


MCSE 652




Software Quality Assurance

Chapter 1: Introduction

Dr. Mehedi Hasan
Stamford University Bangladesh

Quote



“To the optimist, the glass is half full. To the pessimist, the glass is half empty. To the engineer, the glass is twice as big as it needs to be.”

Introduction



- ▶ The term ***software engineering*** is composed of two words, software and engineering.
- ▶ **Software** is more than just a program code. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.
- ▶ **Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.
- ▶ So, we can define ***software engineering*** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.
- ▶ The outcome of software engineering is an efficient and reliable software product.

Software Engineering



- ▶ IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

- ▶ Software engineer must understand the customer's business needs and design software to help meet them. A software engineer starts with *problem definition* and applies tools of the trade to obtain a *problem solution*.

Unlike any other engineering, software engineering seems to require:

- ▶ great emphasis on *methodology* or *method* for managing the development process
- ▶ great skill with tools and techniques.

Experts justify this with the atypical nature of the problems solved by software engineering.

Software Engineering

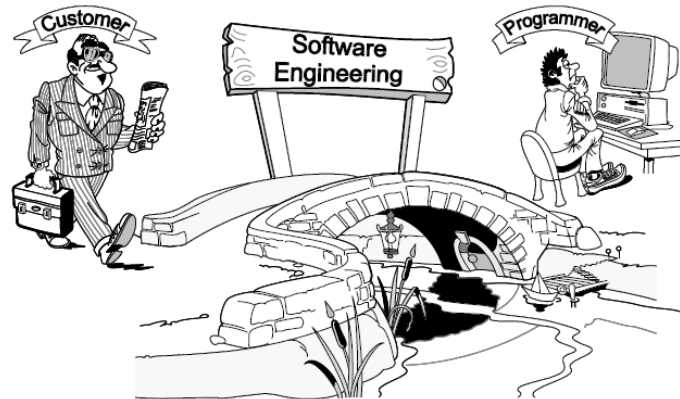


- ▶ The purpose of software engineering is to develop software-based systems that let customers achieve business goals.
 - ✓ The customer may be a hospital manager who needs patient-record software to be used by secretaries in doctors' offices;
 - ✓ a manufacturing manager who needs software to coordinate multiple parallel production activities that feed into a final assembly stage.
- ▶ **Software Engineer Requires:**
 - ▶ Quickly learn new and diverse disciplines and business processes
 - ▶ Extract an abstract model of the problem and formulate a solution
 - ▶ Design a software system and evolve with business needs.


Software Engineer vs Programmer

Software engineering is often confused with programming.

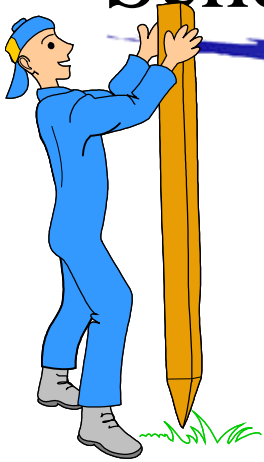
- Software engineering is the creative activity of understanding the business problem, coming up with an idea for solution, and designing the “blueprints” of the solution.
- Programming is the craft of implementing the given blueprints.
- Software engineer’s focus is on *understanding* the interaction between the system-to-be and its users and the environment and *designing* the software-to-be based on this understanding.
- Unlike this, programmer’s focus is on the program code and ensuring that the code faithfully implements the given design.



Introduction: Software is Complex

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

Complexity Example: Scheduling Fence Construction Tasks



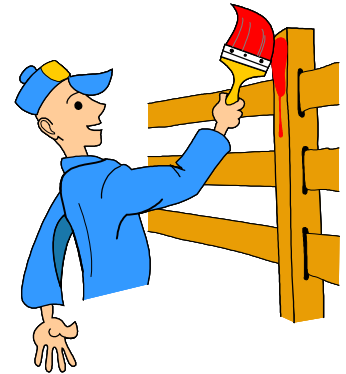
Setting posts
[3 time units]



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 = ?

Principles



- ▶ Large commercial programs is highly complex and difficulty levels of the programs increase exponentially with their sizes.
- ▶ Software engineering helps to reduce this programming complexity.
- ▶ Software engineering principles use two important techniques to reduce problem complexity:
 1. abstraction
 2. decomposition
- ▶ The principle of abstraction implies that a problem can be simplified by omitting irrelevant details.
- ▶ In decomposition, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one.

NEED OF SOFTWARE ENGINEERING



- ▶ **The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.**

1. Large software
2. Scalability
3. Cost
4. Dynamic Nature
5. Quality Management

CHARACTERISTICS OF GOOD SOFTWARE



A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- 1. Operational:** This tells us how well software works in operations.
- 2. Transitional:** This aspect is important when the software is moved from one platform to another
- 3. Maintenance:** This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment

CHARACTERISTICS OF GOOD SOFTWARE



☐ Operational

- ✓ Budget
- ✓ Usability
- ✓ Efficiency
- ✓ Correctness
- ✓ Functionality
- ✓ Dependability
- ✓ Security
- ✓ Safety

☐ Transitional

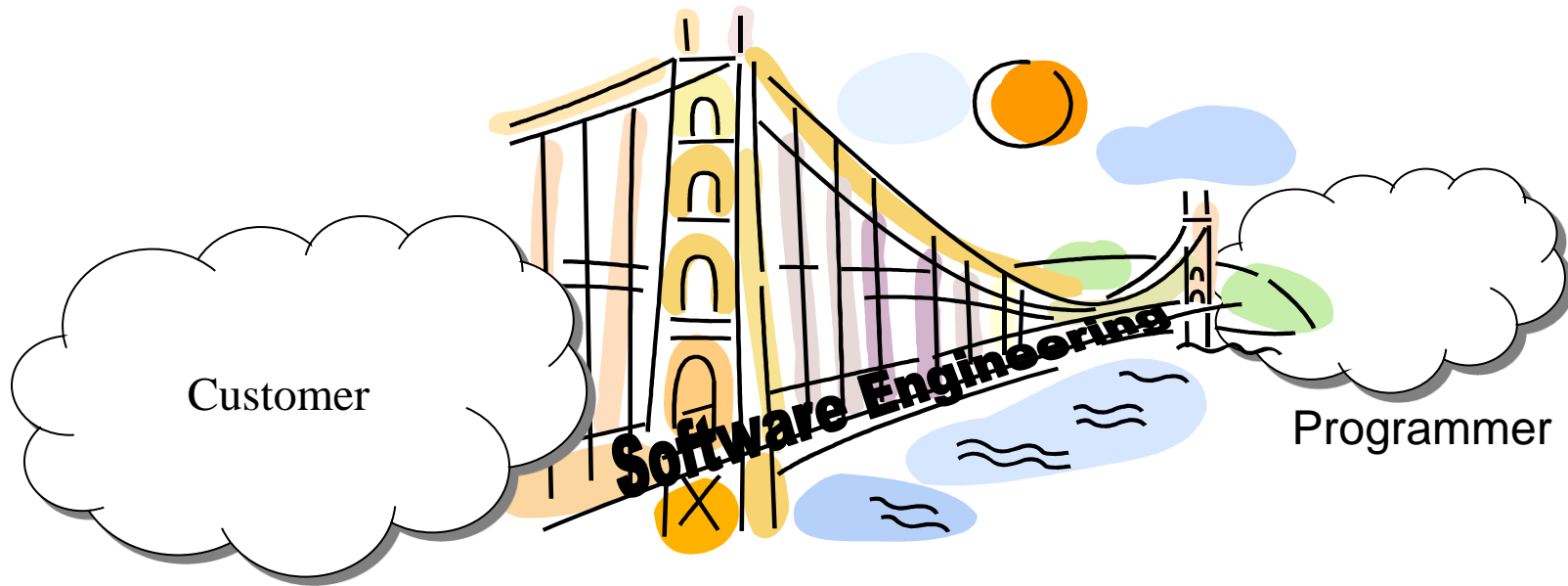
- ☐ Portability
- ☐ Interoperability
- ☐ Reusability
- ☐ Adaptability

☐ Maintenance

- ☐ Modularity
- ☐ Maintainability
- ☐ Flexibility
- ☐ Scalability

The Role of Software Engg. (1)

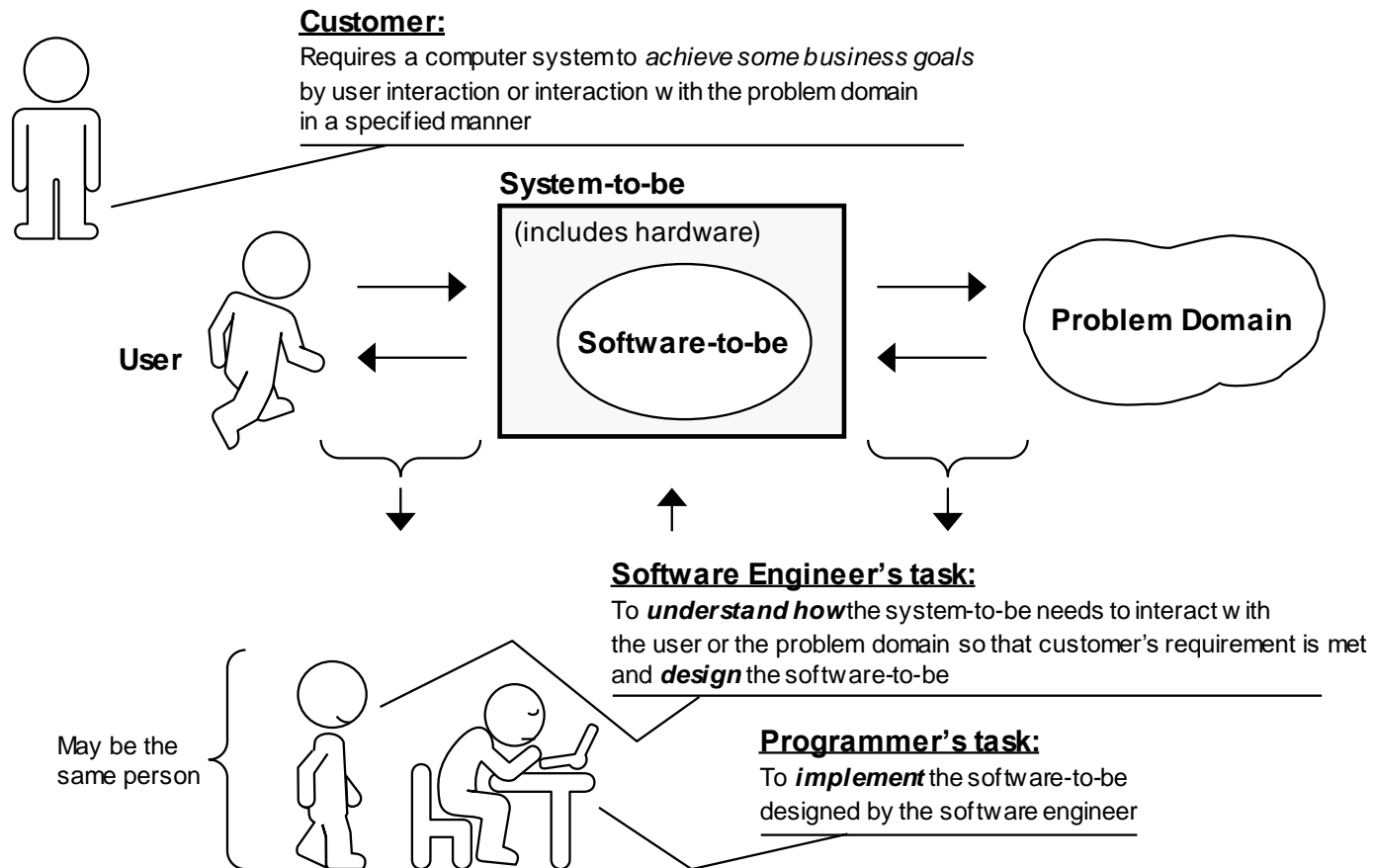
A bridge from customer needs to programming implementation



First law of software engineering

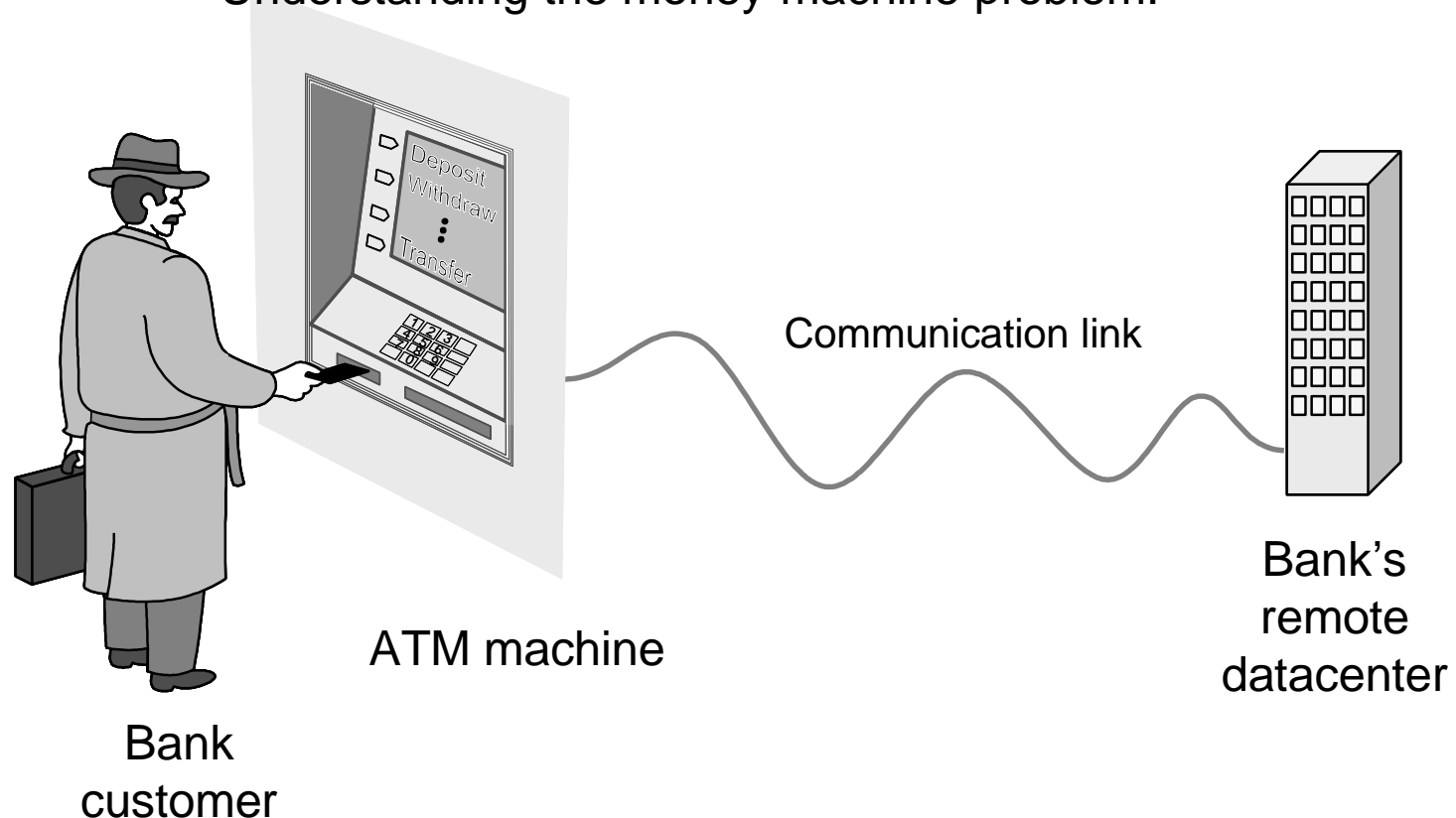
Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

The Role of Software Engg. (2)



Example: ATM Machine

Understanding the money-machine problem:



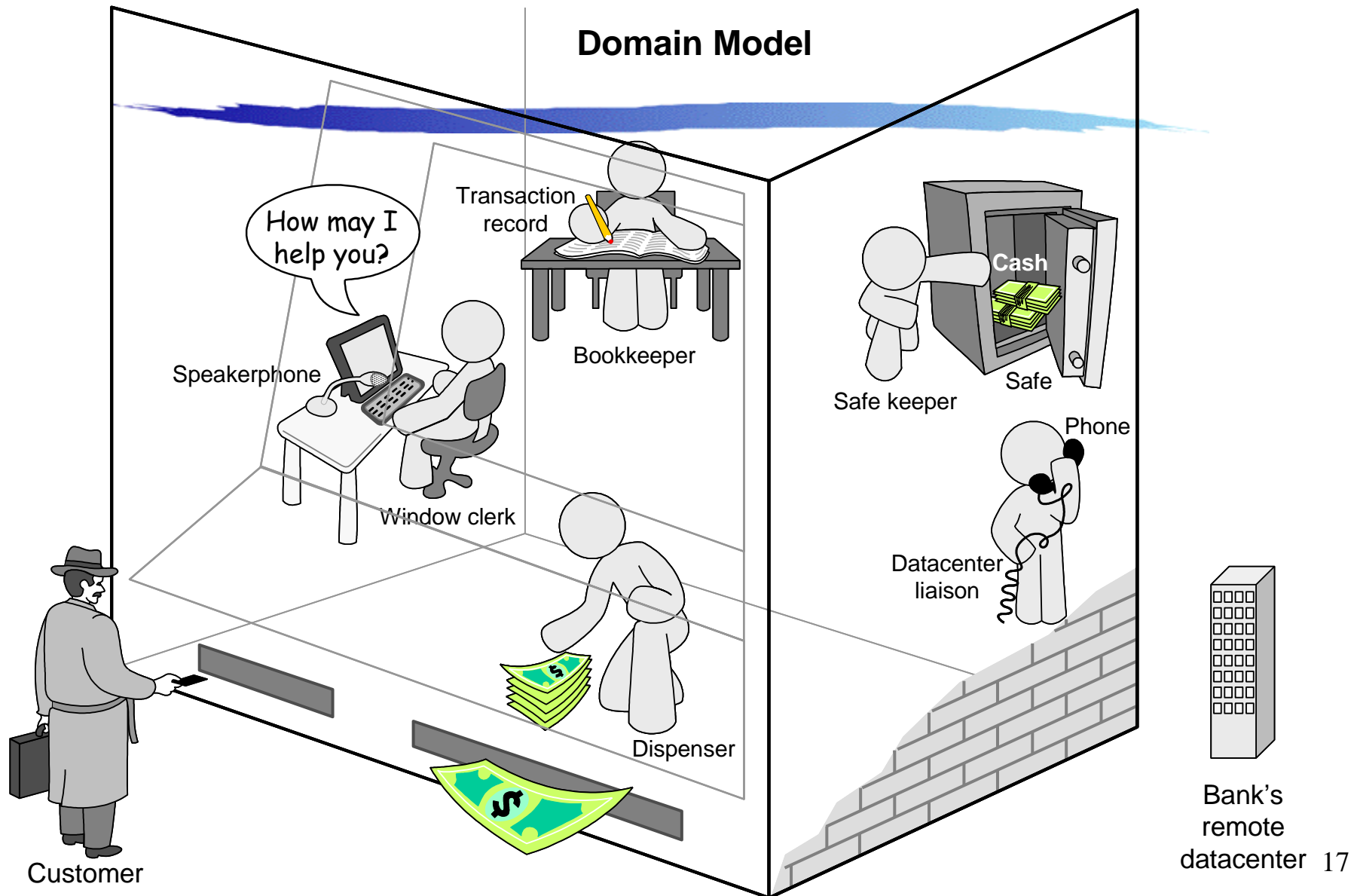
Problem-solving Strategy



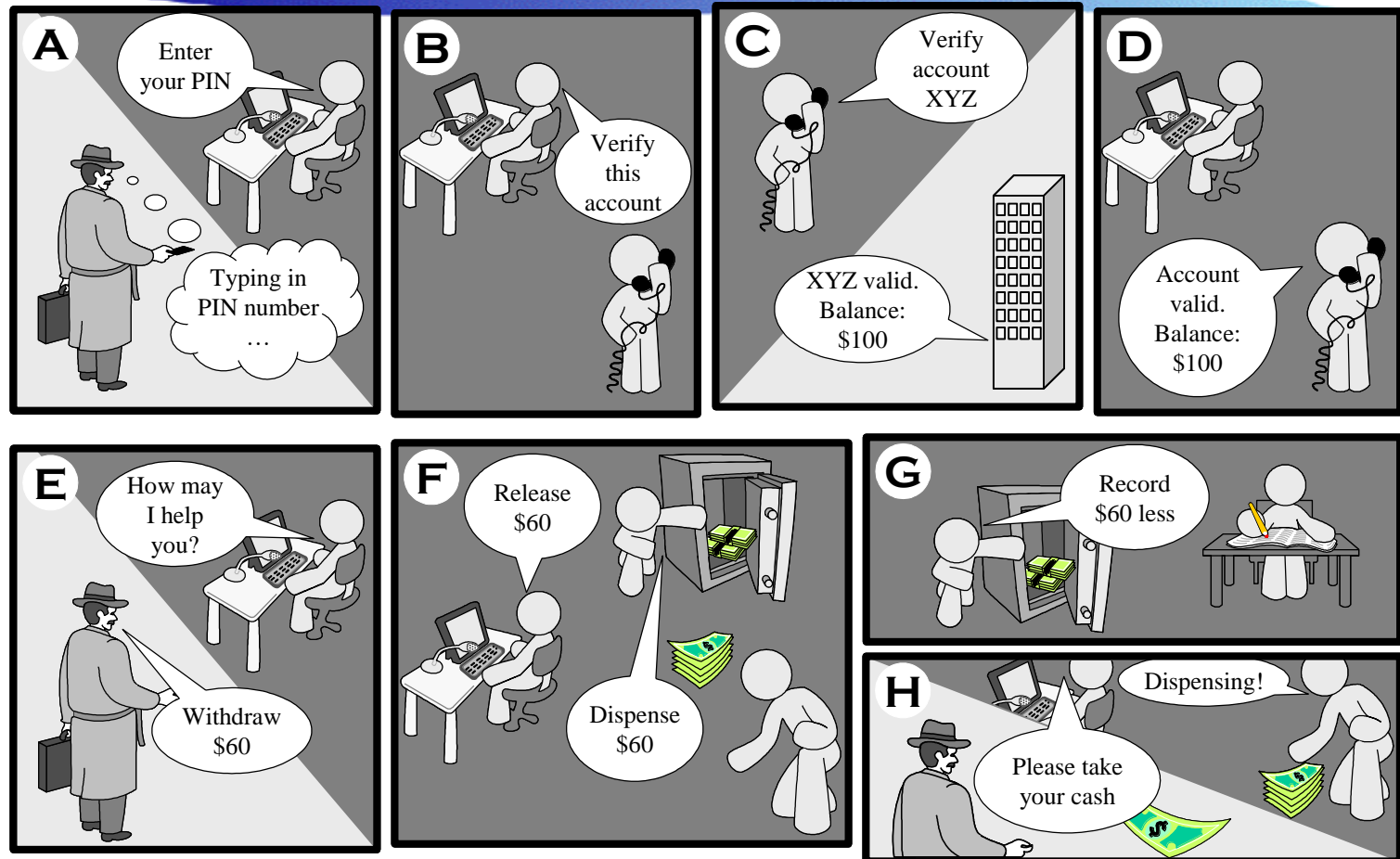
Divide-and-conquer:

- ▶ 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



SOFTWARE DEVELOPMENT LIFE CYCLE

LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

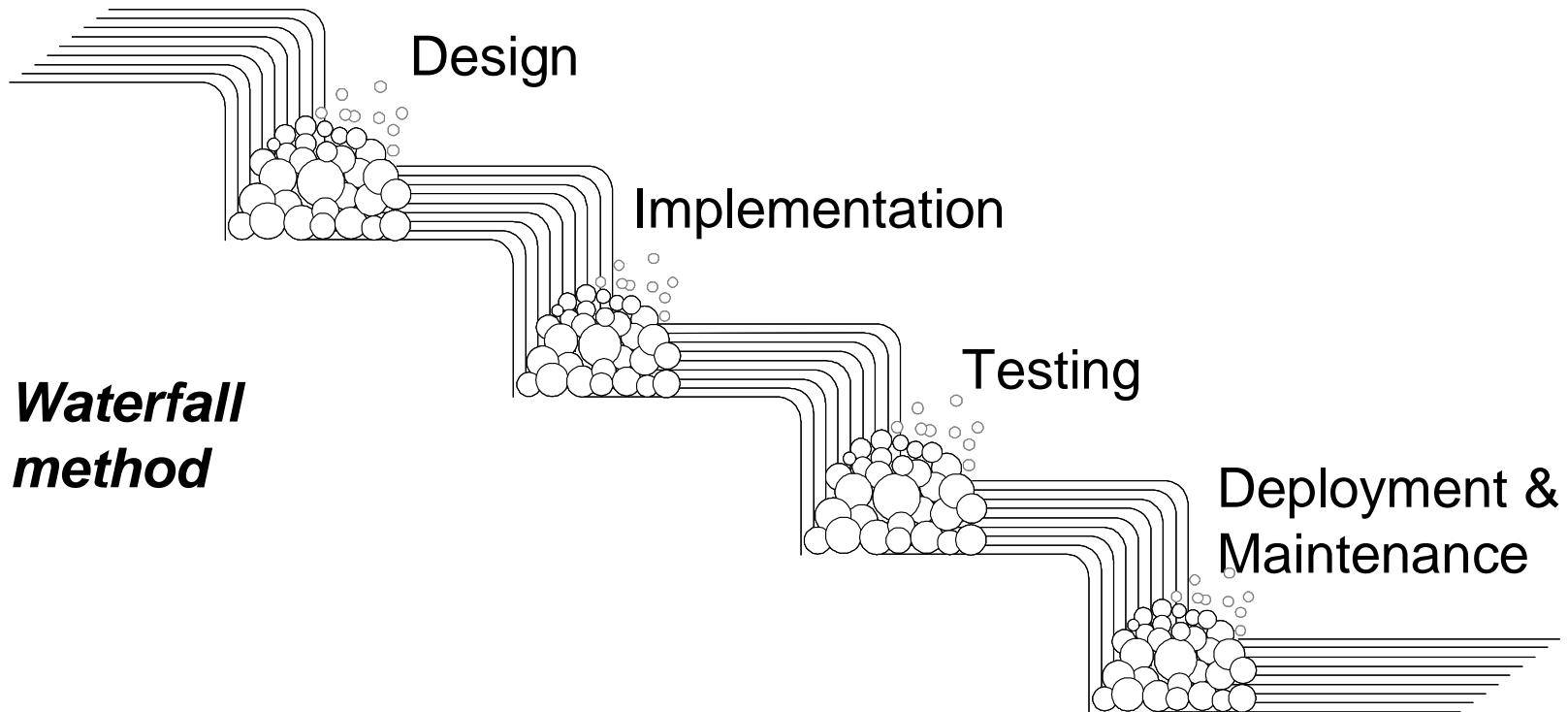
Different software life cycle models



- ▶ Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:
 1. Classical Waterfall Model
 2. Iterative Waterfall Model
 3. Prototyping Model
 4. Evolutionary Model
 5. Spiral Model

Waterfall Model

Requirements



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

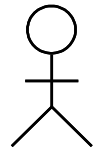
How Much Diagramming?



- ▶ 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

UML – Language of Symbols

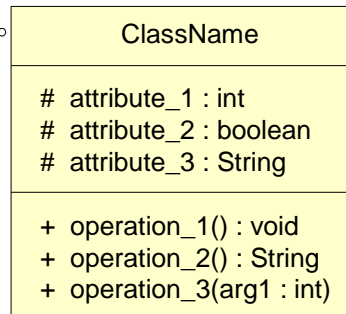
UML = Unified Modeling Language



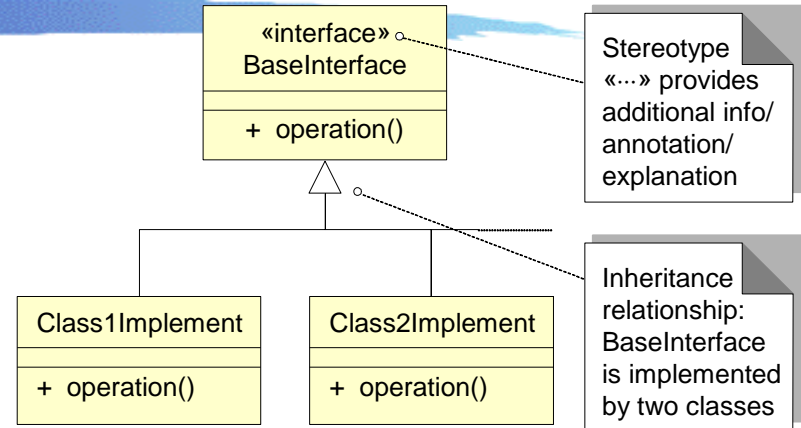
Actor

Three common compartments:
1. Classifier name
2. Attributes
3. Operations

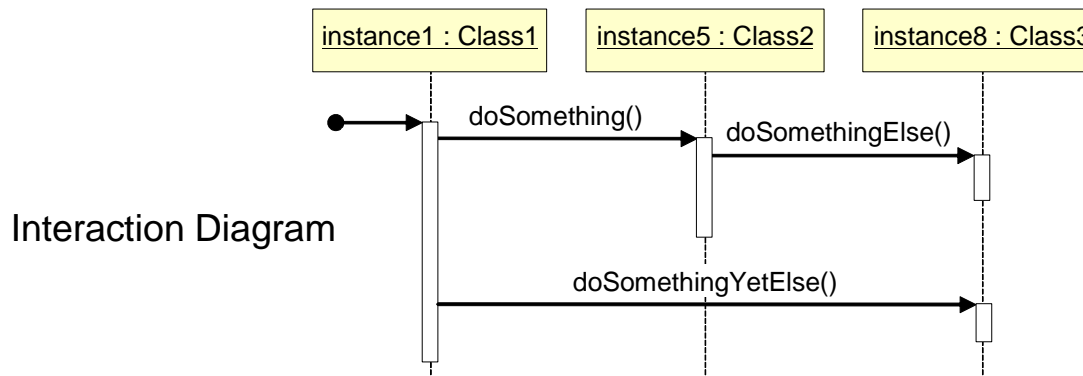
Comment



Software Class



Software Interface Implementation



Interaction Diagram

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

UML



Why UML

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs.

The primary goals in the design of the Object-Oriented Design in UML as follows:

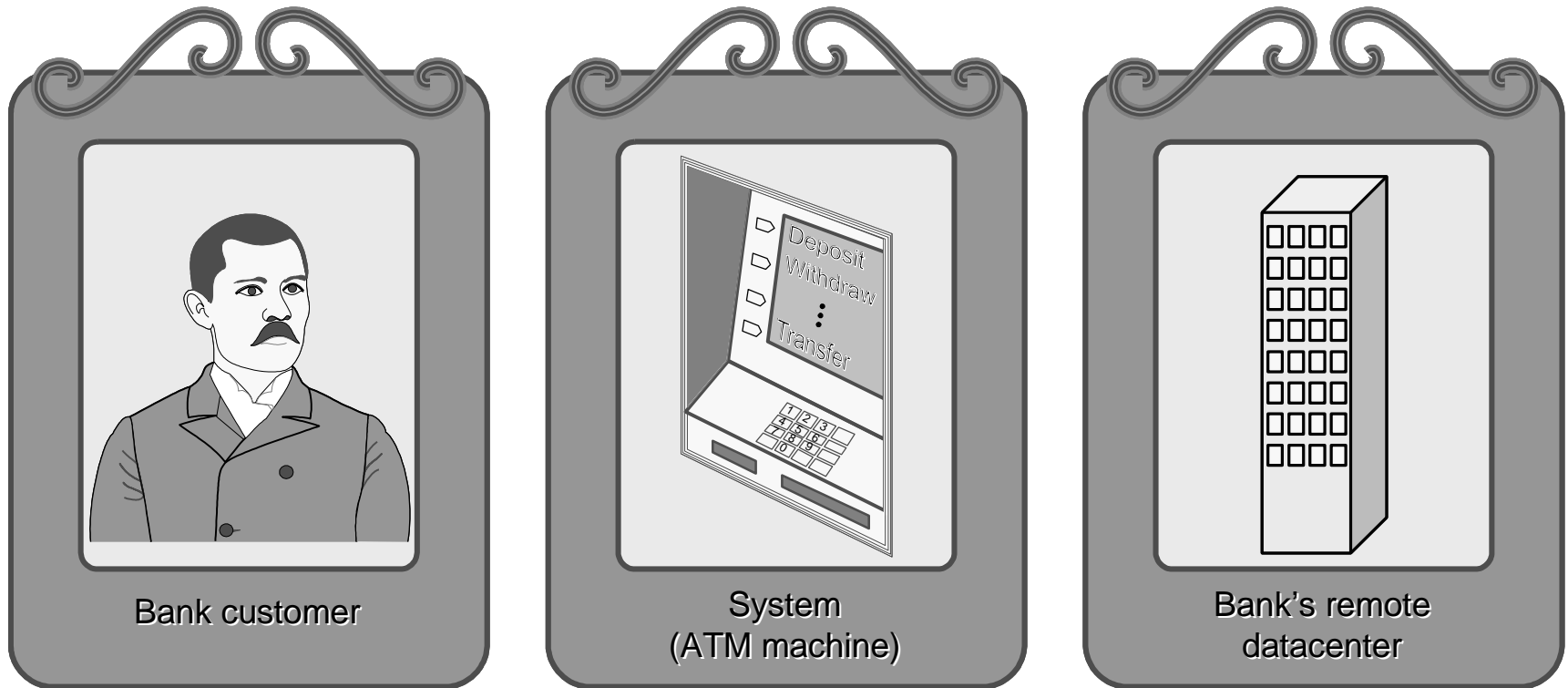
1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OOP tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Understanding the Problem Domain



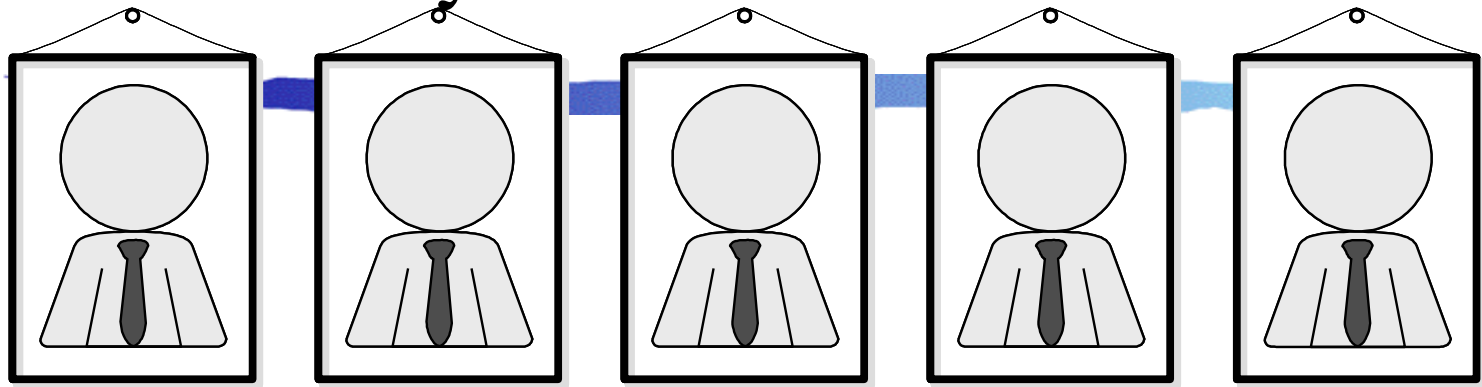
- ▶ System to be developed
- ▶ Actors
 - ▶ Agents external to the system that interact with it
- ▶ Concepts/ Objects
 - ▶ Agents working inside the system to make it function
- ▶ Use Cases
 - ▶ Scenarios for using the system

ATM: Gallery of Players



Actors (Easy to identify because they are visible!)

Gallery of Workers + Tools



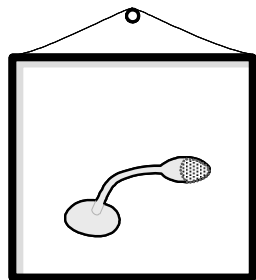
Window clerk

Datacenter
liaison

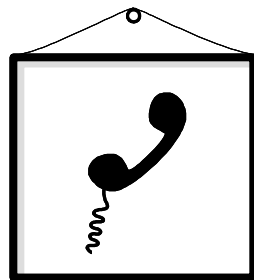
Bookkeeper

Safe keeper

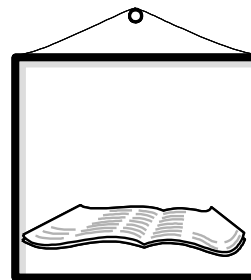
Dispenser



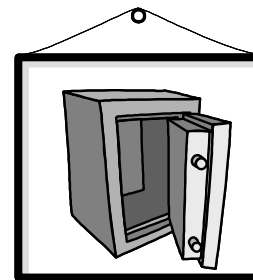
Speakerphone



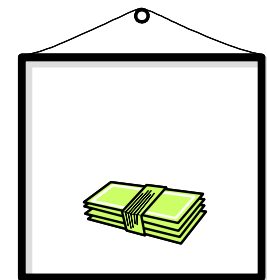
Telephone



Transaction
record



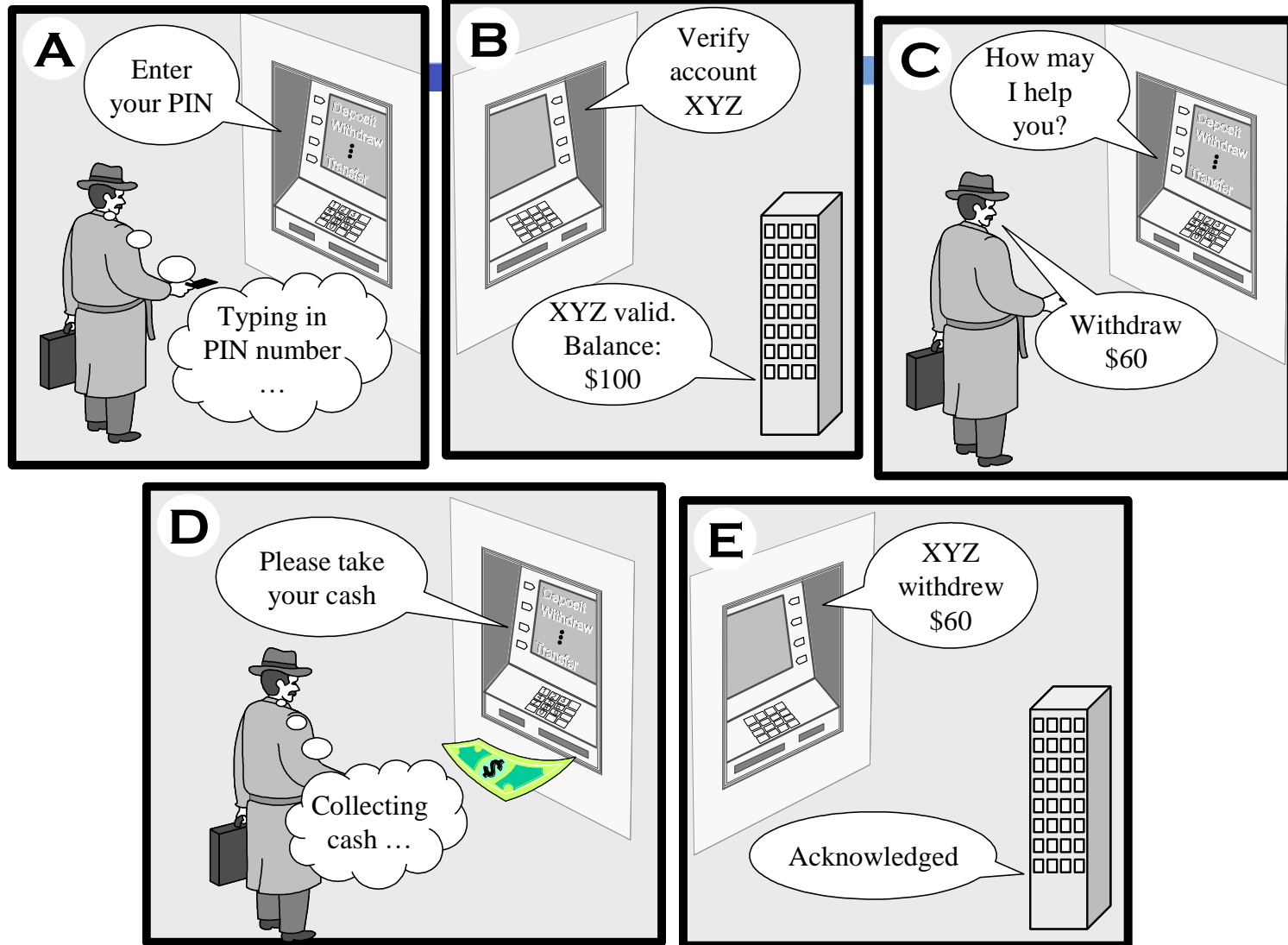
Safe



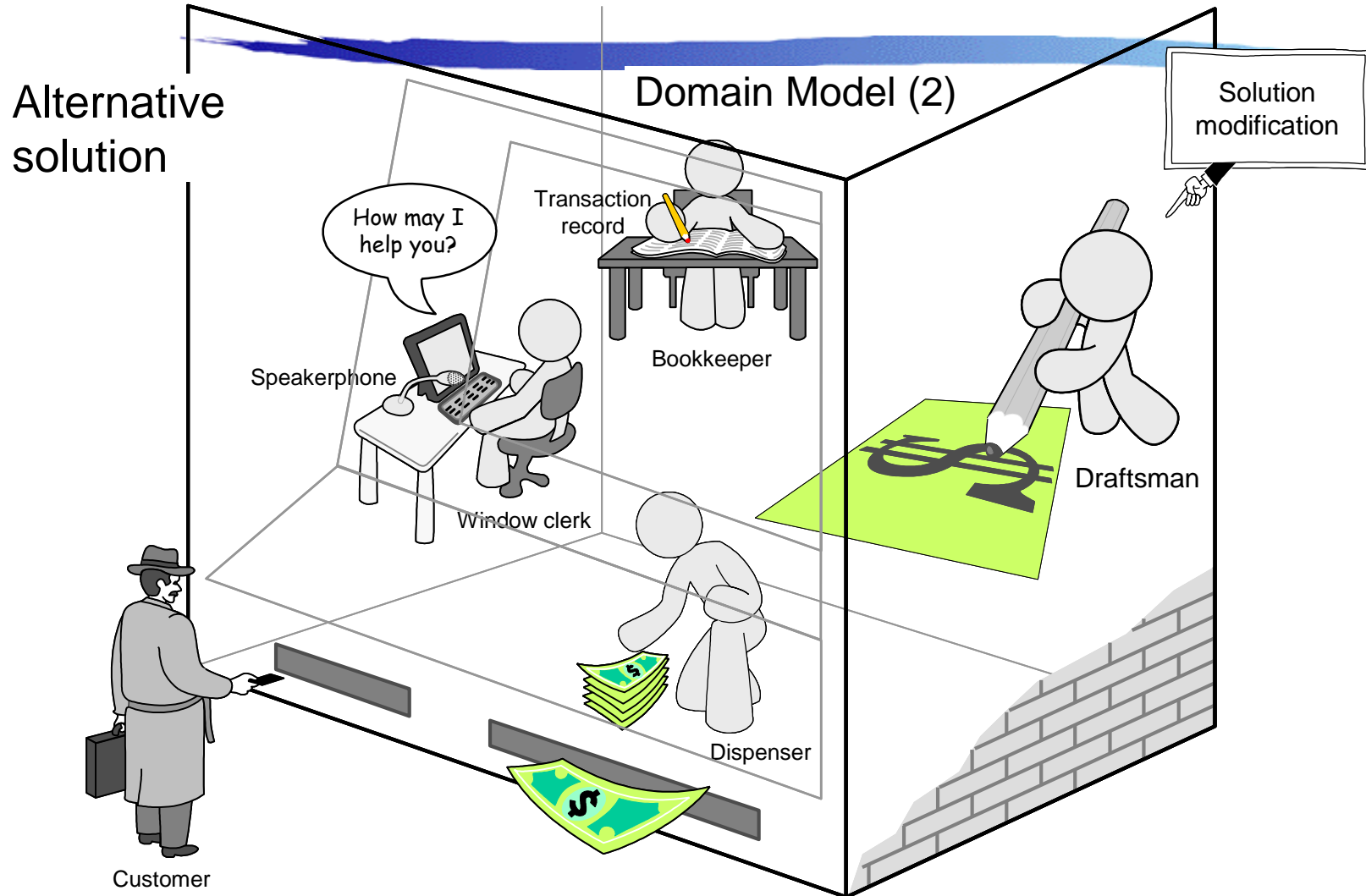
Cash

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

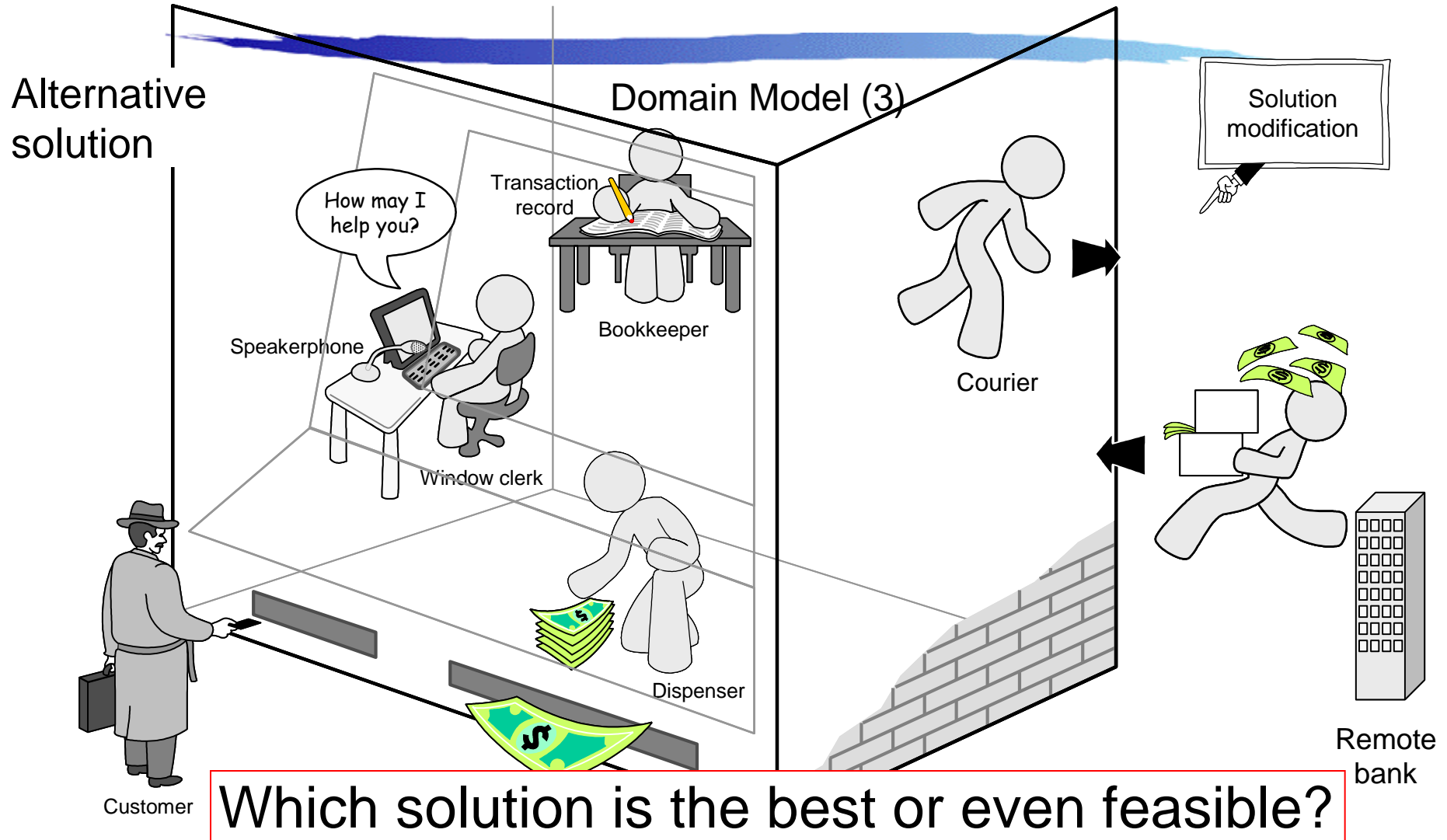
Use Case: Withdraw Cash



How ATM Machine Works (2)

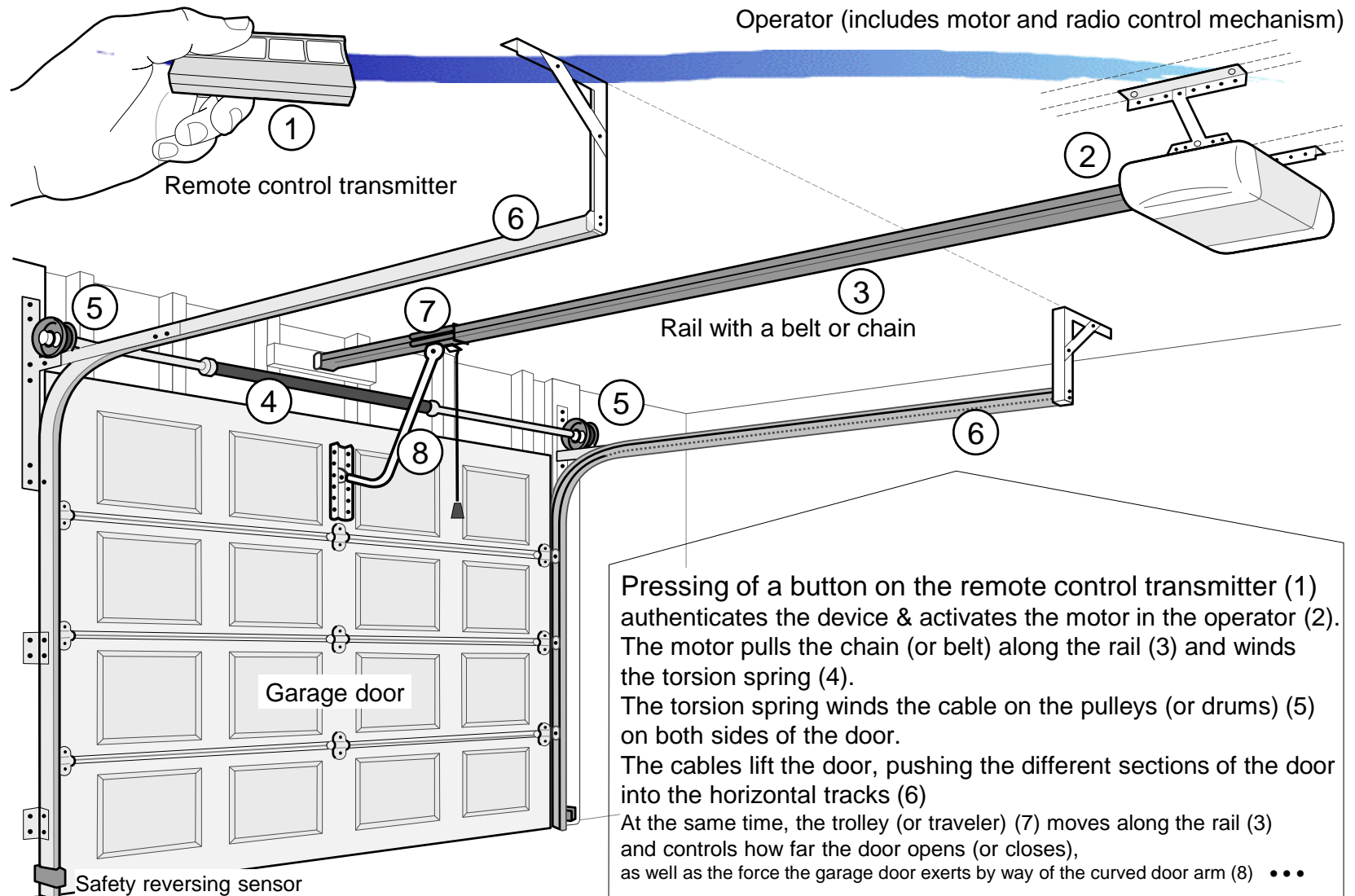


How ATM Machine Works (3)



Which solution is the best or even feasible?

Actual Design



Feasibility & Quality of Designs



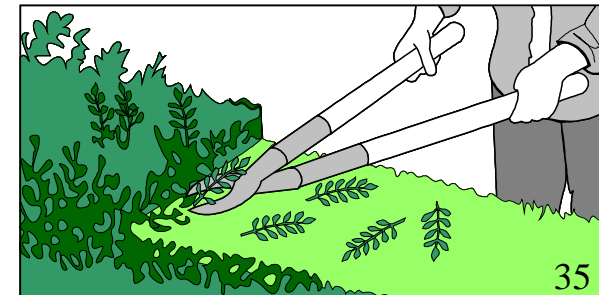
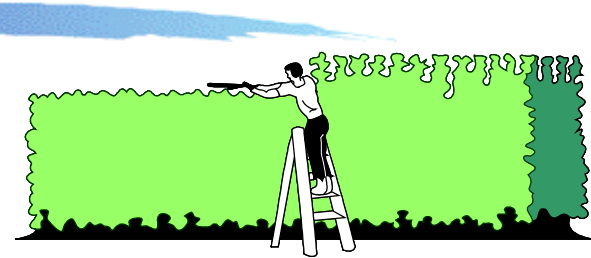
- ▶ Judging feasibility or quality of a design requires great deal of domain knowledge
(and commonsense knowledge!)

Software Measurement

□ What to measure?

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

Formal hedge pruning



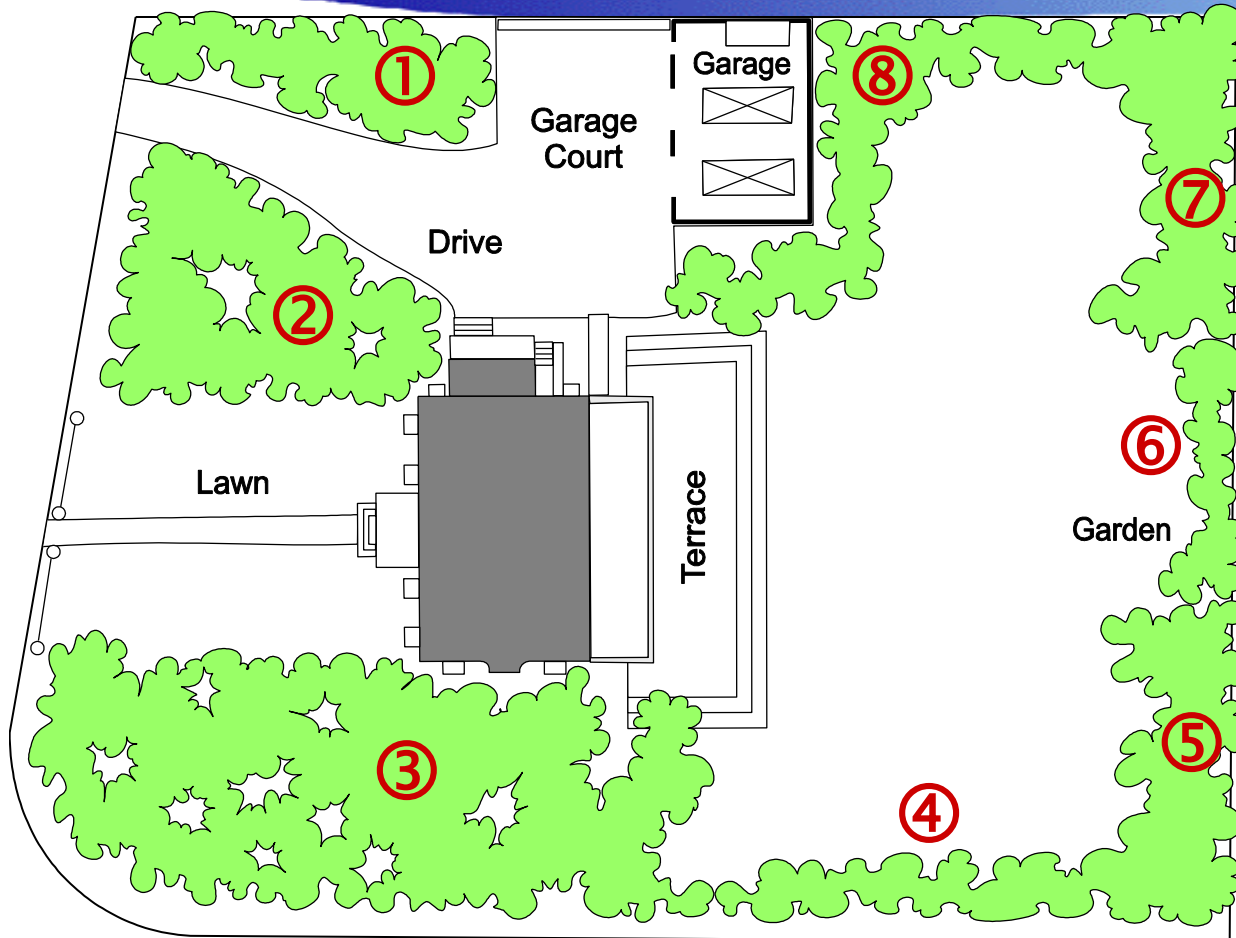
Work Estimation Strategy



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

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

- ▶ 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:

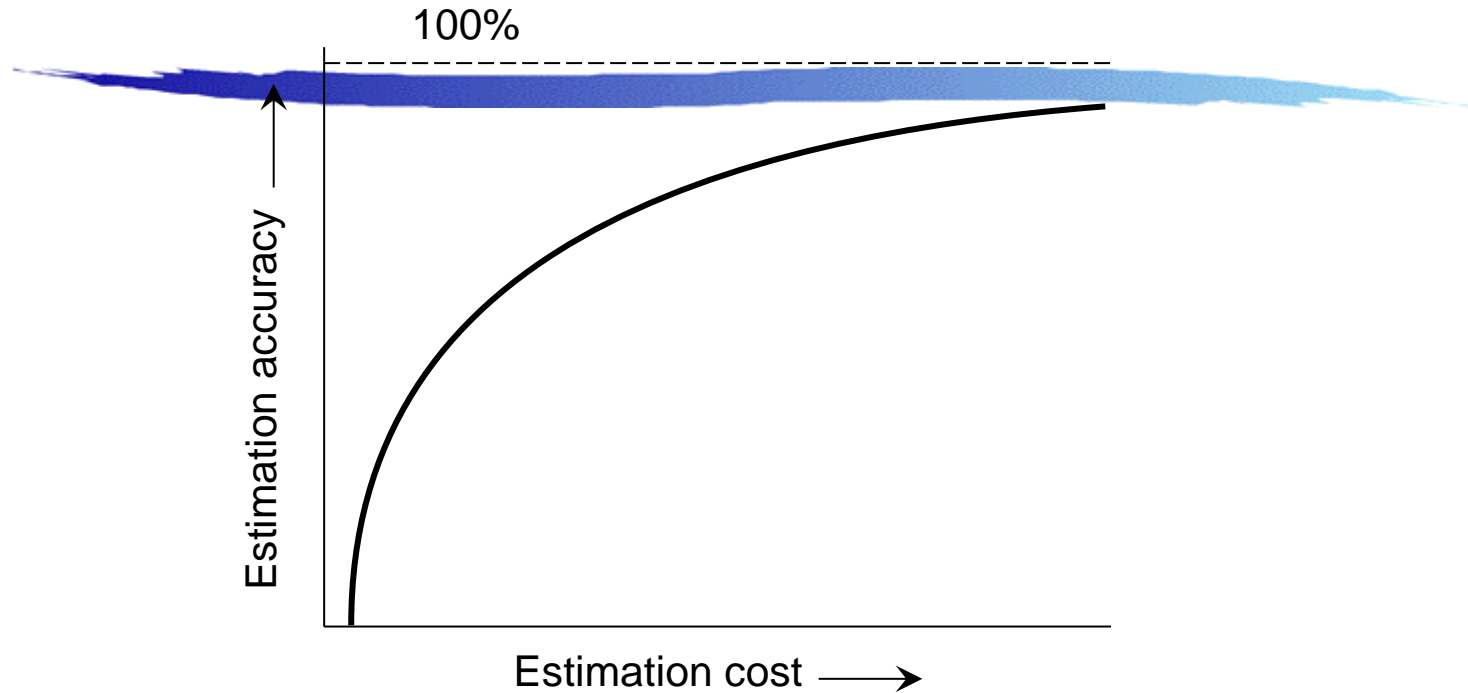
- ▶ Velocity estimate may need to be adjusted (based on observed progress)
- ▶ However, the total duration can be recomputed quickly

🔧 Provided that the relative size estimates of parts are accurate
—accuracy easier achieved if the parts are small and similar-size

Unfortunately:

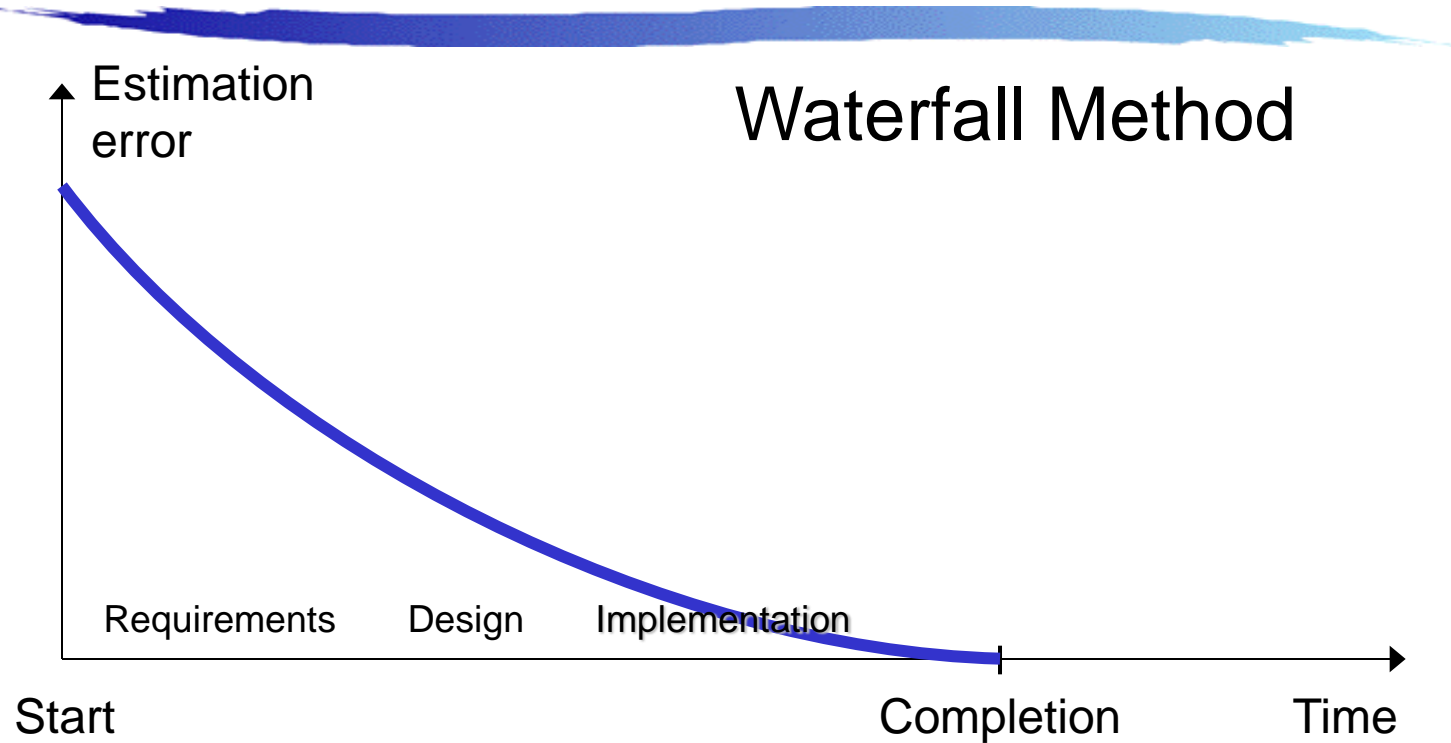
- ▶ Unlike hedges, software is mostly **invisible** and **does not exist** when project is started
 - ➔ The initial estimate hugely depends on experience and imagination

Exponential Cost of Estimation



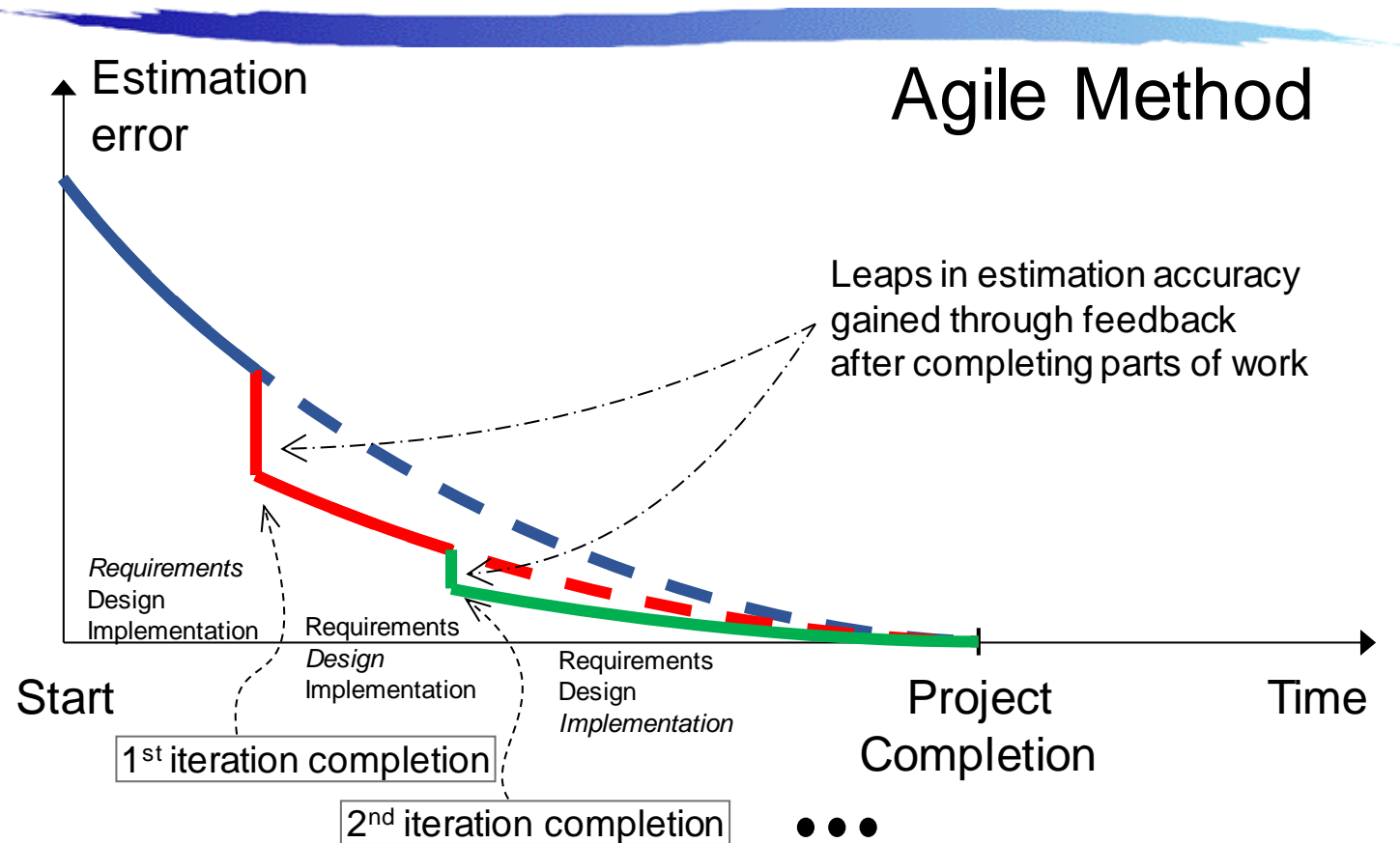
- ❑ Improving accuracy of estimation beyond a certain point requires huge cost and effort (known as the law of diminishing returns)
- ❑ In the beginning of the curve, a modest effort investment yields huge gains in accuracy

Estimation Error Over Time



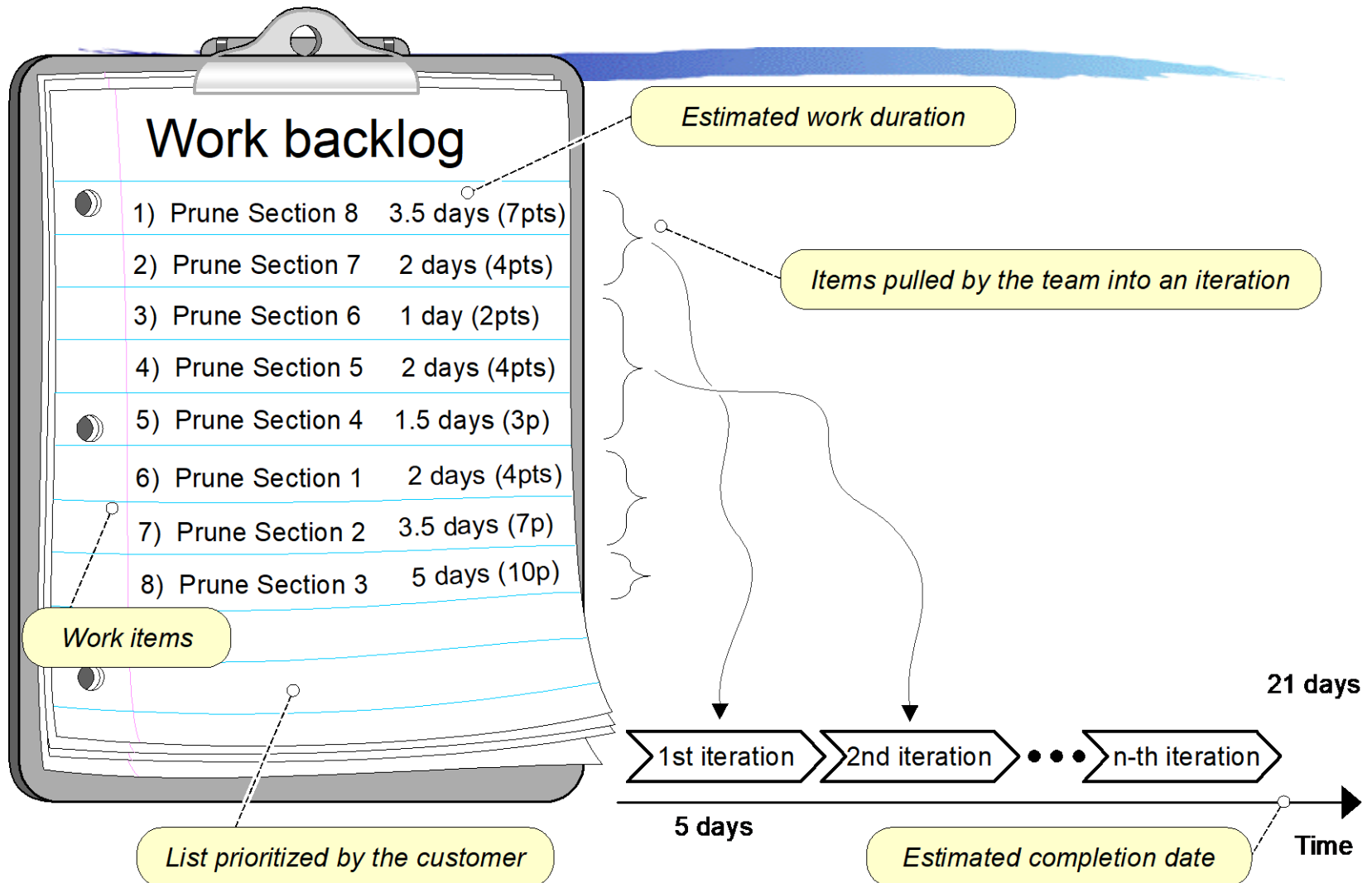
Waterfall method *cone of uncertainty* starts high and *gradually* converges to zero as the project approaches completion.

Estimation Error Over Time

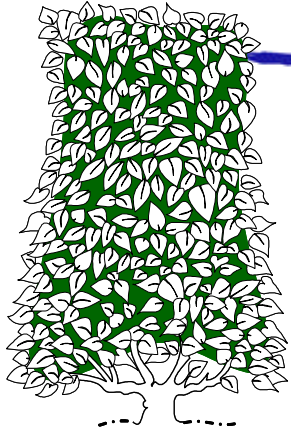


Agile method *cone of uncertainty* starts high and *in leaps* converges to zero as the project approaches completion.

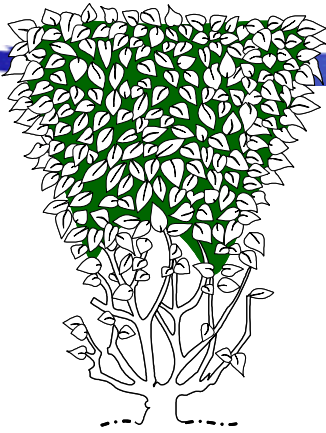
Agile Project Effort Estimation



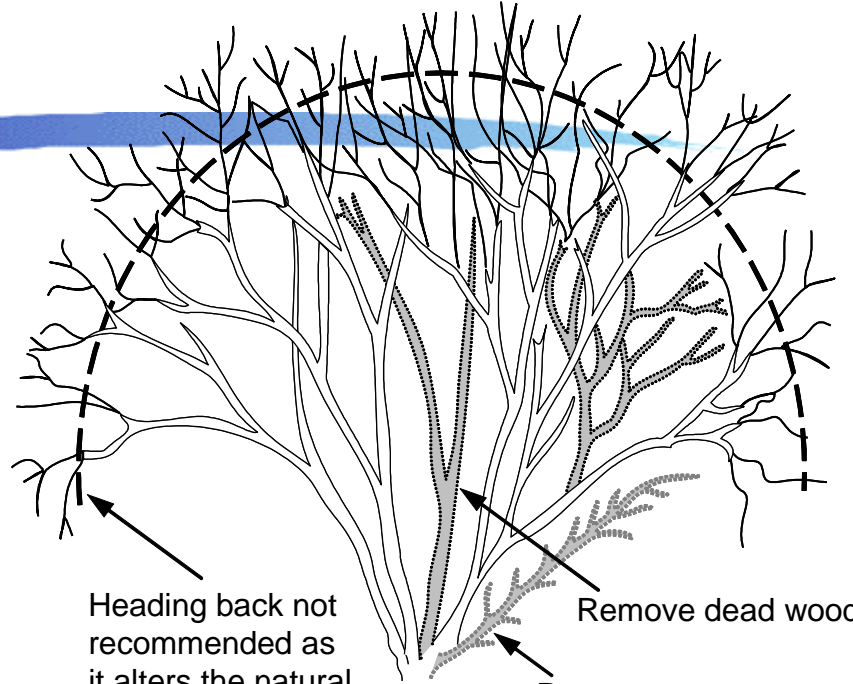
Measuring Quality of Work



Good Shape
(Low branches get sun)



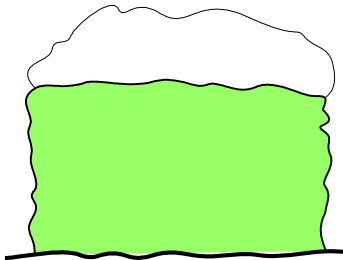
Poor Shape
(Low branches shaded from sun)



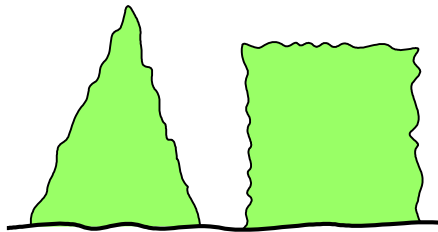
Heading back not recommended as it alters the natural shape of the shrub

Remove dead wood

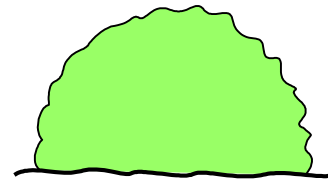
Remove water spouts and suckers



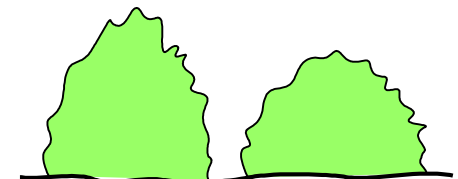
Snow accumulates on broad flat tops



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

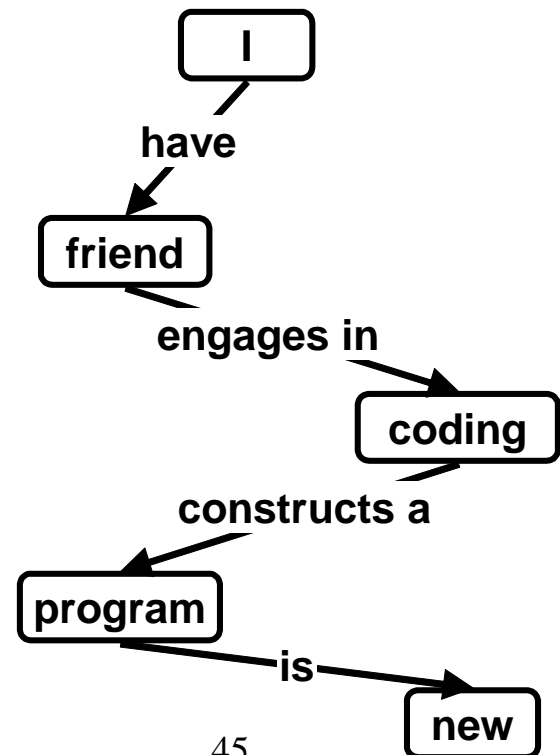
Useful tool for problem domain description

SENTENCE: “My friend is coding a new program”

translated into propositions



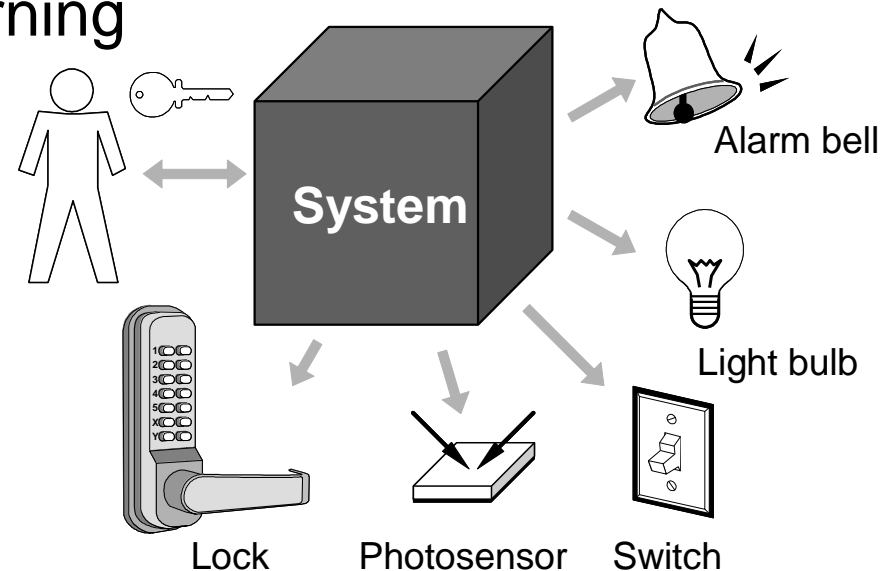
Proposition	Concept	Relation	Concept
1.	I	have	friend
2.	friend	engages in	coding
3.	coding	constructs a	program
4.	program	is	new



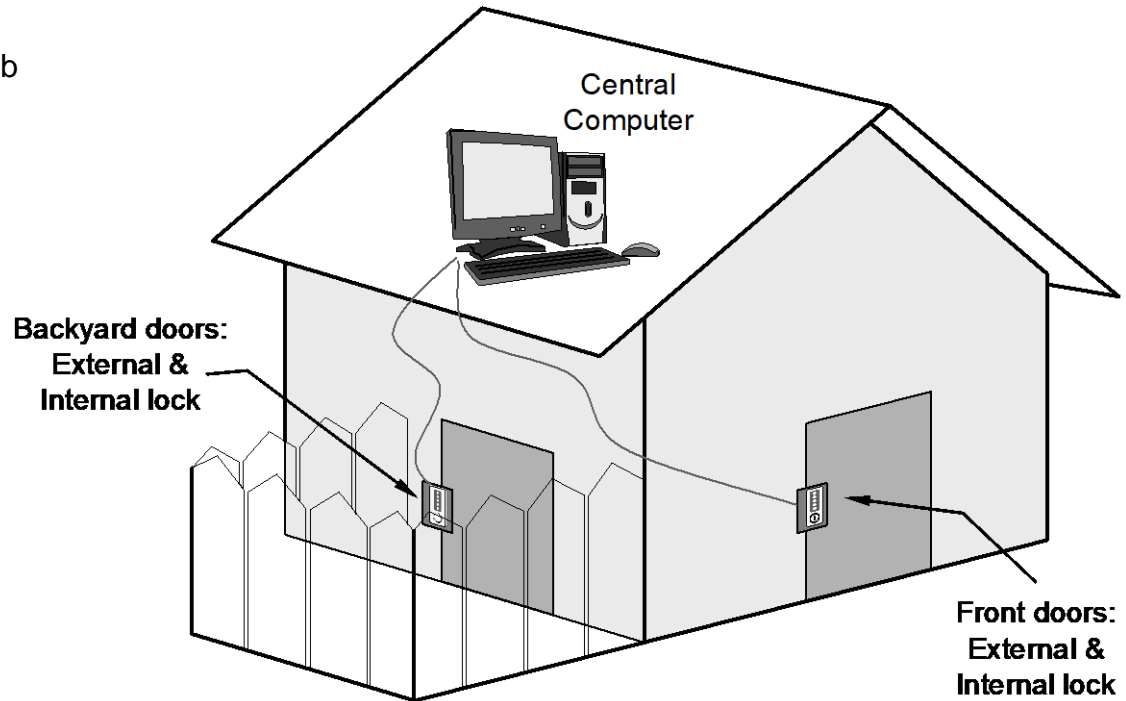
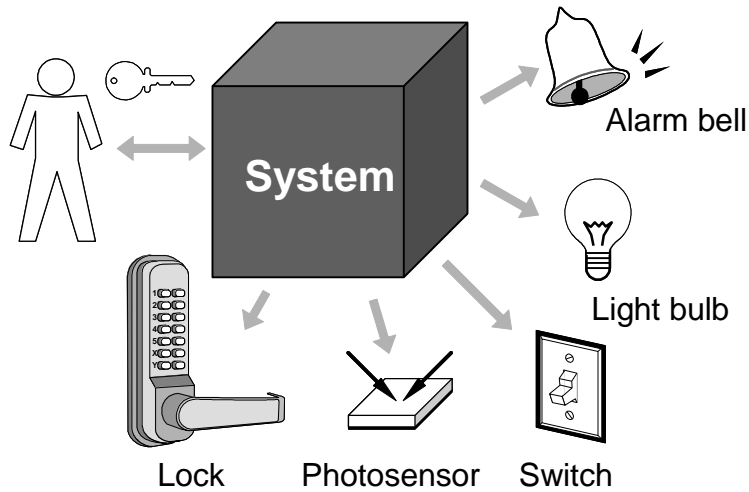
Search the Web for Concept Maps

Case Study: Home Access Control

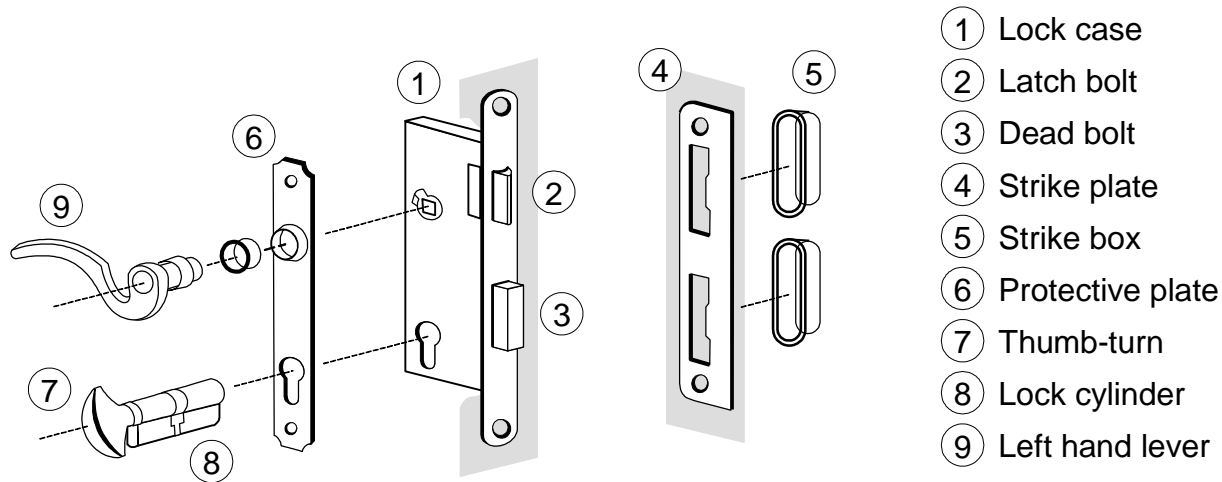
- ▶ Objective: Design an electronic system for:
 - ▶ Home access control
 - ✂ Locks and lighting operation
 - ▶ Intrusion detection and warning



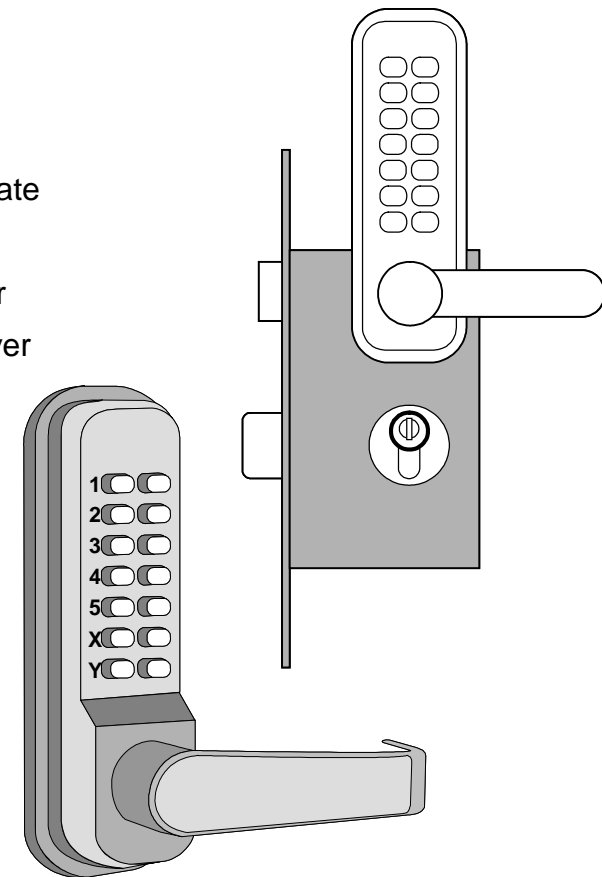
Case Study – More Details



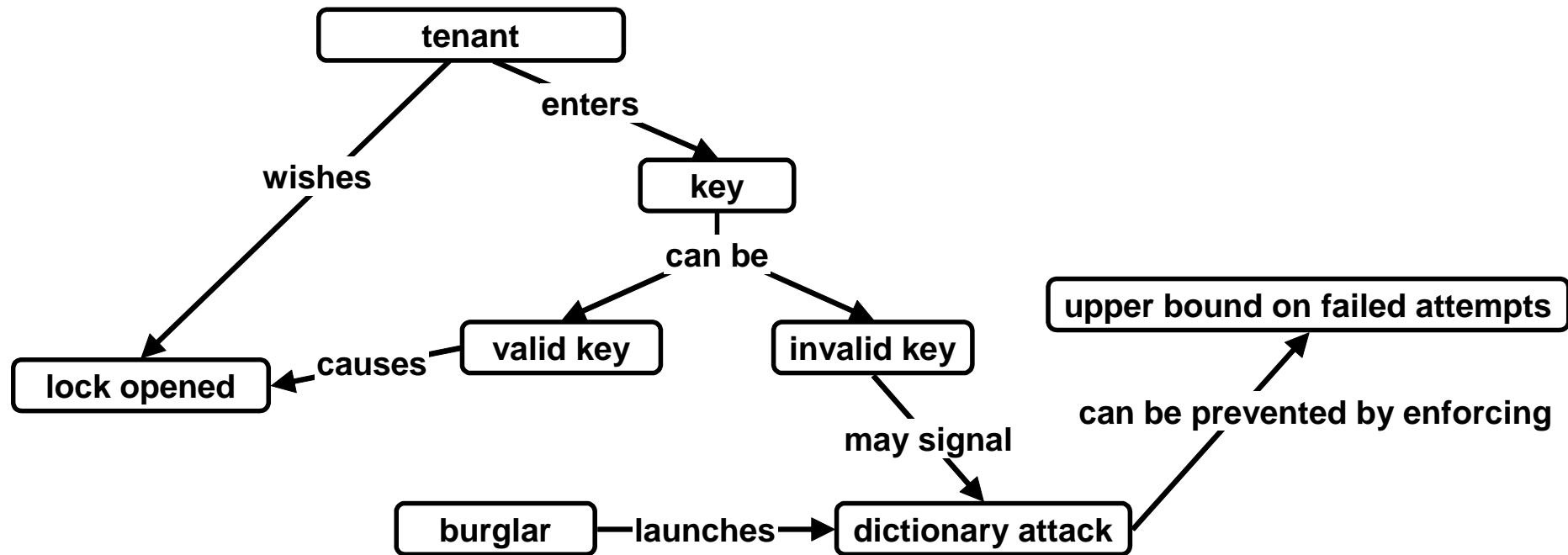
Know Your Problem



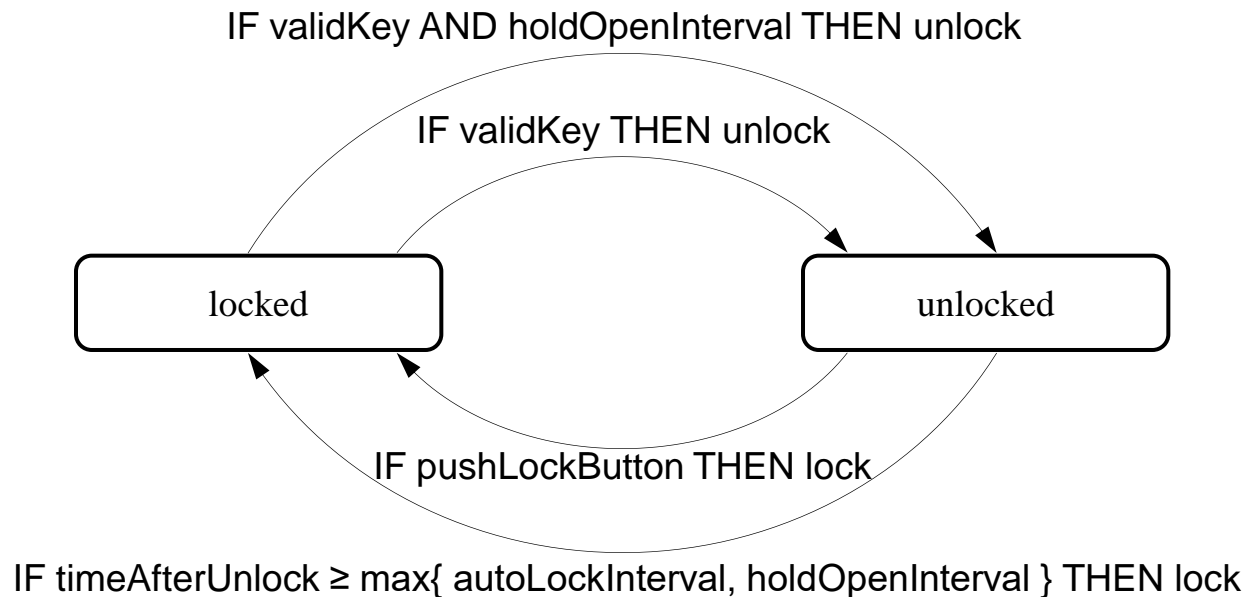
Mortise Lock Parts



Concept Map for Home Access Control



States and Transition Rules



... what seemed a simple problem, now is becoming complex