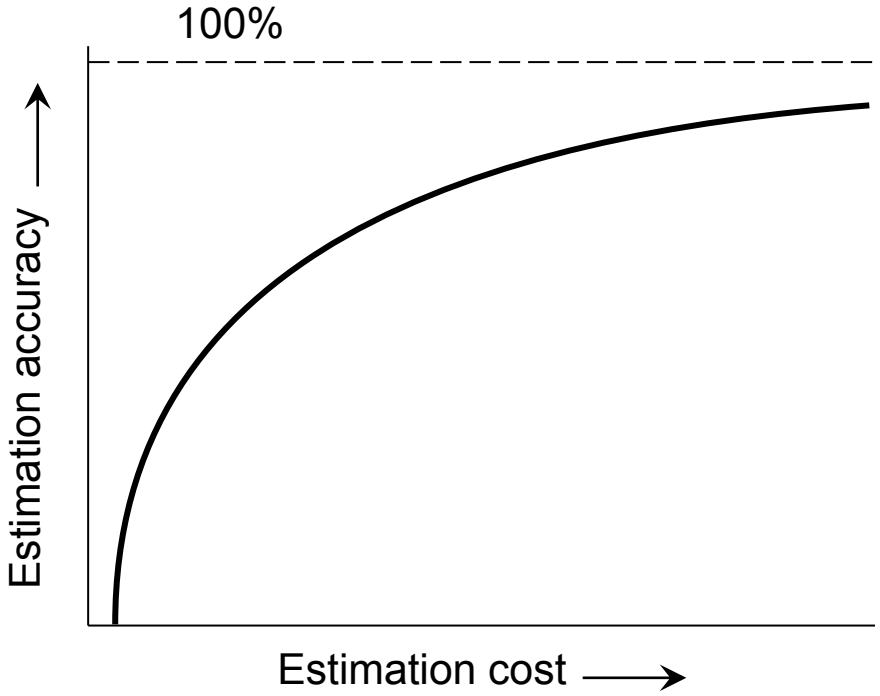
- Step 5: Estimate the work duration

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

Sizing the Problem (3)

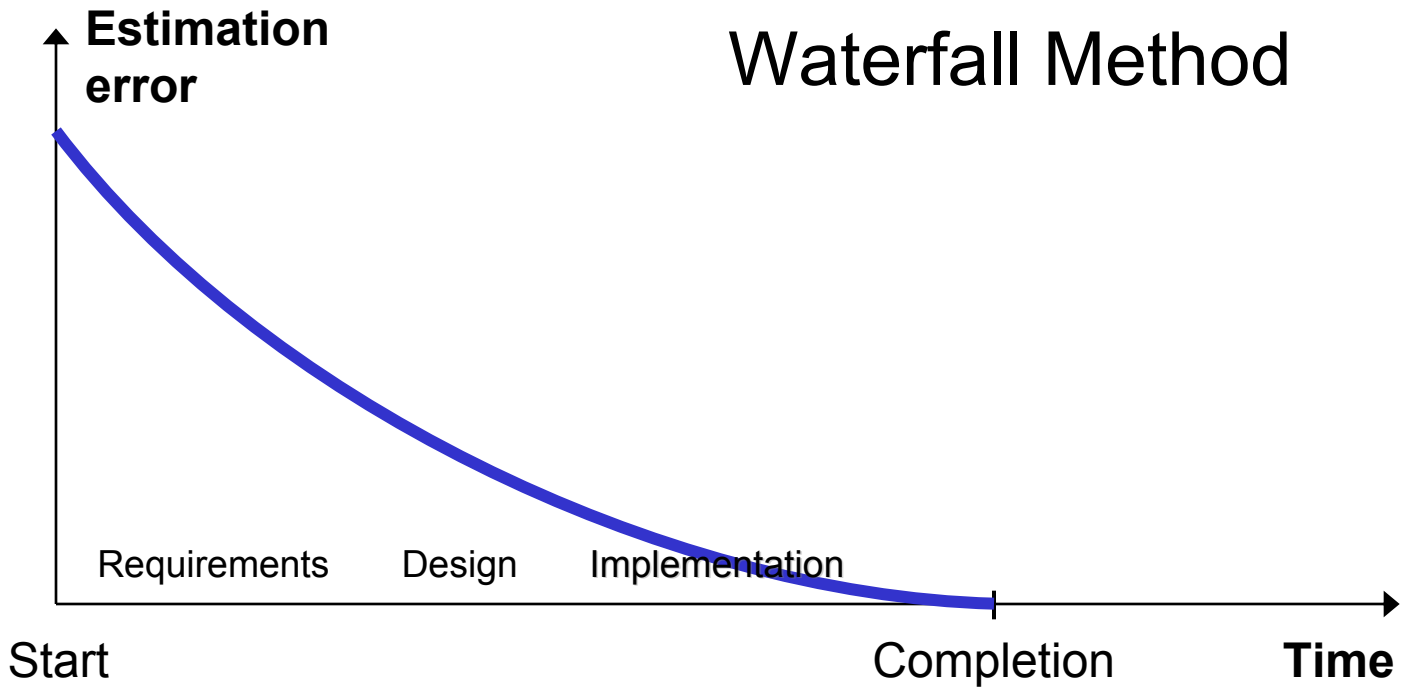
- *Advantages*:
  - Velocity estimate may need to be adjusted (based on observed progress)
  - However, the total duration can be computed quickly (provided that the *relative* size estimates of parts are accurate – easier to achieve if the parts are **small** and **similar-size**)

Exponential Cost of Estimation



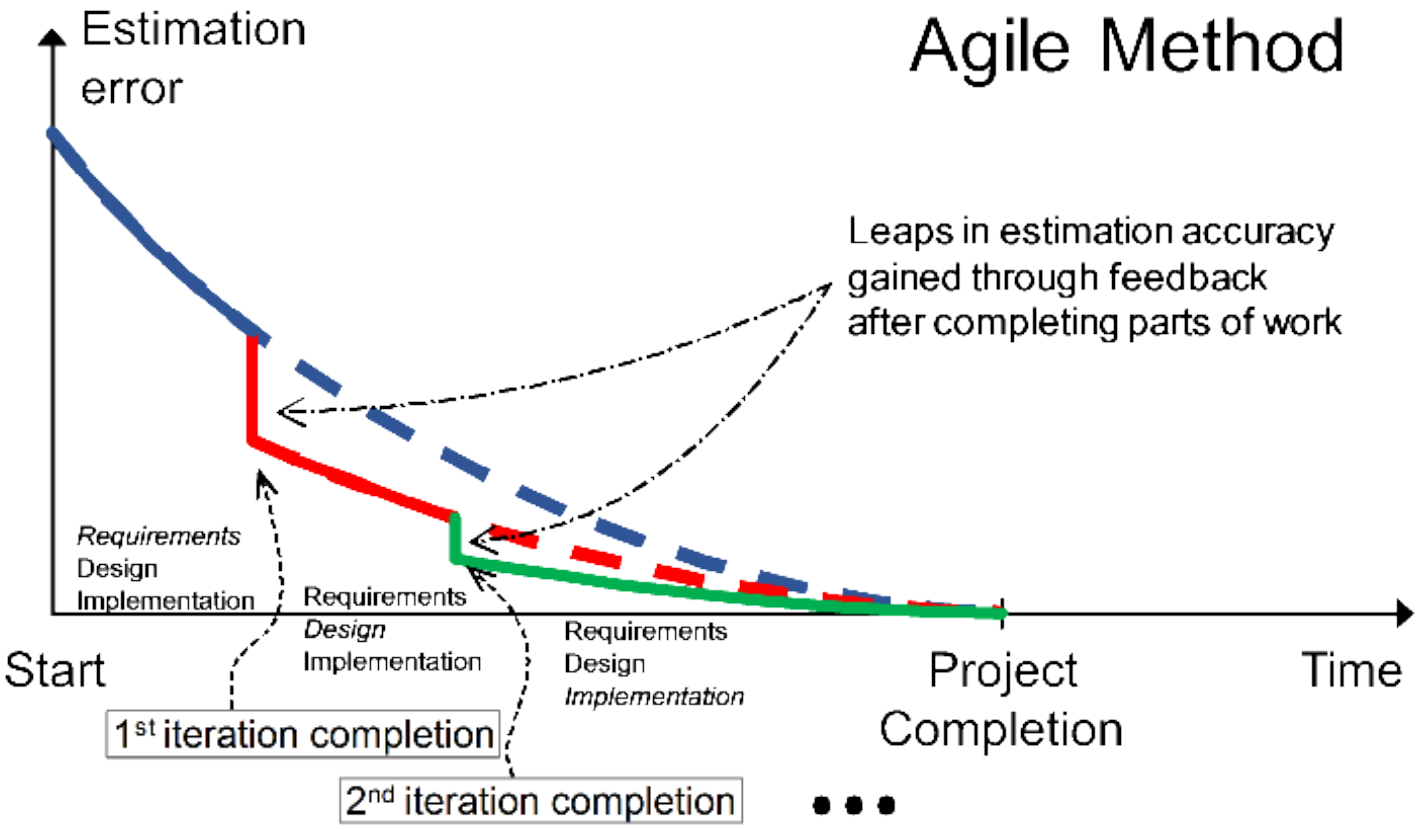
- **Improving accuracy of estimation beyond a certain point requires huge cost and effort (known as the law of diminishing returns)**
- **In the beginning of the curve, a modest effort investment yields huge gains in accuracy**

Estimation Error Over Time



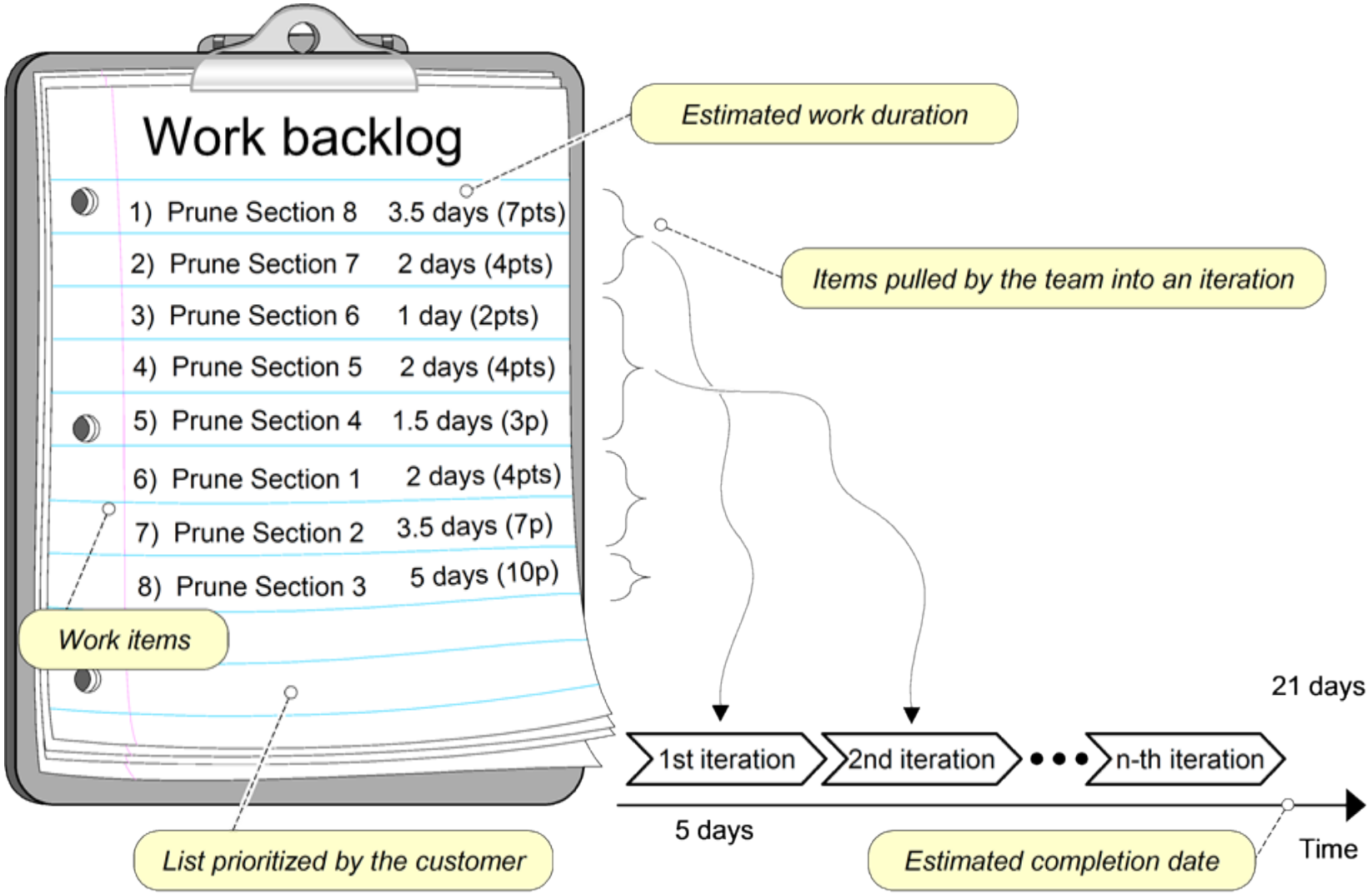
Waterfall method *cone of uncertainty* starts high and *gradually* converges to zero as the project approaches completion.

Estimation Error Over Time

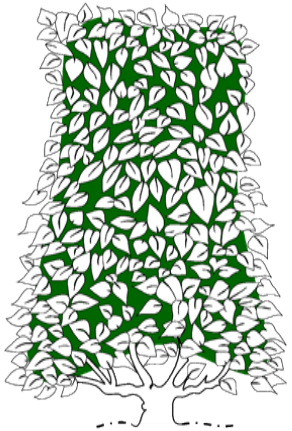


Agile method *cone of uncertainty* starts high and *in leaps* converges to zero as the project approaches completion.

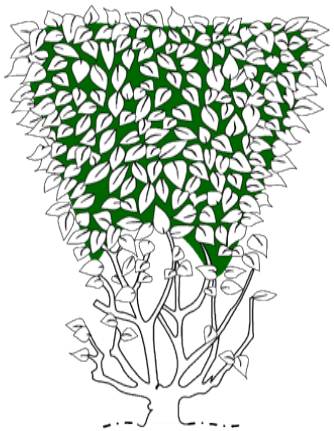
Agile Project Effort Estimation



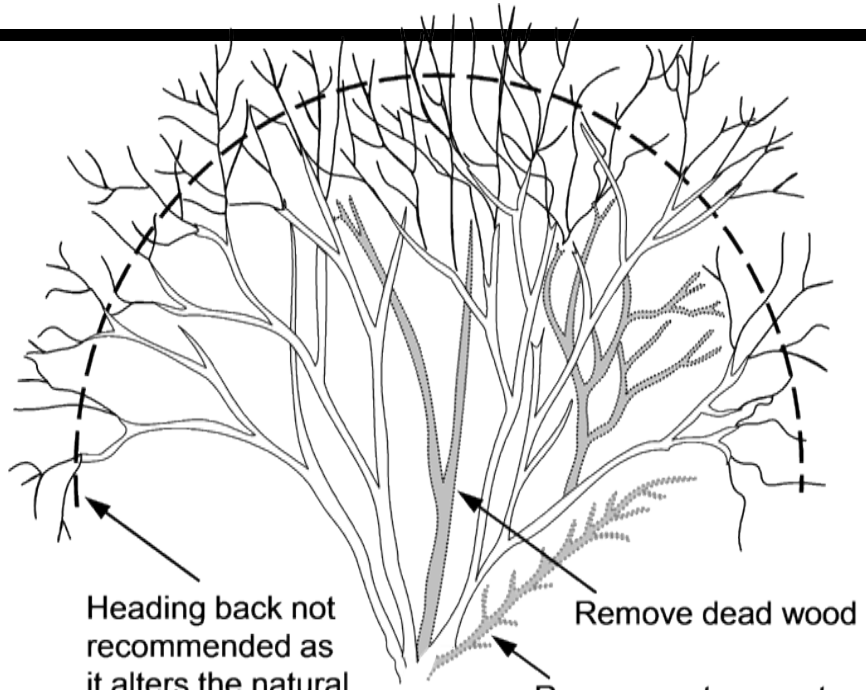
Measuring Quality of Work

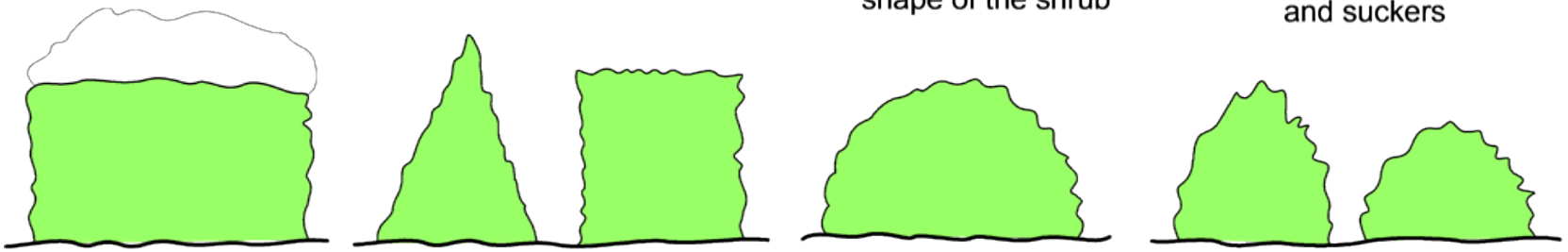


Good Shape
(Low branches get sun)



Poor Shape
(Low branches
shaded from sun)





Snow accumulates on broad flat tops

Straight lines require more frequent trimming

Peaked and rounded tops hinder snow accumulation

Rounded forms, which follow nature's tendency, require less trimming

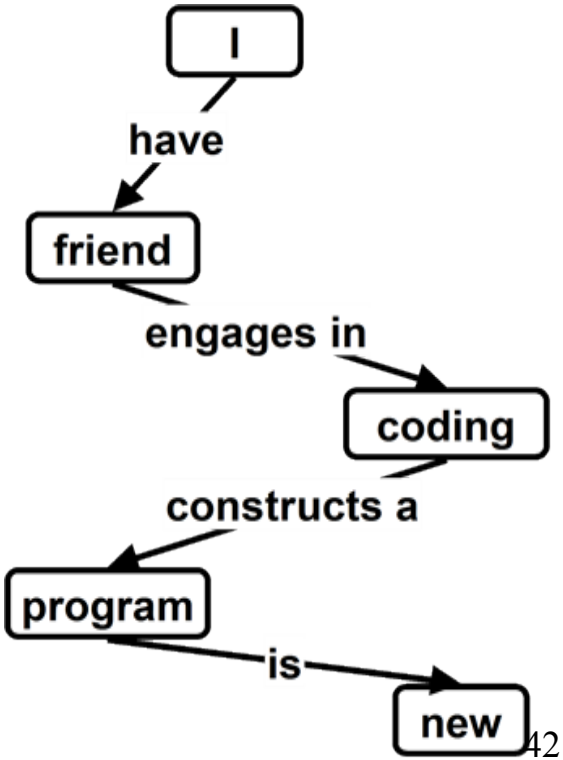
Concept Maps

Useful tool for problem domain description

SENTENCE: "My friend is coding a new program"

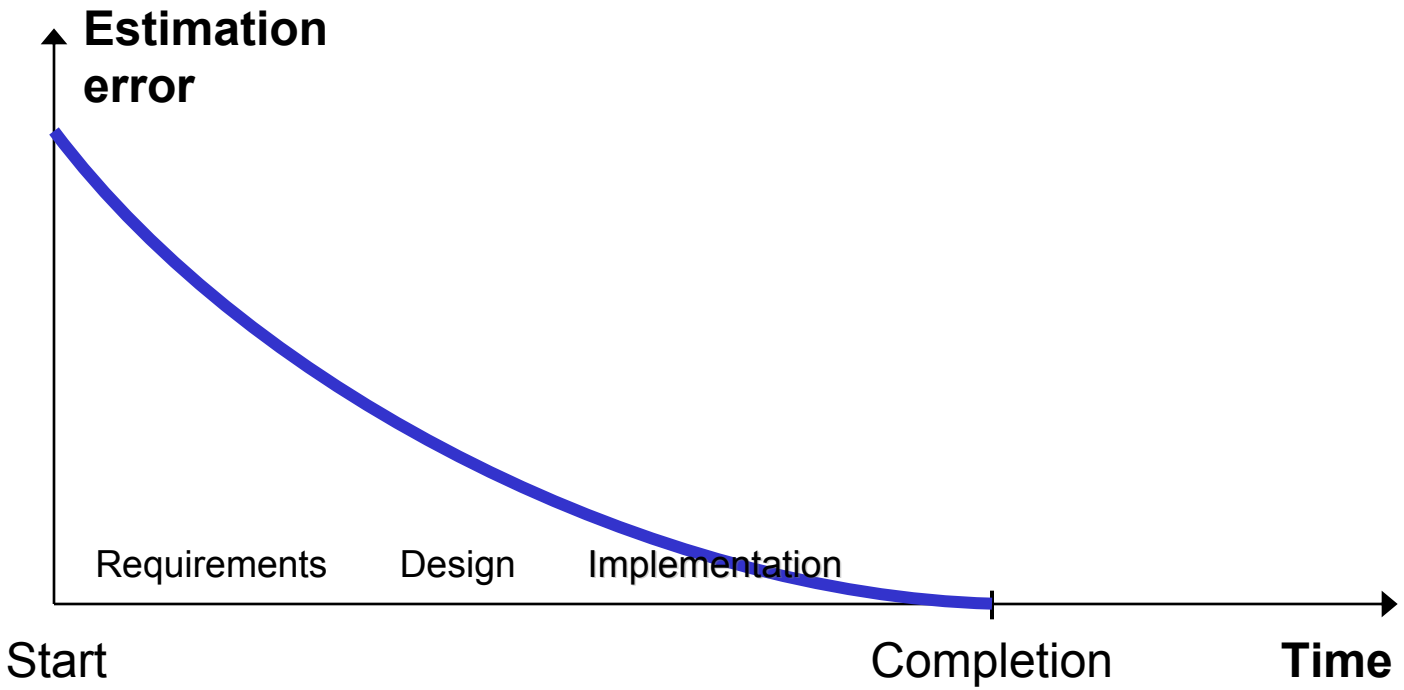
translated into propositions

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



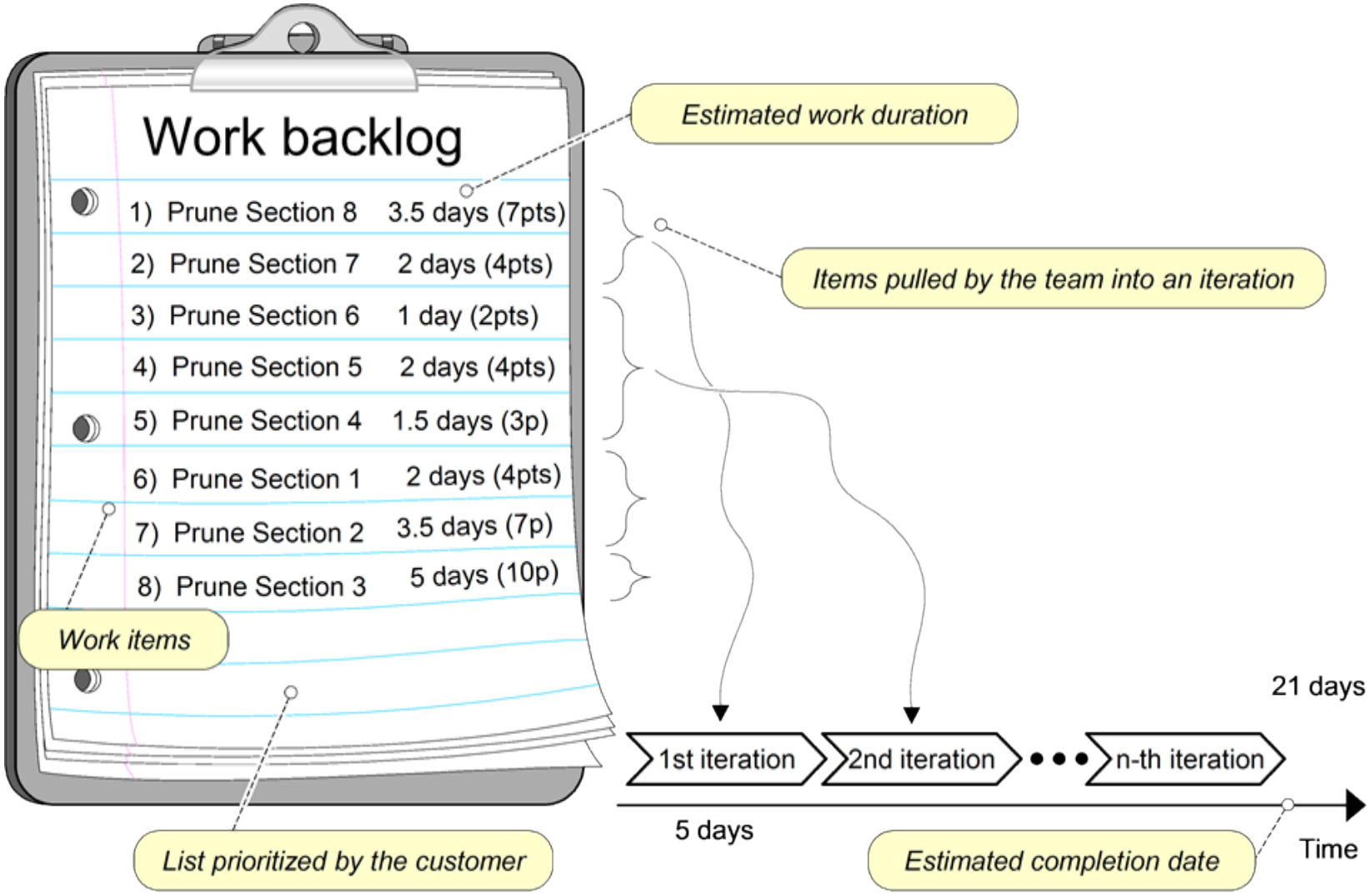
Search the Web for Concept Maps

Estimation Error Over Time

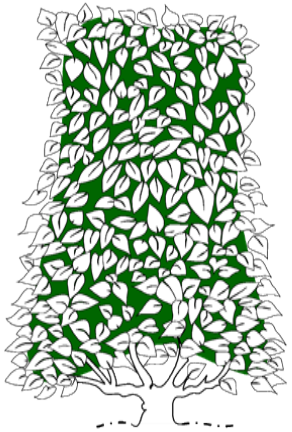


The *cone of uncertainty* starts high and narrows down to zero as the project approaches completion.

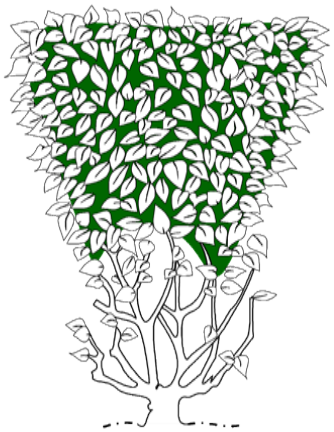
Agile Project Effort Estimation



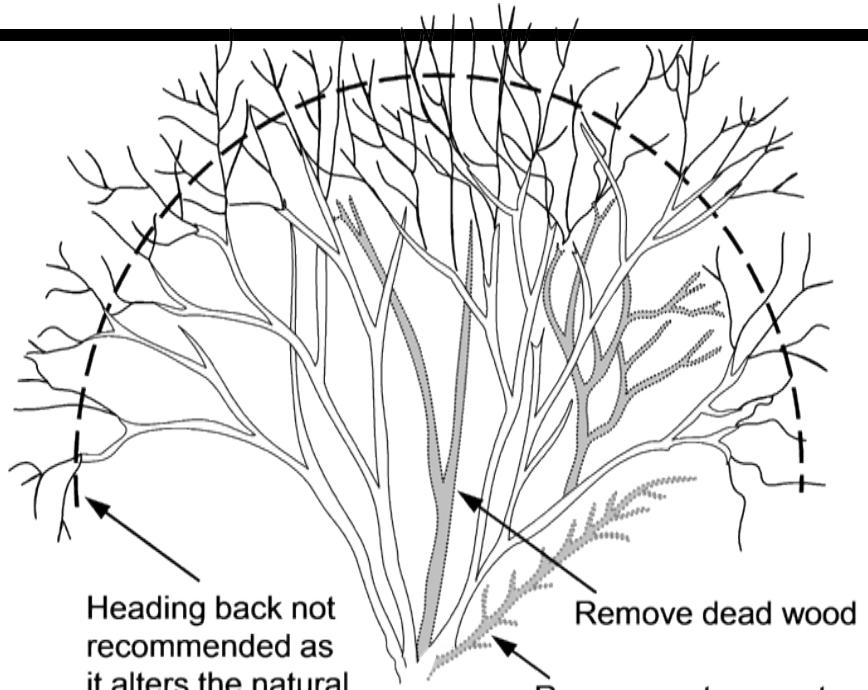
Measuring Quality of Work

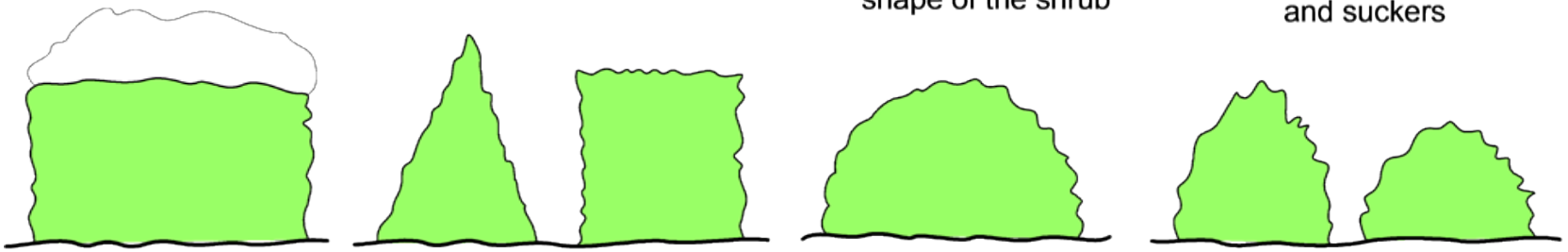


Good Shape
(Low branches get sun)



Poor Shape
(Low branches
shaded from sun)





Snow accumulates on broad flat tops

Straight lines require more frequent trimming

Peaked and rounded tops hinder snow accumulation

Rounded forms, which follow nature's tendency, require less trimming

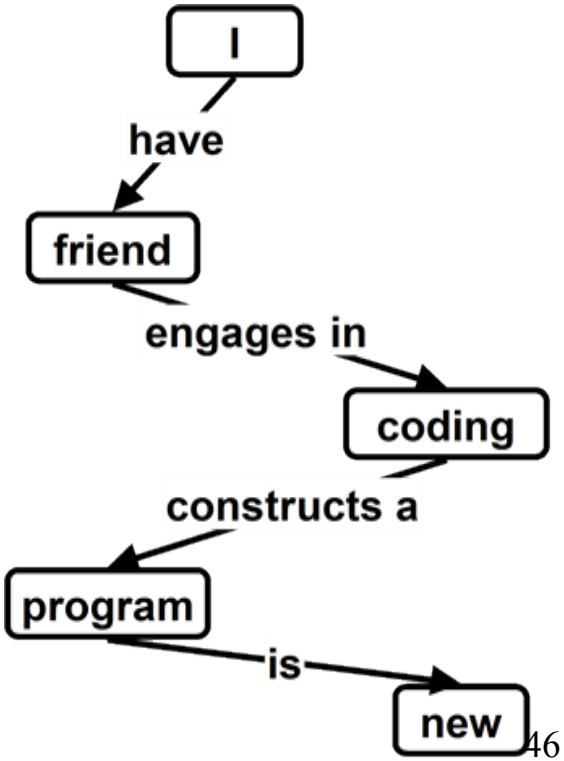
Concept Maps

Useful tool for problem domain description

SENTENCE: "My friend is coding a new program"

translated into propositions

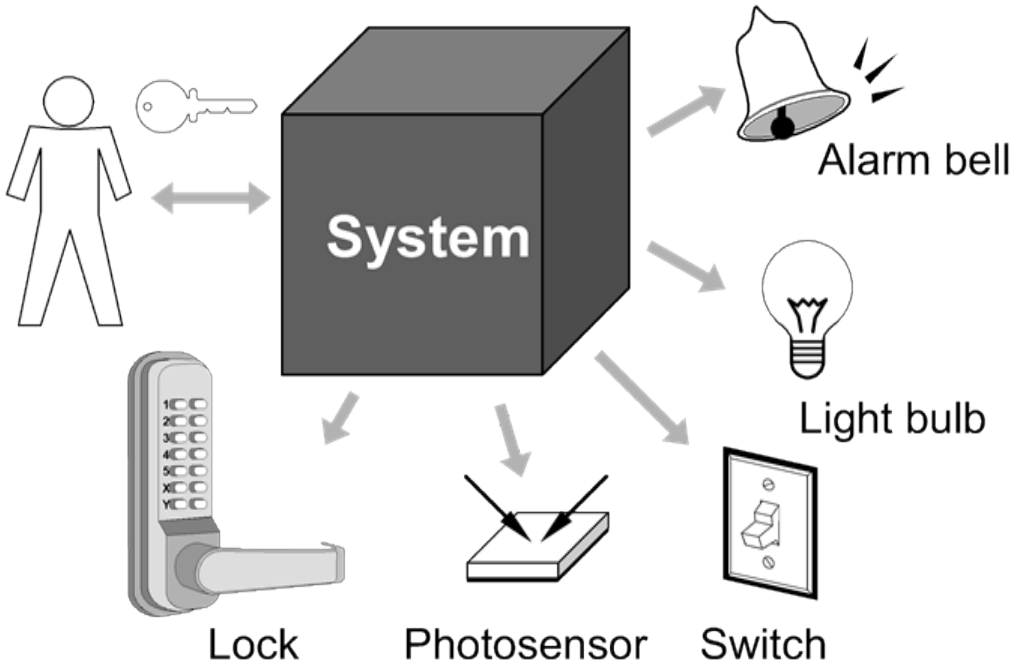
Proposition	Concept	Relation	Concept
1.	I	have	friend
2.	friend	engages in	coding
3.	coding	constructs a	program
4.	program	is	new



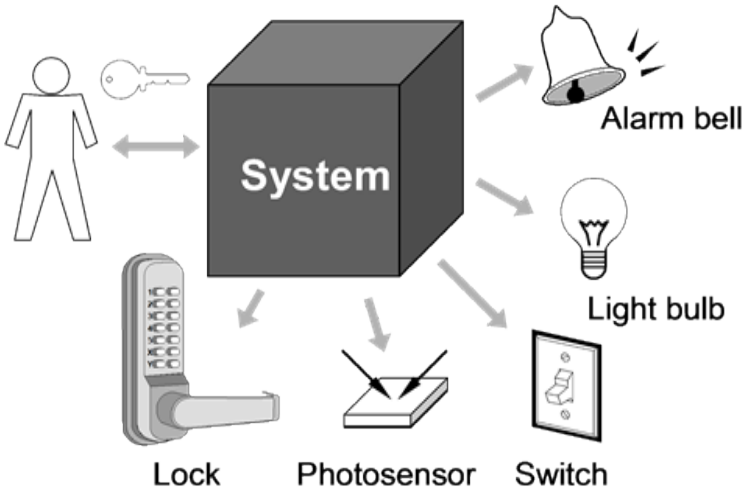
Search the Web for Concept Maps

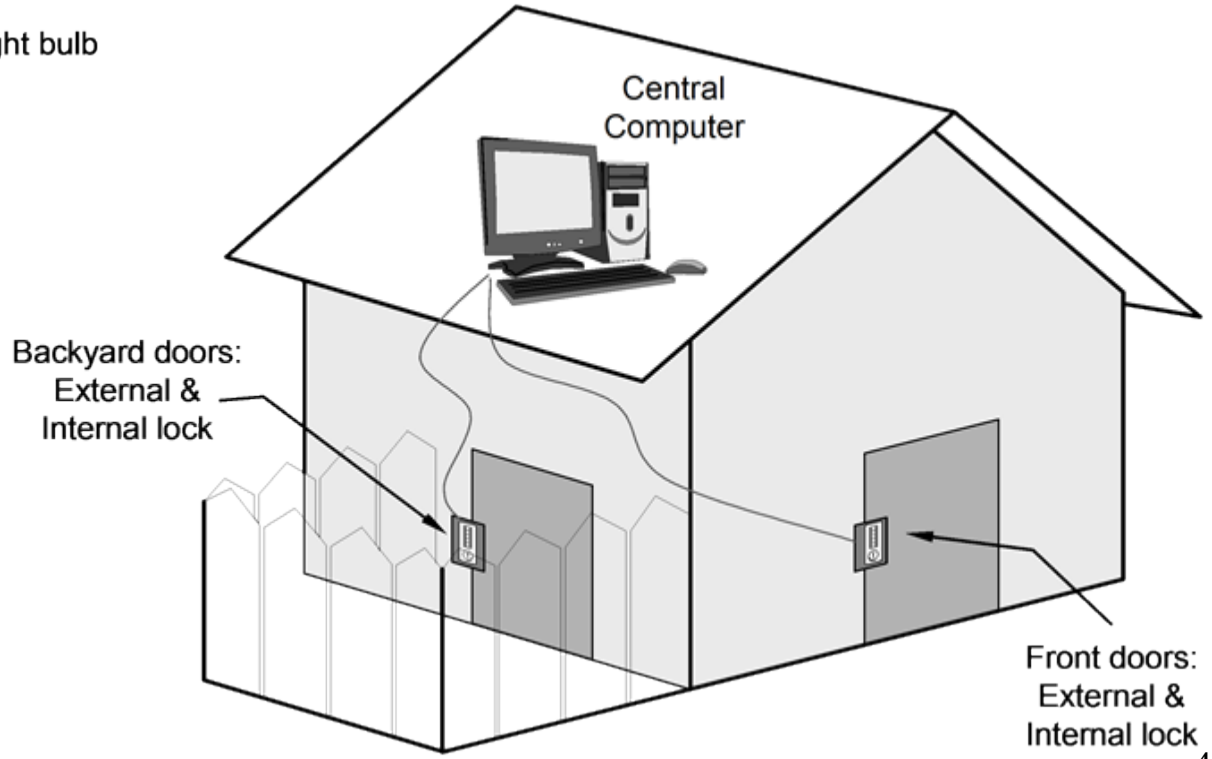
Case Study: Home Access Control

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



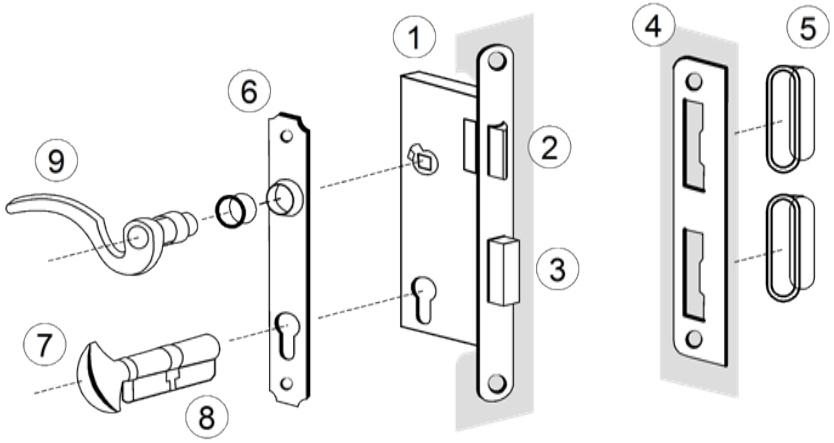
Case Study – More Details



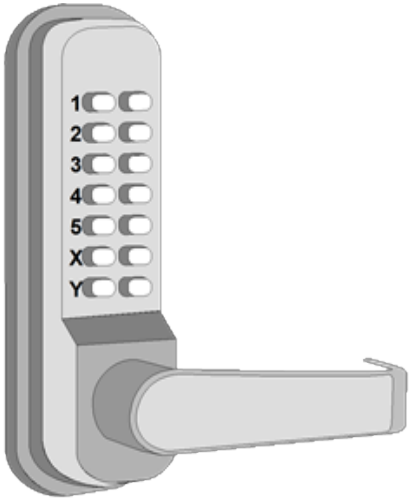


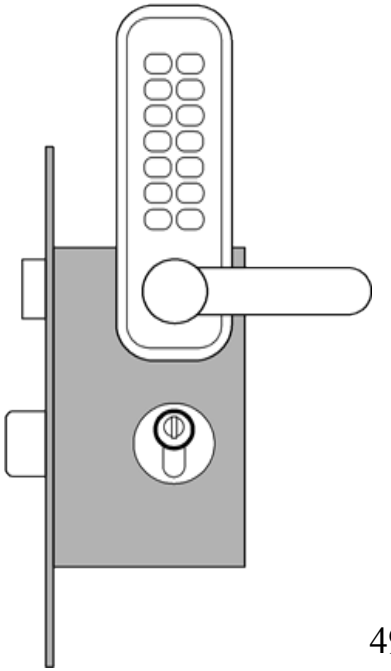
Know Your Problem

Mortise Lock Parts

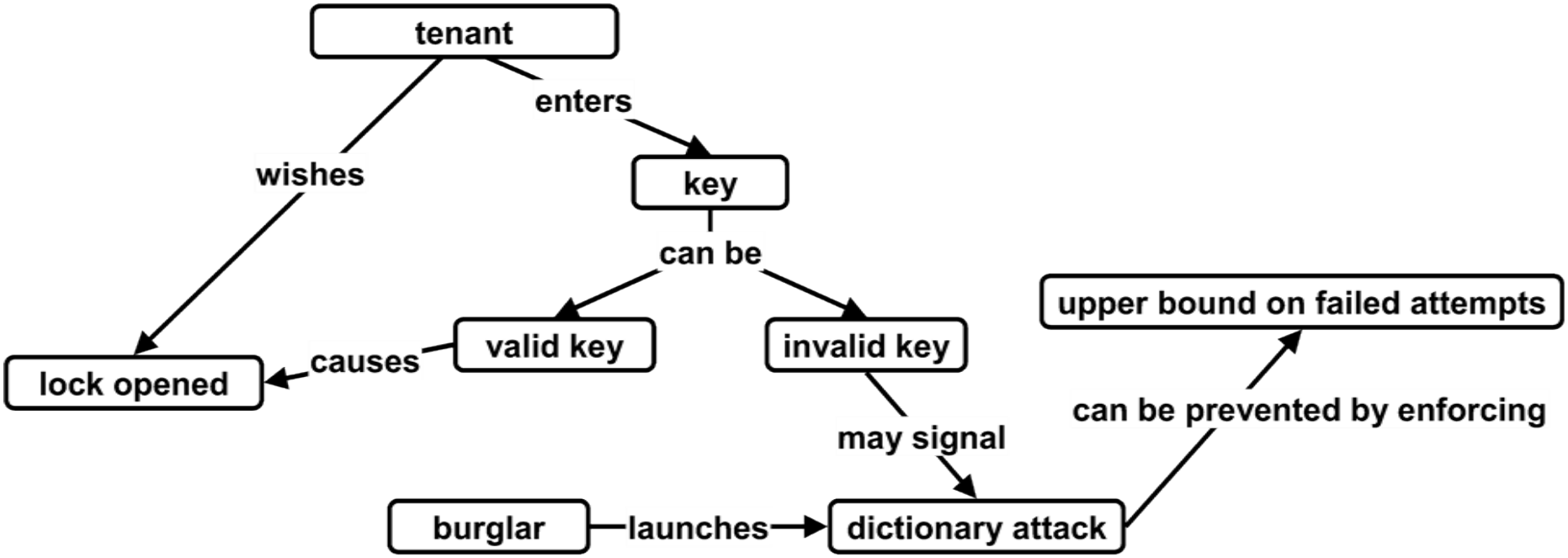


1. Lock case
2. Latch bolt
3. Dead bolt
4. Strike plate
5. Strike box
6. Protective plate
7. Thumb-turn
8. Lock cylinder
9. Left hand lever

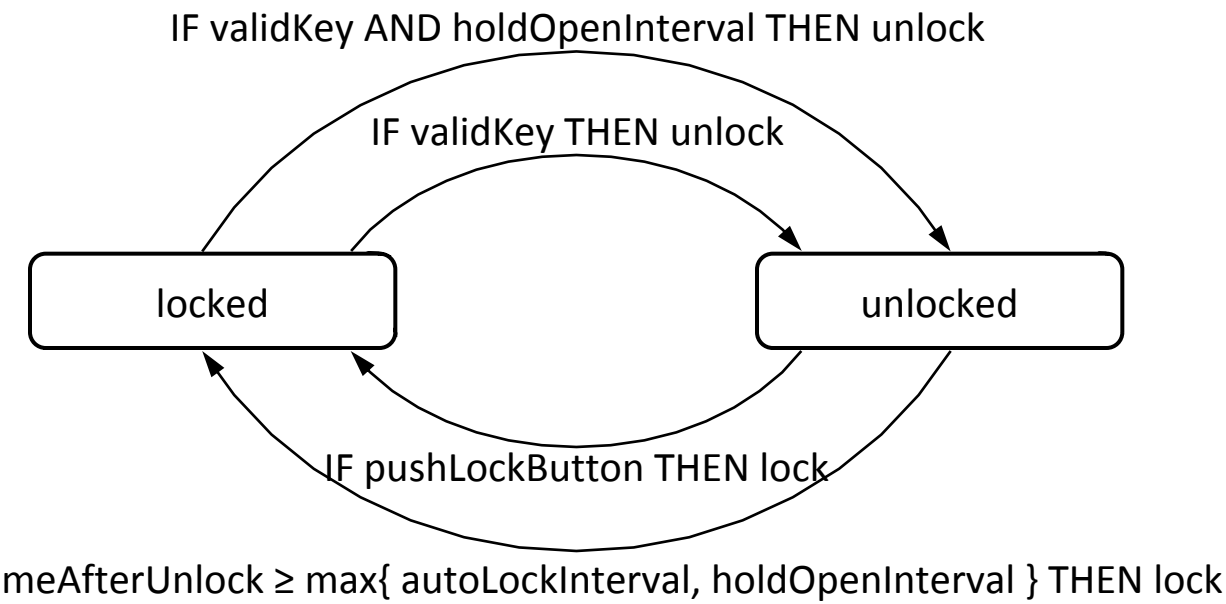




Concept Map for Home Access Control



States and Transition Rules



… what seemed a simple problem, now is becoming complex