Software Engineering (433)

Introduction

Section: 1
Location: Masri109;
Time: Tuesday:09:30-10:50, Thursday:09:30-10:50

Lecturer: Dr. Adel Taweel
E-mail: ataweel@birzeit.edu

web-page:
http://

Why Software Engineering?

- **Software development is Complex**!
- **Important to distinguish "small" systems** (*one developer, one user, experimental use only*) **from "Complex" systems** (*multiple developers, multiple users, products*)
- **Experience with "small" systems is misleading**
  - *One person techniques do not scale up*
- **Analogy with bridge building:**
  - A bridge over a stream = easy, one person job
  - A bridge over a River … ? (*the techniques do not scale)*

Why Software Engineering ?

The problem is ***complexity***

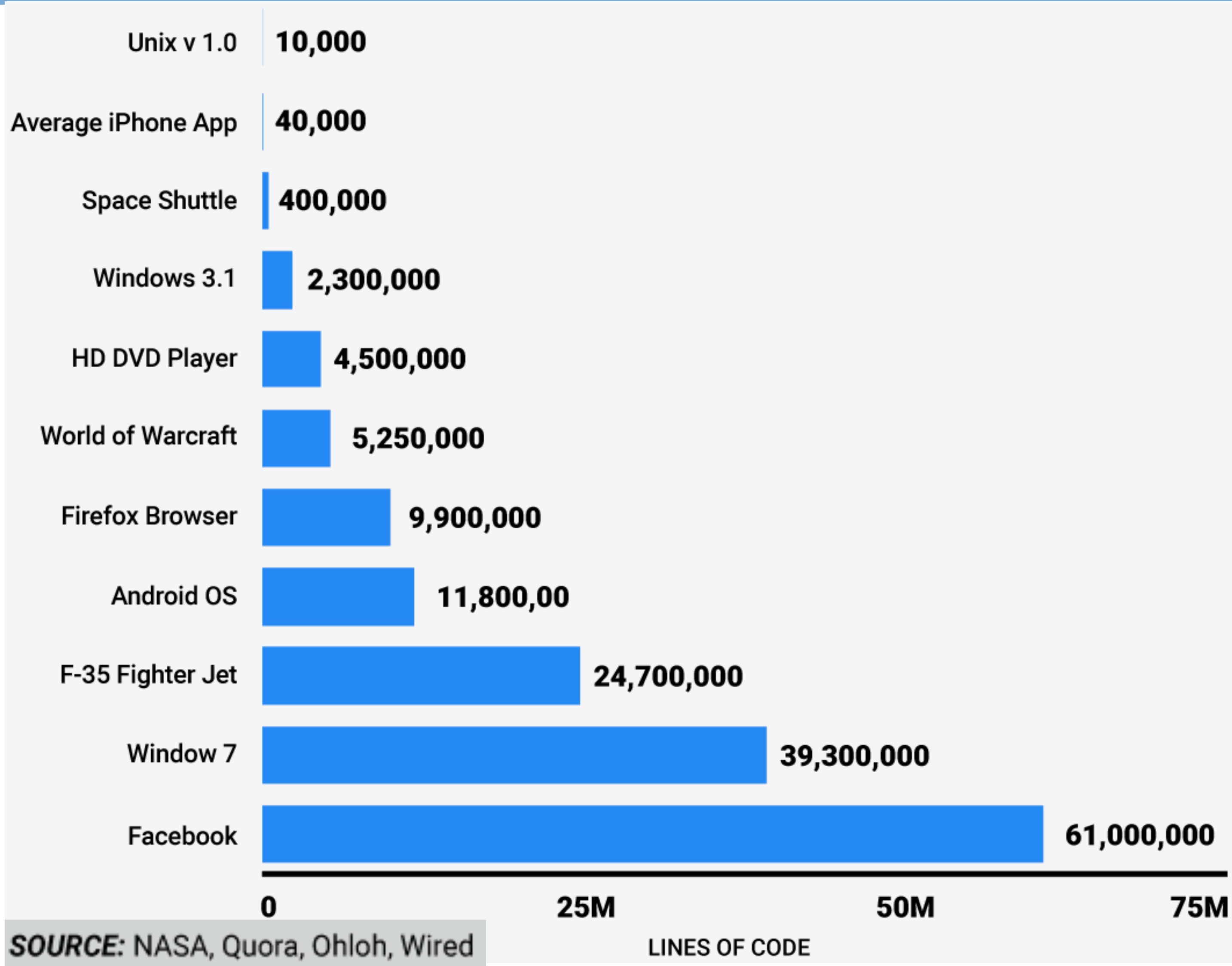
Complexity depends on many factors, but ***size*** is key:

- UNIX:
  - v 1 (1971) contains 10,000 lines of code
  - v 10 (1989) contains 4 million lines of code

- Windows:
  - Windows 2000 contains 100 million lines of code
  - Windows 7 contains 39.3 million lines of code (?)

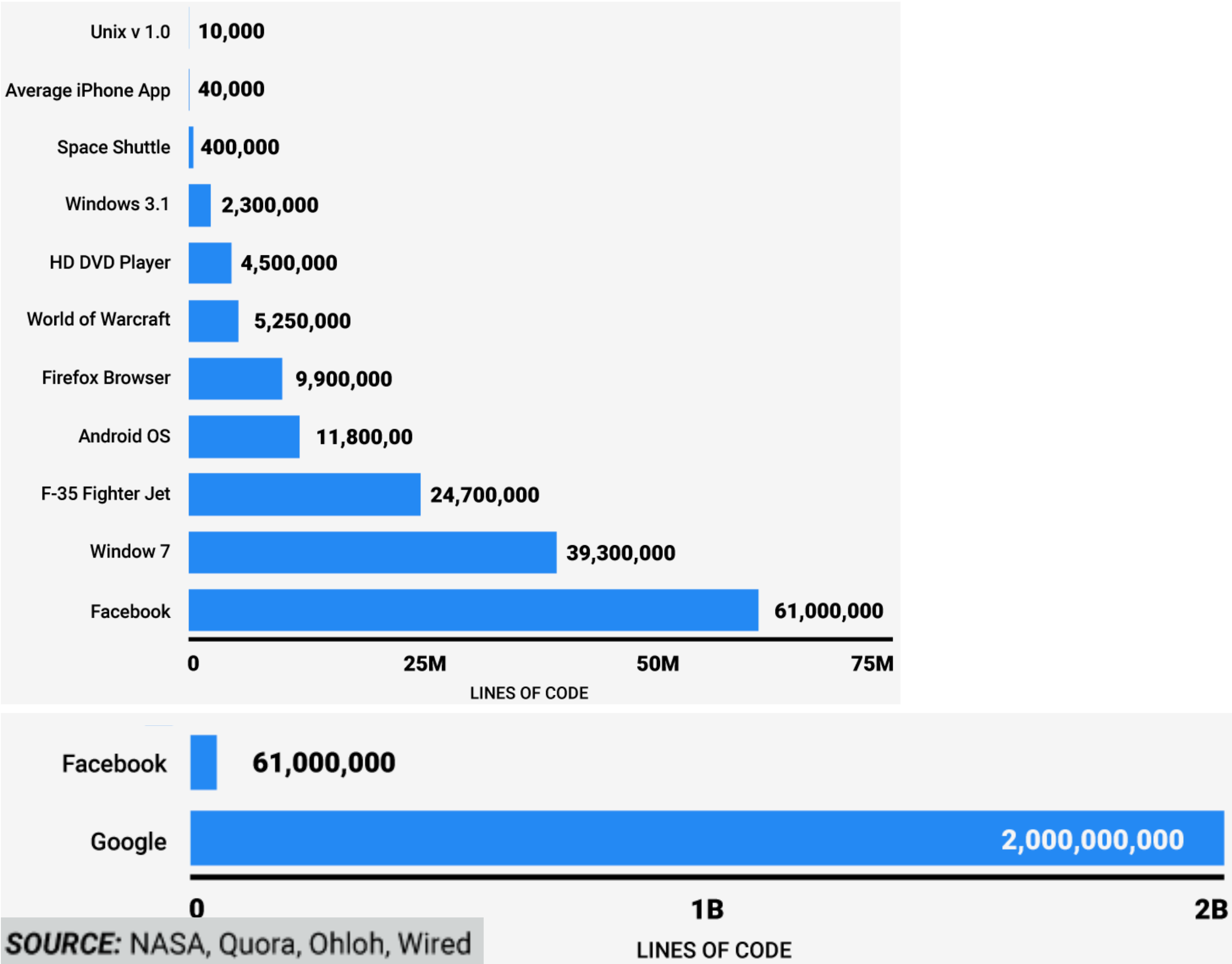
Why Software Engineering ?

Complexity
and
Size
matter!



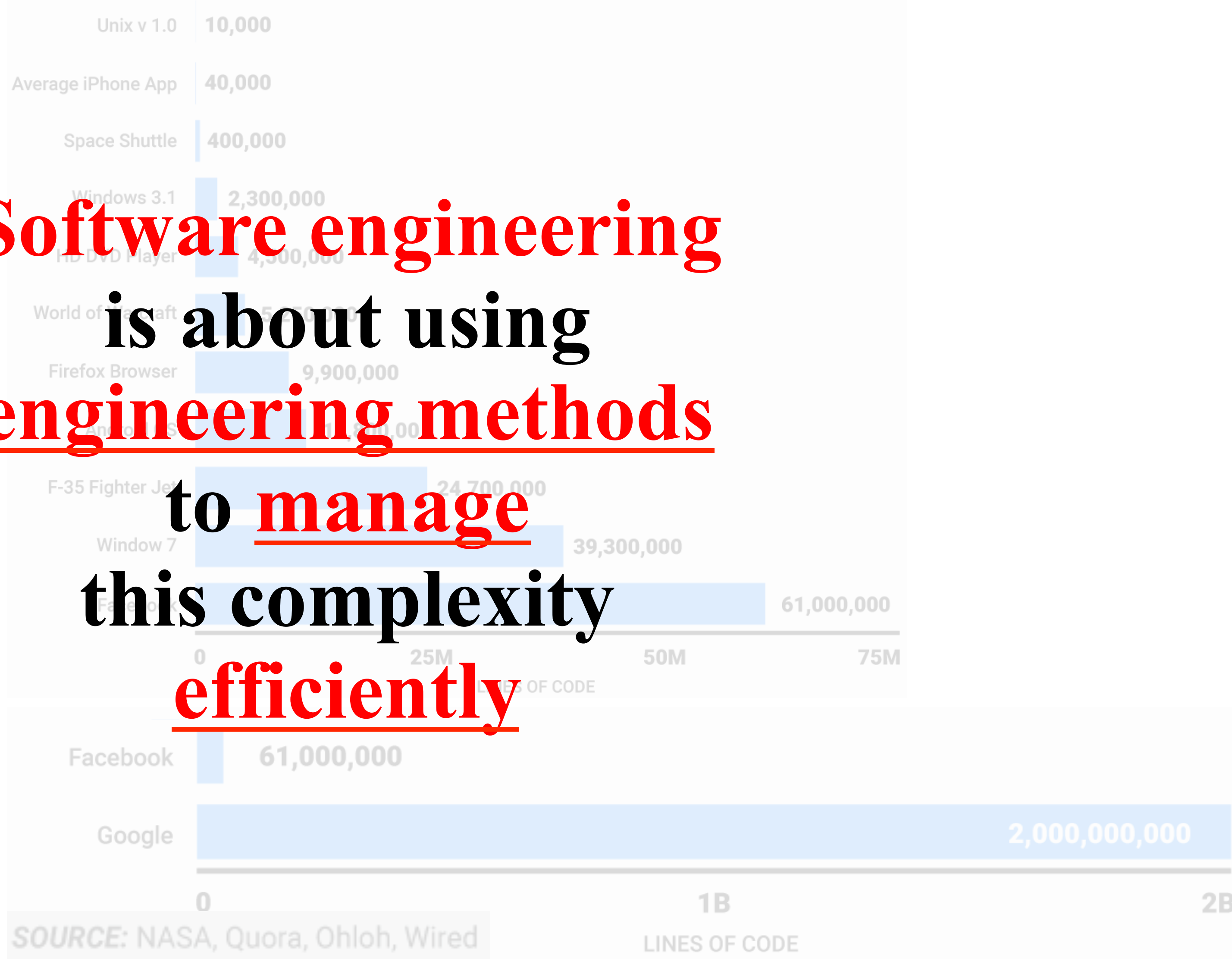
Why Software Engineering ?

Complexity ***increases*** *as* ***Size*** **increases!**



Why Software Engineering ?

Complexity increases as Size increases!



Software engineering is about using engineering methods to manage this complexity efficiently

Teaching method

- Lectures **(~ 3hrs per week)**
  Sec2: Tuesday+Thursday 09:30- 10:50 (Masri109)
- Independent Student Reading
- Practical work **(A group project)**
- Tutorials (**in lectures**)

---------------------- Course Assessment ----------------------

- Mid-term + Quizzes **30%**
- Group Project and Assignment **35%**
- Final **35%**

--

Recommended Course Textbooks

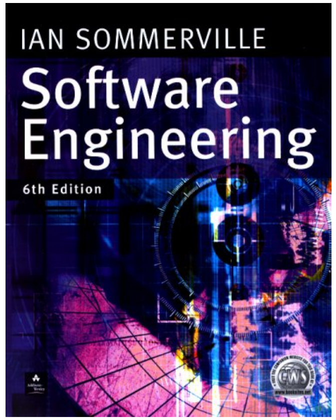
- Sommerville I. (2010)
  Software Engineering, 9th Edition, Addison-Wesley, Harlow, Essex,UK (6th ,7th,or 8th would suffice)

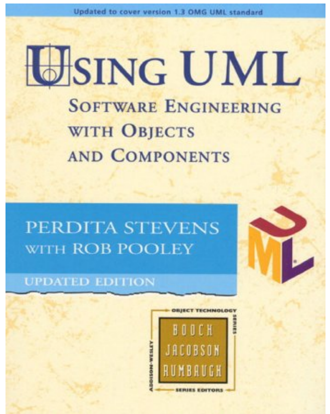
- Bruegge and Dutoit, ***Object-Oriented Software Engineering Using UML, Patterns, and Java***, Prentice Hall **3rd** Edition

- Stevens P. with Pooley, R. (2005)
  Using UML: Software Engineering with Objects and Components,
  2nd Ed., Addison-Wesley, Harlow, Essex, UK

- Jeffrey A. Hoffer, Joey F. George, Joseph S. Valacich. (2005)
  Modern System Analysis and Design 4th - 6th Edition, Prentice Hall.

- *Roger Pressman (2014)*, ***Software Engineering: A Practitioner's Approach*** *6-8th Edition, McGraw-Hill.*





What is the difference between software engineering and computer science?

Computer Science | **Software Engineering**

is concerned with

Computer Science:

- theory
- fundamentals

Algorithms, date structures, complexity theory, numerical methods

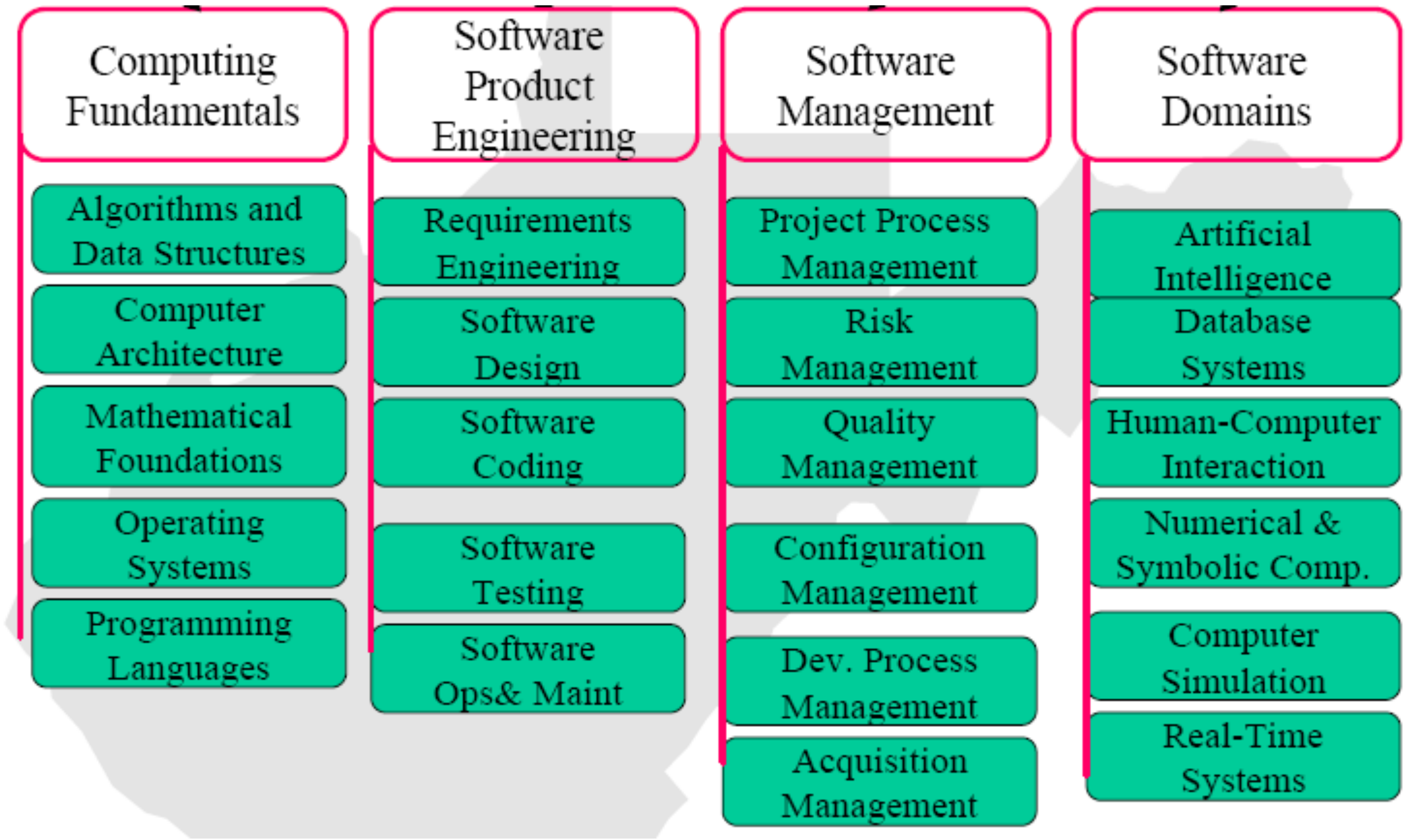
Software Engineering:

- Understanding domain challenges
- the practicalities of developing and
- delivering useful software

SE deals with practical problems in complex software products

Computer science theories are currently insufficient to act as a complete underpinning for software engineering, BUT it is a foundation for practical aspects of software engineering

Software Engineering Body of Knowledge



Source: http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr004.pdf

SE history

- SE introduced first in 1968 – conference about “software crisis” when the introduction of third generation computer hardware led to more complex software systems than before
- Early approaches based on informal methodologies led to
  - Delays in software delivery
  - Higher costs than initially estimated
  - Unreliable, difficult to maintain software
- Thus, there is a need for new methods and techniques to manage the production of complex software, ones that consider the intangible nature of software as a product.

Software myths

- **Management myths**
  - *Standards and procedures for building software exist*
  - *Add more programmers if behind schedule*
- **Customer myths**
  - *A general description of objectives enough to start coding*
  - *Requirements may change as software is flexible*
- **Practitioner myths**
  - *Task accomplished when the program works*
  - *Quality assessment when the program is running*
  - *"Working program" the only project deliverable*

Software failures

- **Therac-25 (1985-1987)**: six people overexposed during treatments for cancer
- **Taurus (1993)**: the planned automatic transaction settlement system for London Stock Exchange cancelled after five years of development
- **Ariane 5 (1996)**: rocket exploded soon after its launch due error conversion (16 floating point into 16-bit integer)
- **The Mars Climate Orbiter** assumed to be lost by NASA officials (1999): different measurement systems (Imperial and metric)

More Software failures

- **Passport System** delays cause backlog (1999, UK)
- **Ferry Company** left thousands of lorries stranded for 12 hours (back up also failed, 1999, UK)
- **Inland Revenue** (IR) 'losing tax records' (2000, UK)

=> IR spokesman said 'All major IT initiatives have some kind of teething problems ....'

=> Guardian (20 July 2000) 'At the centre of the crisis are two computer systems .... Files appear to have gone missing somewhere between the two'

- **General Motors Ford** Cars (2016, USA + Worldwide): A "software bug" that may cause human safety, 4.5M cars recalled.

Even More Software failures

In 1995 annual US spending on software projects reached 250 billion dollars

This involved some 175,000 projects

Of this spend:

Overspend cost 59 billion dollars
Cancelled projects cost 81 billion dollars

Why Software Fail?

Causes of Software Failure

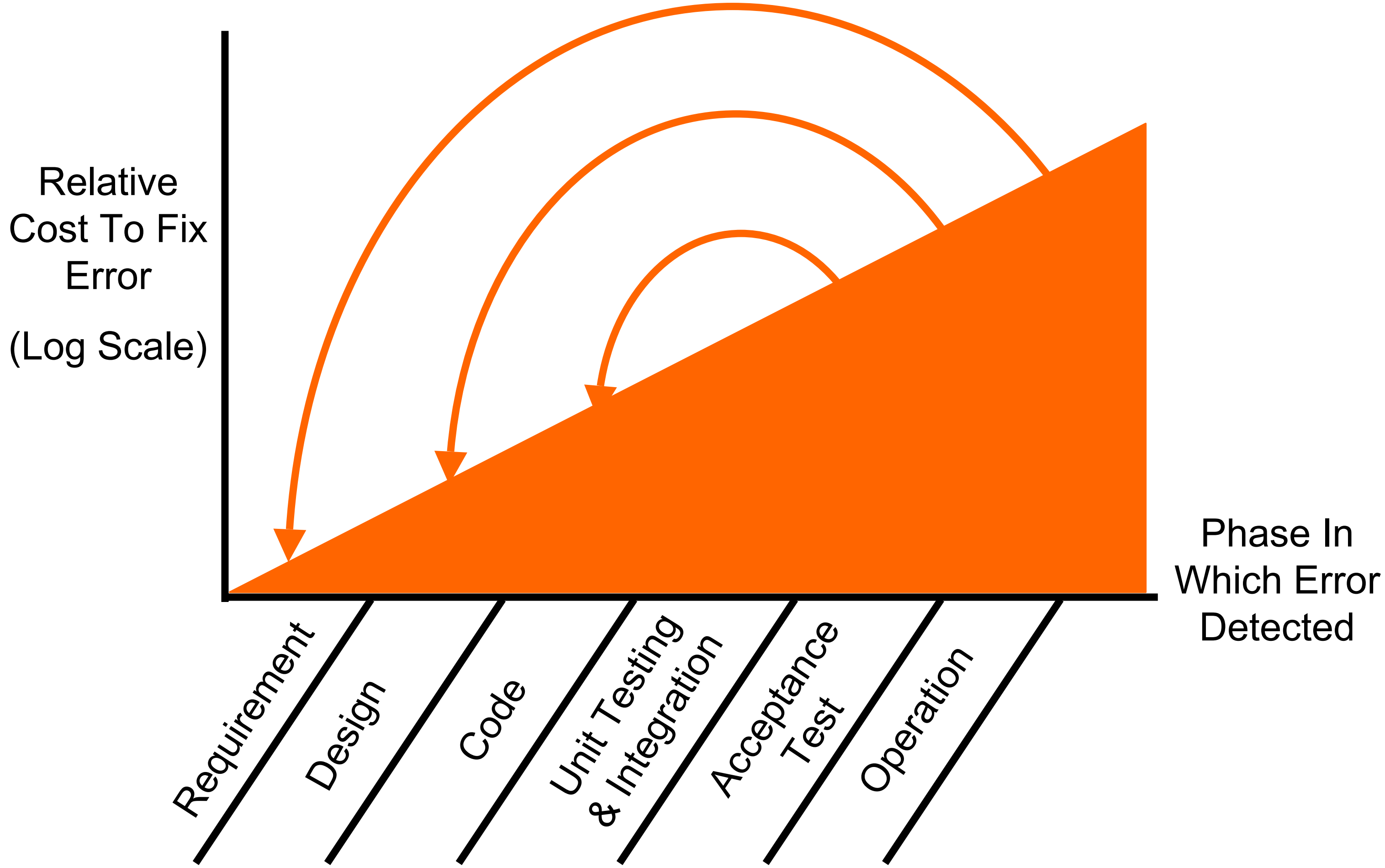
Many factors can cause software failure, however, there are some general causes, including:

- Undetected bugs!
- Co-evolution of software
- Costs factors
- Risk factors

Greater complexity= greater changes = potential errors!

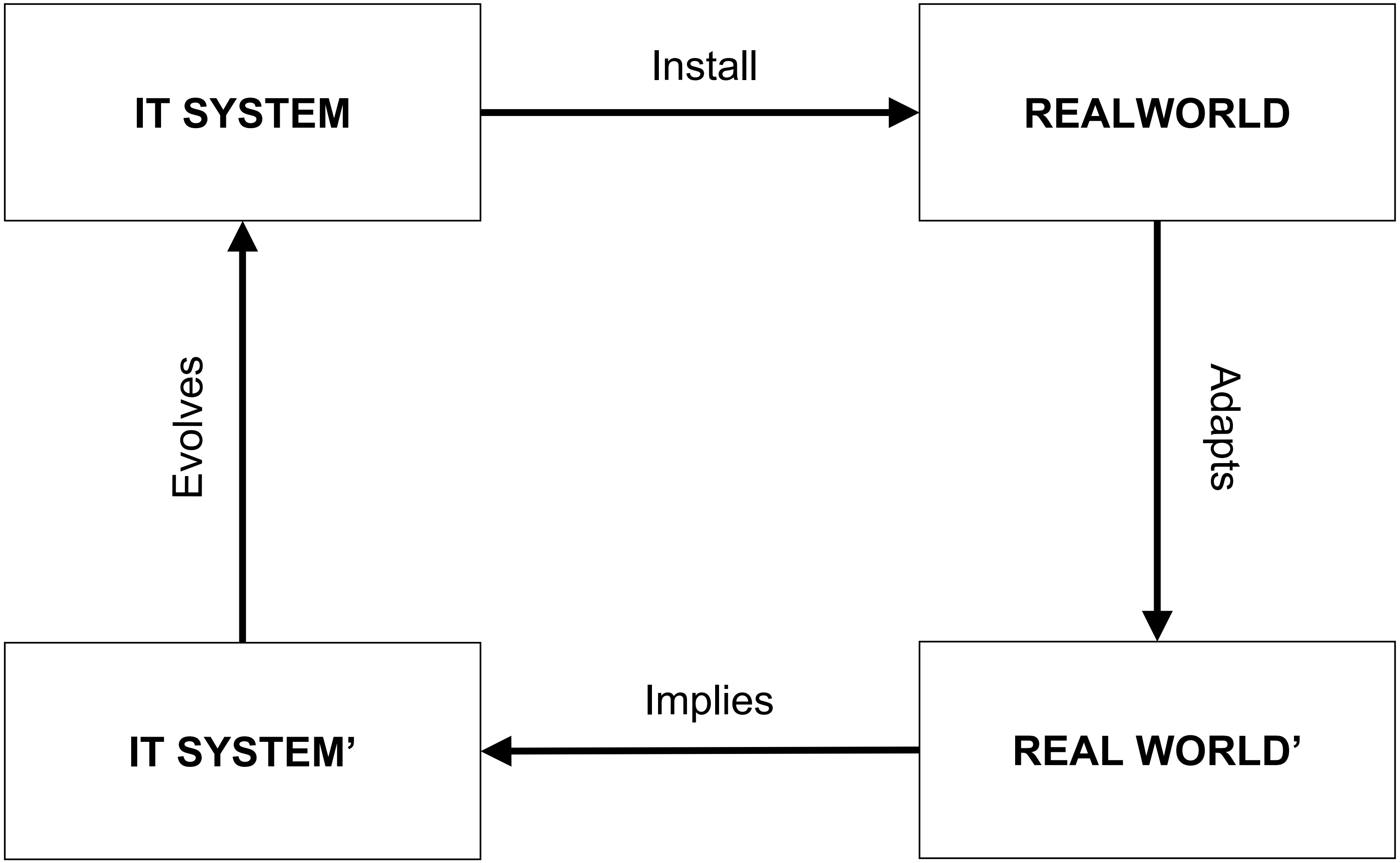
Causes:

Cost (



Relative
Cost To Fix
Error
(Log Scale)
Phase In
Which Error
Detected
Requirement
Design
Code
Unit Testing
& Integration
Acceptance
Test
Operation

Causes: System co-evolution- eternal loop



Causes: Costs

***System Development*.**

System Requirements	2
Hardware Requirements	8
Software Requirements	10
Software Design	12
Coding	13
Unit Test	24
Integration Test	13
Documentation	6
System Test	12
TOTAL	**100**

Causes: But total Costs

- **Pre-Delivery**
  - System Development 100
  - Installation 15

- **Post-Delivery - Maintenance**
  - Defect Removal 60
  - Environmental Changes 60
  - Enhancements 180

- **TOTAL 415**

What Do Coders Actually Do?

Reading Code (Code Reviewing)	16%
Job Communications	25%
Personal & Business Calls	9%
Training	6%
Electronic Mail	9%
Surfing The Web	9%
Other	13%
Writing Code	13%

- Initial writing code is 13% of 100/415 of 13% of development.

=>THUS **CODING** IS ONLY 0.004 of TOTAL DEVELOPMENT COST

Risk Factors: DELPHI Study

9.5	Lack of top management commitment to the project.	♣
8	Failure to gain user commitment.	♣
8	Misunderstanding the requirements.	♦
7.5	Lack of adequate user involvement.	♦
7	Failure to manage end user expectation.	♦
6.5	Change of scope of the project.	♦
6.5	Lack of required skills in the development project.	♣
6.5	Lack of frozen requirements.	♦
6	Introduction to new technology.	♠
6	Insufficient staffing.	♣
5	Conflicts between end user departments.	♦

1 = less important
10 = most important

4 organisation factors ♣ 6 requirements ♦ 1 new technology ♠

However …

Important progress:

- Ability to produce more **complex** software has increased
- New technologies have led to **new SE approaches**
- A better understanding of the **activities** involved in software development
- Effective **methods** to specify, design and implement software have been developed
- New **notations** and **tools** have been produced

What is a software process?

Software Process (SP) is a **set of activities** whose goal is the development or evolution of software

Fundamental activities in all software processes are:

- **Specification** - what the system should do and its development constraints
- **Development** - production of the software system (design and implementation)
- **Validation** - checking that the software is what the customer wants
- **Evolution** - changing the software in response to changing demands

What is a Software Process Model?

SPM is a simplified representation of a software process, presented from a specific perspective

- **Examples of process perspectives:**
  Workflow perspective represents inputs, outputs and dependencies
  Data-flow perspective represents data transformation activities
  Role/action perspective represents the roles/activities of the people involved in software process
- **Generic process models**
  - **Waterfall**
  - **Evolutionary development (commonly known as agile)**
  - **Formal transformation**
  - **Reuse-oriented: Integration from reusable components**

What are the costs of software engineering?

Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs

Costs vary depending on the type of system being developed **and the requirements** of system attributes such as performance and system reliability

Distribution of costs depends on the development model that is used

What is CASE ?
(Computer-Aided Software Engineering)

Software systems which are intended to provide automated support for software process activities, such as requirements analysis, system modelling, debugging and testing

Upper-CASE

Tools to support the early process a
requirements and design

Lower-CASE

Tools to support later activities such as programming, debugging and testing

What are the attributes of good software?

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable

- **Maintainability**
  - Software must evolve to meet changing needs
- **Dependability**
  - Software must be trustworthy
- **Efficiency**
  - Software should not make wasteful use of system resources
- **Acceptability and Usability**
  - Software must be acceptable and usable by the users for the purpose it was designed for.

What are the key challenges facing software engineering?

Software engineering in the 22st century faces three key challenges:

- **Legacy systems**
  - Old, valuable systems must be maintained and updated
- **Increasing Diversity and Heterogeneity**
  - Systems are distributed and include a mix of different hardware and software
- **Dependability and Delivery**
  - Having trustworthy software with faster delivery of software product (time-to-market)

Next lecture

Software Processes