Software Engineering

LECTURE 1: Introduction

Ting Su
East China Normal University

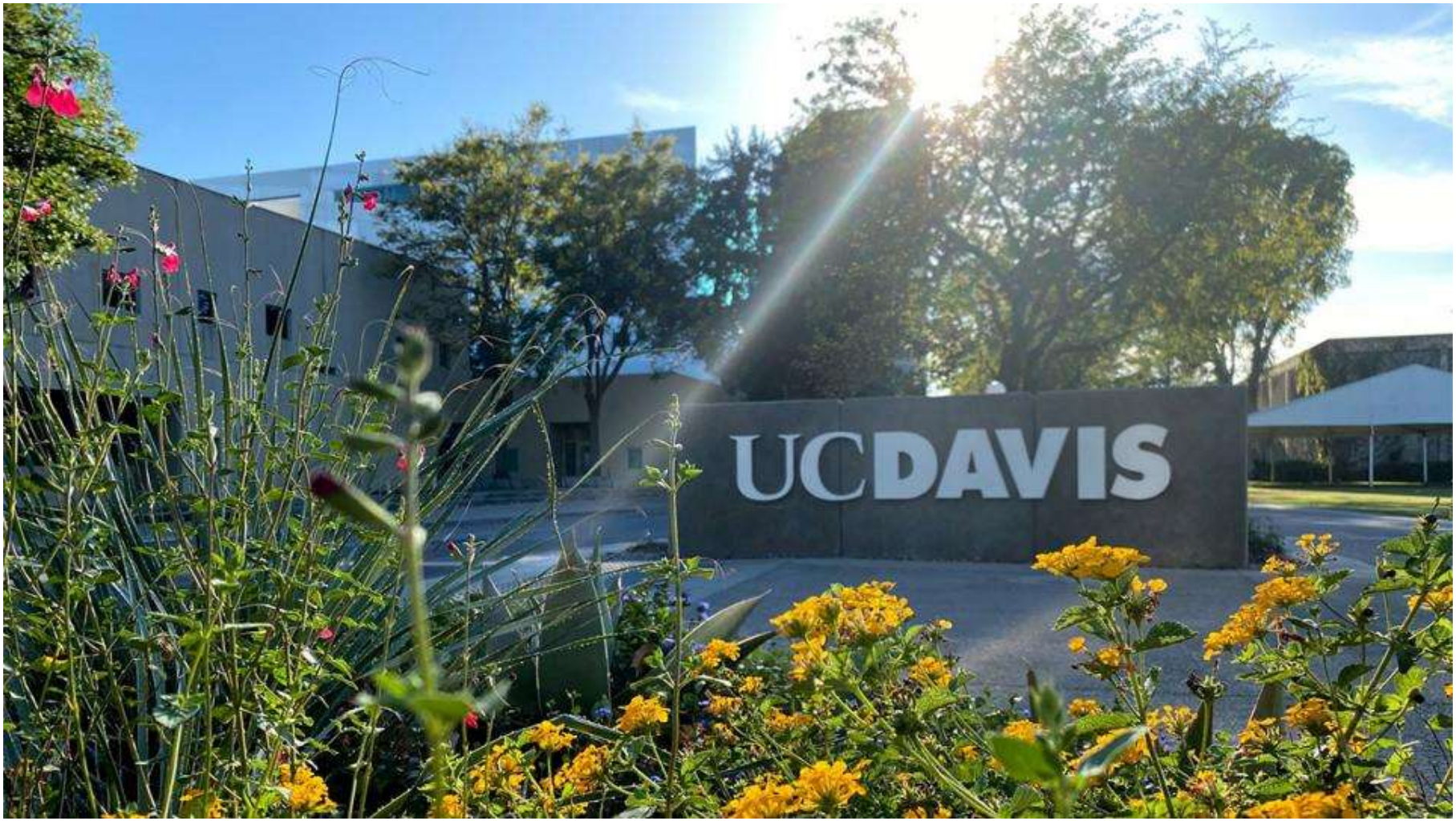
The slides are adapted from Ivan Marsic's lectures from Rutgers University.

Self-Introduction

- 苏亭，软件科学与技术系，教授
- 个人主页:
  http://tingsu.github.io (英文)
  https://faculty.ecnu.edu.cn/_s43/st2/main.psp (中文)
- 研究方向:
  软件分析与验证、软件测试、软件安全、可信人工智能、系统软件
- 实验室
  软件与系统可靠性研究小组 (理科楼B416)
- 联系方式
  理科楼B1103，tsu@sei.ecnu.edu.cn

ECNU

ECNU -> UCD



ECNU -> UCD

ECNU -> UCD

ECNU -> UCD -> NTU


NANYANG TECHNOLOGICAL UNIVERSITY
UNIVERSITI TEKNOLOGI NANYANG
南洋理工大學

ECNU -> UCD -> NTU

ECNU -> UCD -> NTU

ECNU -> UCD -> NTU -> ETH

ECNU -> UCD -> NTU -> ETH

ECNU -> UCD -> NTU -> ETH -> ECNU

Self-Introduction

- 苏亭，软件科学与技术系，教授
- 个人主页:
  http://tingsu.github.io (英文)
  https://faculty.ecnu.edu.cn/_s43/st2/main.psp (中文)
- 研究方向:
  软件分析与验证、软件测试、软件安全、可信人工智能、系统软件
- 实验室
  软件与系统可靠性研究小组 (理科楼B416)
- 联系方式
  理科楼B1103，tsu@sei.ecnu.edu.cn

Course Information

- 课程目标
  1. **掌握和熟悉软件工程的理论、概念、方法、和工具**
  2. **具备分析、设计、开发和管理软件项目的能力**
- 课程形式:
  1. **理论课：每周一下午第5-6节课**
  2. **实践课：双周二下午第5-6节课**
  3. **考核形式：出勤：5%；项目、作业：45%；期末考试：50%**
- 参考教材 (see more on the course website)
  - 《Software Engineering textbook》, by I. Marsic. （电子书）
  - 《Software Engineering-A Practitioner's Approach (Eighth Edition)》, Roger S. Pressman著, 郑人杰等译. 北京: 机械工业出版社, 2015年.
  - 软件测试(原书第二版), Patton,R.著, 张小松等译, 北京: 机械工业出版社, 2006.4.

Course Information

- 课程网站:

  https://tingsu.github.io/files/courses/se.html

- 助教
  - 唐文兵 (20级博士)
  - 熊一衡 (20级硕士)

What is Software Engineering?

- The IEEE definition:
  - *Software Engineering: (1) The application of a **systematic, disciplined, quantifiable approach** to the **development, operation, and maintenance** of software; that is, the application of engineering to software. (2) The study of approaches as in (1).*

History of Software Engineering

- Its origin: 1945 to 1965
- The software crisis: 1965 to 1985
- "No Silver Bullet": 1985 to 1989
- Prominence of the Internet: 1990 to 1999
- Lightweight methodologies: 2000 to 2015

https://en.wikipedia.org/wiki/History_of_software_engineering

History of Software Engineering

History of Software Engineering

Margaret Heafield Hamilton is an American computer scientist, systems engineer, and business owner. She was director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for NASA's Apollo program. She later

ONFERENCE ON
NG MAY 27 - JUNE 3 2018 GOTHENBURG, SWEDEN

What is Software?

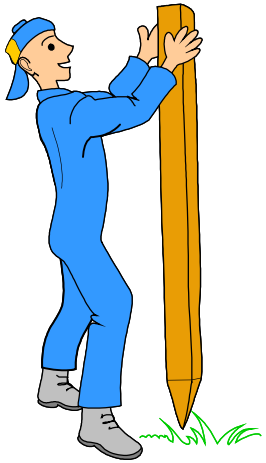
What is Software?

- Software is: (1) *instructions* (computer programs) that when executed provide desired features, function, and performance; (2) *data structures* that enable the programs to adequately manipulate information and (3) *documentation* that describes the operation and use of the programs.

Introduction: Software is Complex

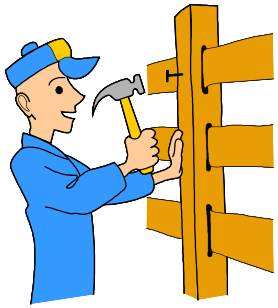
- Complex ≠ complicated
- Complex = composed of many simple parts related to one another
- Complicated = not well understood, or explained

Complexity Example: Scheduling Fence Construction Tasks

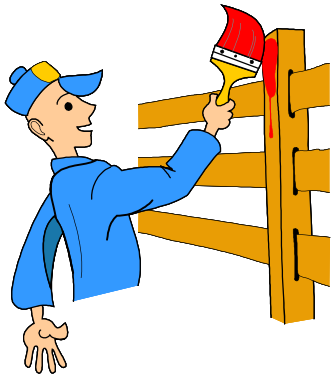


Setting posts
[3 time units]

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 = ?

[$\Rightarrow$ “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]

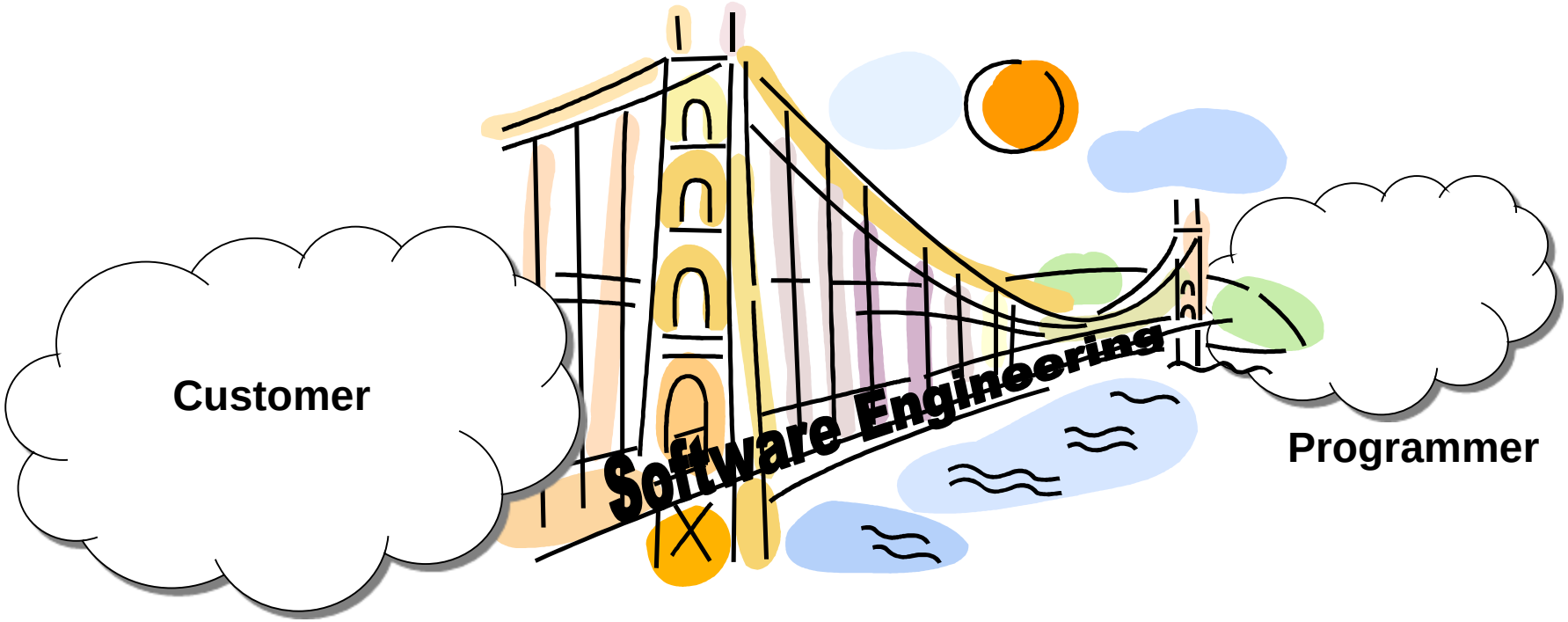
The Frog in Boiling Water

- 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 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” 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)

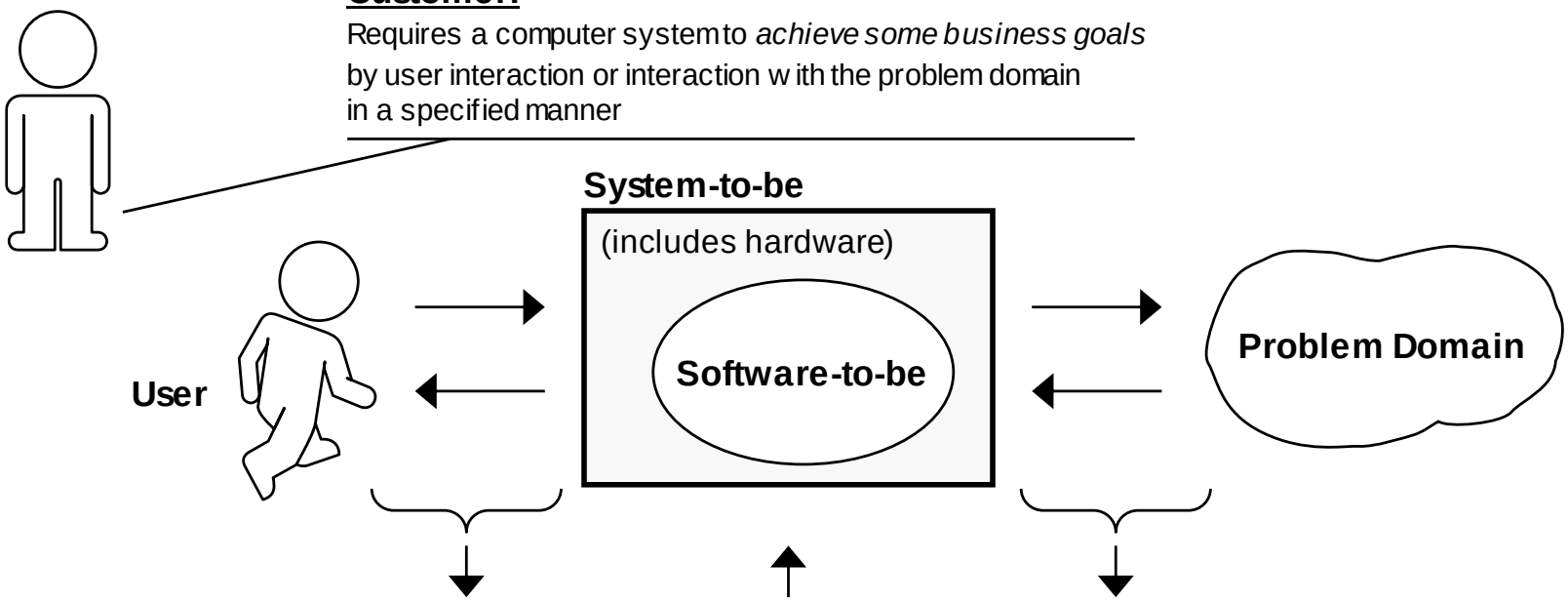
A bridge from customer needs to programming implementation



First law of software engineering

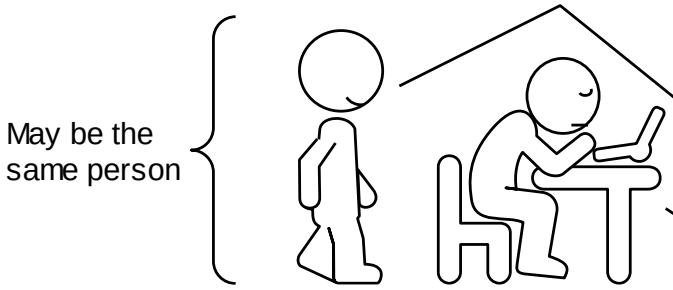
Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

The Role of Software Engg. (2)



Software Engineer's task:

To ***understand how*** the system-to-be needs to interact with the user or the problem domain so that customer's requirement is met and ***design*** the software-to-be

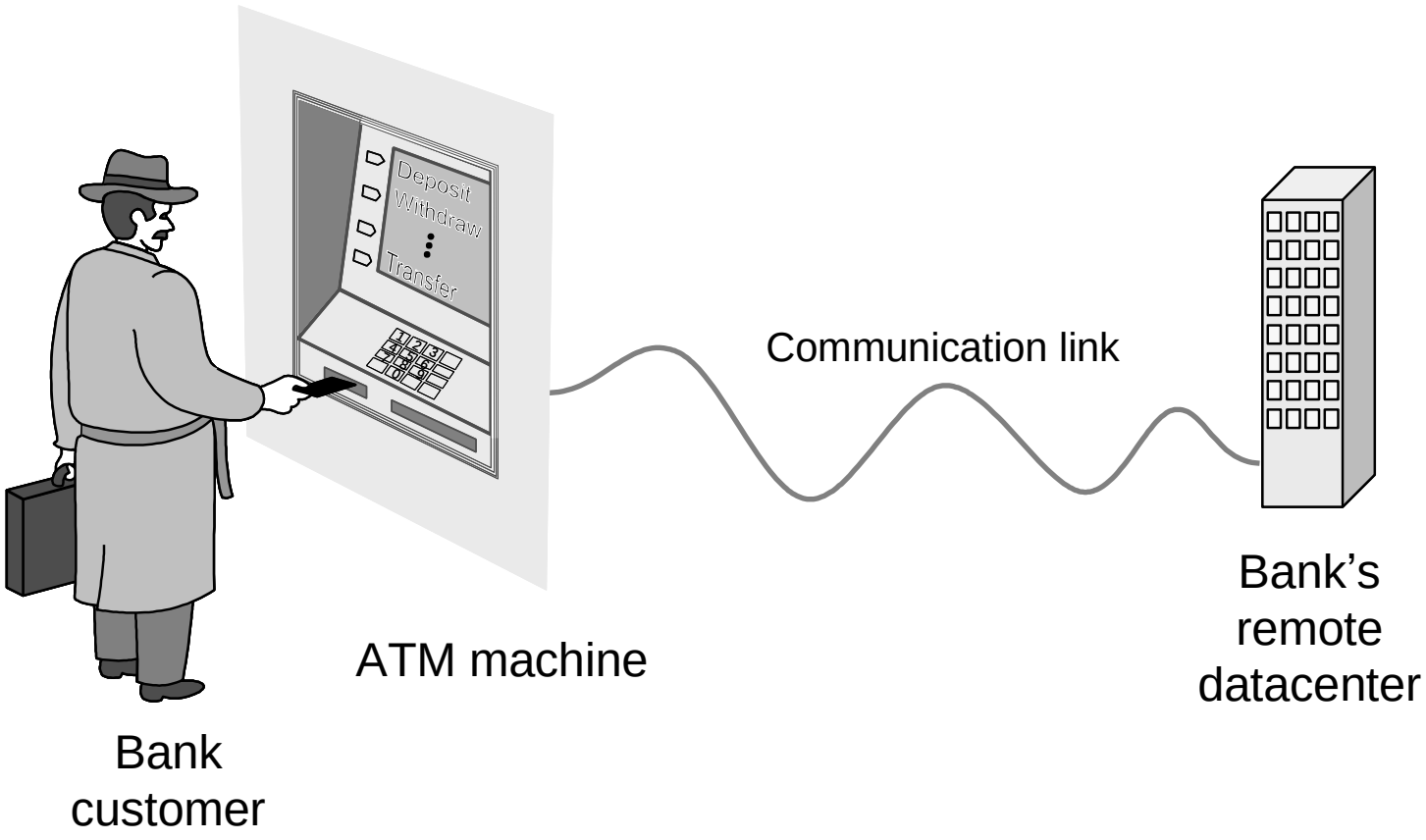


Programmer's task:

To ***implement*** the software-to-be designed by the software engineer

Example: ATM Machine

Understanding the money-machine problem:

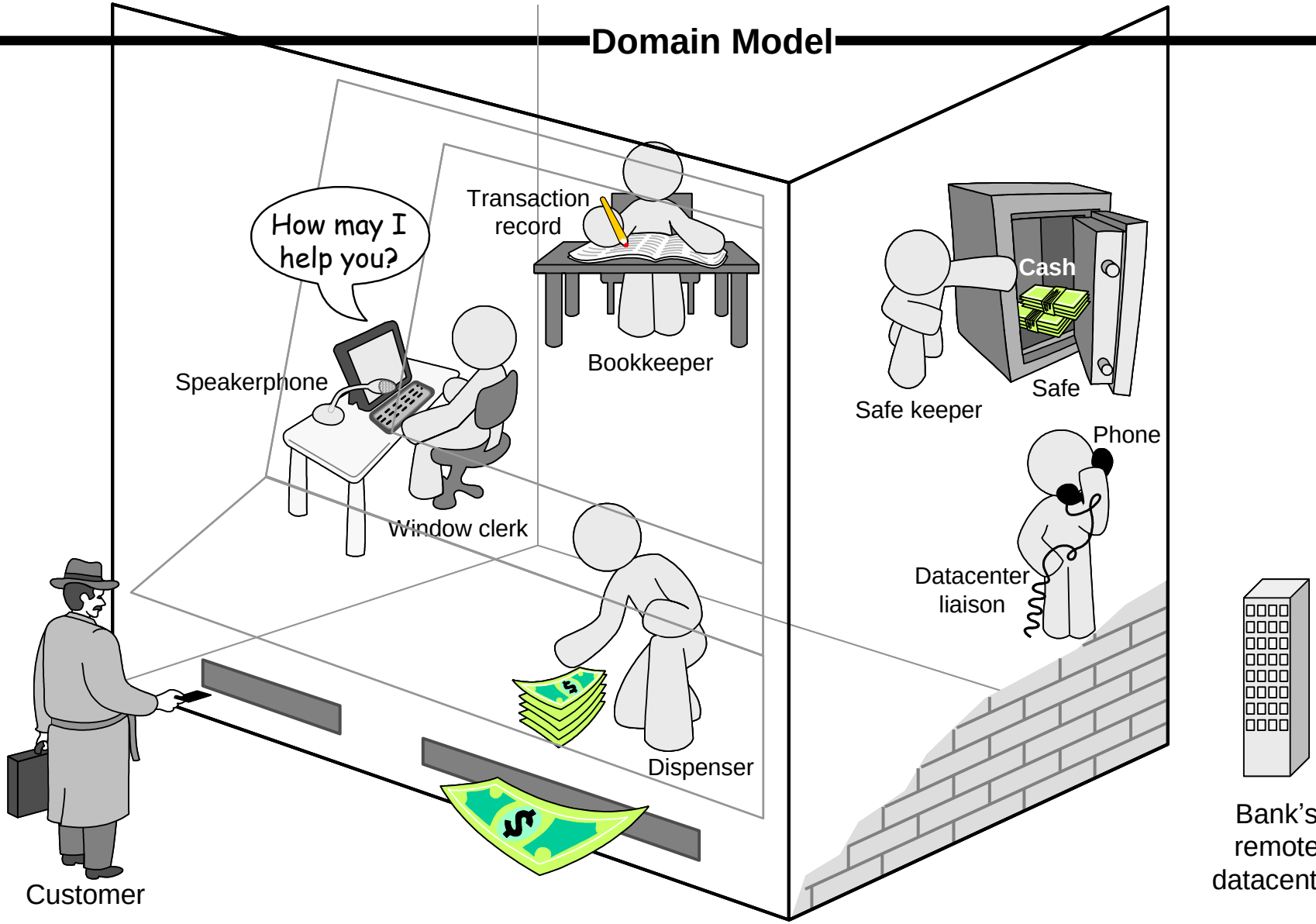


Problem-solving Strategy

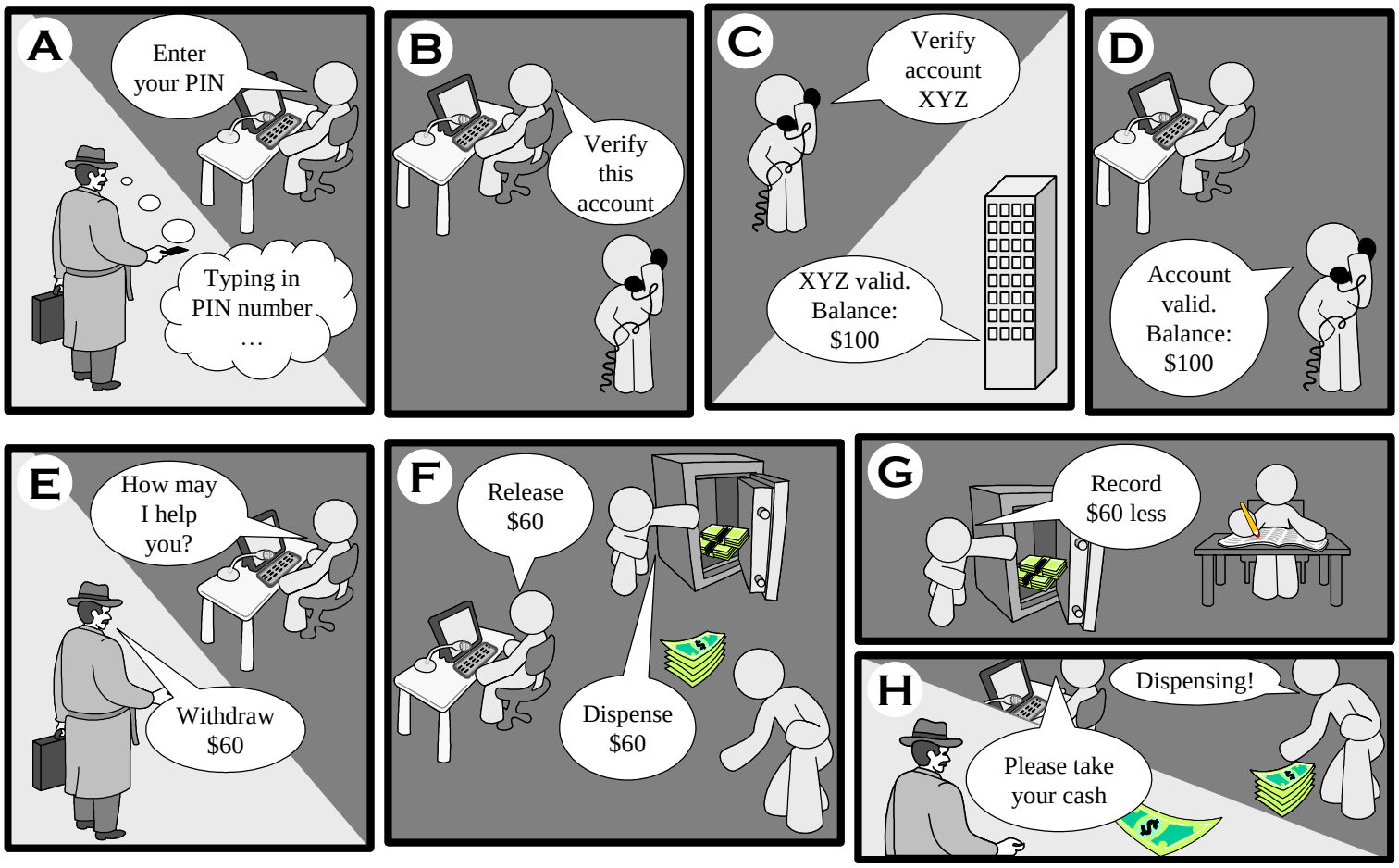
Divide-and-conquer:

- 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 …

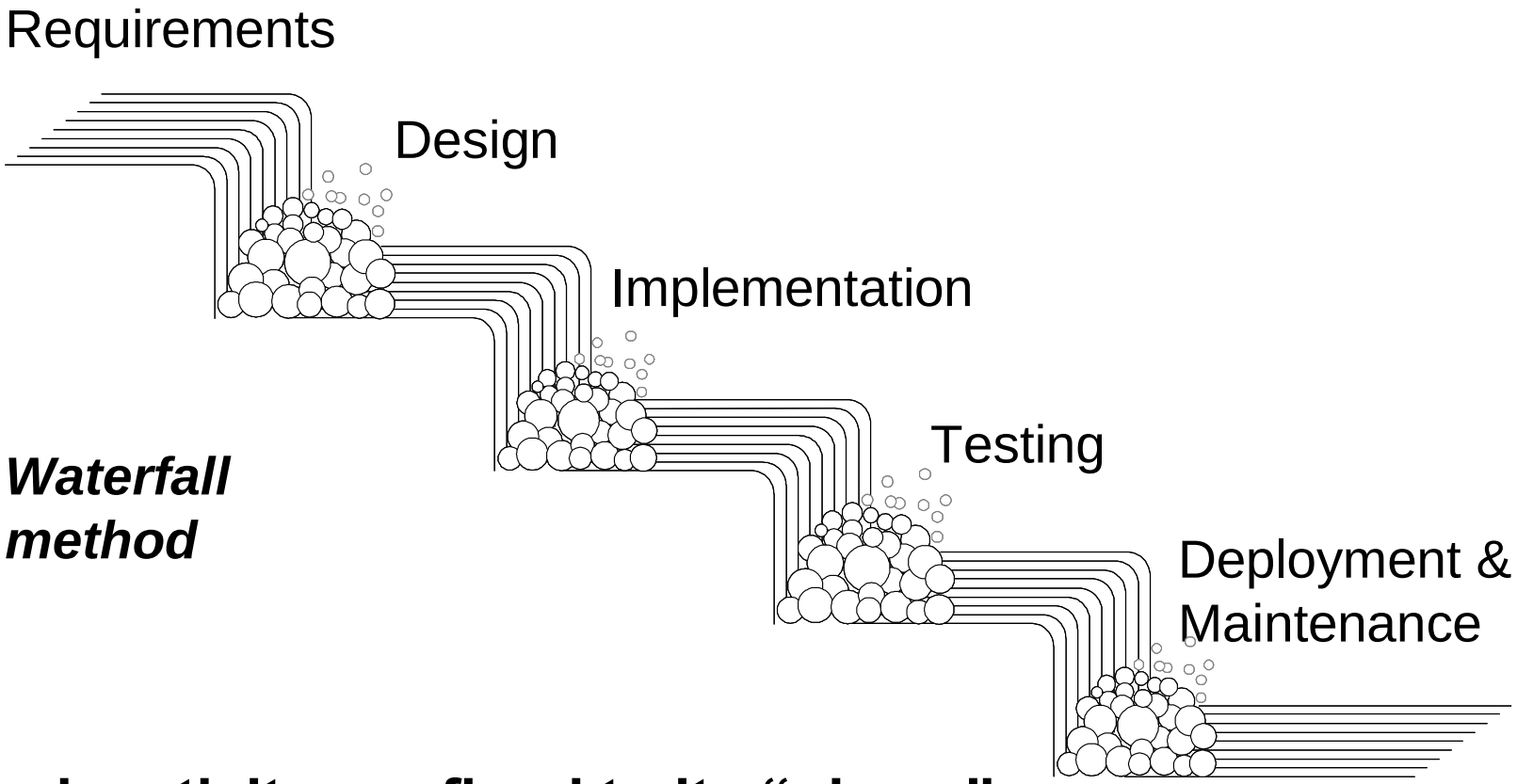
Second Law of Software Engineering

- 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

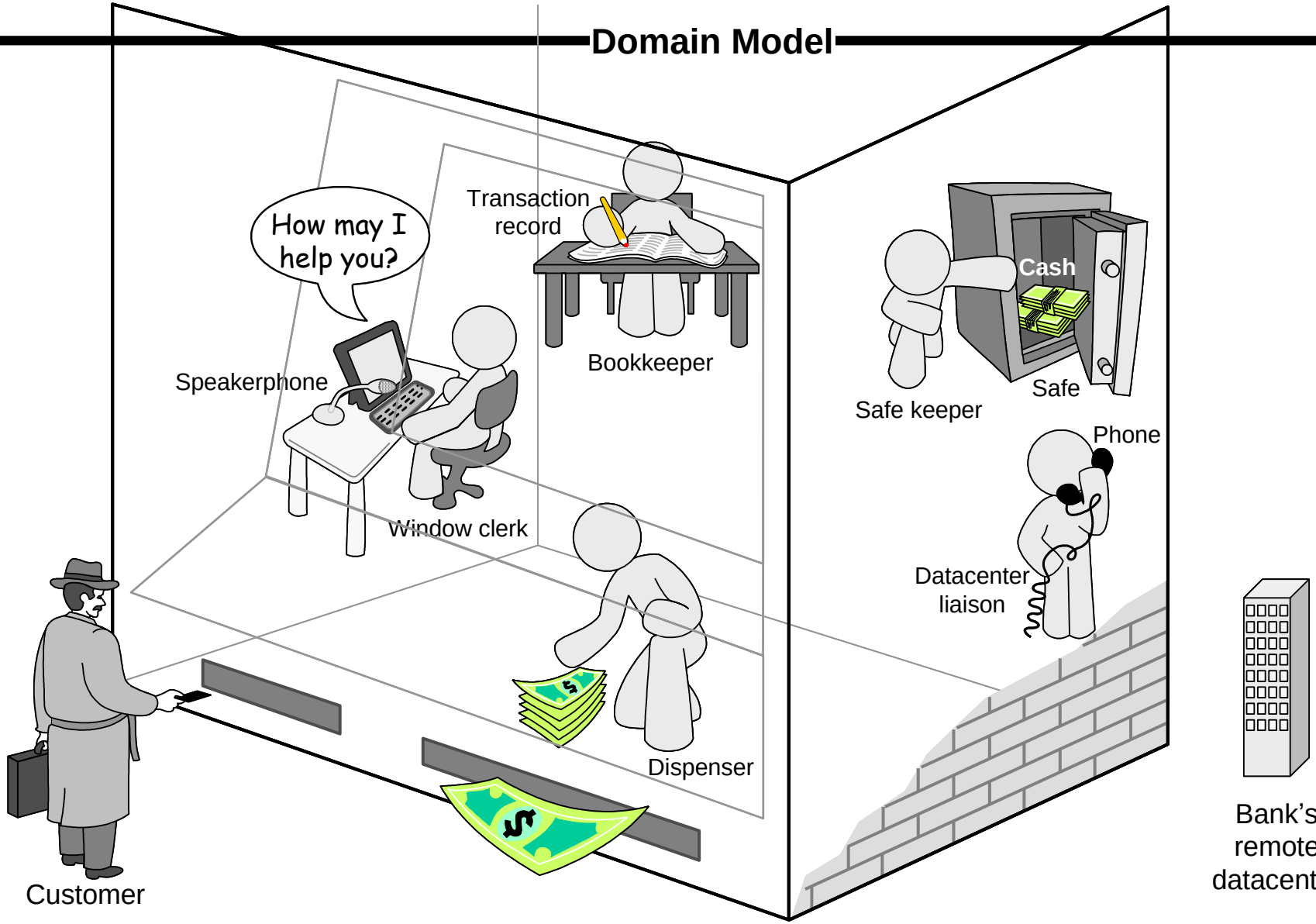
Waterfall Method



Waterfall method

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

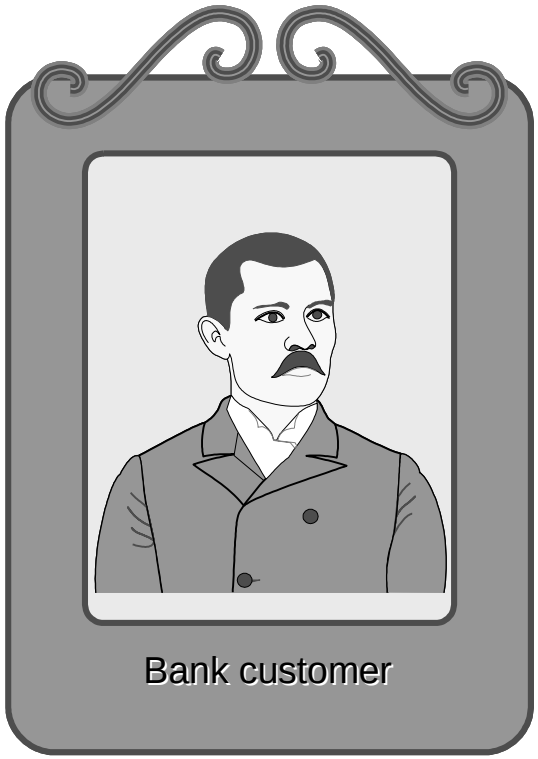
How ATM Machine Might Work



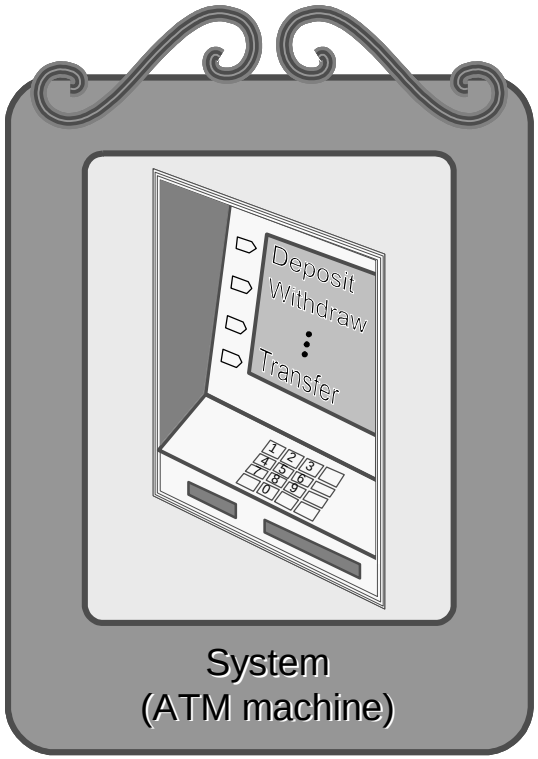
Understanding the Problem Domain

- 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

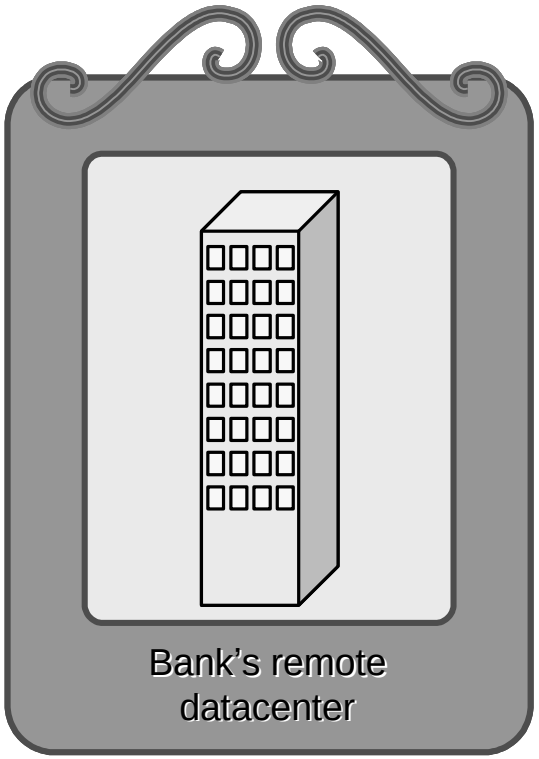
ATM: Gallery of Players



Bank customer



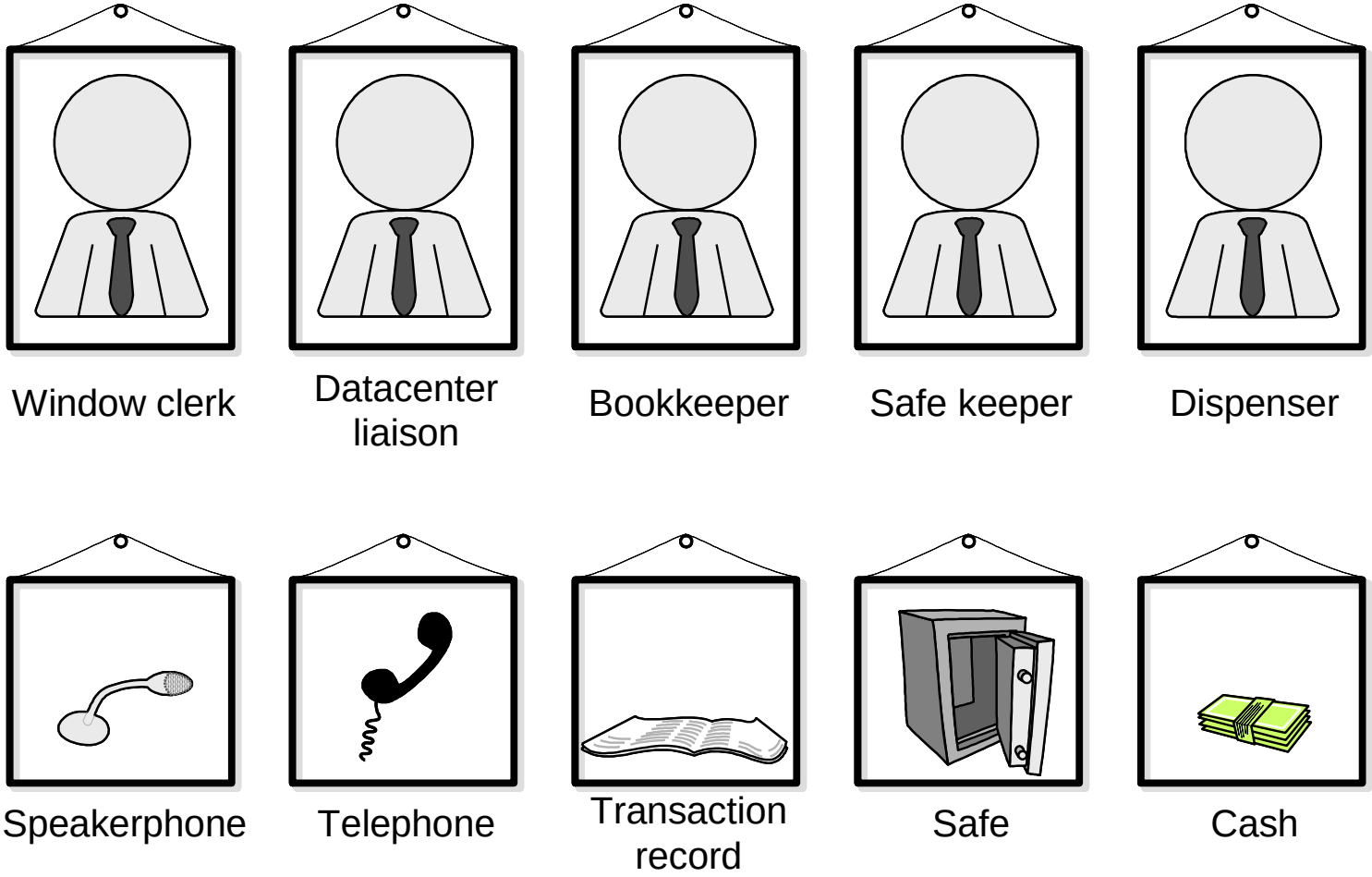
System
(ATM machine)



Bank's remote
datacenter

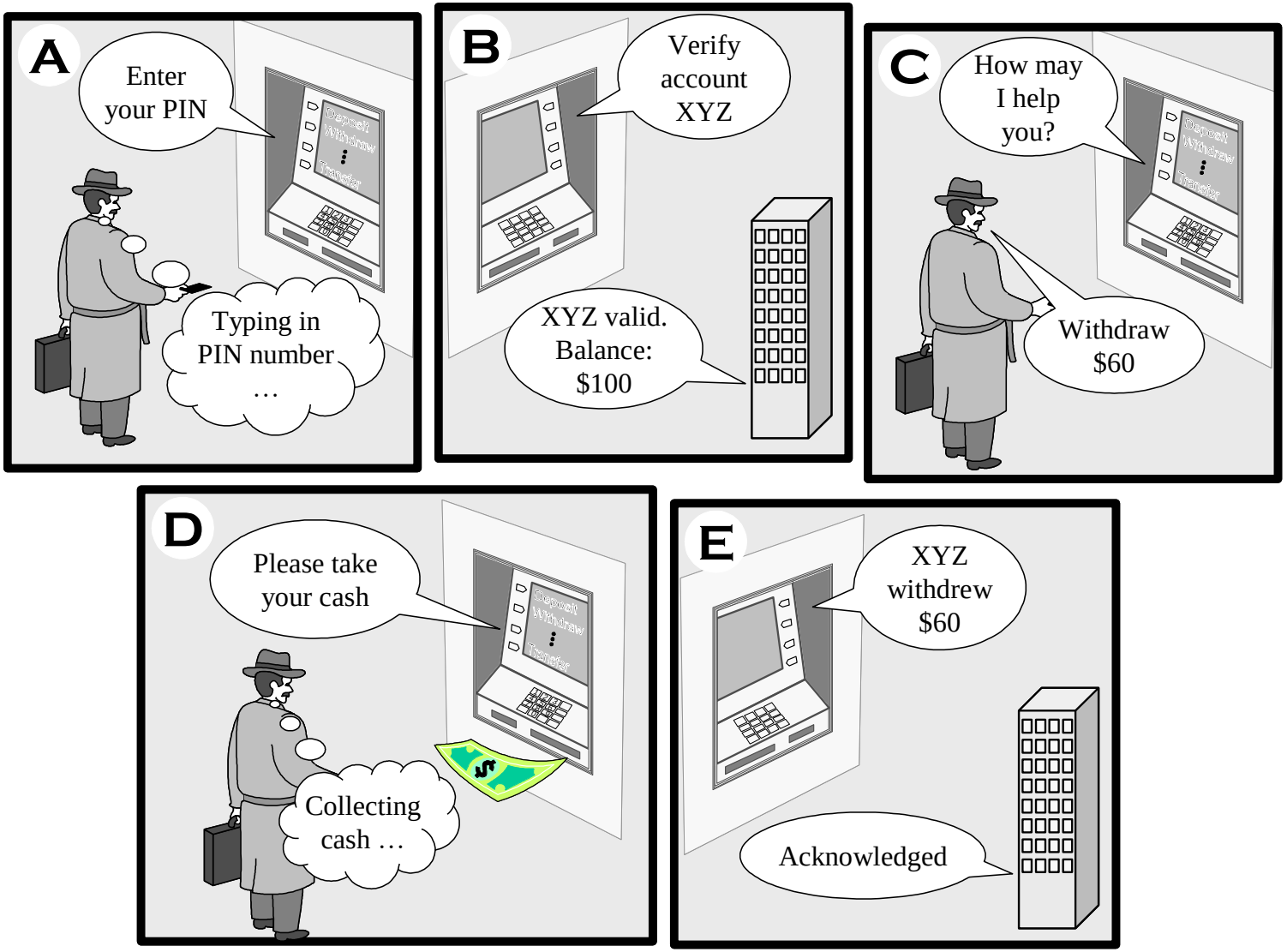
Actors (Easy to identify because they are visible!)

Gallery of Workers + Tools



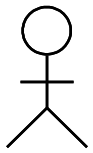
Concepts (Hard to identify because they are invisible/imaginary!)

Use Case: Withdraw Cash

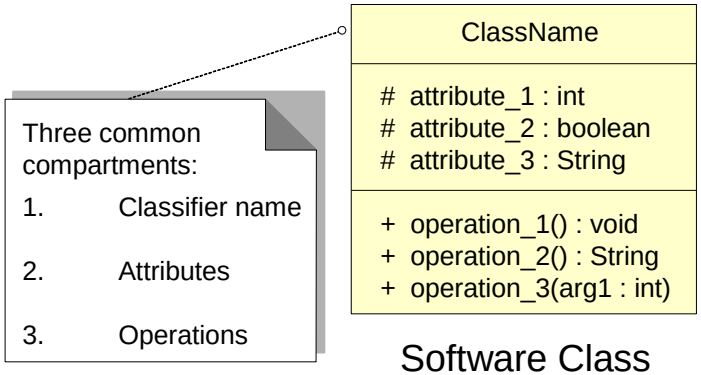


UML – Language of Symbols

UML = Unified Modeling Language

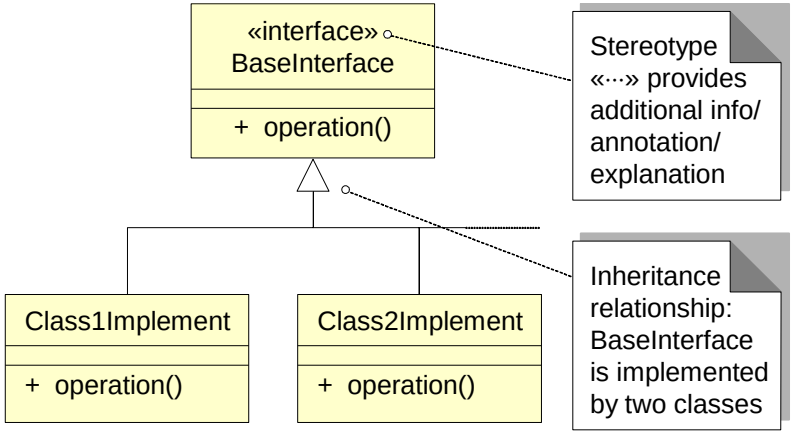


Actor

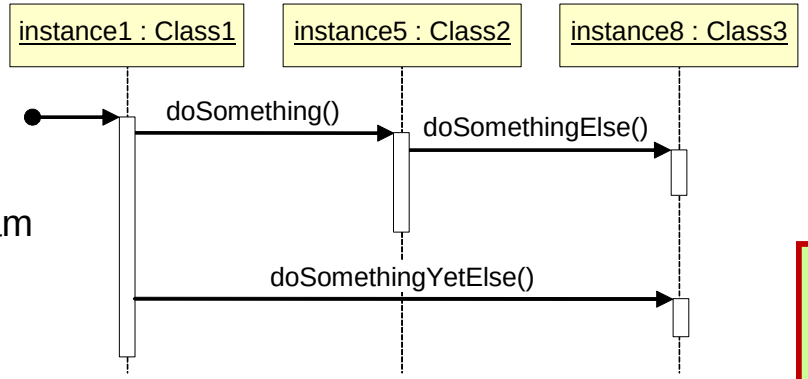


Comment

Software Class



Software Interface Implementation



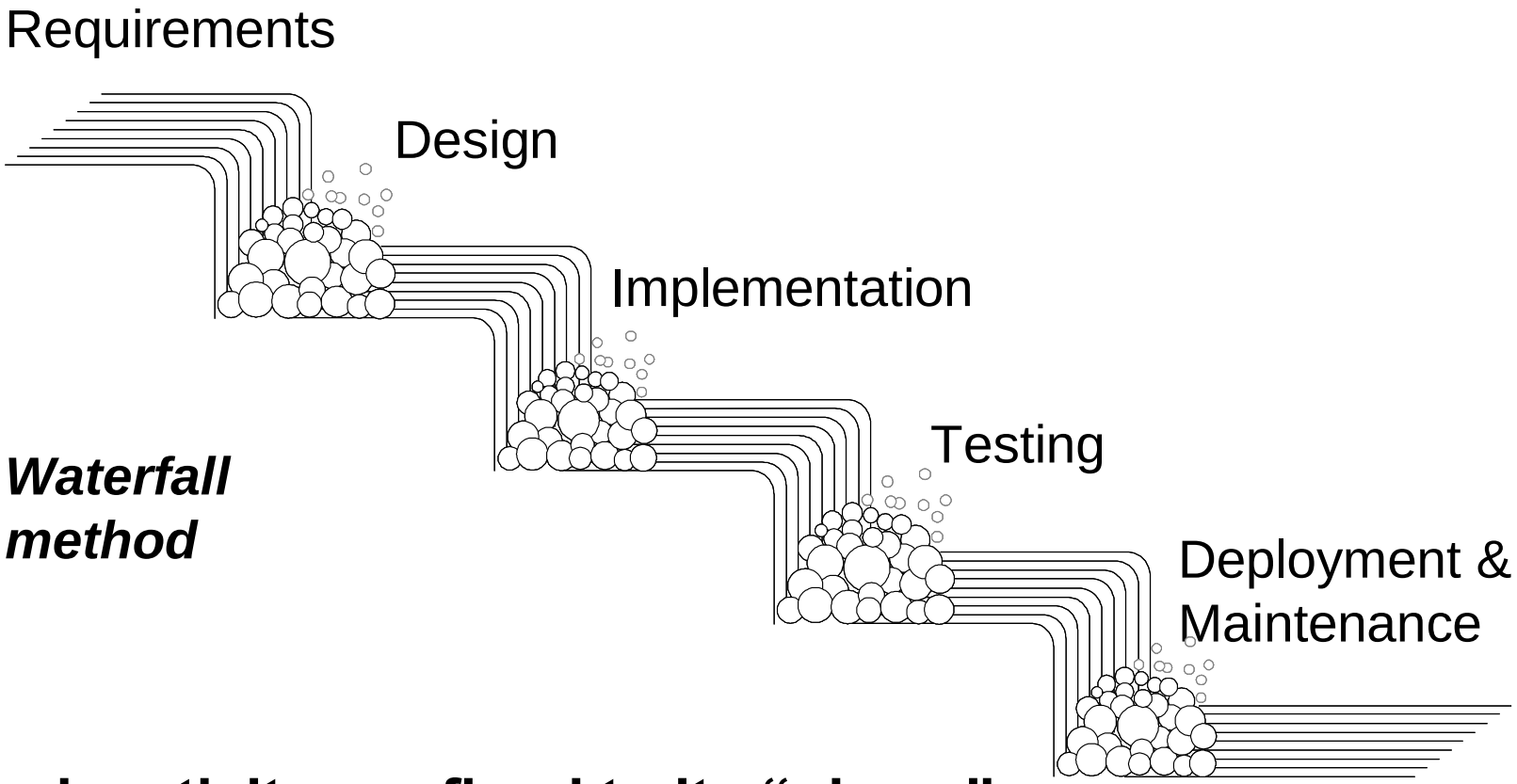
Interaction Diagram

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

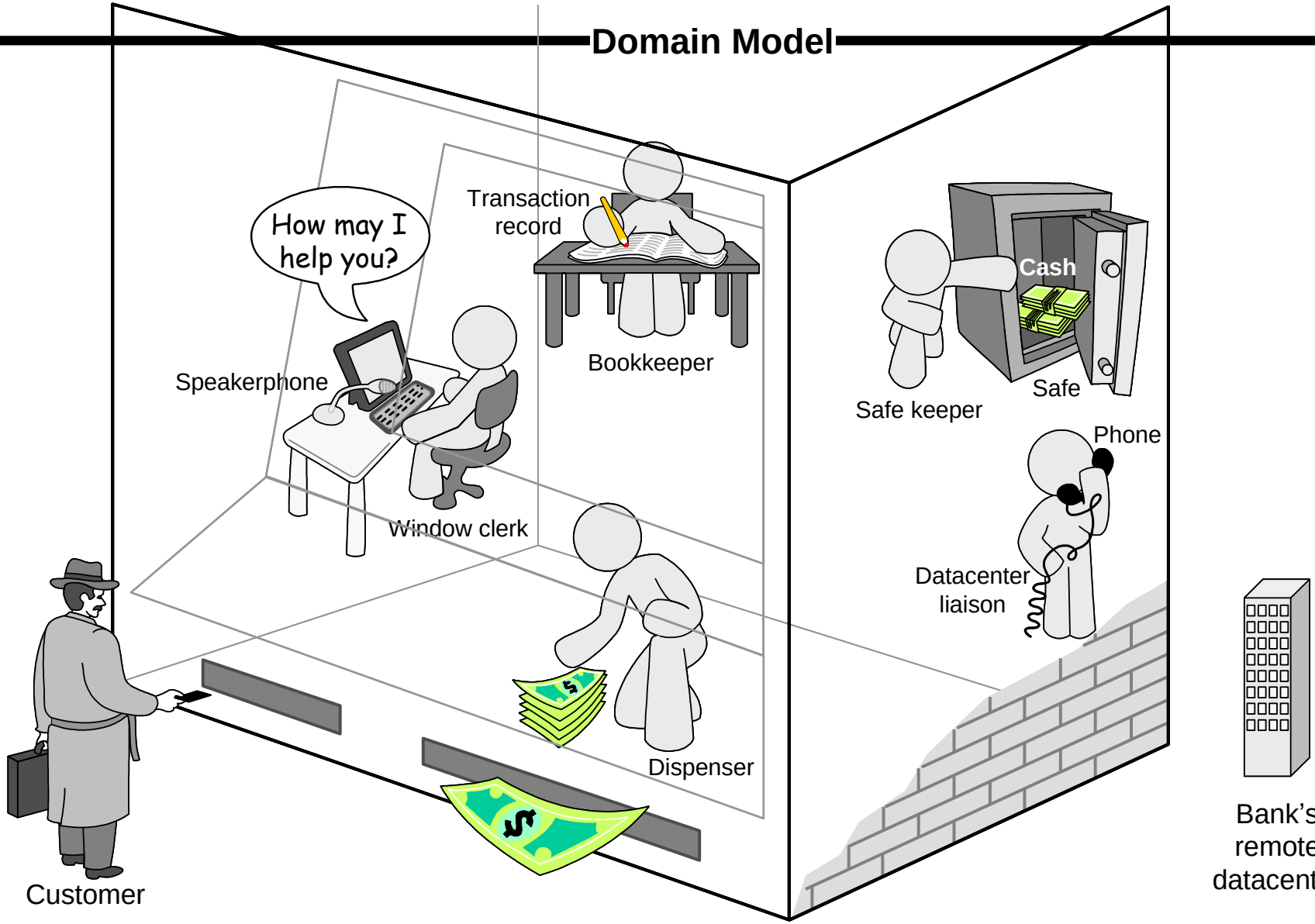
Waterfall Method



Waterfall method

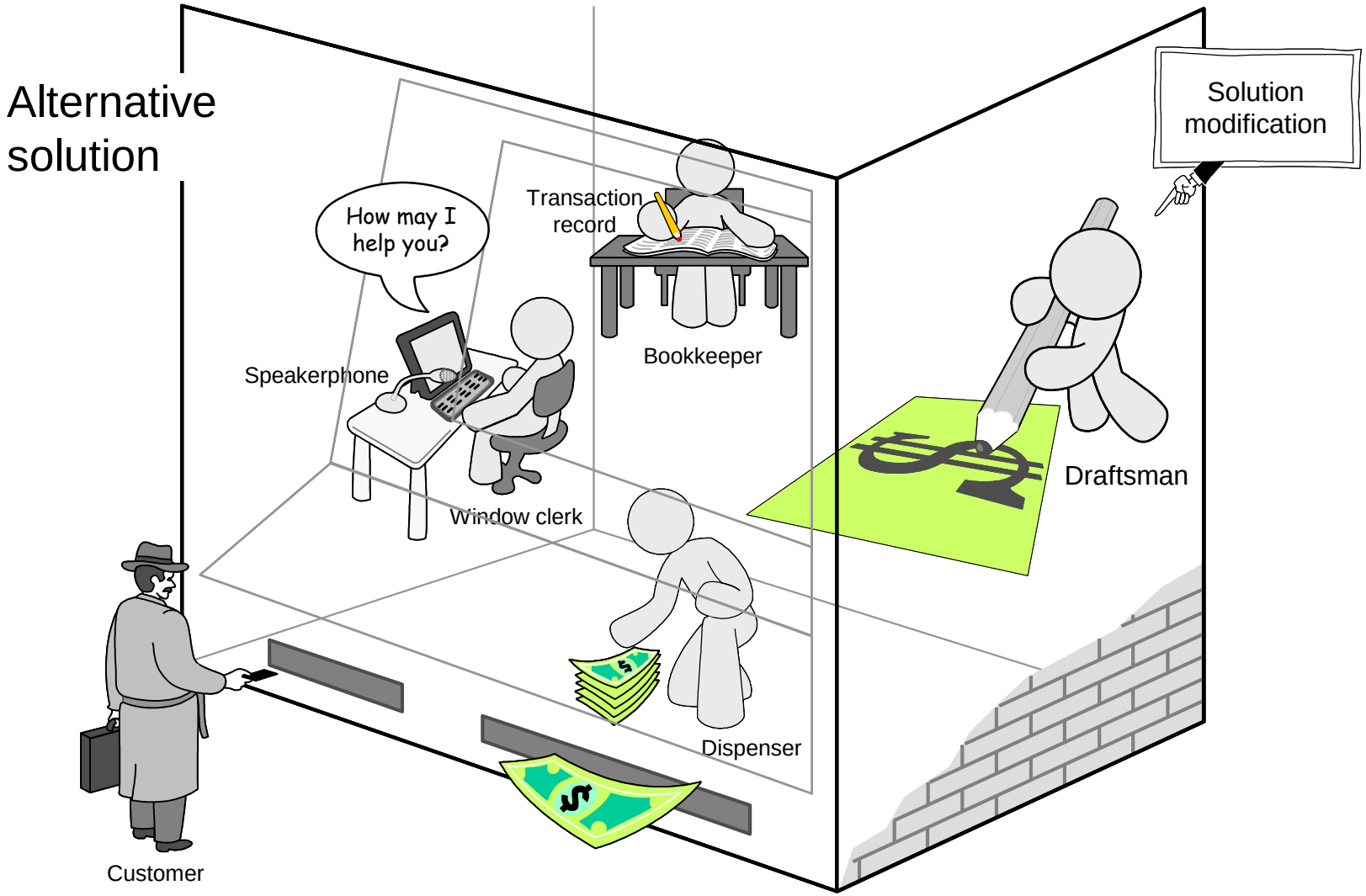
Each activity confined to its "phase".
Unidirectional, no way back;
finish this phase before moving to the next

How ATM Machine Might Work



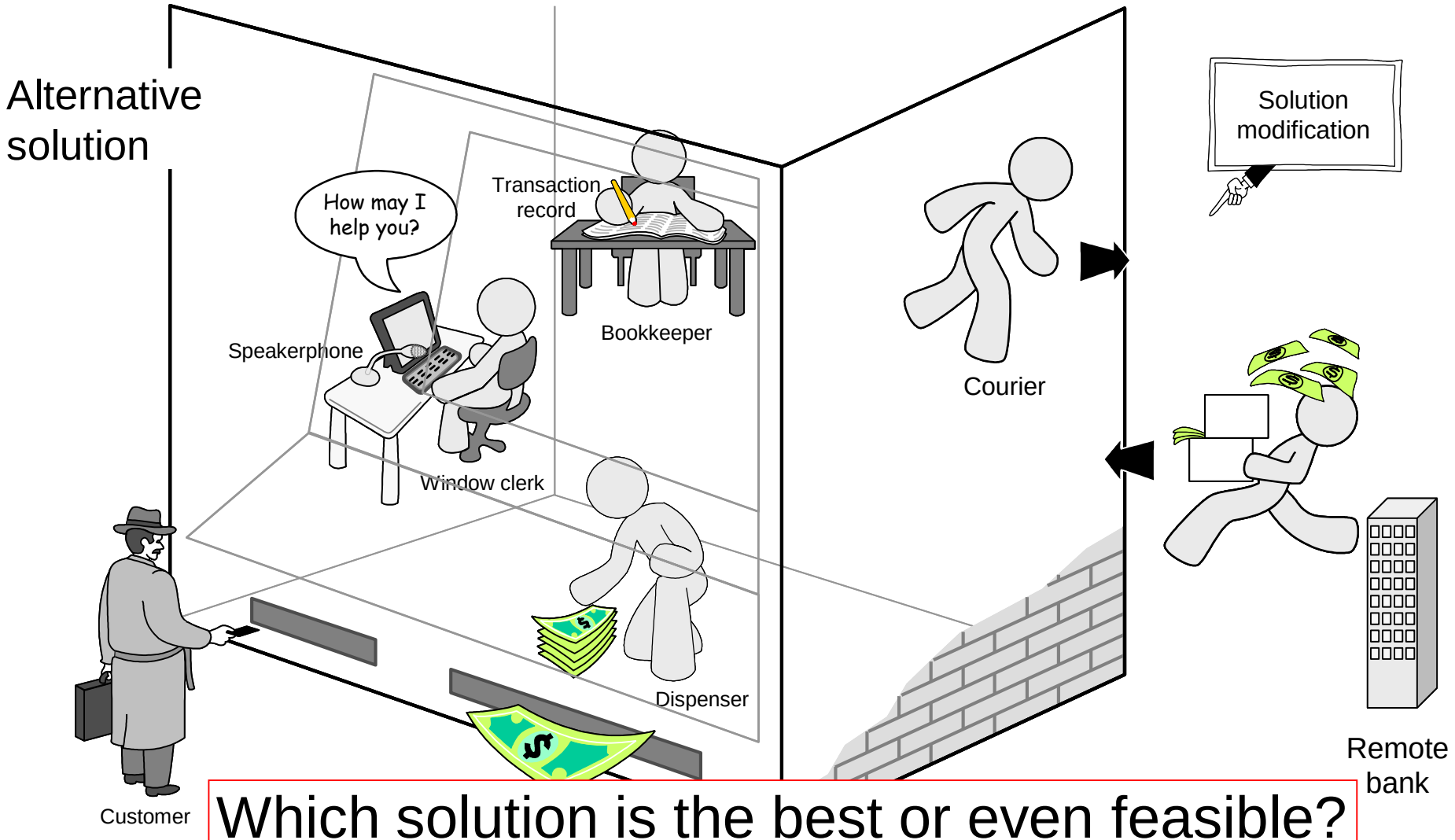
How ATM Machine Works (2)

Domain Model (2)



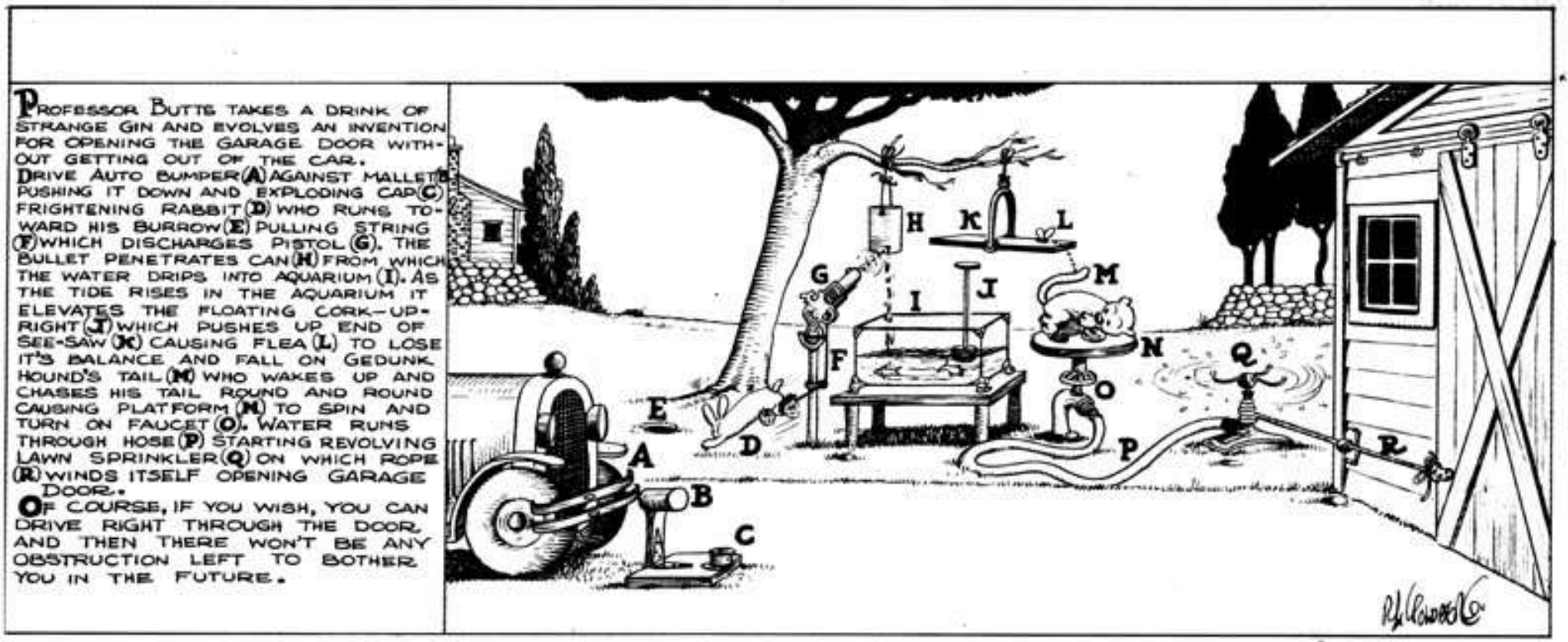
How ATM Machine Works (3)

Domain Model (3)

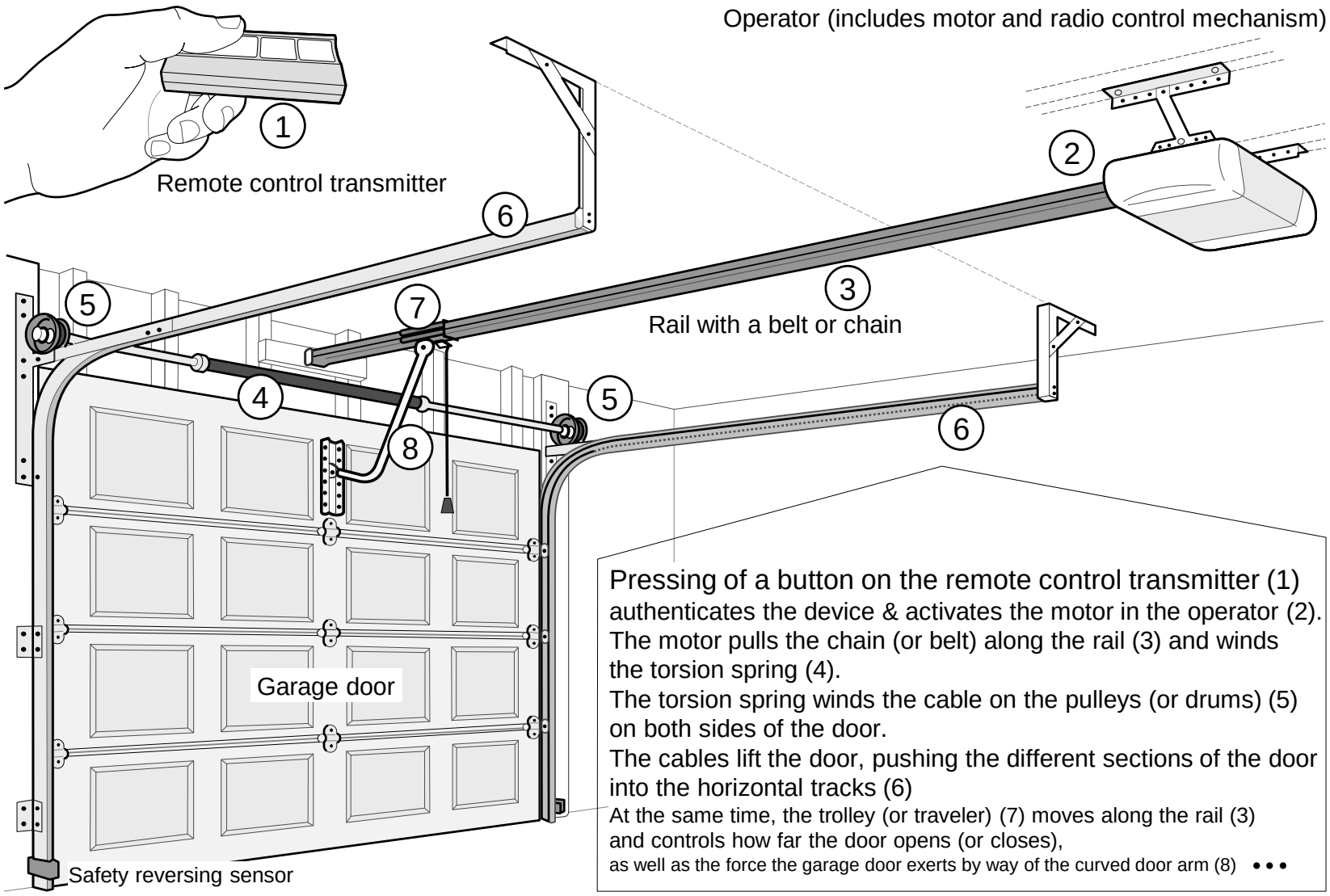


Rube Goldberg Design

Garage door opener



Actual Design

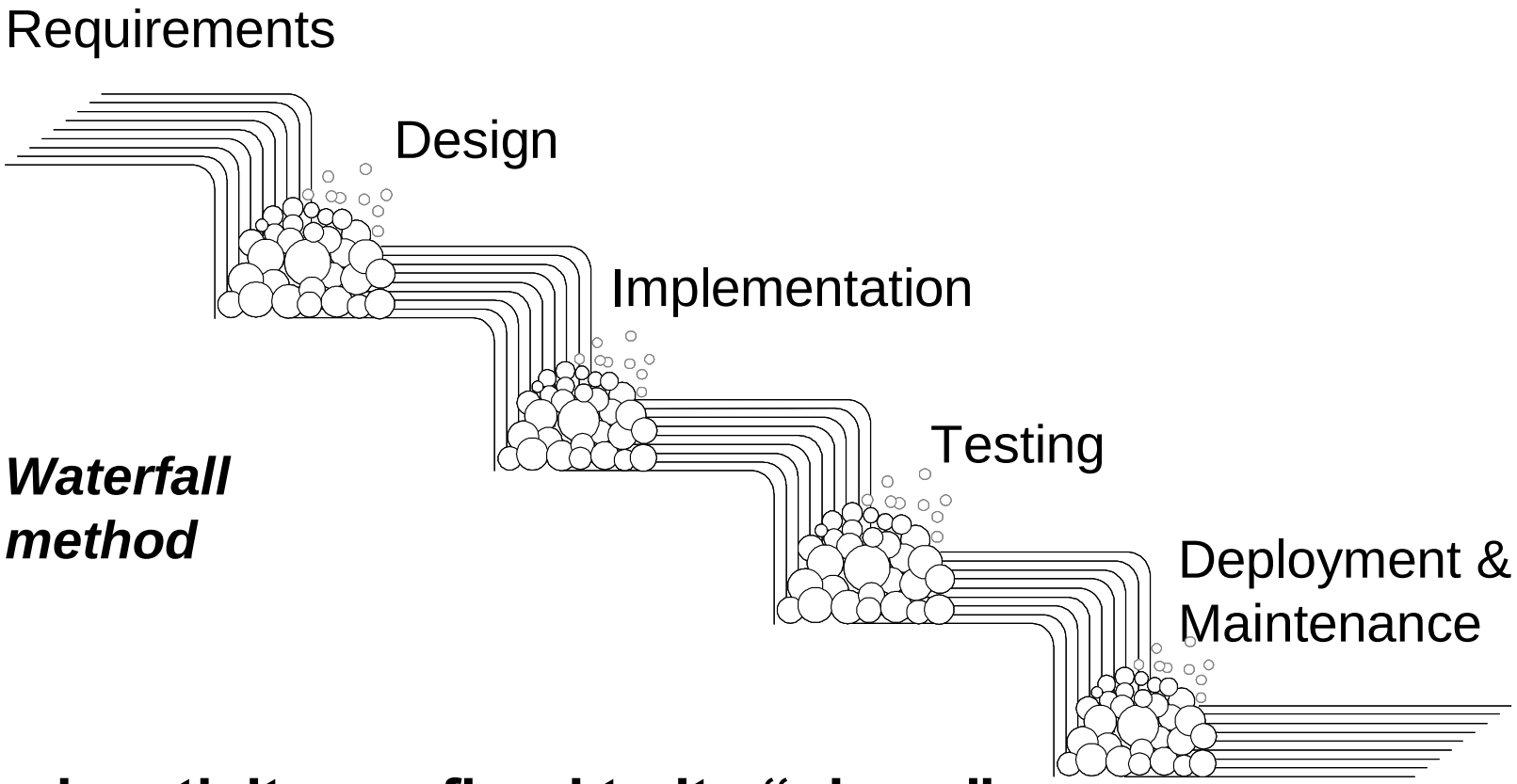


Pressing of a button on the remote control transmitter (1) authenticates the device & activates the motor in the operator (2).
The motor pulls the chain (or belt) along the rail (3) and winds the torsion spring (4).
The torsion spring winds the cable on the pulleys (or drums) (5) on both sides of the 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!)

Waterfall Method



Waterfall method

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

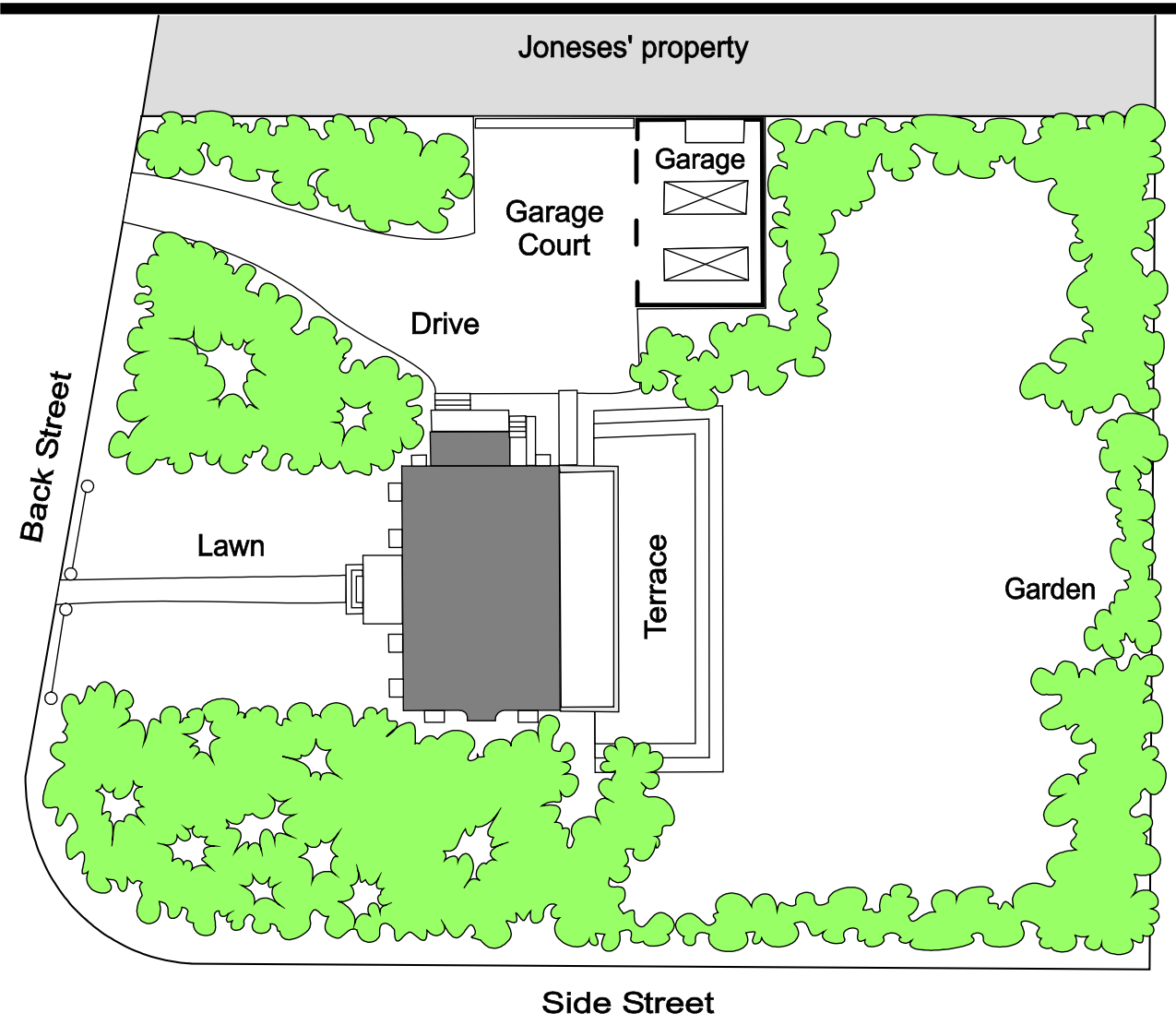
Software Measurement

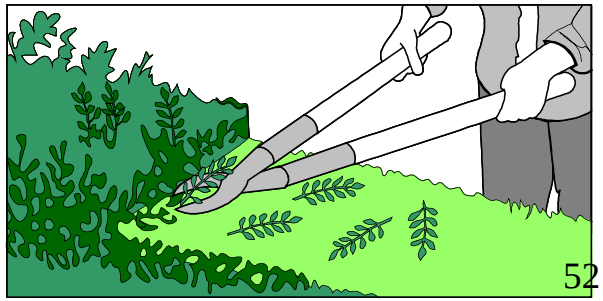
- What to measure?
  - Project (developer's work),

    for budgeting and scheduling
  - Product,

    for quality assessment

Formal hedge pruning



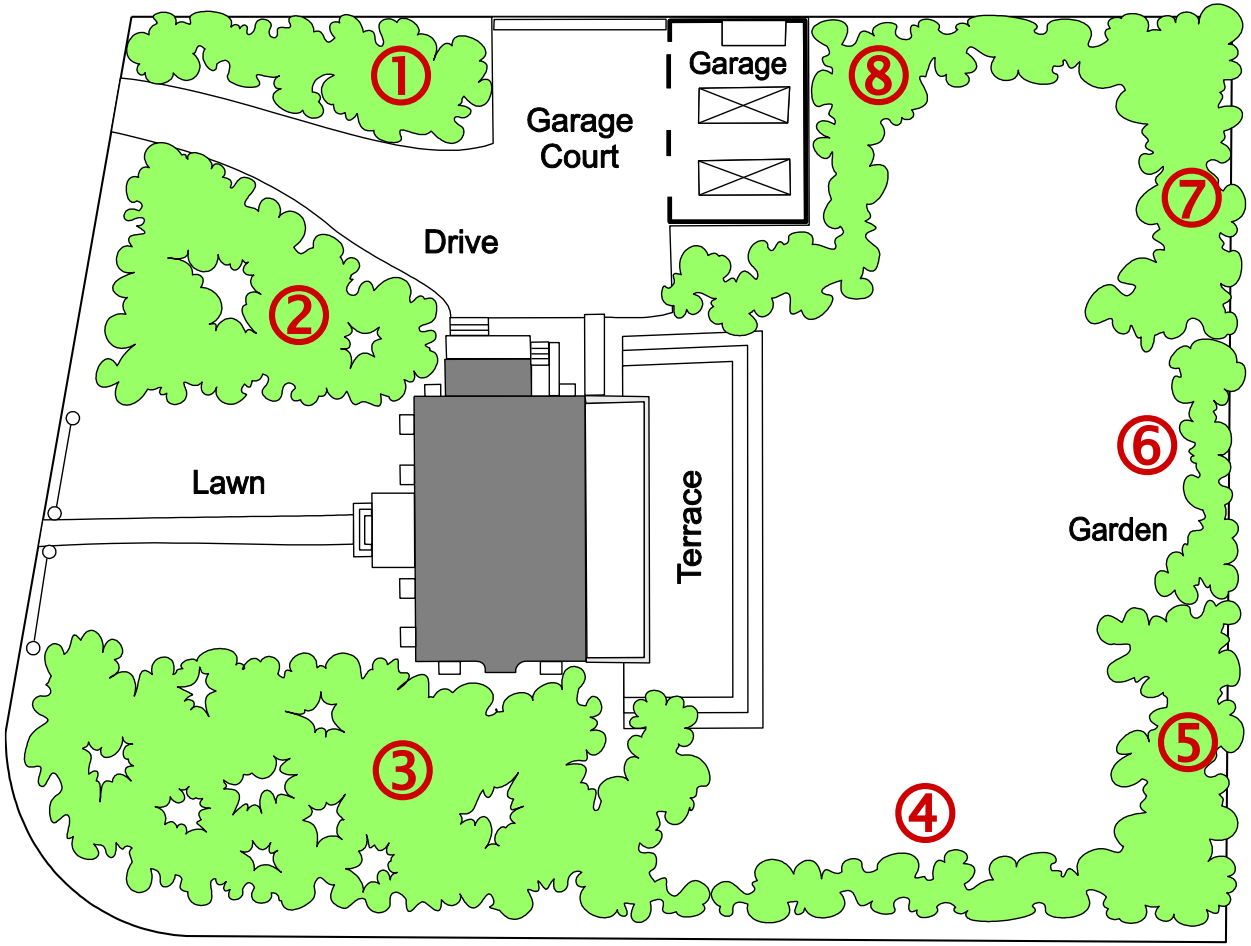


Work Estimation Strategy

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)

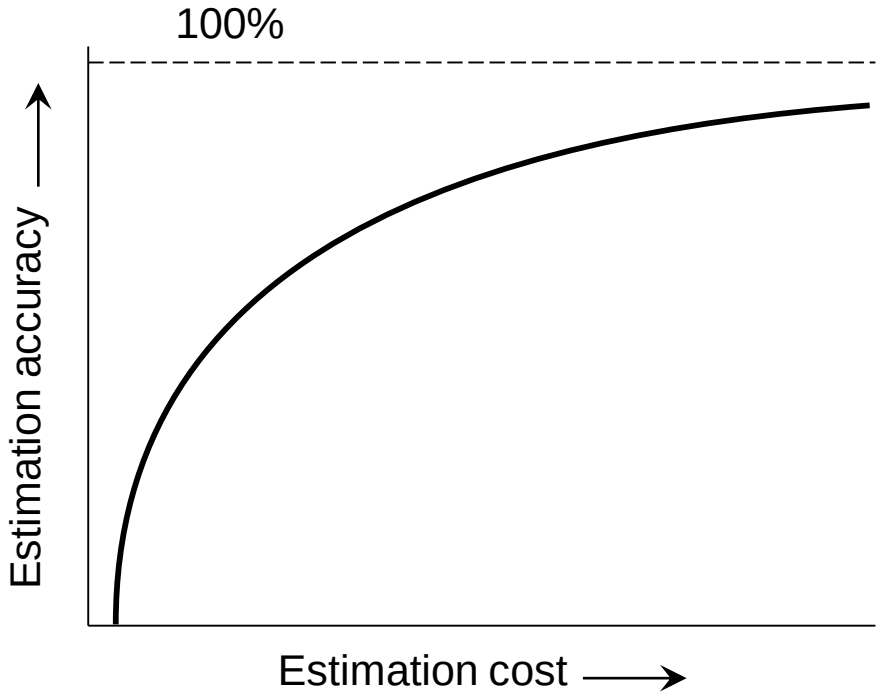
- Assumptions:
  - Relative size estimates are accurate
    - That's why parts should be small & similar-size!
- Advantages:
  - Velocity estimate may need to be adjusted (based on observed progress)
  - However, the total duration can be recomputed quickly
    - Provided that the relative size estimates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

<u>Unfortunately</u>:

- Unlike hedges, software is mostly invisible and does not exist when project is started

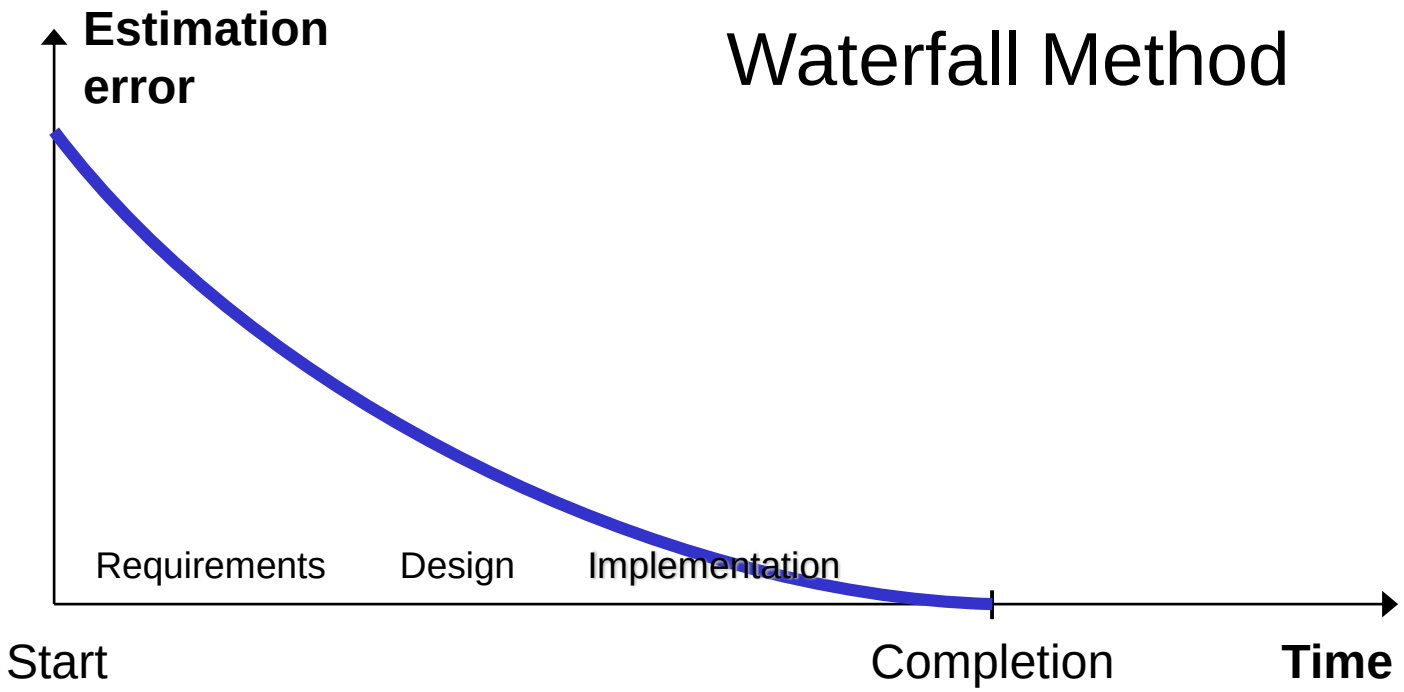
  ➔ The initial estimate hugely depends on experience and imagination

Exponential Cost of Estimation



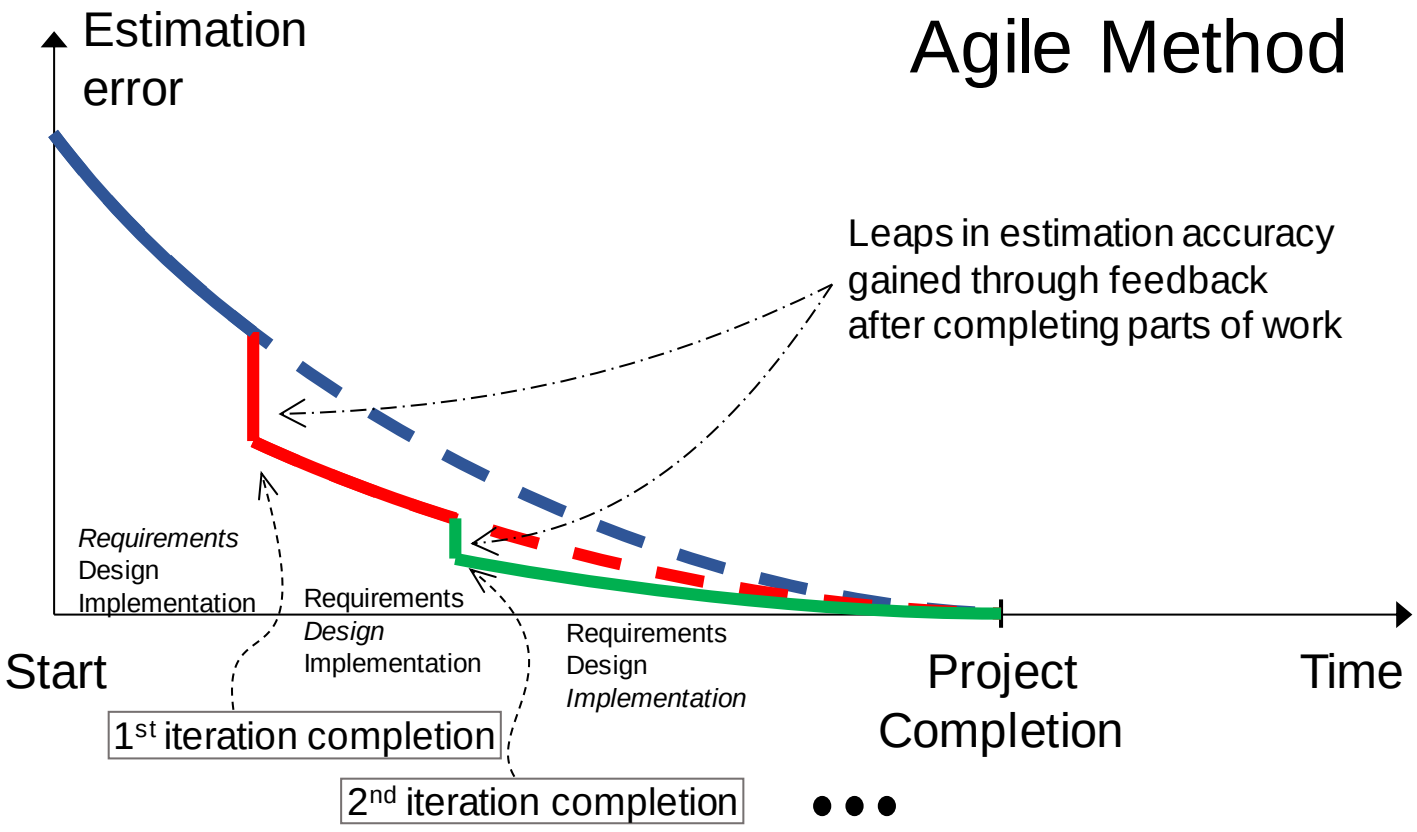
- **Improving accuracy of estimation beyond a certain point requires huge cost and effort (known as the law of diminishing returns)**
- **In the beginning of the curve, a modest effort investment yields huge gains in accuracy**

Estimation Error Over Time



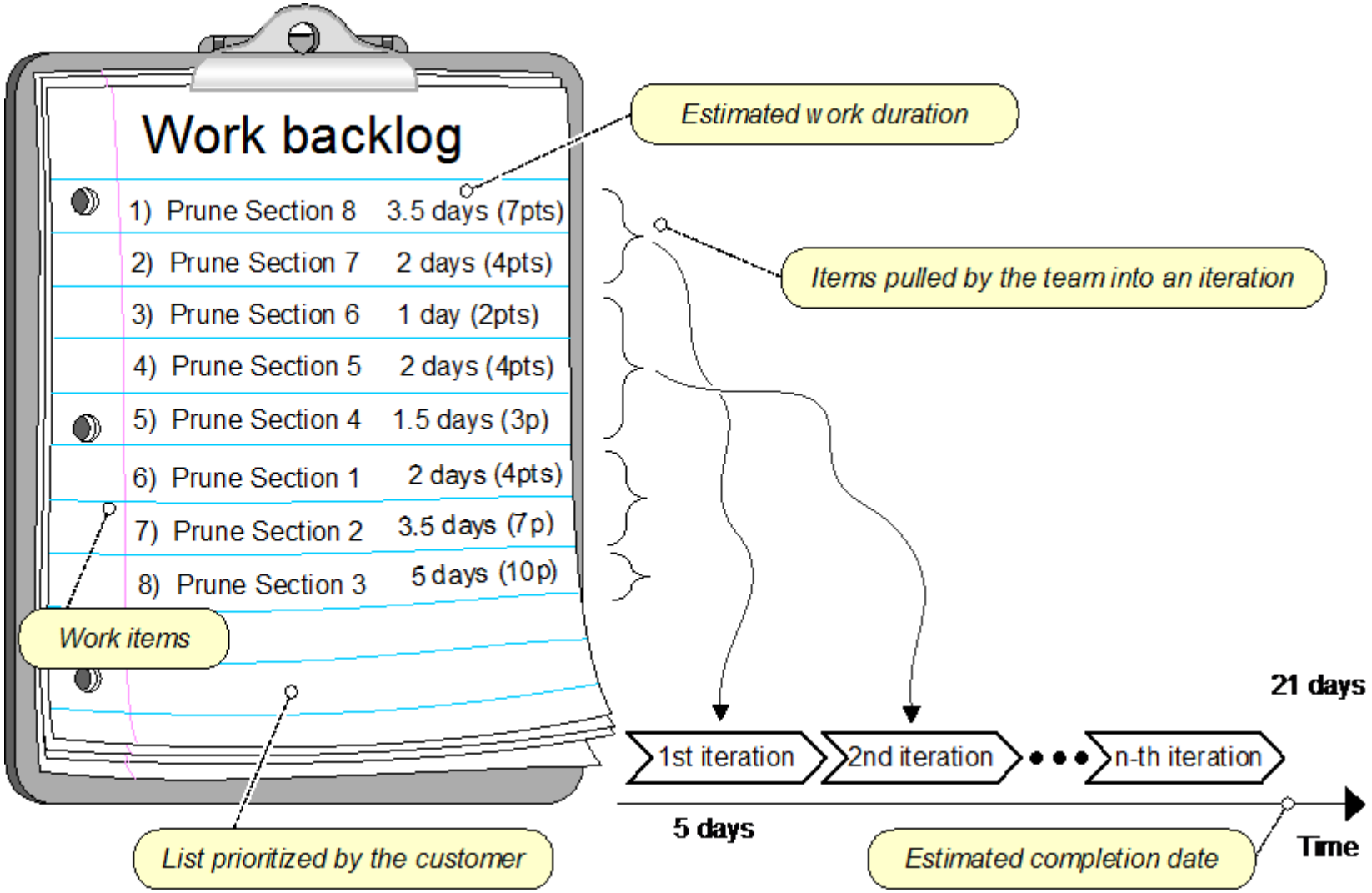
Waterfall method *cone of uncertainty* starts high and *gradually* converges to zero as the project approaches completion.

Estimation Error Over Time

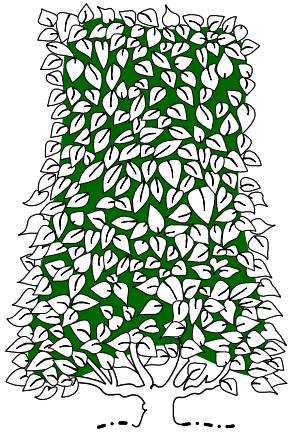


Agile method *cone of uncertainty* starts high and *in leaps* converges to zero as the project approaches completion.

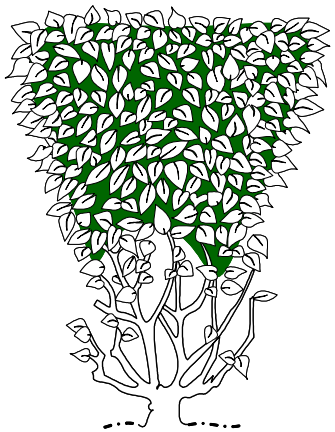
Agile Project Effort Estimation



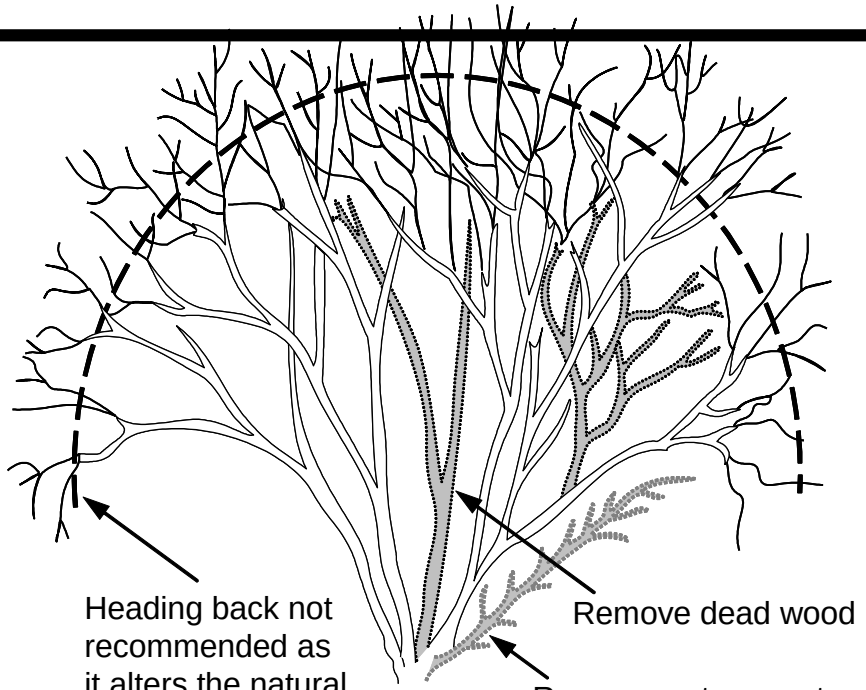
Measuring Quality of Work

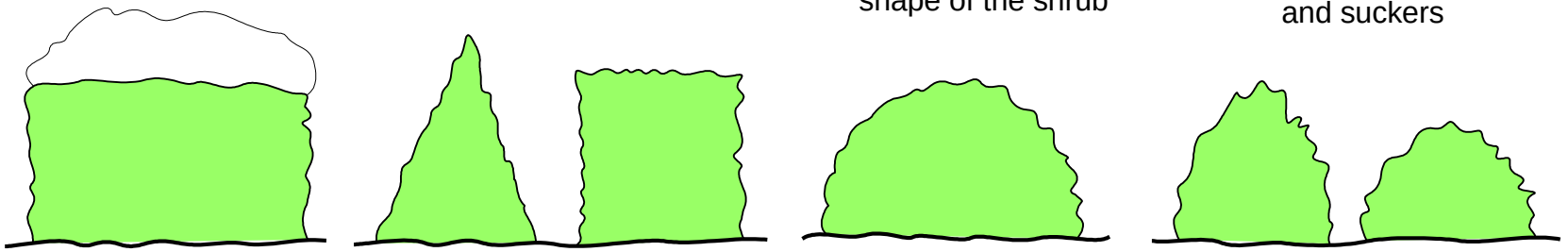


Good Shape
(Low branches get sun)



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

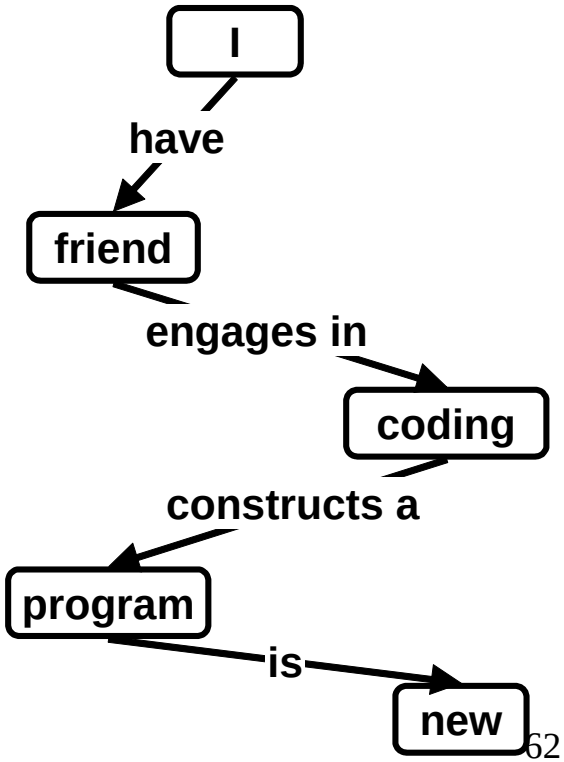
Concept Maps

Useful tool for problem domain description

SENTENCE: "My friend is coding a new program"

translated into propositions

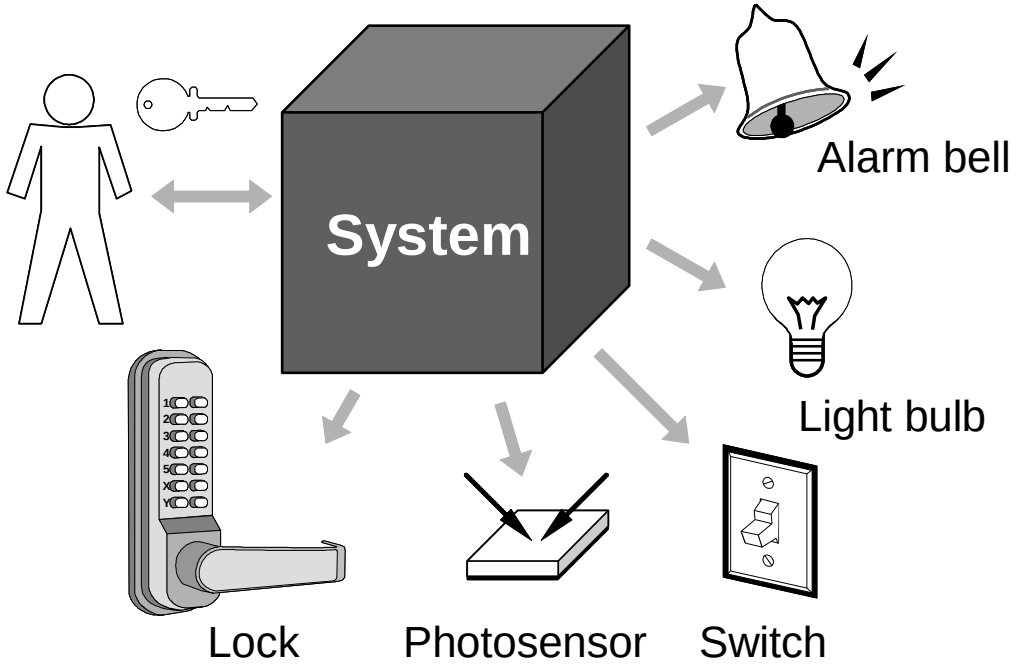
Proposition	Concept	Relation	Concept
1.	I	have	friend
2.	friend	engages in	coding
3.	coding	constructs a	program
4.	program	is	new



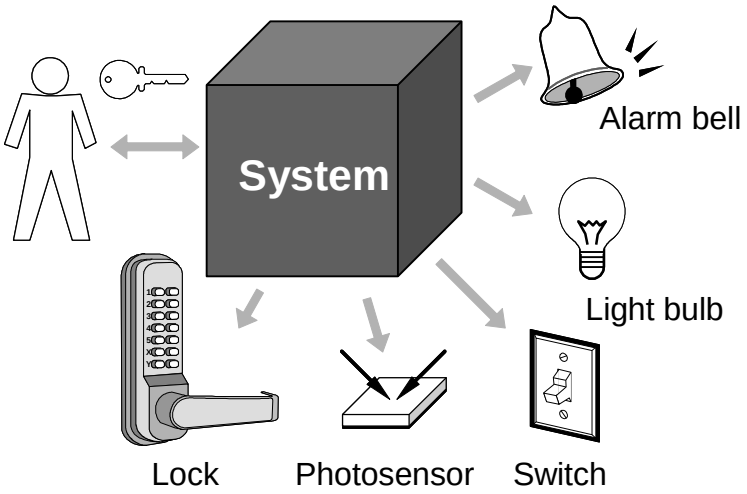
Search the Web for Concept Maps

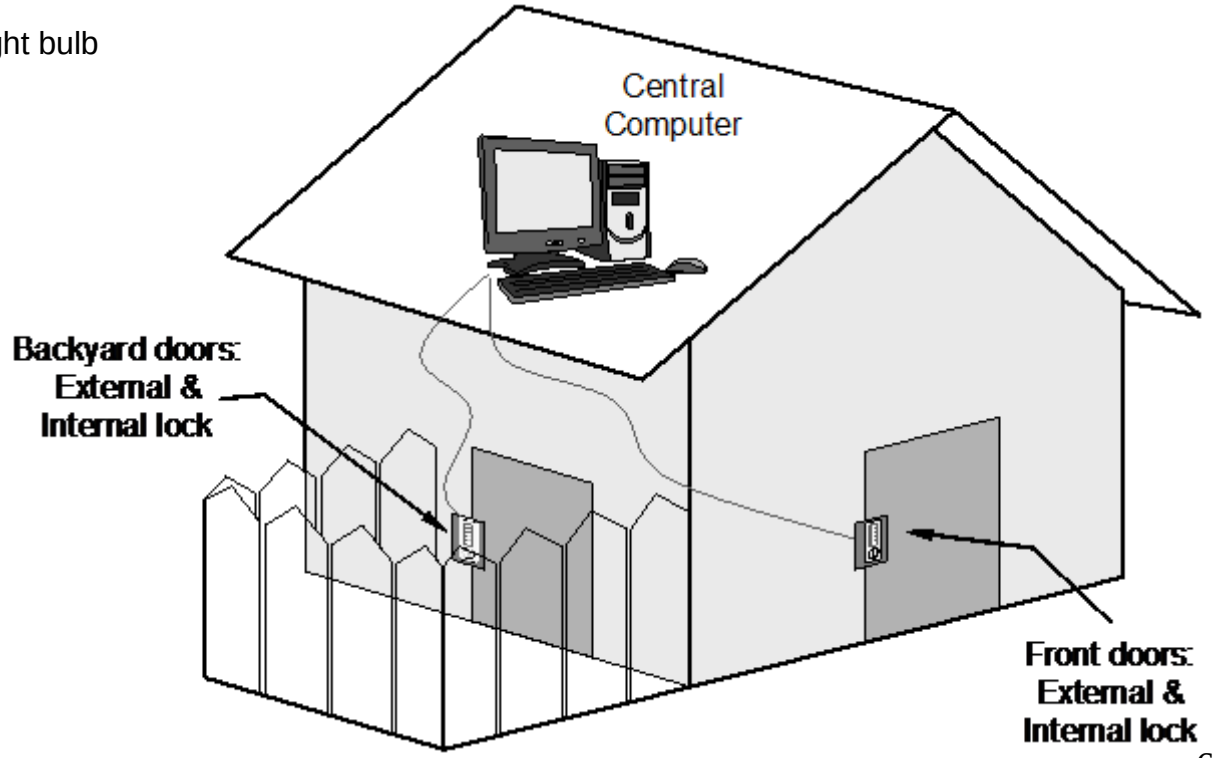
Case Study: Home Access Control

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



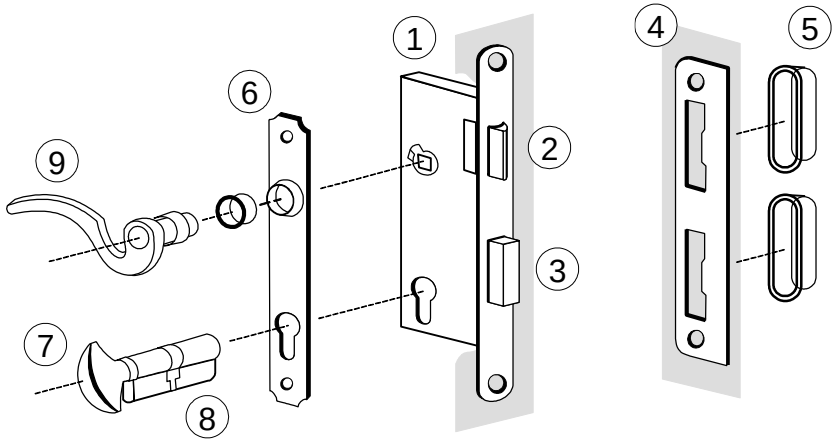
Case Study – More Details



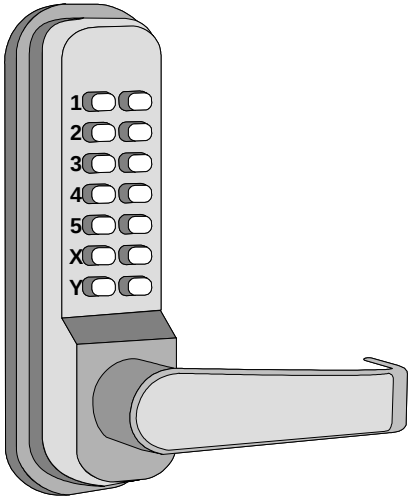


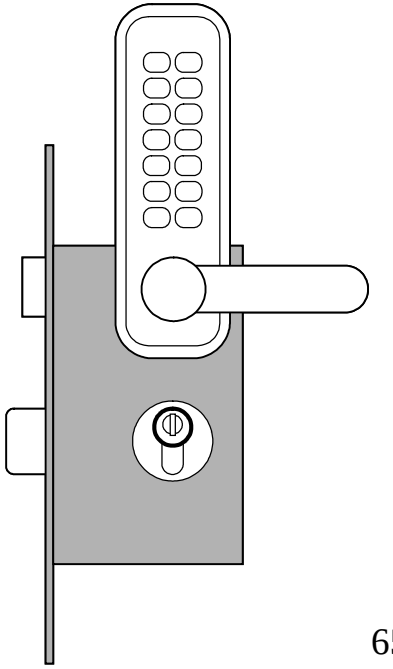
Know Your Problem

Mortise Lock Parts

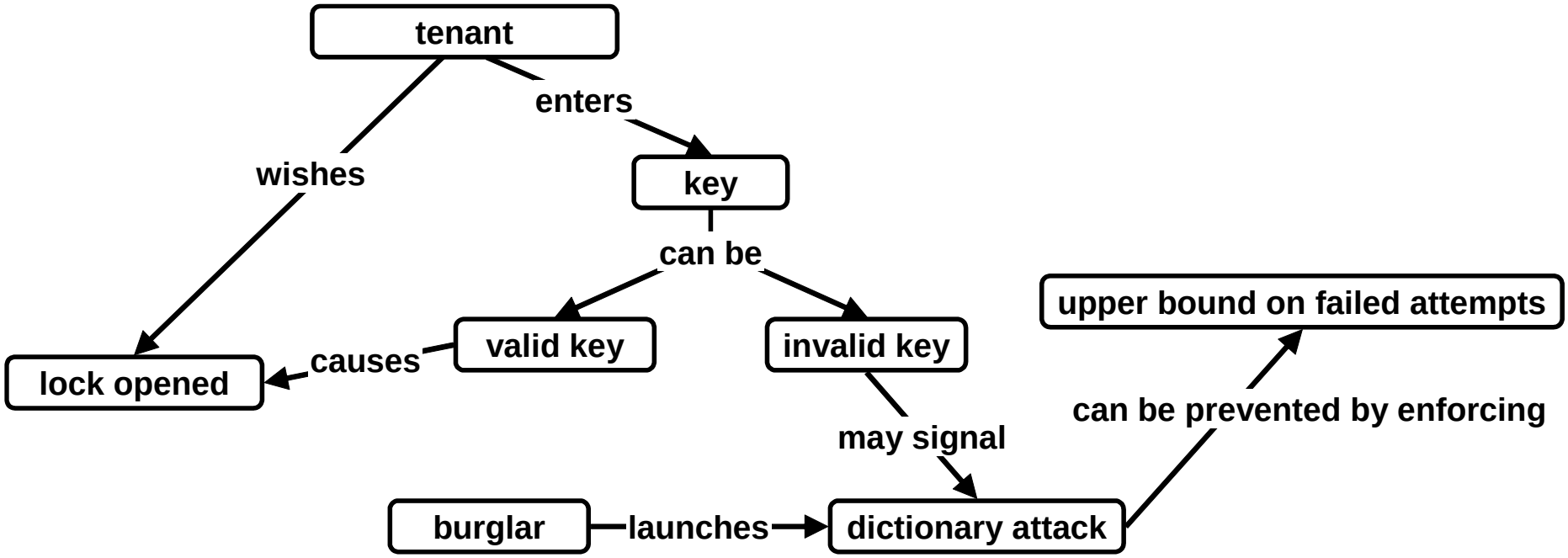


1. Lock case
2. Latch bolt
3. Dead bolt
4. Strike plate
5. Strike box
6. Protective plate
7. Thumb-turn
8. Lock cylinder
9. Left hand lever

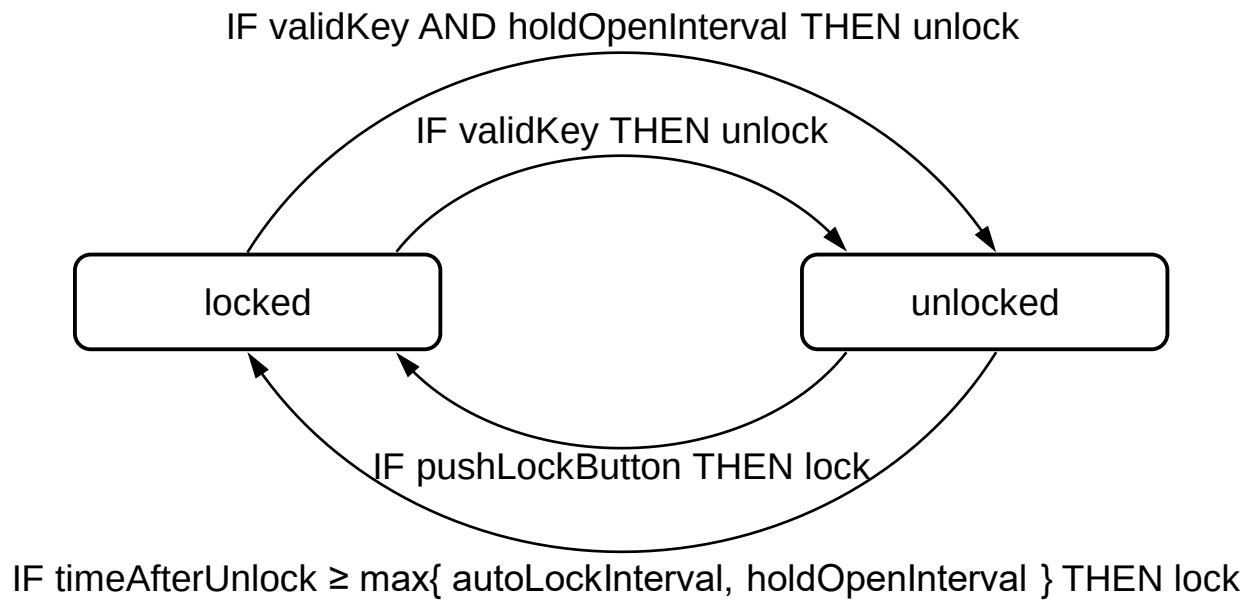




Concept Map for Home Access Control



States and Transition Rules



… what seemed a simple problem, now is becoming complex