CTIS359
Principles of Software Engineering

Introduction to Software Engineering (SWE) & SWE Terminology

"The purpose of software engineering is to control complexity, not to create it."

Pamela Zave

This Lecture

- What is a computer program?
- What is software?
  - Software usage in our lives
- What is engineering?
- What is software engineering (SWE)?
- Complexity barrier(s) and engineering the software
- Goals of software engineering
- Software disasters and failures
- Software engineering activities
  - 4P’s of Software engineering
- Software engineering principles
- Software characteristics

What is a Computer Program?

- A list of **instructions** that tell the computer how to perform the **four basic (input, processing, output, storage) operations** to accomplish a task.
- Written by making use of programming languages (PLs)
    - C, Fortran, Pascal, Basic etc.
    - ADA, Java, C++, etc.
    - ASP, PHP, etc.
    - Assembly

Source: Computers are Your Future, 12th Edition, Catherine Laberta, Prentice Hall.

What is Software?

- Computer Programs + Associated Documentation + Configuration Data.
- Software products
  - may be developed for a particular customer
    - (a.k.a. *Bespoke, customized*) OR
  - may be developed for a general market
    - (a.k.a. *generic products*)

Source: Software Engineering, 9th Edition, Ian Sommerville, Addison-Wesley.

Where do we use software?

Where do we use software?

- @ your home
  - refrigerator, coffee-machine, TV set, camera (most of the electrical products) etc.
- @ your phone
  - Smart phones (OS, application software etc.)
- @ your car
  - Cruise-control, ESP, TCS, etc.
- @ your school
  - registrations
  - many software you use @ labs
- @ the cafeteria
  - make the payment
- @ online shopping …
- @ all financial system …
- @ national infrastructures and utilities
  - Electricity grid
- @ the industrial manufacturing
- @ the public transportation
  - Planes, high-speed trains, airport, coaches, metro, etc.
- @ the health services
  - Integrated health IS, imaging technologies

Where do we use software?

- @ your home
  - refrigera
- @ your
  -
- @
-
-
-
- @
  -
- @ the ind
- @ the public tr
  - Planes, high-spee
- @ the health services
  - Integrated health IS, imag

More and more systems are software controlled **everyday**.
We can NOT run the **modern world without software**.

Why (HW+SW) widespread?

- As computers became cheaper, they became available to a larger selection of the population and in parallel, their application areas widened.
  - In the beginning, computers were nothing but calculators—
    - they added, subtracted, multiplied, and divided numbers to get new numbers.
- Probably the major driving force of the computing technology is the realization that every piece of info can be represented as numbers.
  - This in turn implies that the computer, which until then was used to process numbers, can be used to process all types of info.

Source: Machine Learning: The New AI, The MIT Press, Ethem Alpaydin , 2016.

What is Engineering?

- The application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people.

Source: http://www.webster.com/

- Engineers make things WORK by appyling theories + methods + tools where appropirate.
- Try to discover solutions within CONSTRAINTS.
- Engineering disciplines such as civil, mechanical, and electrical involve the design, analysis, and construction of an artifact* for some practical purpose.
  - SWE is NO exception to this software products certainly have practical purposes.

*insan eliyle yapılmış şey

What is Engineering?

- Principles of <u>any</u> engineering activity

Projects should be completed

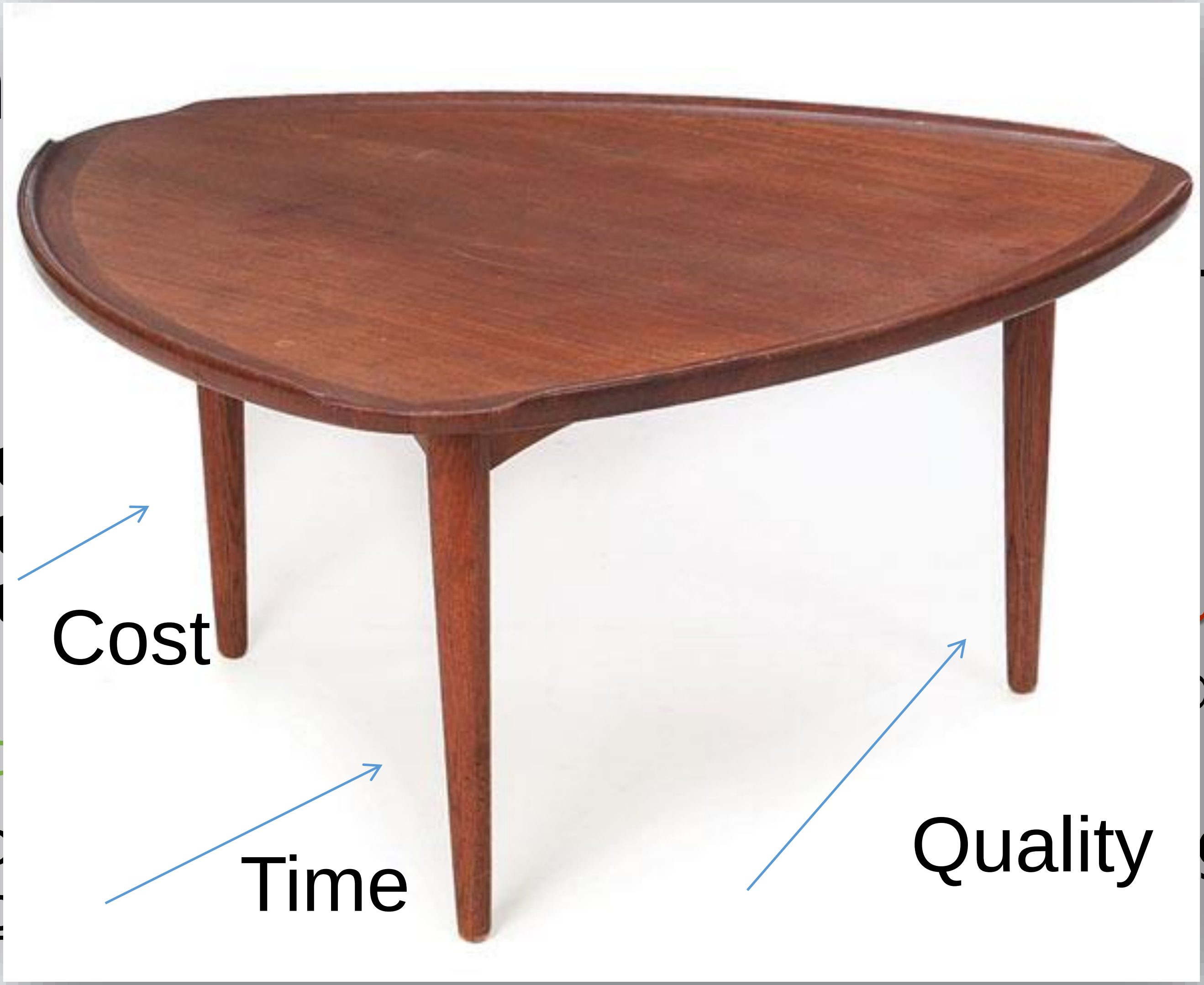
- Within anticipated budget **Cost**
- Within anticipated schedule **Time**
- With conformance to customers' requirements **Quality**

- Engineering is about <u>getting results</u> of the required quality within the schedule and budget.
- This often involves making compromises—engineers <u>cannot</u> be <u>perfectionists</u>.

Wha

- Prin tivity

Project
  - Wit
  - Wit
  - Wit ality
- Engin quired qualit
- This c gineers canno

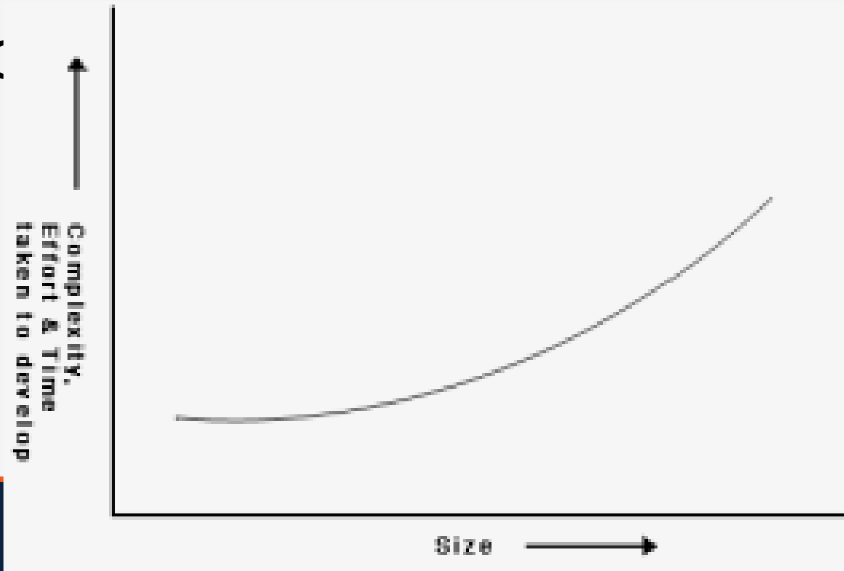


History of Software Engineering

- In early days of computing,
  - “hardware” is “hard” to change, while “software” is “soft”
  - If the software did not work, change it until it will work
  - Because software were NOT complex
- "*The Complexity Barrier*"
- Become difficult for one person develop, track, test, maintain, etc.
- Person→Team

History of Software Engineering

- In early days of computing,
  - “hardware” is “hard” to change, while “software” is “soft”
  - If the software did not work, change it until it will work
  - Because software were NOT complex
- "*The Complexity Barrier*"
- Become difficult for one persc
  maintain, etc.
- Person→Team



An analogy

Carpenter

- Simple house

$\Big\}$ Early programs

Engineer

- Powerplant
- Skyscraper

$\Big\}$ Todays, complex sofware

An analogy

Carpenter
- Simple house

$\}$ Early programs

Engineer
- Powerplant
- Skyscraper

$\}$ Todays, complex sofware

An analogy

Carpenter

- Simple house

} Early programs

Engineer

- Powerplant
- Skyscraper

} Todays, complex sofware

Large and/or Complex Systems

Building a skyscraper requires a different approach than building a slum.

How Program Size Affects…

How Program Size Affects Construction

- Scaling up in software development isn't a simple matter of taking a small project and making each part of it bigger.
- Suppose you wrote the 25,000-line Gigatron 1.0 software package in 20 ~~**man-months**~~ **person-months** and found 500 errors in field testing.
    - Gigatron 1.0 is successful.
- Suppose you start work on the Gigatron Deluxe, a greatly enhanced version of the program that's expected to be 250,000 LOC.
- Even though Deluxe's 10 times as large as the original Gigatron 1.0, the Gigatron Deluxe won't take 10 times the effort to develop; it'll take **30 times the effort**.
- Moreover, **30 times the total effort** doesn't imply 30 times as much construction.
    - It probably implies 25 times as much construction and 40 times as much architecture and system testing.
    - You won't have 10 times as many errors either; you'll have 15 times as many—or more.

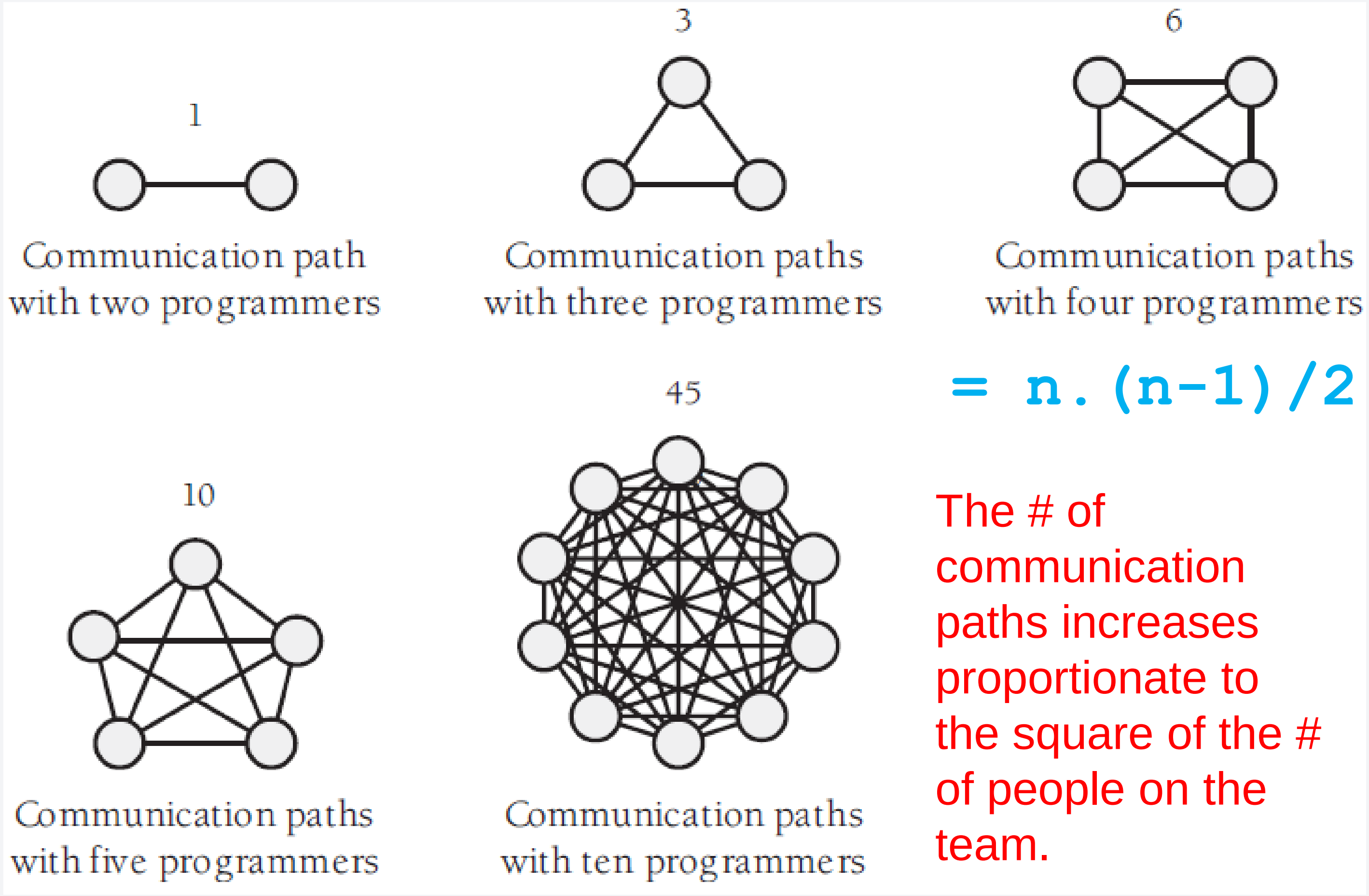
Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Communication and Size

- If you're the only person on a project, the only **communication path** is between you and the customer, unless you count the path across your *corpus callosum*, the path that connects the left side of your brain to the right.
- As the # of people on a project increases, the # of communication paths increases, too.
- The # doesn't increase additively as the # of people increases. It increases multiplicatively, proportionally to **the square of the # of people**.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Communication and Size


1
Communication path with two programmers
3
Communication paths with three programmers
6
Communication paths with four programmers
10
Communication paths with five programmers
45
Communication paths with ten programmers

= n.(n-1)/2

The # of communication paths increases proportionate to the square of the # of people on the team.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Communication and Size ★

- The 10% of projects that have 50+ programmers have at least 1,200 potential paths.
- The more communication paths you have, the more time you spend communicating and the more opportunities are created for communication mistakes.
- Larger-size projects demand organizational techniques that **streamline communication** or **limit it** in a sensible way.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Communication and Size

- The typical approach taken to streamlining communication is to formalize it in **documents**.
  - Instead of having 50 people talk to each other in every conceivable combination, 50 people read and write **documents**.
- Some are **text** documents; some are **graphic**. Some are printed on paper; others are kept in electronic form.

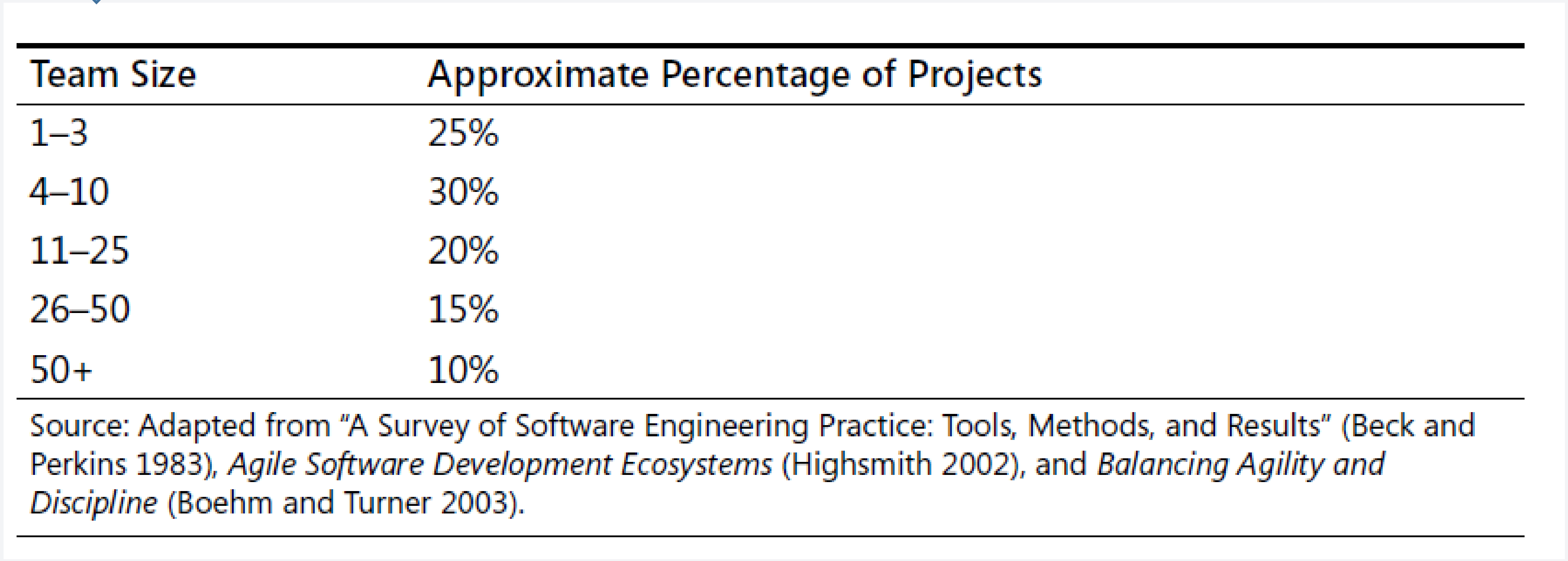
Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Range of Project Sizes vs. Team Size

- Is the size of the project you're working on typical?
- The wide range of project sizes means that you can't consider any single size to be typical.
- One way of thinking about project size is to think about the size of a project **team**.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Range of Project Sizes vs. Team Size



Team Size	Approximate Percentage of Projects
1–3	25%
4–10	30%
11–25	20%
26–50	15%
50+	10%

Source: Adapted from "A Survey of Software Engineering Practice: Tools, Methods, and Results" (Beck and Perkins 1983), *Agile Software Development Ecosystems* (Highsmith 2002), and *Balancing Agility and Discipline* (Boehm and Turner 2003).

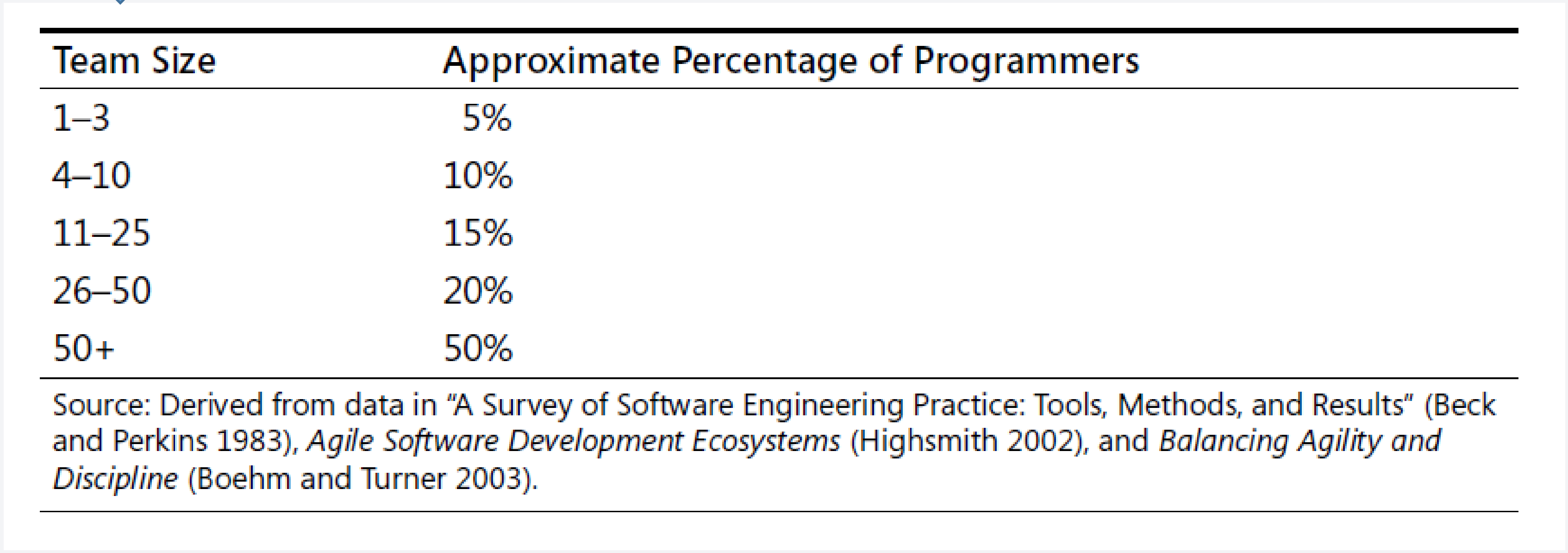
Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Range of Project Sizes vs. Distribution of Programmers

- One aspect of project size data that might NOT be immediately apparent is the difference between **the % of projects of various sizes** and **the # of programmers who work on projects of each size**.
  - Because larger projects use more programmers on each project than do small ones, they employ a large percentage of all programmers.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Range of Project Sizes vs. Distribution of Programmers



Team Size	Approximate Percentage of Programmers
1–3	5%
4–10	10%
11–25	15%
26–50	20%
50+	50%

Source: Derived from data in "A Survey of Software Engineering Practice: Tools, Methods, and Results" (Beck and Perkins 1983), *Agile Software Development Ecosystems* (Highsmith 2002), and *Balancing Agility and Discipline* (Boehm and Turner 2003).

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Effect of Project Size on Errors

- Both quantity and type of errors are affected by project size.
- You might not think that error type would be affected, but as project size increases, a larger % of errors can usually be attributed to mistakes in **requirements** and **design**.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Effect of Project Size on Errors

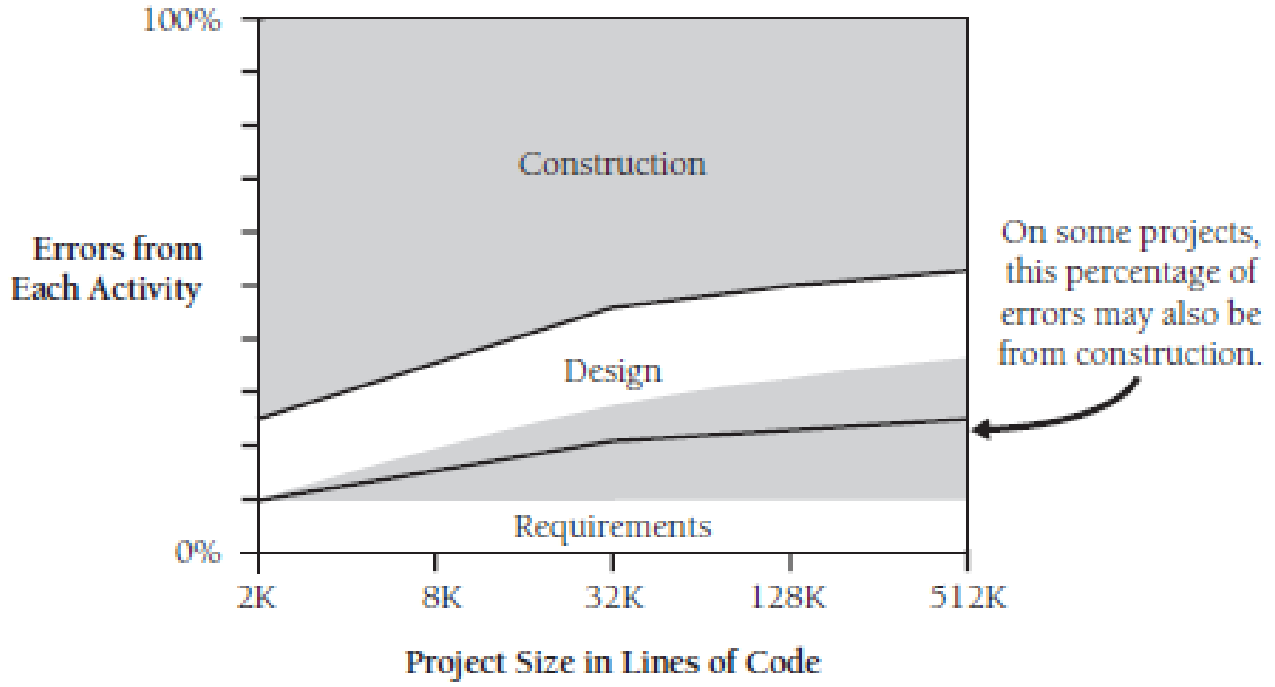


Figure 27-2 As project size increases, errors usually come more from requirements and design. Sometimes they still come primarily from construction (Boehm 1981, Grady 1987, Jones 1998).

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Effect of Project Size on Errors

- On small projects, construction errors make up about 75% of all the errors found.
  - Methodology has less influence on code quality, and the biggest influence on program quality is often the skill of the individual writing the program (Jones 1998).
- On larger projects, construction errors can taper off to about 50% of the total errors; requirements and architecture errors make up the difference.
  - Presumably this is related to the fact that more requirements development and architectural design are required on large projects, so the opportunity for errors arising out of those activities is proportionally larger.
  - In some very large projects, however, the proportion of construction errors remains high; sometimes even with 500 KLOC, up to 75 % of the errors can be attributed to construction (Grady 1987).
- As the kinds of defects change with size, so do the #s of defects.
- You would naturally expect a **project** that's twice as large as another to have twice as many errors. But the **density of defects**—the # of defects per 1000 LOC—increases. The **product** that's twice as large is likely to have more than twice as many errors.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Effect of Project Size on Errors

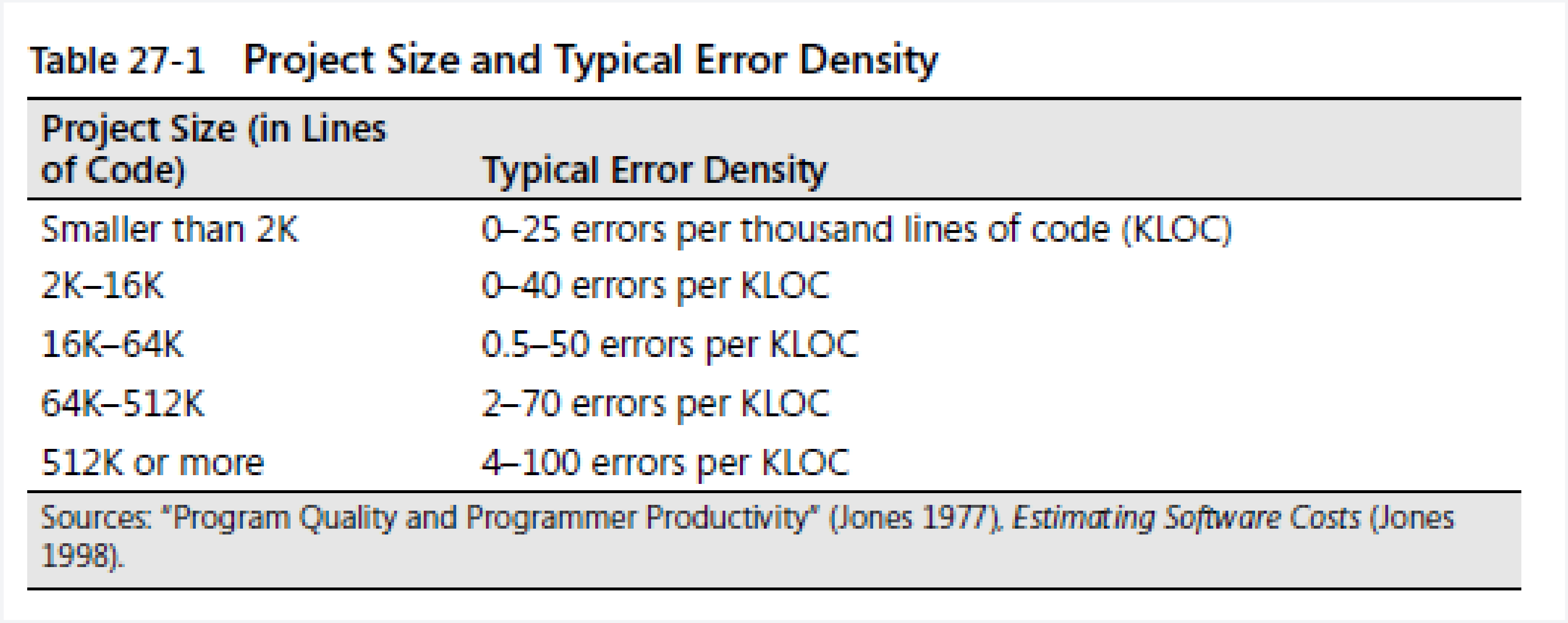


Table 27-1 Project Size and Typical Error Density

Project Size (in Lines of Code)	Typical Error Density
Smaller than 2K	0–25 errors per thousand lines of code (KLOC)
2K–16K	0–40 errors per KLOC
16K–64K	0.5–50 errors per KLOC
64K–512K	2–70 errors per KLOC
512K or more	4–100 errors per KLOC

Sources: "Program Quality and Programmer Productivity" (Jones 1977), *Estimating Software Costs* (Jones 1998).

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Effect of Project Size on Development Activities

- If you are working on a one-person project, the biggest influence on the project's success or failure is you.
- If you're working on a 25-person project, it's conceivable that you're still the biggest influence, but it's more likely that no one person will wear the medal for that distinction; your organization will be a stronger influence on the project's success or failure.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- As project size increases and the need for formal communications increases, the kinds of activities a project needs change dramatically.

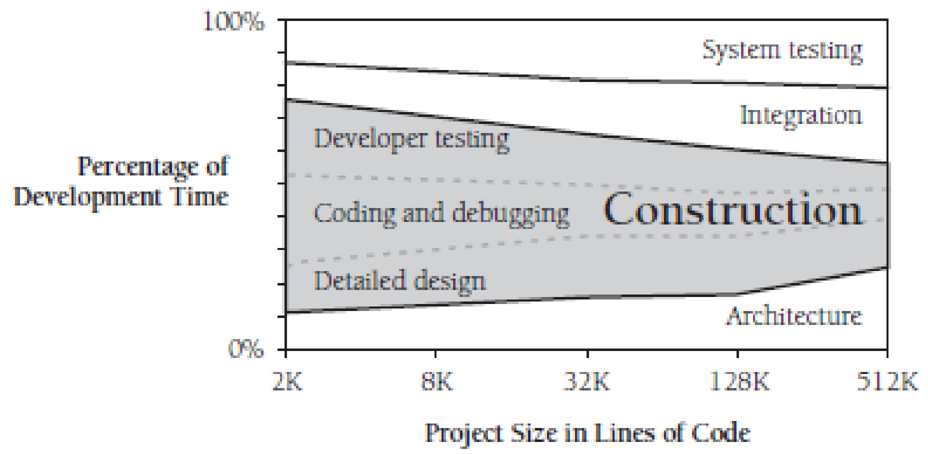


Figure 27-3 Construction activities dominate small projects. Larger projects require more architecture, integration work, and system testing to succeed. Requirements work is not shown on this diagram because requirements effort is not as directly a function of program size as other activities are (Albrecht 1979; Glass 1982; Boehm, Gray, and Seewaldt 1984; Boddie 1987; Card 1987; McGarry, Waligora, and McDermott 1989; Brooks 1995; Jones 1998; Jones 2000; Boehm et al. 2000).

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- On a **small project**, **construction** is the most prominent activity by far, taking up as much as **65% of the total development time**.
- On a medium-size project, construction is still the dominant activity but its share of the total effort falls to about 50%.
- On very large projects, architecture, integration, and system testing take up more time and construction becomes less dominant.
- ***In short, as project size increases, construction becomes a smaller part of the total effort.***
- The chart looks as though you could extend it to the right and make construction disappear altogether.
    - It is cut off at 512K.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- Construction becomes less predominant because as project size increases, the construction activities—detailed design, coding, debugging, and unit testing—scale up proportionately but many other activities scale up faster.

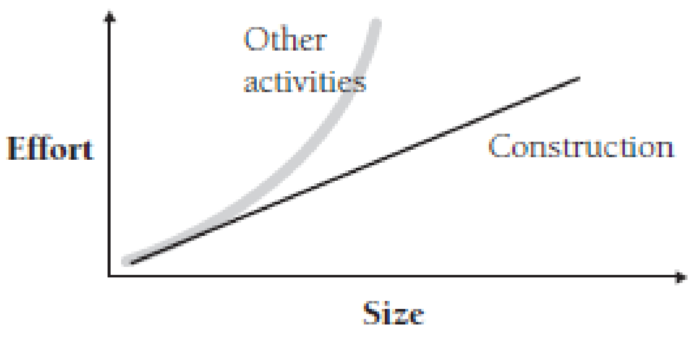


Figure 27-4 The amount of software construction work is a near-linear function of project size. Other kinds of work increase nonlinearly as project size increases.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- Projects that are close in size will perform similar activities, but as sizes diverge, the kinds of activities will diverge, too.
- As the running example is described, when the Gigatron Deluxe comes out at 10X the size of Gigatron 1.0, it will need
  - **25X** more construction effort,
  - **25–50X** more planning effort,
  - **30X** more integration effort,
  - **40X** more architecture and system testing effort.
- Proportions of **activities vary** because different activities become critical at different project sizes.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- Here's a list of activities that grow at a more-than-linear rate as project size increases:
  - Communication
  - Planning
  - Management
  - Requirements development
  - System functional design
  - Interface design and specification
  - Architecture
  - Integration
  - Defect removal
  - System testing
  - Document production

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

Distribution in the Activities and Project Size

- Regardless of the size of a project, a few techniques are always valuable:
  - disciplined coding practices,
  - design and code inspections by other developers,
  - good tool support, and
  - use of high-level languages.
- These techniques are valuable on small projects and invaluable on large projects.

Source: Code Complete: A Practical Handbook of Software Construction, Steve McConnell, 2004

When the term "software engineering" was used?

- One of the first uses of the phrase "software engineering" was in 1968, by a NATO Study Group on Computer Science.
- A conference was organized at that time, motivated by the rapidly increasing importance of computer systems in *many* activities of society.

When the term "software engineering" was used?

- The Study Group discussed possible techniques & methods that might lead to solving the problems.
- They deliberately and provocatively used the term "software engineering," with an emphasis on engineering, as they wanted to "imply the need for software manufacture to be based on the type of theoretical foundations and practical disciplines that are traditional in the established branches of engineering".
- Today, many of the issues they identified are addressed by evolving SWE techniques and practices even as the scope of applications has increased dramatically.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

History of SWE - First Definition

- “Software engineering is the establishment of sound (kuvvetli,sağlam) engineering principles in order to obtain <u>economical</u> software that is <u>reliable</u> and works <u>efficiently</u> on real machines”

  [Naur and Randell, 1969]

History of Software Engineering – Other Definitions

- "Software engineering is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation to develop, operate and maintain them."

  [Boehm, 1976].

History of Software Engineering – Other Definitions

- 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)

  [IEEE Computer Society, 1990].
  - IEEE definition suggests
- it's NOT ONLY what is produced that's important but also *how* it is produced.
- Engineering disciplines employ an established set of *systematic*, *disciplined*, and *quantifiable* approaches to the development of artifacts.

History of Software Engineering – Textbook Definitions

- Software engineering is a discipline whose aim is the <u>production of fault-free</u> software, delivered on <u>time</u> and within <u>budget</u>, that satisfies <u>the client's needs</u>.

Source: OO & Classical Software Engineering, 7th Edition, Stephen Schach

History of Software Engineering – Textbook Definitions

- Software engineering is an engineering discipline which is concerned with ALL aspects of software production.
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

Source: Software Engineering, 9th Edition, Ian Sommerville, Addison-Wesley.

History of Software Engineering – Textbook Definitions

- In Sommerville's definition, there are 2 key phrases:
- 1. Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are NO applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints.
- 2. Software engineering is NOT just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production.

Source: Software Engineering, 9th Edition, Ian Sommerville, Addison-Wesley.

What is Software Engineering? – Textbook Definitions

- Software engineering is an engineering discipline that involves ALL aspects of <u>developing</u> and <u>maintaining</u> a software product.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

What is Software Engineering?

- Software engineering is a **modeling activity**.
  - SW engineers deal with complexity through modeling, by focusing at any one time on only the relevant details and ignoring everything else.
  - In the course of development, SW engineers build many different models of the system and of the application domain.
- SWE is a **problem-solving activity**. Models are used to search for an acceptable solution.
- Software engineering is a **rationale-driven** activity.

Source: Object-Oriented Software Engineering Using UML, Patterns, and Java™ 3rd Edt. B. Bruegge, A. H. Dutoit

Overall

- SWE activities include managing, costing, planning, modeling, analyzing, specifying, designing, implementing, testing, and maintaining.
- By "engineering approach," we mean that each activity is understood and controlled, so that there are **few surprises** as the software is specified, designed, built, and maintained.
- Whereas CS provides the **theoretical foundations** for building software, SWE focuses on **implementing the software** in a controlled and scientific way.

Source: Software Metrics: A Rigorous and Practical Approach, 3rd Edt., Norman Fenton, James Bieman, 2014.

The goal of Software Engineering

- The goal of software engineering, is the development of software system that meet the needs of customers and are <u>reliable</u>, <u>efficient</u>, and <u>maintainable</u>.
- In addition, the system should be produced in an economical fashion, meeting project schedules and budget.
- This is no easy task, especially for **<u>large, complex applications</u>**.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Why SWE is **important**?

- 1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- 2. It is usually cheaper, in the long run, to use SWE methods and techniques for software systems rather than just write the programs as if it was a personal programming project.
  - For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

Source: Software Engineering, 9th Edition, Ian Sommerville, Addison-Wesley.

The 4 "P's" of SWE

- The production of software systems can be extremely complex and present many challenges.
- Systems, especially large ones, require the coordination of many people called stakeholders, who must be organized into teams and whose primary objective is to build a product that meets defined requirements.
- The entire **effort** must be organized into a cohesive project, with a solid plan for success.
- Finally, to successfully develop the product, the activities of the people must be organized through use of an orderly and well-defined process.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

The 4 "P's" of SWE

- Collectively, these are 4 P's (People, Product, Project, and Process) that constitute SWE.
- Successful software projects must adequately plan for and address ALL P’s.
- Sometimes, the needs of each of the P's conflict with each other, and a proper balance must be achieved for a project to be successful.
- Concentrating on one P without the others can lead to a project's failure.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

4 P's

- People
  - Project stakeholders
- Product
  - The software product + associated documents.
- Project
  - The activities carried out to produce the product.
- Process
  - Framework within which the <u>team carries out the activities</u> necessary to build the product.

People

- People are the most important resource on a software project.
    - It is through their efforts that software is successfully constructed and delivered.
- Competent people must be recruited, trained, motivated, and provided with a growth path, which is no easy task.

People

- Typically, several groups of people are **involved** with and have a stake in a project's outcome.
- These are called its **stakeholders**.
- They include business management, project management, the development team, customers, and end users.
- Although each group is motivated to see the project succeed, given their diverse roles each has **a different perspective on the process**.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

People - Business Management

- These are people responsible for the business side of the company developing the software. They include **senior management** (e.g., V.P. Finance), **marketing** (e.g., Product Manager), and **development managers**.
- They primary focus on business issues including profit, cost effectiveness, market competitiveness, and customer satisfaction.
- They are typically not particularly knowledgeable about or involved in the technical aspects of the project.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

People - Project Management

- Project managers are responsible for planning and tracking a project.
- They are involved throughout, managing the people, process, and activities.
- They continuously monitor progress and proactively implement necessary changes and improvements to keep the project on schedule & within budget.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

People - Development Team

- SWEs are responsible for **developing** & **maintaining** the software.
- Software development includes **many tasks** such as requirement gathering, software architecture and design, implementation, testing, configuration management, and documentation.
- SWEs are motivated by many factors including technical innovation, low overhead (e .g., a minimum of business-type meetings), and having the time and support to stay involved in technology.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

People - Customers

- Customers are responsible for purchasing the software.
- They may or may not actually use the software.
- Customers may be purchasing it for use by others in their organization. They are primarily interested in software that is cost-effective, meets specific business needs, and is of high quality.
- They are typically involved in some aspect of specifying requirements, and since they are paying for the project, they have the ultimate say in defining the requirements.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

People - End Users

- End users are people who interact with and use software after it is finished being developed.
- End users are motivated by software that's easy to use and helps them perform their jobs as efficiently as possible.
  - **Ex:** Once they become accustomed to and are effective using a particular UI, they are typically reluctant to accept major changes to it.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Product

- The products of a software development effort consist of **much more than** the source & object code.
- They also include project documentation
  - requirements document
  - design specification, and etc.
  - test plans & results
- customer documentation
  - installation guide
  - command reference
- productivity measurements.
- These products are often called artifacts*

*insan eliyle yapılmış şey

Product

- Project documentation
  - Documents produced during software definition and development.
- Code
  - Source and object.
- Test documents
  - Plans, cases, and results.
- Customer documents
  - Documents explaining how to use and operate product.
- Productivity measurements
  - Analyze project productivity.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Project

- A software project defines the activities and associated results needed to produce a software product.
- Every project involves a similar set of activities: planning, determining what's required, determining how the software should be built to meet the requirements, implementing the software, testing the software, and maintaining it once delivered to customers.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Major activities of a software Project

- Planning
  - Plan, monitor, and control the software project.
- Requirements analysis
  - Define what to build.
- Design
  - Describe how to build the software.
- Implementation
  - Program the software.
- Testing
  - Validate that software meets the requirements.
- Maintenance
  - Resolve problems, adapt software to meet new requirements.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Process

- A software process is a framework for carrying out the activities of a project in an organized and disciplined manner.
- It imposes structure and helps **guide** the many people and activities in a coherent manner.
- A software project progresses through different phases, each interrelated and bounded by time.
- A software process expresses the **interrelationship** among the phases by defining their **order** and **frequency** , as well as defining the **deliverables of the project**.
- Specific software process **implementations** are called software process models. There are several such models, but most are based on either the waterfall or iterative models.

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Process

- The **waterfall process model**
  - the simplest model
  - forms the basis for most others
- A pure waterfall process model
  - dictates that phases are implemented in sequence, with NO phase starting before the previous one has almost completed.
  - That is, phases are executed in a strictly sequential order, usually with small overlaps.
- Software development rarely occurs in the strict waterfall sequence ☹

Source: Software Engineering: Modern Approaches 2nd Edt. E. Braude, E. Bernstein, 2011

Process

- In practice, then, we often use **iterative processes** for software development, in which all or some of the waterfall process is repeated <u>several times</u>.
- Some processes dictate that activities may be carried out in parallel.

Software Characteristics

- 1) Software is developed or engineered, it is NOT manufactured in the classical sense.

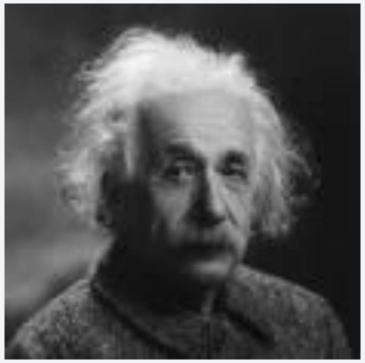
Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Software Characteristics

1) Software is developed or engineered, it is NOT manufactured in the classical sense.

Raw material

+



Human Creativity

=

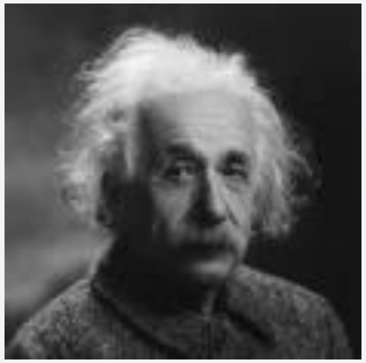
Physical **Element**

e.g. $500

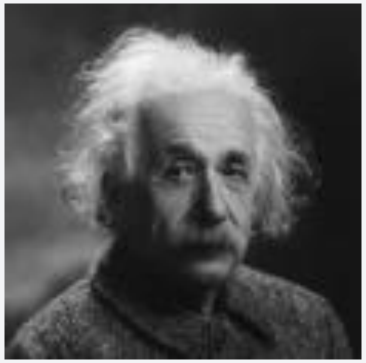
Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Software Characteristics

1) Software is developed or engineered, it is NOT manufactured in the classical sense.



+



=

+

Human Creativity

Human Creativity

Logical Element

Physical Elemen

e.g. $5M

e.g. $0.05

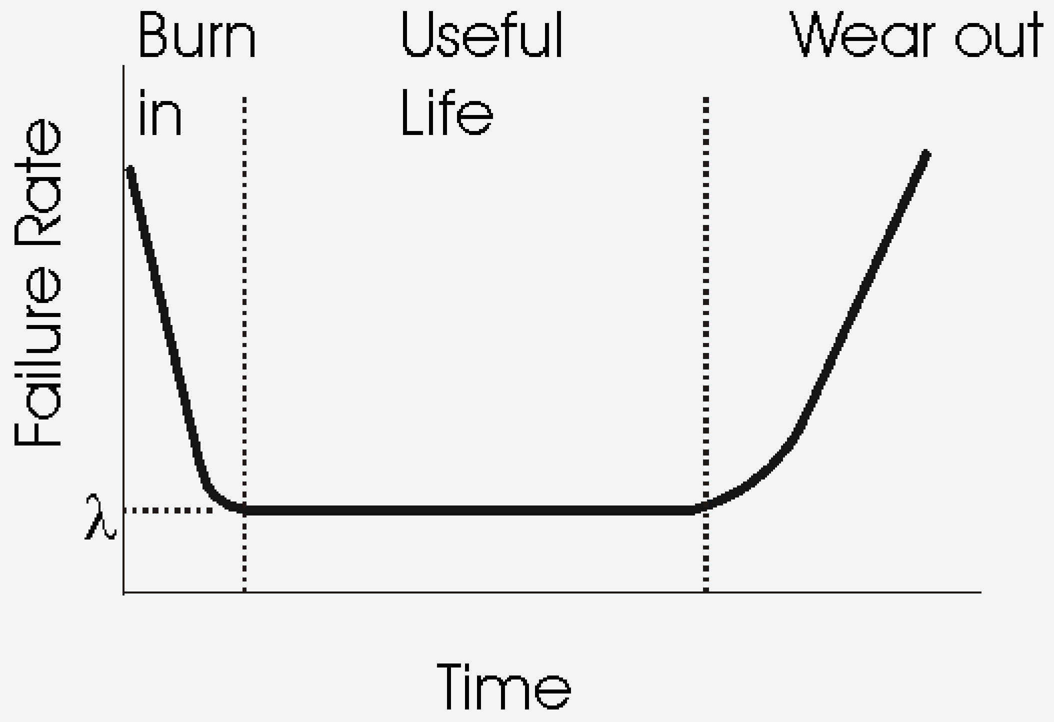
Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Software Characteristics

2) Software does NOT wear out (~aşınmak, yıpranmak) but deteriorates (~gerileme).

Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

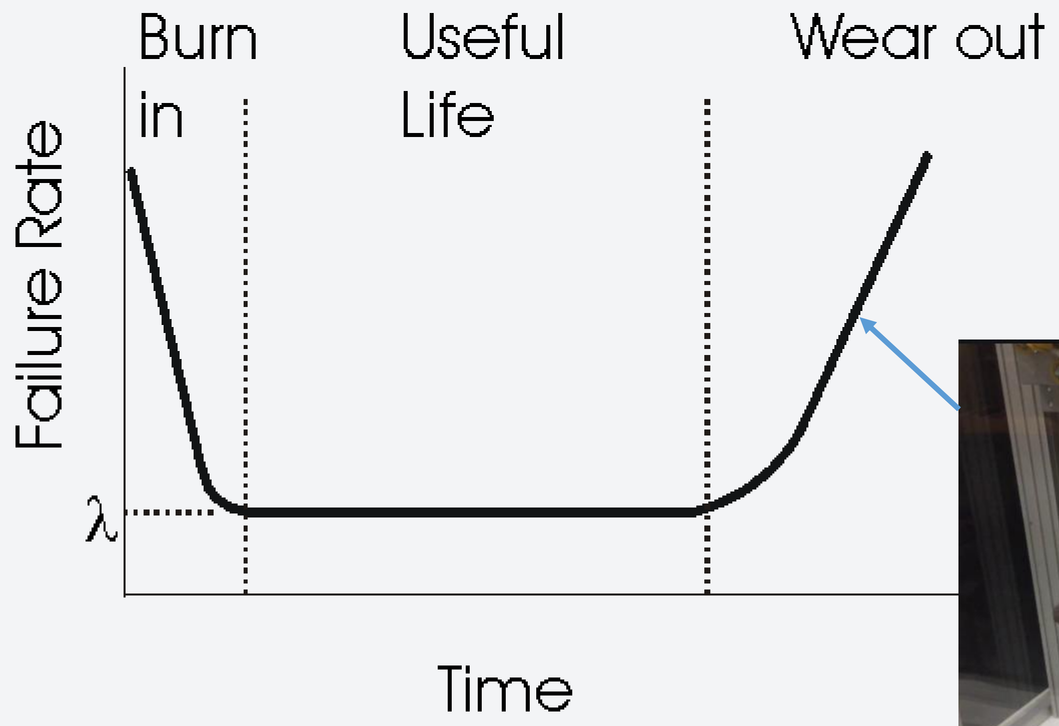
Hardware Characteristics



Failure Curve for **Hardware**

Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

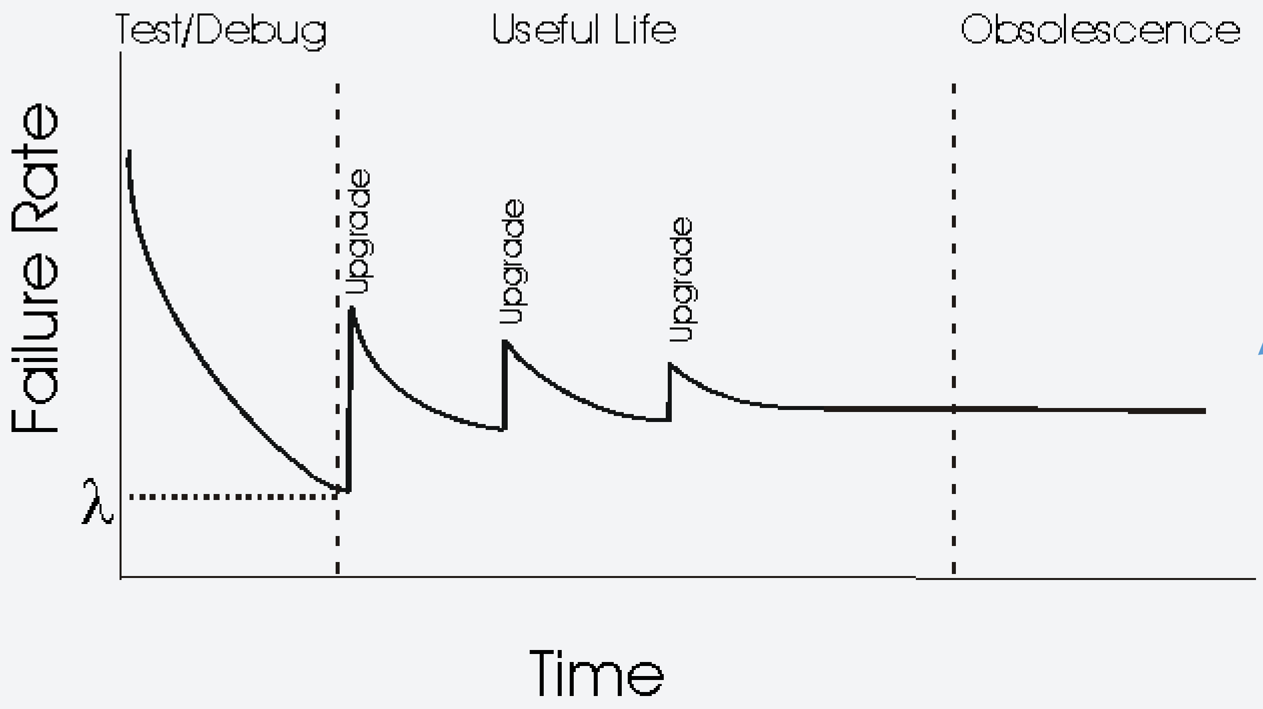
Hardware Characteristics



Failure Curve for **Hardware**

Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Software Characteristics



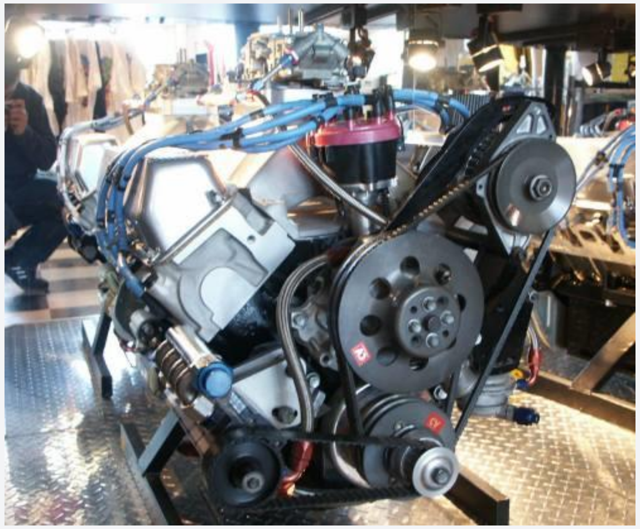
Failure Curve for **Software**

Deterioration
Not Ware
Out

Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Software Characteristics

3) Although industry is moving towards the component based assembly, most software continues to be custom built.



```
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < columns; j++) {
      nextStates [i][j] =
         getCellAt (i, j).getNextState ();
   }
}
```

Source: Software Engineering: A Practitioner's Approach 8th Edition by R. Pressman, B. Maxim

Wrap Up- Software engineering involves …

- Software engineering involves the **multi-person** construction of **multi-version** programs.
    - **Ex:** A typical modern software consists of tens, hundreds, if not thousands of classes.

Source: Concise Guide to Software Engineering From Fundamentals to Application Methods, O'Regan Gerard, 2017

Software engineering includes:

1. Methodologies to design, develop and test SW to meet customers' needs.
2. SW is engineered.
   - That is, the SW products are properly designed, developed and tested in accordance with engineering principles.
3. Quality and safety are properly addressed.

Source: Concise Guide to Software Engineering From Fundamentals to Application Methods, O'Regan Gerard, 2017

Software engineering includes:

4. Mathematics **may be** employed to assist with the design and verification of SW products.
   - The level of mathematics employed will depend on the safety critical nature of the product.
   - Systematic peer reviews and rigorous testing will often be sufficient to build quality into the SW, with heavy mathematical techniques reserved for safety and security critical SW.
5. Sound project management (**PM**) and quality management (**QM**) practices are employed.
6. Support and maintenance of the SW is properly addressed.

Source: Concise Guide to Software Engineering From Fundamentals to Application Methods, O'Regan Gerard, 2017

References

1. Computers are Your Future, 12th Edition, Catherine Laberta, Prentice Hall.
2. Software Engineering, 9th Edition, Ian Sommerville, Addison-Wesley.
3. OO & Classical Software Engineering, 7th Edition, Stephen Schach
4. Software Engineering: Modern Approaches 2nd Edition Eric J. Braude, Michael E. Bernstein, Wiley, 2011
5. Systems Analysis and Design, 9th Edition, Gary B. Shelly Harry J. Rosenblatt
6. Software Engineering: A Practitioner's Approach 8th Edition by Roger Pressman, Bruce Maxim
7. Object-Oriented Software Engineering Using UML, Patterns, and Java™ 3rd Edt. B. Bruegge, A. H. Dutoit
8. Concise Guide to Software Engineering From Fundamentals to Application Methods, O'Regan Gerard, 2017