INTRODUCTION TO SOFTWARE ENGINEERING

ECC 822 – SOFTWARE ENGINEERING

Tuesday, February 18, 2020

WHAT IS SOFTWARE ENGINEERING?

- Software engineering is a branch of computer science which includes the development and building of computer systems software and applications software.

INDUSTRIAL STRENGTH SOFTWARE

- An industrial strength software system is built to solve some problem of a client and is used by the clients organization for operating some part of business (we use the term "business" in a very broad sense—it may be to manage inventories, finances, monitor patients, air traffic control, etc.).
- Important activities depend on the correct functioning of the system.
- A malfunction of such a system can have huge impact in terms of financial or business loss, inconvenience to users, or loss of property and life.
- Industrial software system needs to be of high quality with respect to properties like dependability, reliability, user-friendliness, etc.

REQUIREMENTS FOR INDUSTRIAL STRENGTH SOFTWARE

- **Requires that the software be thoroughly tested before being used.** The need for rigorous testing increases the cost considerably. In an industrial strength software project, 30% to 50% of the total effort may be spent in testing (while in a student software even 5% may be too high!).
- **Requirement that the development be broken into phases** such that output of each phase is evaluated and reviewed so bugs can be removed. This requires more documentation, standards, processes, etc. All these increase the effort required to build the software— hence the productivity of producing industrial strength software is generally much lower than for producing student software.
- **Requirements of backup and recovery, fault tolerance, following of standards, portability,** etc. These generally have the effect of making the software system more complex and larger. The size of the industrial strength software system may be two times or more than the student system for the same problem.

TYPES OF SOFTWARE PRODUCTS

There are two fundamental types of software products, i.e

1. **Generic Products:** Stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples include software for PCs such as databases, word processors, drawing packages and project management tools.
2. **Customized (or bespoke) Products:** Systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process and air traffic control systems.

FUNDAMENTAL ACTIVITIES OF SOFTWARE DEVELOPMENT

There are four fundamental activities that are common to all software processes. These activities are:

1. **Software Specification:** Customers and engineers define the software that is to be produced and the constraints on its operation.
2. **Software Development:** The software is designed and programmed.
3. **Software Validation:** the software is checked to ensure that it is what the customer requires.
4. **Software Evaluation:** The software is modified to reflect changing customer and market requirements.

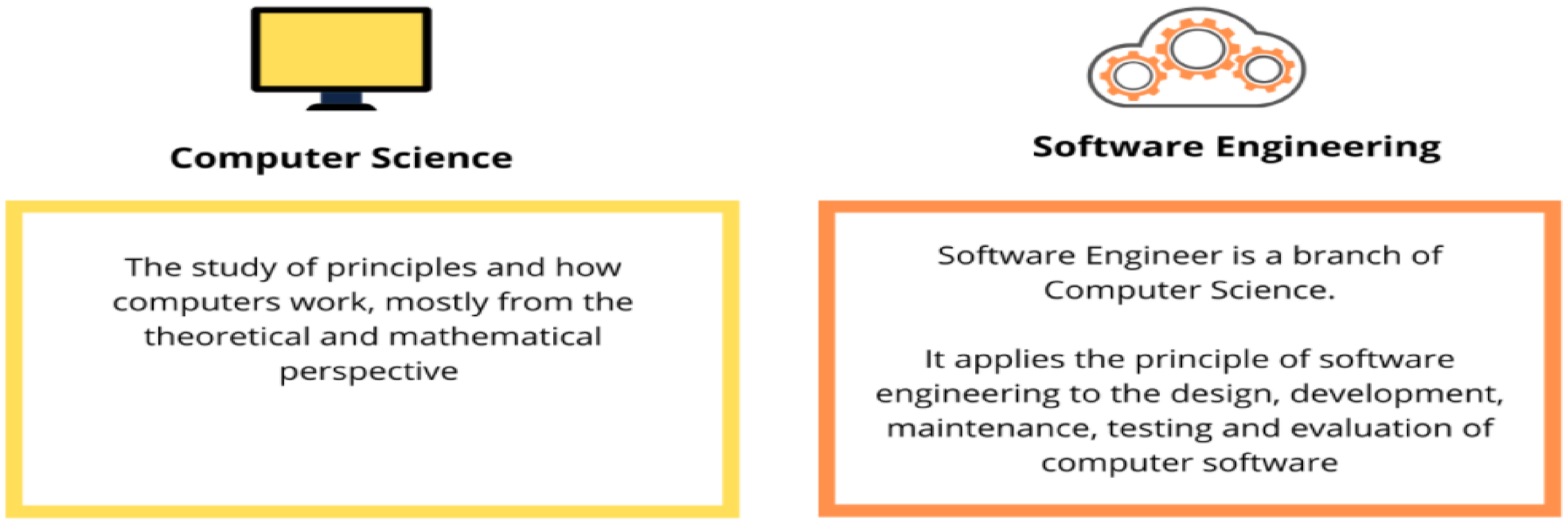
Specification
Development
Validation
Evaluation

Four Activities of Software Process Framework

SOFTWARE ENGINEERING Vs COMPUTER SCIENCE

Software engineering is related to both computer science.

Computer Science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software.



Computer Science

The study of principles and how computers work, mostly from the theoretical and mathematical perspective

Software Engineering

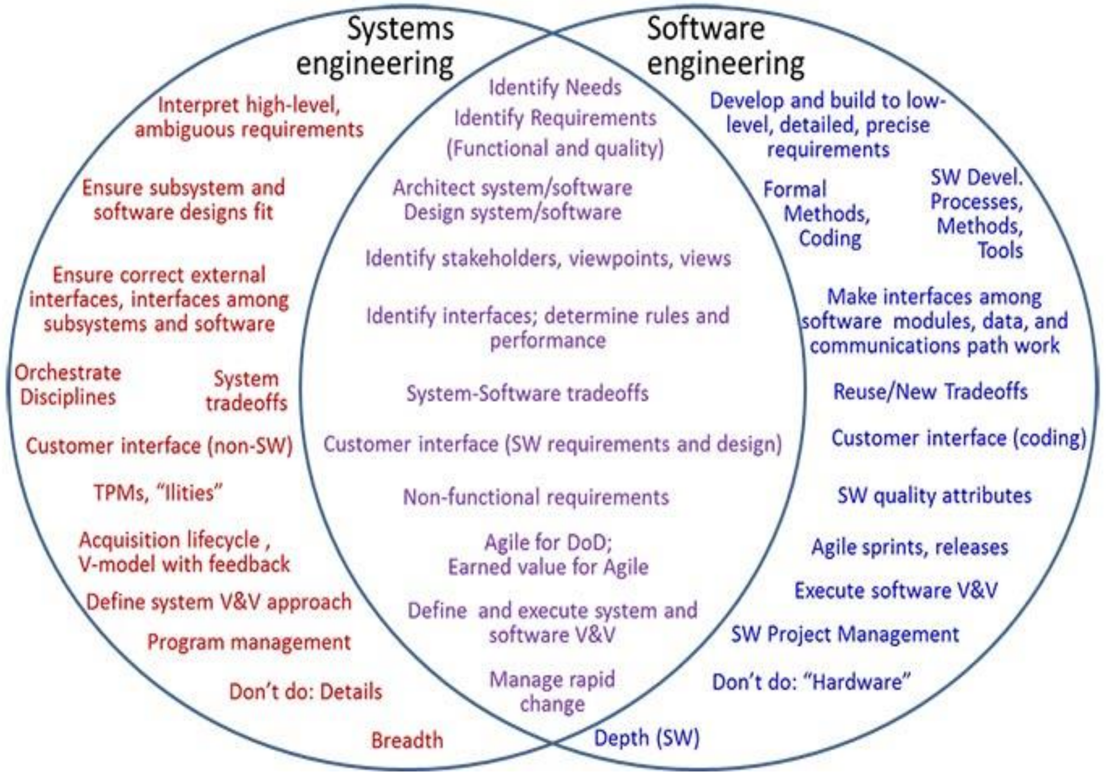
Software Engineer is a branch of Computer Science.

It applies the principle of software engineering to the design, development, maintenance, testing and evaluation of computer software

SOFTWARE ENGINEERING Vs SYSTEMS ENGINEERING

Software engineering is related to systems engineering.

- **System Engineering** is concerned with all aspects of the development and evolution of complex systems where software plays a major role.
- Focus is on hardware development, policy and process design and system deployment, as well as software engineering.
- System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system.
- They are less concerned with the engineering of the system components (hardware, software, etc.)


Systems engineering
Software engineering
Interpret high-level, ambiguous requirements
Ensure subsystem and software designs fit
Ensure correct external interfaces, interfaces among subsystems and software
Orchestrate Disciplines
System tradeoffs
Customer interface (non-SW)
TPMs, "Ilities"
Acquisition lifecycle , V-model with feedback
Define system V&V approach
Program management
Don't do: Details
Breadth
Identify Needs
Identify Requirements (Functional and quality)
Architect system/software
Design system/software
Identify stakeholders, viewpoints, views
Identify interfaces; determine rules and performance
System-Software tradeoffs
Customer interface (SW requirements and design)
Non-functional requirements
Agile for DoD; Earned value for Agile
Define and execute system and software V&V
Manage rapid change
Develop and build to low-level, detailed, precise requirements
Formal Methods, Coding
SW Devel. Processes, Methods, Tools
Make interfaces among software modules, data, and communications path work
Reuse/New Tradeoffs
Customer interface (coding)
SW quality attributes
Agile sprints, releases
Execute software V&V
SW Project Management
Don't do: "Hardware"
Depth (SW)

GENERAL ISSUES AFFECTING SOFTWARE

There are three general issues that affect many different types of software:

1. **Heterogeneity:** Refers to integration of new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope with this heterogeneity.
2. **Business and Social Change:** Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. Software need to be able to change their existing software and to rapidly develop new software.
3. **Security and Trust:** As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface.

TYPES OF SOFTWARE APPLICATIONS / 01

There are many different types of software application including:

1. **Stand-alone applications:** Application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.
2. **Interactive transaction-based applications: A**pplications that execute on a remote computer and that are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications where users can interact with a remote system to buy goods and services.
3. **Embedded control systems:** Software control systems that control and manage hardware devices. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.
4. **Batch processing systems:** Business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.

TYPES OF SOFTWARE APPLICATIONS / 02

There are many different types of software application including:

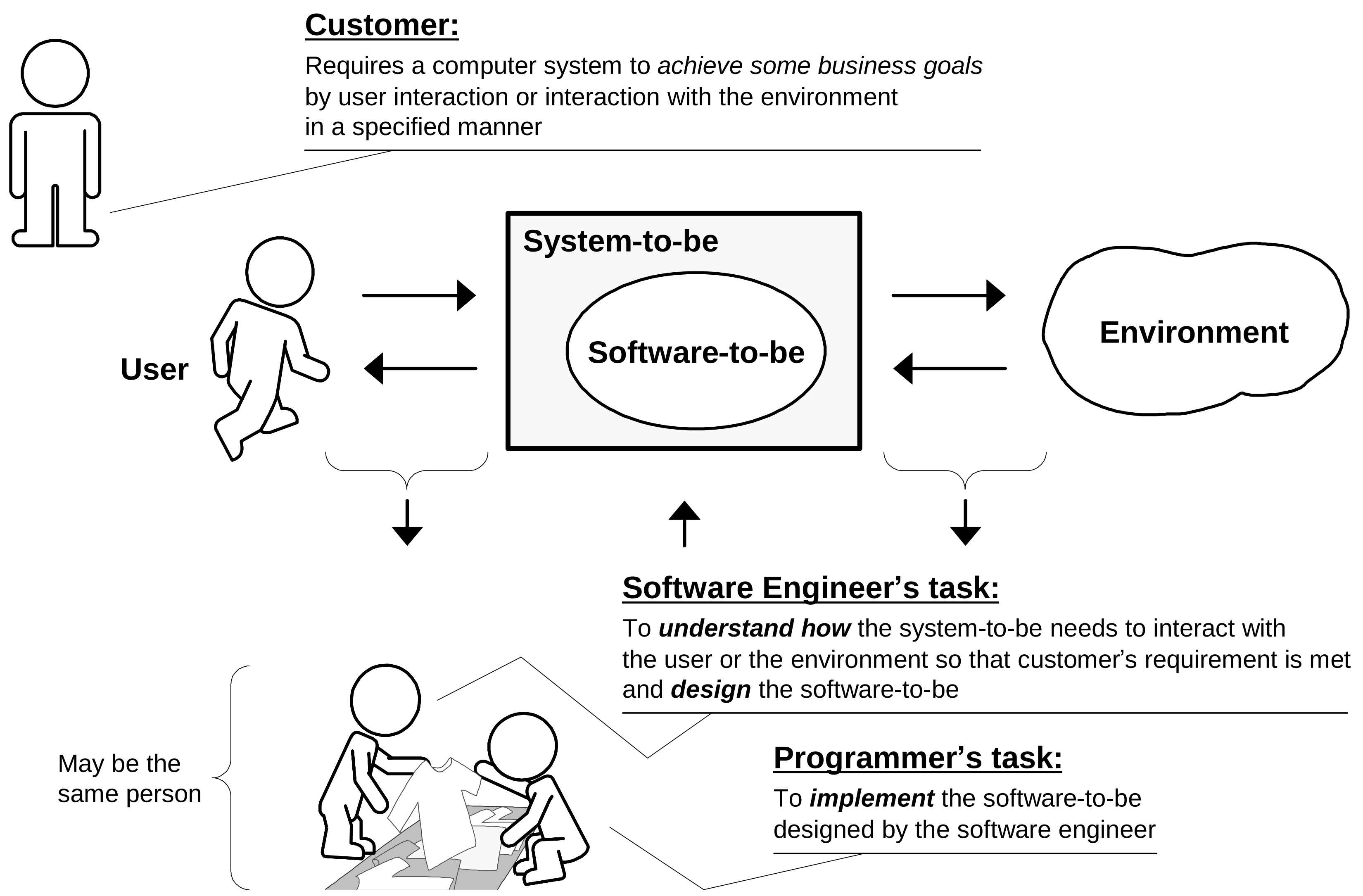
5. **Entertainment systems:** Systems that are primarily for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.
6. **Systems for modeling and simulation:** Systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.
7. **Data collection systems:** Systems that collect data from their environment using a set of sensors and send that data to other systems for processing. The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location.
8. **Systems of systems:** Systems that are composed of a number of other software systems. Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

SOFTWARE ENGINEERING & THE WEB

Changes in the software organization, led to changes in the ways that web-based systems are engineered. For example:

1. **Software reuse** has become the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.
2. **Incremental development:** It is now generally recognized that it is impractical to specify all the requirements for such systems in advance. Web-based systems should be developed and delivered incrementally.
3. **User interfaces** are constrained by the capabilities of web browsers. Web forms with local scripting are more commonly used. Application interfaces on web-based systems are often poorer than the specially designed user interfaces on PC system products.

THE ROLE OF SOFTWARE ENGINEER /01



Customer:

Requires a computer system to *achieve some business goals* by user interaction or interaction with the environment in a specified manner

Software Engineer's task:

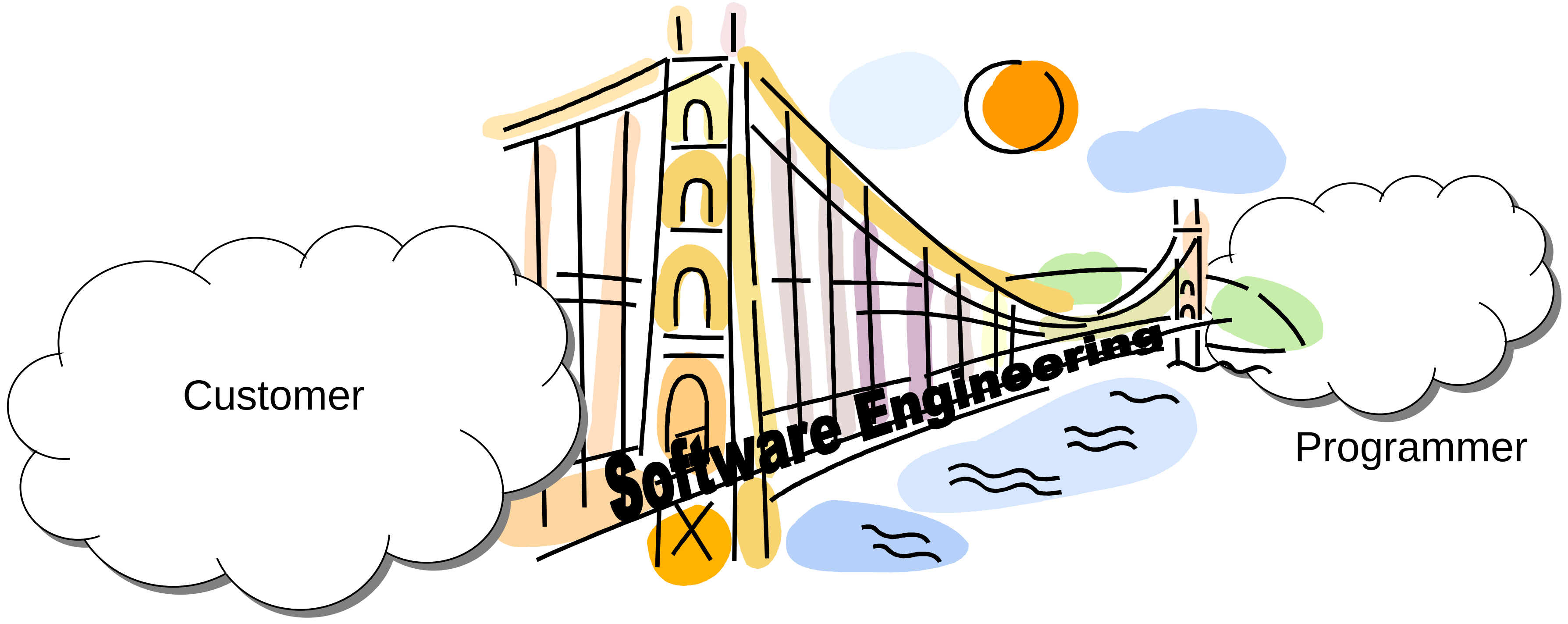
To ***understand how*** the system-to-be needs to interact with the user or the environment so that customer's requirement is met and ***design*** the software-to-be

Programmer's task:

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

THE ROLE OF SOFTWARE ENGINEER / 02

A bridge from customer needs to programming implementation



<u>First law of software engineering</u>

Software engineer is willing to learn the problem domain

(problem cannot be solved without understanding it first)

SOFTWARE ENGINEERING ETHICS

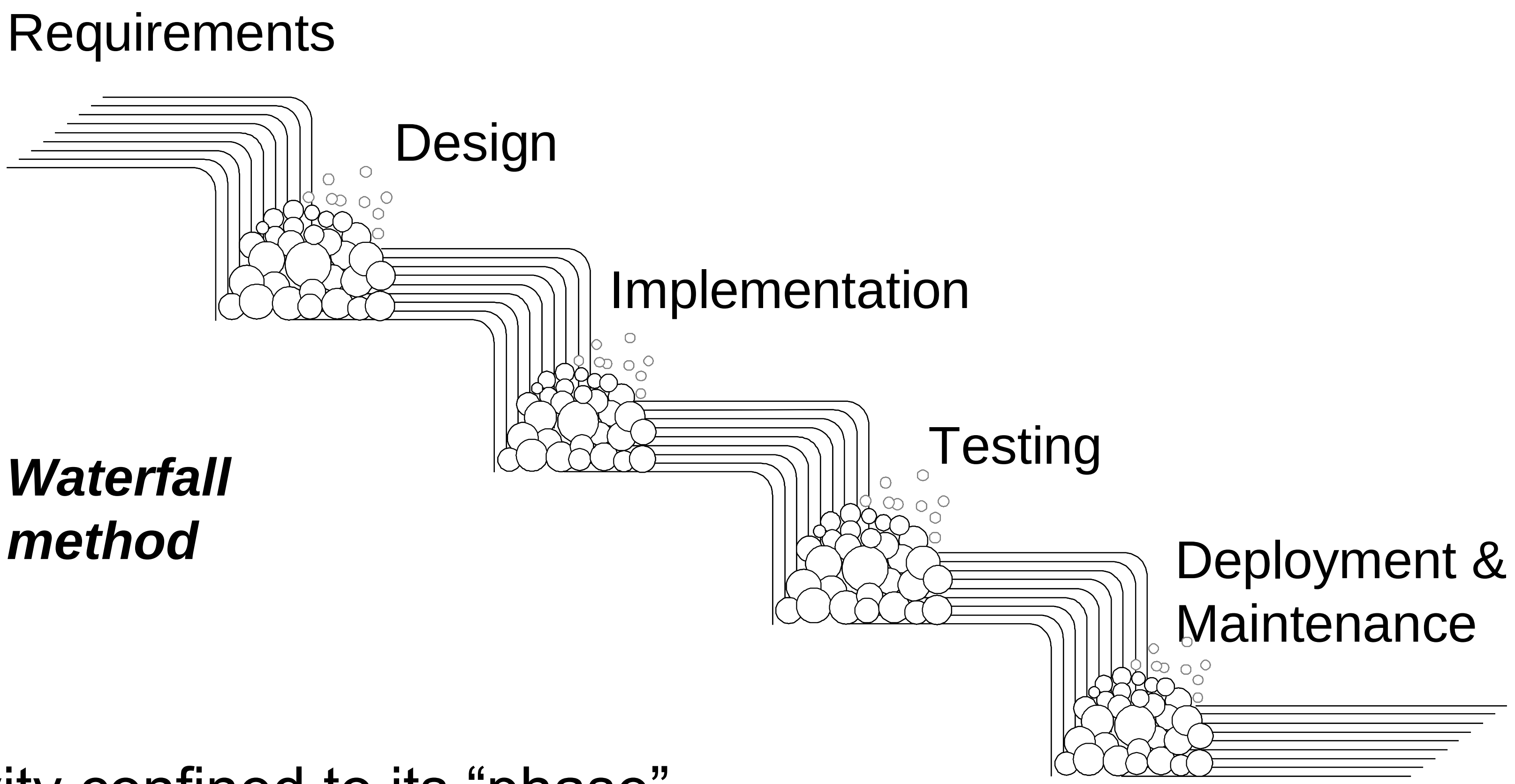
Some professional responsibilities includes:

1. **Confidentiality:** You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
2. **Competence:** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. **Intellectual Property Rights:** You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. **Computer Misuse:** You should not use your technical skills to misuse other people’s computers.

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

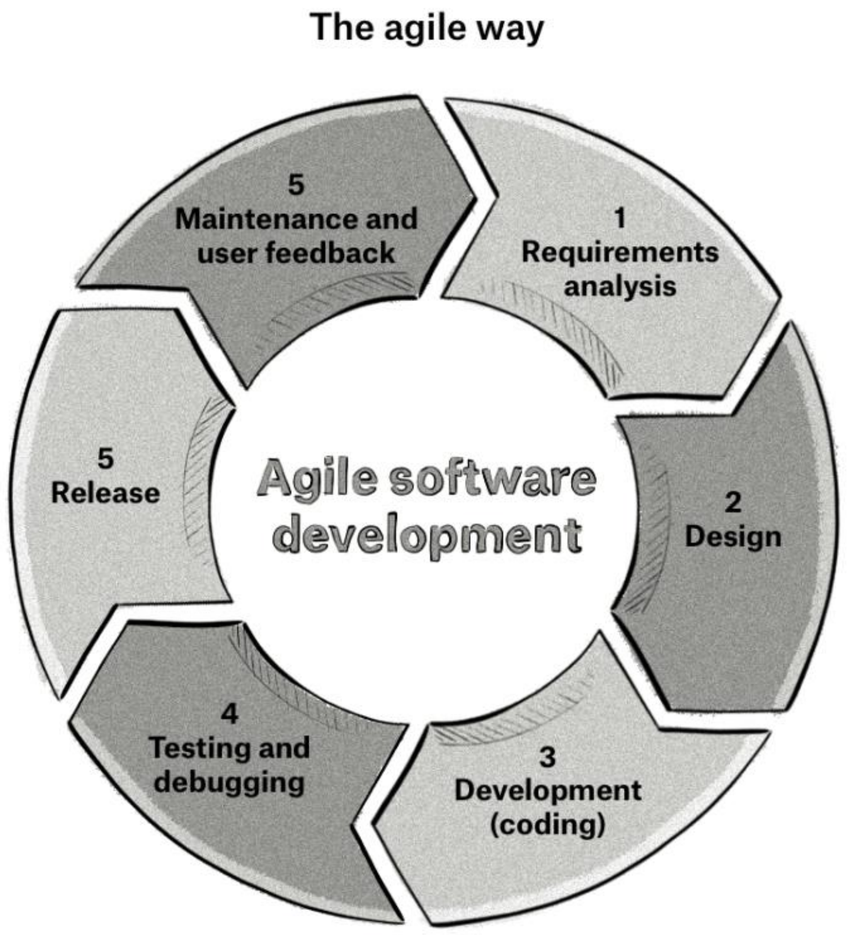
WATERFALL METHOD



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

AGILE SOFTWARE DEVELOPMENT

- **Agile software development** refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.
- Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation.



SOFTWARE MEASUREMENT

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

    for budgeting and scheduling
  - Product,

    for quality assessment

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

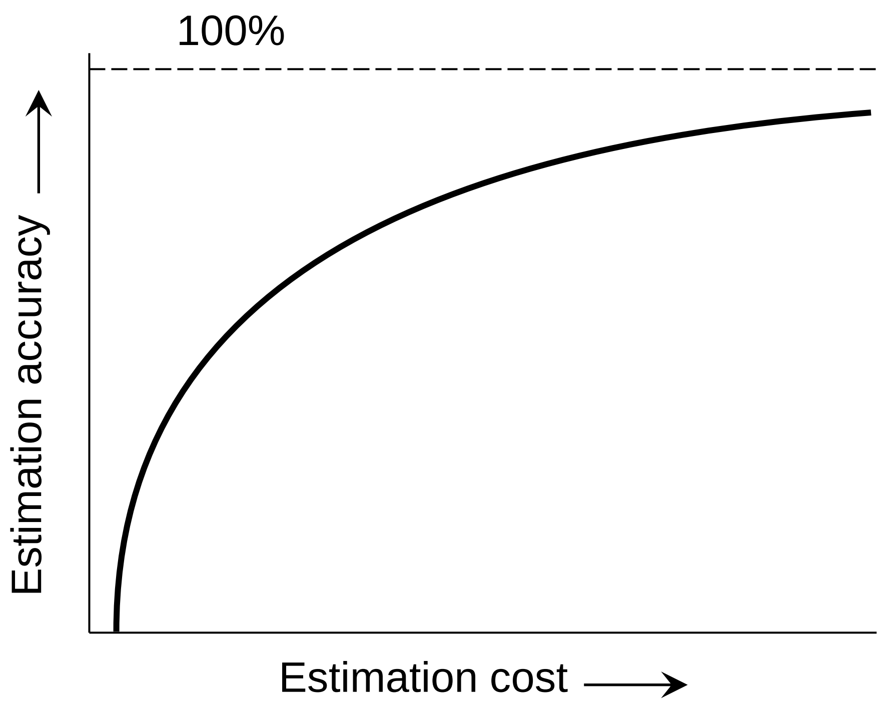
$$\text{Total size} = \sum \text{points-for-section } \boldsymbol{i} \quad (i = 1..\text{N})$$

- Step 4: Estimate speed of work (velocity)

- Step 5: Estimate the work duration

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

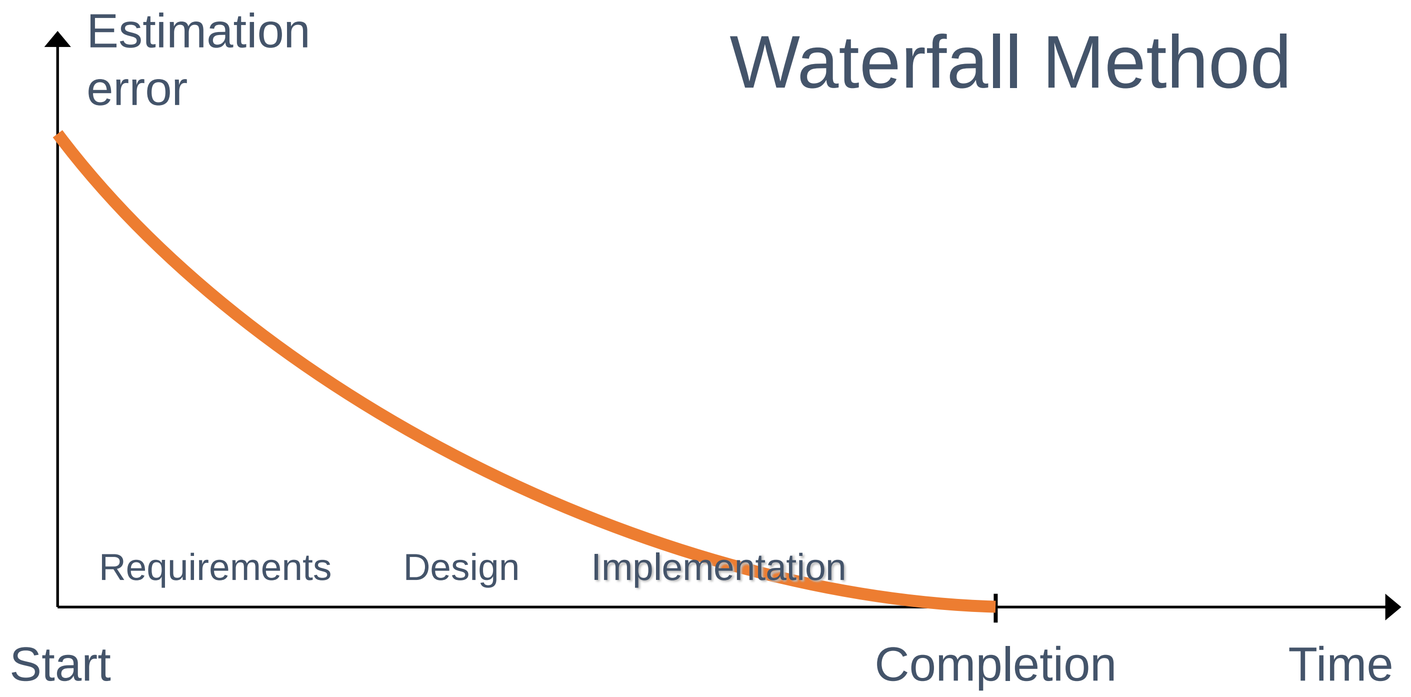
EXPONENTIAL COST OF ESTIMATION



❑ Improving accuracy of estimation beyond a certain point requires huge cost and effort (known as the law of diminishing returns)

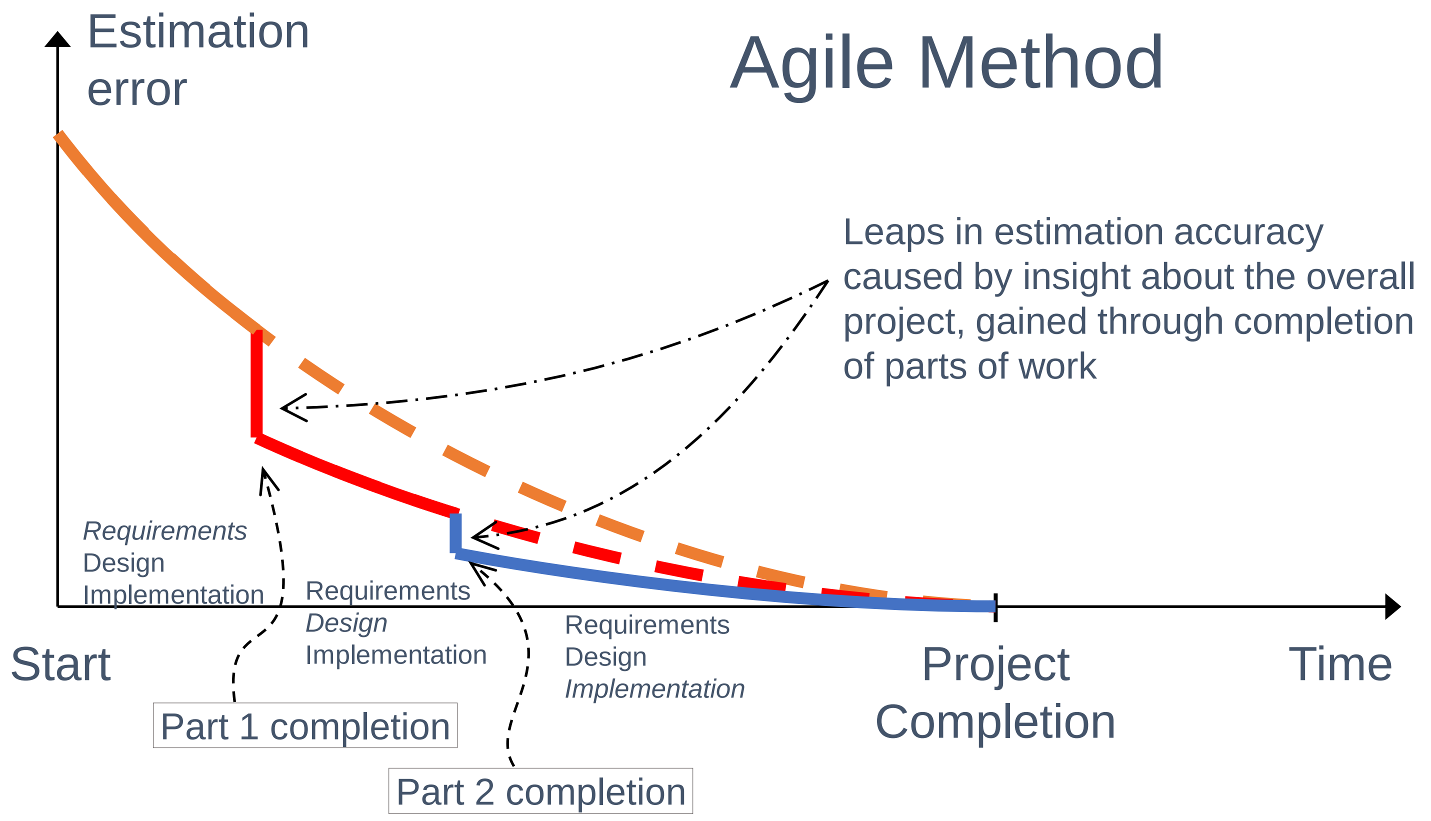
❑ In the beginning of the curve, a modest effort investment yields huge gains in accuracy

ESTIMATION ERROR OVER TIME



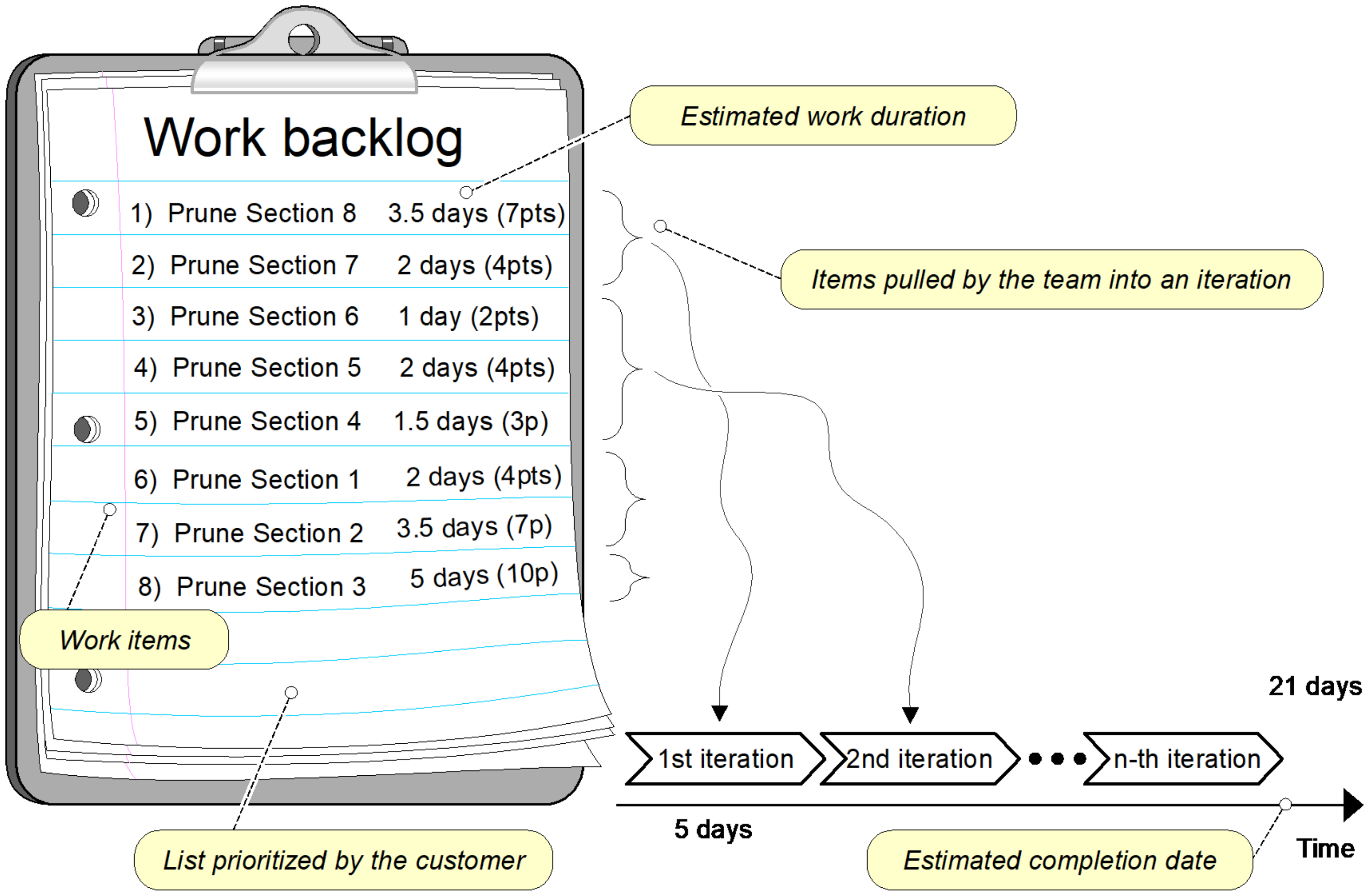
Waterfall method *cone of uncertainty* starts high and *gradually* converges to zero as the project approaches completion.

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


Work backlog
1) Prune Section 8 3.5 days (7pts)
2) Prune Section 7 2 days (4pts)
3) Prune Section 6 1 day (2pts)
4) Prune Section 5 2 days (4pts)
5) Prune Section 4 1.5 days (3p)
6) Prune Section 1 2 days (4pts)
7) Prune Section 2 3.5 days (7p)
8) Prune Section 3 5 days (10p)
Estimated work duration
Items pulled by the team into an iteration
Work items
List prioritized by the customer
21 days
1st iteration
2nd iteration
n-th iteration
5 days
Time
Estimated completion date

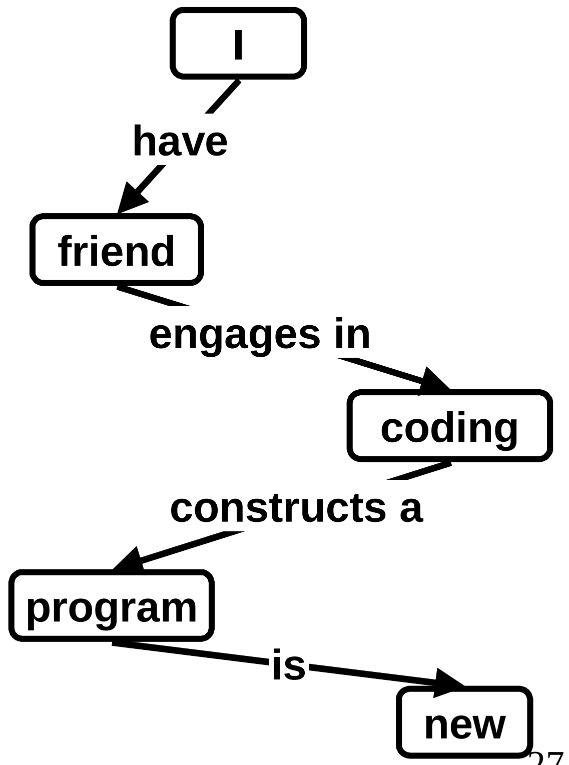
CONCEPT MAPS

Useful tool for problem domain description

SENTENCE: “My friend is coding a new program”

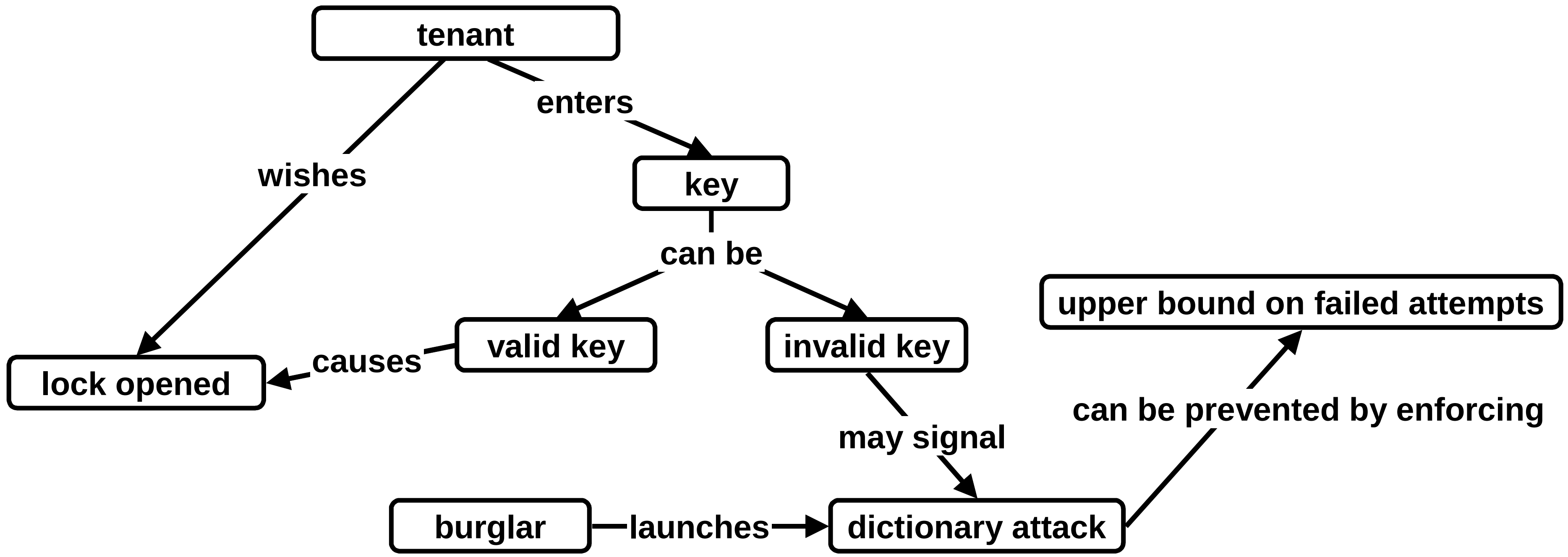
translated into propositions

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

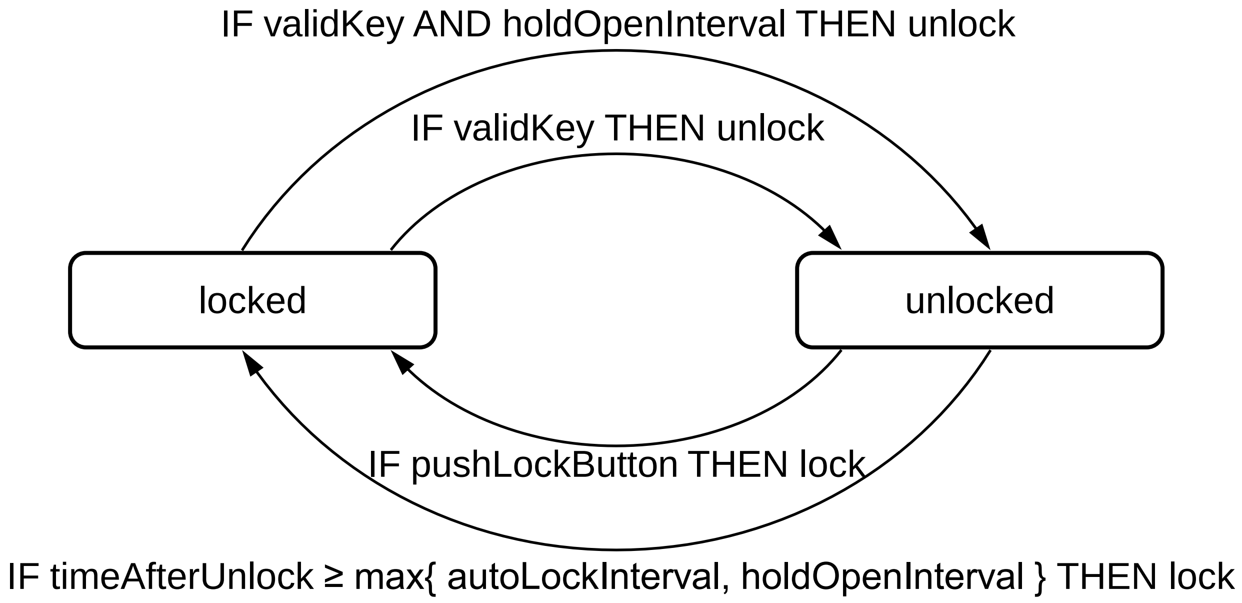


Search the Web for Concept Maps

CONCEPT MAP FOR HOME ACCESS CONTROL



STATES AND TRANSITION RULES


IF validKey AND holdOpenInterval THEN unlock
IF validKey THEN unlock
locked
unlocked
IF pushLockButton THEN lock
IF timeAfterUnlock ≥ max{ autoLockInterval, holdOpenInterval } THEN lock

… what seemed a simple problem, now is becoming complex