



# Introduction to Software Engineering Chapter One - Part Two

CMPT 276

# Objectives

We will complete **Chapter One** of the **textbook** (Sommerville, 10<sup>th</sup> Ed.)

Today

Part 1 Introduction to Software Engineering

Chapter 1: Introduction

Chapter 2: Software processes

Chapter 3: Agile software development

Chapter 4: Requirements engineering

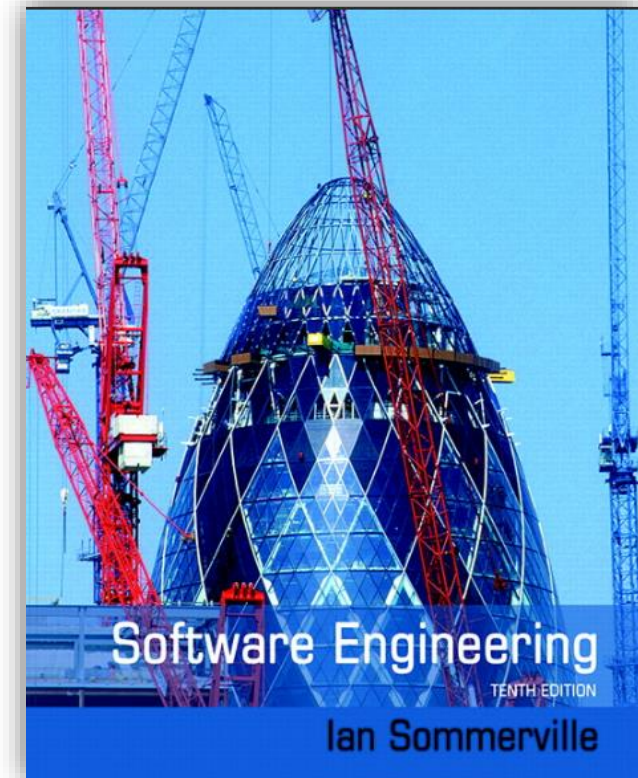
Chapter 5: System modeling

Chapter 6: Architectural design

Chapter 7: Design and Implementation

Chapter 8: Software testing


Chapter 9: Software Evolution





---

# Table of Contents

<b>Part 1</b>	<b>Introduction to Software Engineering</b>	<b>1</b>
<hr/>		
<b>Chapter 1</b>	<b>Introduction</b>	<b>3</b>
	1.1 Professional software development	5
	1.2 Software engineering ethics	14
	1.3 Case studies	17

# Table of Contents

## Part 1 Introduction to Software Engineering

1

### Chapter 1

## Objectives

The objectives of this chapter are to introduce software engineering and to provide a framework for understanding the rest of the book. When you have read this chapter you will:

- understand what software engineering is and why it is important;
- understand that the development of different types of software systems may require different software engineering techniques;
- understand some ethical and professional issues that are important for software engineers;
- have been introduced to three systems, of different types, that will be used as examples throughout the book.

Today



# Class Exercise

Please organize yourselves into your teams and collaborate on the following questions...



# Preliminary Quiz...

1. The typical software development project takes \_\_\_\_\_ months.
2. To be considered a large system, a system today must contain at least \_\_\_\_\_ lines of executable code.
3. What percent of the \$92 billion software market is commercial "shrink-wrap" personal computer products?
4. For a medium-sized software system, how many lines of executable source code are typically produced per day per person (averaged over the entire period of development).
5. The approximate number of errors found in every 1000 lines of executable source code during development of a software system is \_\_\_\_\_.
6. Approximately what percentage of software system that begin development are finally completed?



# Preliminary Quiz...

7. Most errors found by users in software are the result of
  - a. coding errors.
  - b. difficulties understanding the problem statement.
  - c. system integration errors.
  - d. errors in the design of the solution.
8. The cost of owning and maintaining software is typically \_\_\_\_\_ times as expensive as developing the software.
9. A 1979 study by the Government Accounting Office found that for software development contracts:
  - \_\_\_\_\_ % were used as delivered.
  - \_\_\_\_\_ % were used after rework.
  - \_\_\_\_\_ % were abandoned or reworked.
  - \_\_\_\_\_ % delivered but not used.
  - \_\_\_\_\_ % paid for but not delivered.



# Preliminary Quiz...

10. A more recent study (1995) of 365 Information Systems professionals found the statistics on software development project:

- \_\_\_\_ % successful.
- \_\_\_\_ % operational (but less than successful).
- \_\_\_\_ % cancelled.

11. During the 1990's, software maintenance accounted for what percent of the software budget for an information system organization? In 2015, what percent of the Federal government's IT budget was spent on operations and maintenance?

12. A 2002 study by the U.S. Department of Commerce estimates that software defects cost the U.S. economy \_\_\_\_\_ per year.

- a. \$100 million.
- b. \$500 million.
- c. \$1 billion.
- d. \$60 billion.

13. The average project spends about \_\_\_\_\_ percent of its time on unplanned rework; fixing mistakes that were made earlier in the project.

14. A defect introduced in requirements and removed after release costs how much more to repair than if detected during requirements?





# Preliminary Quiz...

**Answers and references:**

[http://users.csc.calpoly.edu/~jdalbey/crisis\\_quiz.html](http://users.csc.calpoly.edu/~jdalbey/crisis_quiz.html)

**CAL POLY**  
SAN LUIS OBISPO

**Department of Computer Science  
and Software Engineering**

*Cal Poly College of Engineering*

# Answers

1. The typical software development project takes **12 to 23 months**.
2. To be considered a large system, a system today must contain at least **50,000** lines of executable code.
3. What percent of the \$92 billion software market is commercial "shrink-wrap" personal computer products? **less than 10%.**
4. For a medium-sized software system, how many lines of executable source code are typically produced per day per person (averaged over the entire period of development). **Less than 10 (1000 LOC/year for procedural coding of relatively small projects)**
5. The approximate number of errors found in every 100 lines of executable source code during development of a software system is **5 to 6**.
6. Approximately what percentage of software system that begin development are finally completed? **70 to 79%.**
7. Most errors found by users in software are the result of **b. difficulties understanding the problem statement (REQUIREMENTS).**
8. The cost of owning and maintaining software is typically **2 times** as expensive as developing the software.
9. A 1979 study by the Government Accounting Office found that for software development contracts:
  - 2 %** were used as delivered.
  - 3 %** were used after rework.
  - 45 %** were abandoned or reworked.
  - 20 %** delivered but not used.
  - 30 %** paid for but not delivered.
10. A more recent study (1995) of 365 Information Systems professionals found the statistics on software development project:
  - 16 %** successful.
  - 53 %** operational (but less than successful).
  - 31 %** cancelled.
11. a) During the 1990's, software maintenance accounted for what percent of the software budget for an information system organization?  
b) Fed govt in 2015? **a) 60% b) 75%**
12. A 2002 study by the U.S. Department of Commerce estimates that software defects cost the U.S. economy \_\_\_\_\_ per year. **d. \$60 billion.**
13. The average project spends about **80** percent of its time on unplanned rework; fixing mistakes that were made earlier in the project.
14. A defect introduced in requirements and removed after release costs how much more to repair than if detected during requirements?  
**50 - 200 times.**

# Software Project Classification

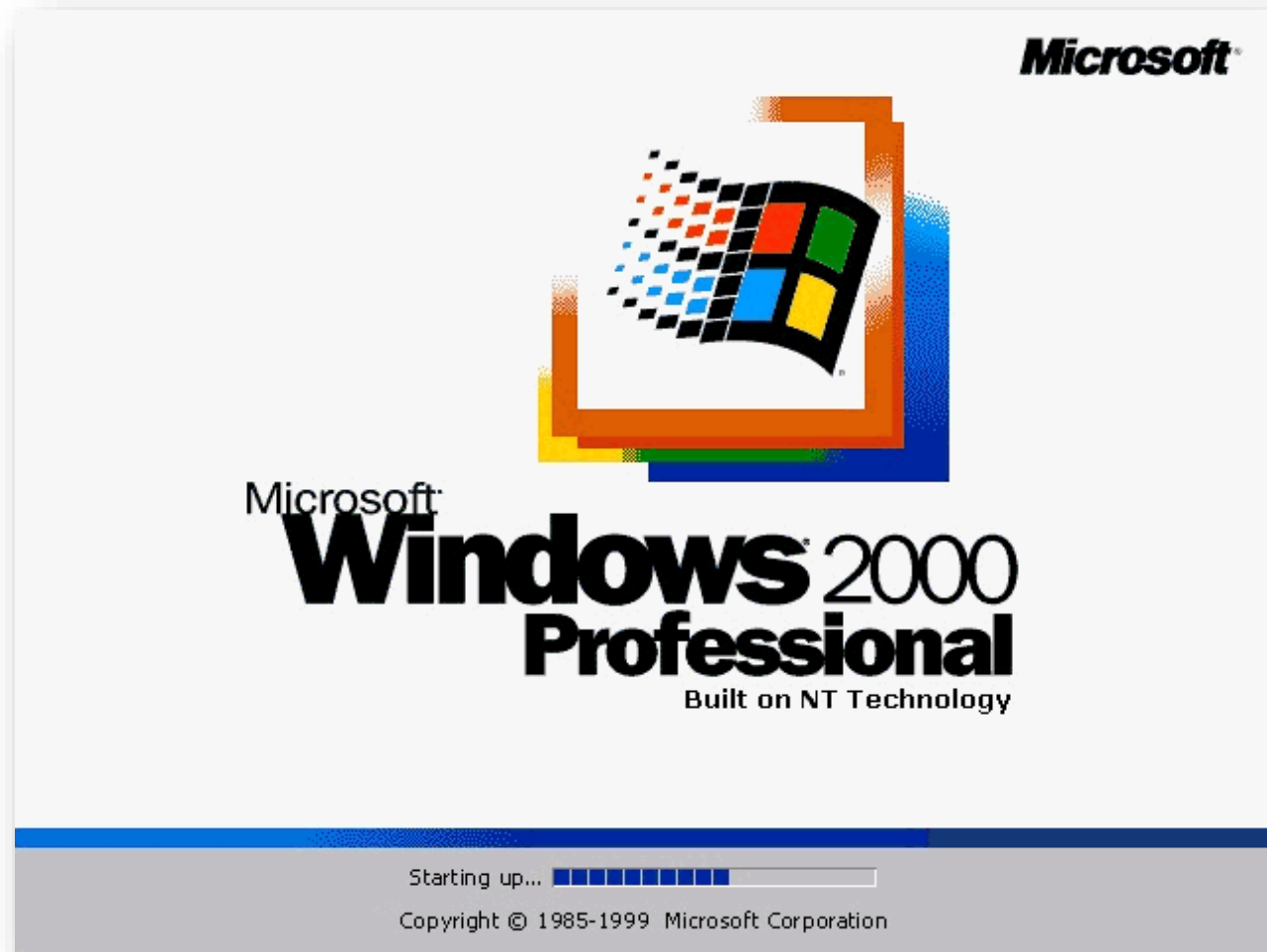
...from

- **Trivial** ( $< 10^2$  lines of code)

...to

- **Titanic** ( $> 10^8$  lines of code)

# Example



SLOC ~ 45 million

# Titanic Projects

Estimate how long it would take for one person to build  
Windows 2000?



# Titanic Projects

Estimate how long it would take for one person to build Windows 2000?

- Total Lines of code  $\sim 10^7$  lines
- Code building rate =  $10^3$  lines/year
- On the order of the age of civilization.  
(the physical limit is on the order of decades)

# Titanic Projects

Estimate how long it would take for one person to build Windows 2000?

- Total Lines of code  $\sim 10^7$  lines
- Code building rate =  $10^3$  lines/year
- On the order of the age of civilization.

(the physical limit is on the order of decades)

This is why large, complex S/W projects require *teams of people* that are specialized and synchronized for a common goal (like building the Port Mann Bridge), which is a highly *nonlinear* process  $\Rightarrow$  **unpredictability**.

# Professional Software Development



© Can Stock Photo - csp15267587

# A Programmer Versus a Software Engineer



**Olayinka S. Folorunso**, Software Engineer

Written Mar 8, 2016

A **programmer** is simply someone who writes computer programs, can also be called a **coder**.

I'm gonna quote directly from wikipedia since software engineer is the more controversial term.

A **software engineer** is a person who applies the principles of [software engineering](#) to the design, development, maintenance, testing, and evaluation of the software and systems that make computers or anything containing software work.

wikipedia.org

**Software engineering** is the study and an application of [engineering](#) to the [design](#), [development](#), [implementation](#) and [maintenance](#) of [software](#) in a systematic method.

# A Programmer Versus a Software Engineer

There is no precise checklist as to what it takes to be a software engineer but I believe that to be an engineer you *must* have a university degree in that field. Is it possible to be an efficient Software Engineer without a university degree? Yes. But in my opinion, the skills listed above can rarely be obtained without a university degree.

*But who am I to assign titles and roles. If your company gives you a Software Engineering title, then I suppose you can call yourself a software engineer.*

To sum up, here is why I call myself a Software Engineer;

- I have a university degree in Computer Engineering, this would imply that I am a Computer Engineer but look at the next point...
- I suck at computer hardwares, I had average grades in my computer hardware courses (microelectronics, computer architecture, embedded systems, ...) and excelled and was actually a lot more interested in the theoretical ones (algorithm design, SE, SDLC, embedded systems, cryptography, computability, AI, ML, ...). So I may have an idea on what is wrong with your computer and how to fix it but please take it away from me, if it is not a software issue.
- I have a job with role/title Software Engineer. If ever my role changes, I would still call myself a Software Engineer.



# A Programmer Versus a Software Engineer



**Antonius Lin**, a father and a friend

Written Aug 22, 2015

The same difference between the title: Janitorial service provider and Cleaning crew.

We can say that there are distinct differences in the skills of those who want to refer to themselves as software engineers and those who call themselves programmers; but all points are arguable considering that there's no single standard of what makes a software engineer "an engineer". In fact legally, we, programmers, aren't supposed to call ourselves using the word "engineer" in our titular rank because it is a "legal word". But for all that we do with software, it makes us feel "good" and valued when we are called software engineers instead of programmers.

Anyone who argues otherwise by saying "but computer science this, versus self-taught that, or algorithm this versus perusing existing frameworks that, and on and on" are really just, in my humble opinion, showing a sense of elitist mind-set where we can joke that actually the difference between a software engineer and a programmer is really just the size of their ego, and not necessarily the skill or output the produce.

[Illegal use of engineer title raises ire of profession - Greensboro - Triad Business Journal](#) 



About PEO

> What is PEO?

> How we Govern Licence and  
Certificate Holders

> How we Protect the Public

> Complaints,  
Investigations

> Discipline

> Disputes Resolution  
and Hearings

> Enforcement

> Licences

Resources

Public / Student

Applicants

Practising Engineers / EITs

Members

Enforcement

- > About
- > PEO and the practice of natural science
- > Misleading certifications
- > Software engineering
- > More information
- > Contact information

Enforcement is legal action taken against individuals or entities who practise engineering without a licence, or offer engineering services while holding a licence that is not in good standing. With only a few exceptions, only people who are holding a licence issued by the PEO can offer or practise engineering services to the public.

Sections 39 and 40 of the *Professional Engineers Act* give PEO the authority to impose maximum fines for individuals and firms practising with an expired licence. These fines are \$25,000 for a first offence and \$50,000 for each subsequent offence. Section 40(2) authorizes fines of up to \$10,000 for a first offence and \$25,000 for subsequent offences, for people who:

- > use the title "professional engineer" or an abbreviation or variation as an occupational designation;
- > use a term, title or description that will lead to the belief that the person may engage in the practice of professional engineering; or
- > use a seal that will lead to the belief that the person is a professional engineer.

PEngs

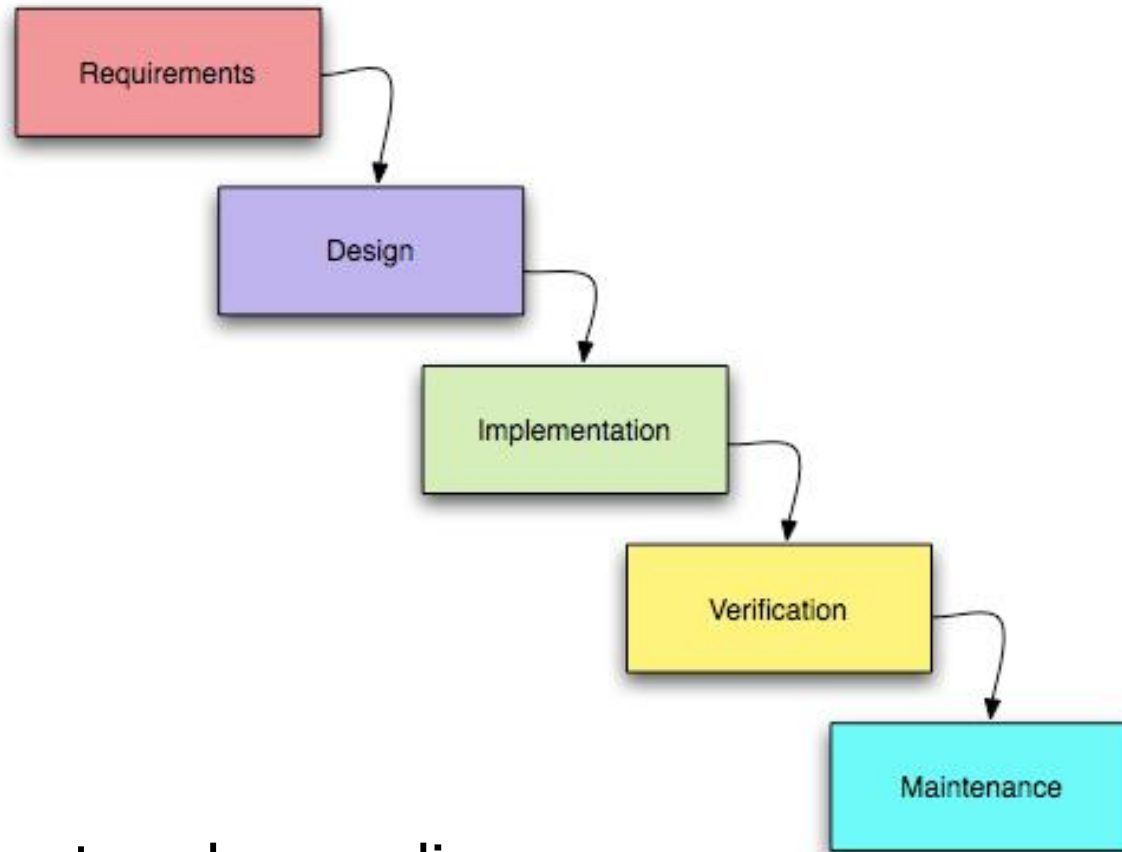
# Software engineering professionalism

---

From Wikipedia, the free encyclopedia

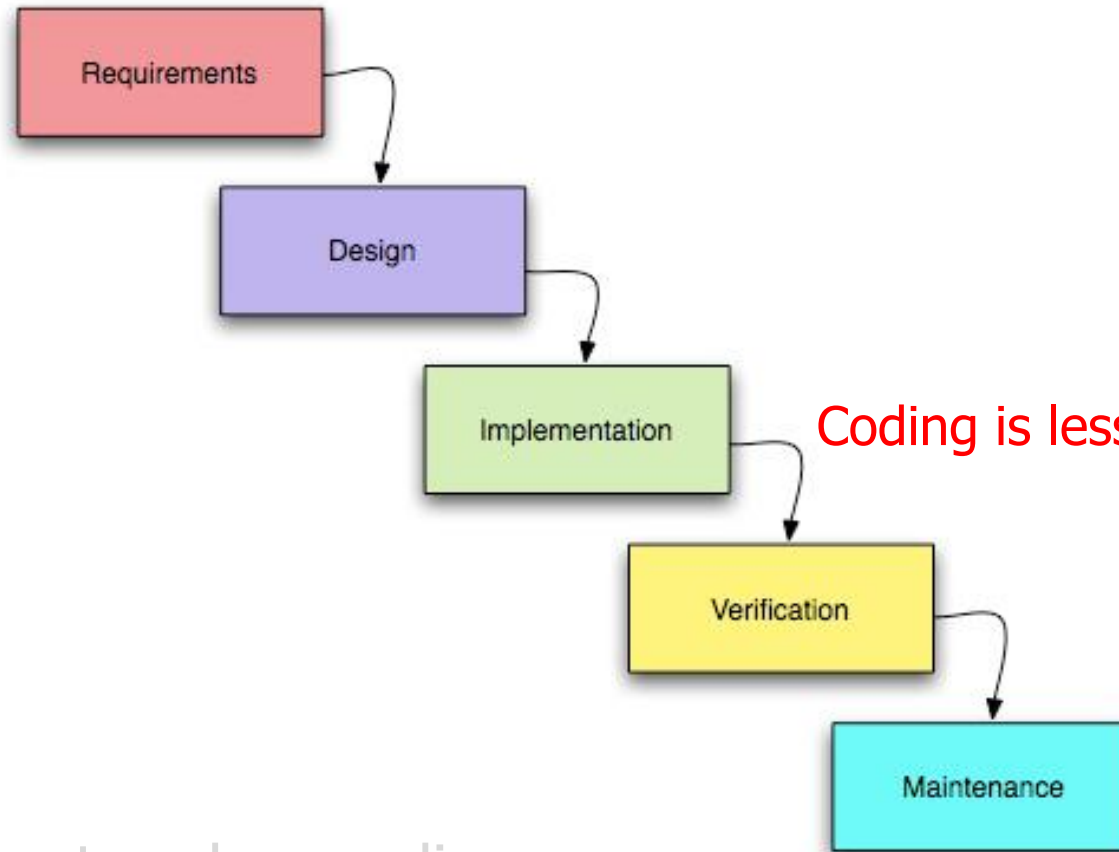
**Software engineering professionalism** is a movement to make **software engineering** a **profession**, with aspects such as degree and certification programs, **professional associations**, **professional ethics**, and government licensing. The field is a licensed discipline in Texas in the United States<sup>[1]</sup> (**Texas Board of Professional Engineers**, since 2013), Engineers Australia<sup>[2]</sup> (Course Accreditation since 2001, not Licensing), and many provinces in Canada.

# A Programmer Versus a Software Engineer



SWE is NOT centered on coding.  
It is a *process* involving distinct phases (next chapter).

# A Programmer Versus a Software Engineer



Coding is less than 10% of the "life cycle".

SWE is NOT centered on coding.  
It is a *process* involving distinct phases (next chapter).



# Software Engineering Job Outlook in Canada

Rank ↕	Job Title ↕	Median Salary ↕	Wage Growth ↕	5-year employee growth ↕	Outlook * ↕
1	Mining or Forestry Manager	\$104,000	19%	44%	↗
2	Urban Planner	\$85,010	15%	30%	↗
3	Pharmacist	\$99,840	9%	38%	↗
4	Pilot or Flying Instructor	\$79,997	28%	5%	↗
5	Public Administration Director	\$101,920	23%	6%	↗
6	Construction Manager	\$79,997	15%	26%	↗
7	Software Engineer	\$88,005	17%	60%	→
8	Oil & Gas Well Operator	\$72,800	17%	47%	↗
9	Financial Manager	\$97,074	21%	17%	→
10	Engineering Manager	\$100,006	10%	17%	→

Canadian Business Top Jobs 2015:

<http://www.canadianbusiness.com/lists-and-rankings/best-jobs/2015-top-100-jobs-in-canada/>

# Software Engineering Job Outlook in Canada

This begs the issue of the quality of people that enter this field – **career driven rather than knowledge driven** as in physics, for example.

Rank ↕	Job Title ↕	Median Salary ↕	Wage Growth ↕	5-year employee growth ↕	Outlook * ↕
1	Mining or Forestry Manager	\$104,000	19%	44%	↗
2	Urban Planner	\$85,010	15%	30%	↗
3	Pharmacist	\$99,840	9%	38%	↗
4	Pilot or Flying Instructor	\$79,997	28%	5%	↗
5	Public Administration Director	\$101,920	23%	6%	↗
6	Construction Manager	\$79,997	15%	26%	↗
7	Software Engineer	\$88,005	17%	60%	→
8	Oil & Gas Well Operator	\$72,800	17%	47%	↗
9	Financial Manager	\$97,074	21%	17%	→
10	Engineering Manager	\$100,006	10%	17%	→

# Software Engineering Job Outlook in Canada

This begs the issue of the quality of people that enter this field – **career driven rather than knowledge driven** as in physics, for example.

This point was eluded to earlier in our discussion of the so-called Software Crisis and the observation *that management (people) failures generally outweigh technical failures.*

Rank ⬆️	Job Title ⬆️	Median Salary ⬆️	Wage Growth ⬆️	5-year employee growth ⬆️	Outlook * ⬆️
1	Mining or Forestry Manager	\$104,000	19%	44%	↗️
2	Urban Planner	\$85,010	15%	30%	↗️
3	Pharmacist	\$99,840	9%	38%	↗️
4	Pilot or Flying Instructor	\$79,997	28%	5%	↗️
5	Public Administration Director	\$101,920	23%	6%	↗️
6	Construction Manager	\$79,997	15%	26%	↗️
7	Software Engineer	\$88,005	17%	60%	➡️
8	Oil & Gas Well Operator	\$72,800	17%	47%	↗️
9	Financial Manager	\$97,074	21%	17%	➡️
10	Engineering Manager	\$100,006	10%	17%	➡️

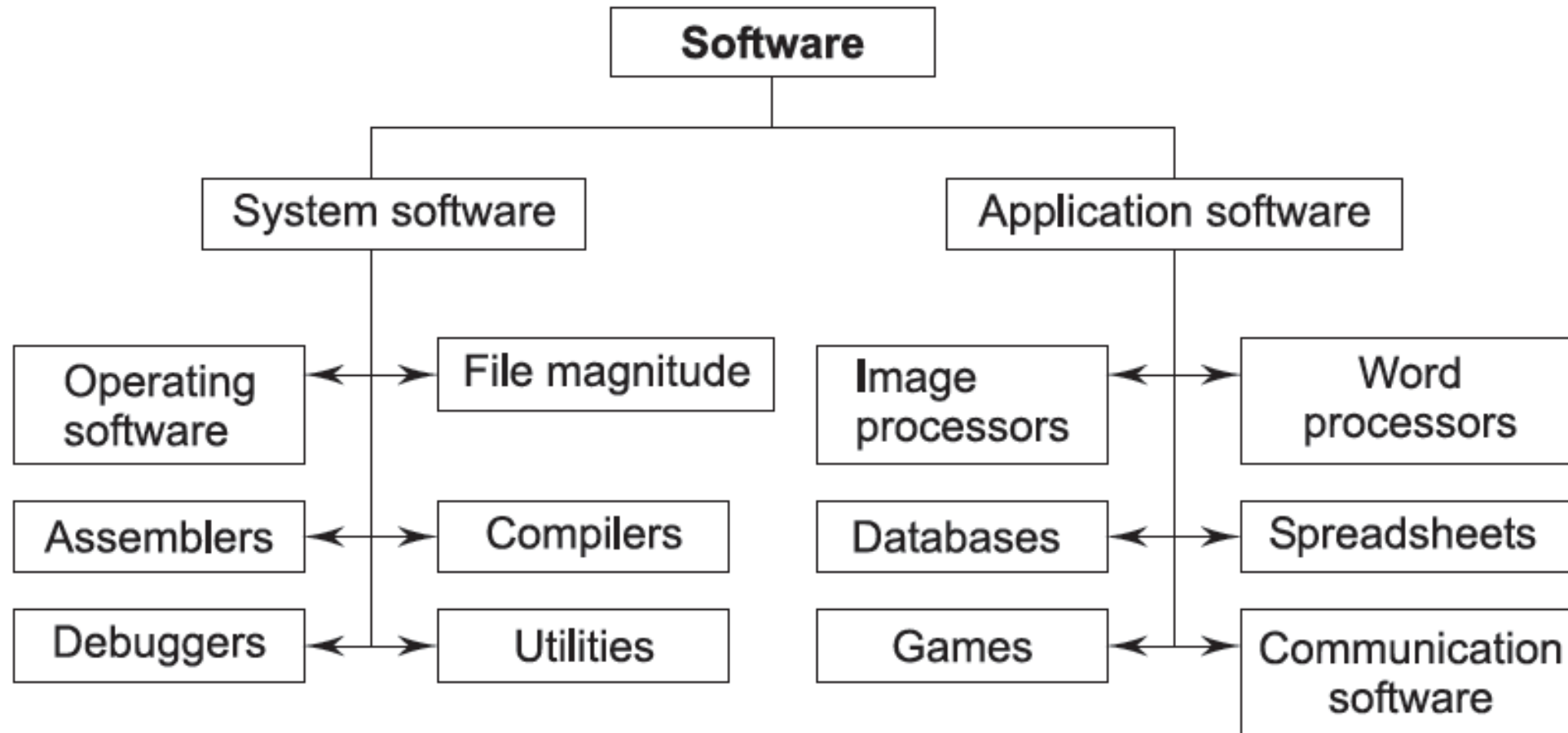
# Scuttlebutt

- It is common to hear upper division undergraduates in CS complain that a significant percentage of their peers “CANNOT PROGRAM!”
- If this is true, it begs the question, “Why are these people here?”
- Probably, career/lifestyle or parental pressure related rather.
- Of course, this further begs the question of the university’s admittance criteria.

# Characteristics/Classification of Software



# Classification of Software



# Classification of Software

Software can be applied in countless fields such as business, education, social sector, and other fields. It is designed to suit some specific goals such as data processing, [information](#) sharing, communication, and so on. It is classified according to the range of potential of applications. These classifications are listed below.

- **System software:** This class of software manages and controls the internal operations of a computer system. It is a group of programs, which is responsible for using computer resources efficiently and effectively. For example, an [operating system](#) is a system software, which controls the hardware, manages [memory](#) and multitasking functions, and acts as an interface between application programs and the computer.
- **Real-time software:** This class of software observes, analyzes, and controls real world events as they occur. Generally, a real-time system guarantees a response to an external event within a specified period of time. An example of real-time software is the software used for weather forecasting that collects and processes parameters like temperature and humidity from the external environment to forecast the weather. Most of the defence organizations all over the world use real-time software to control their military hardware.
- **Business software:** This class of software is widely used in areas where management and control of financial activities is of utmost importance. The fundamental component of a business system comprises payroll, inventory, and accounting software that permit the user to access relevant data from the database. These activities are usually performed with the help of specialized business software that facilitates efficient framework in business operations and in management decisions.

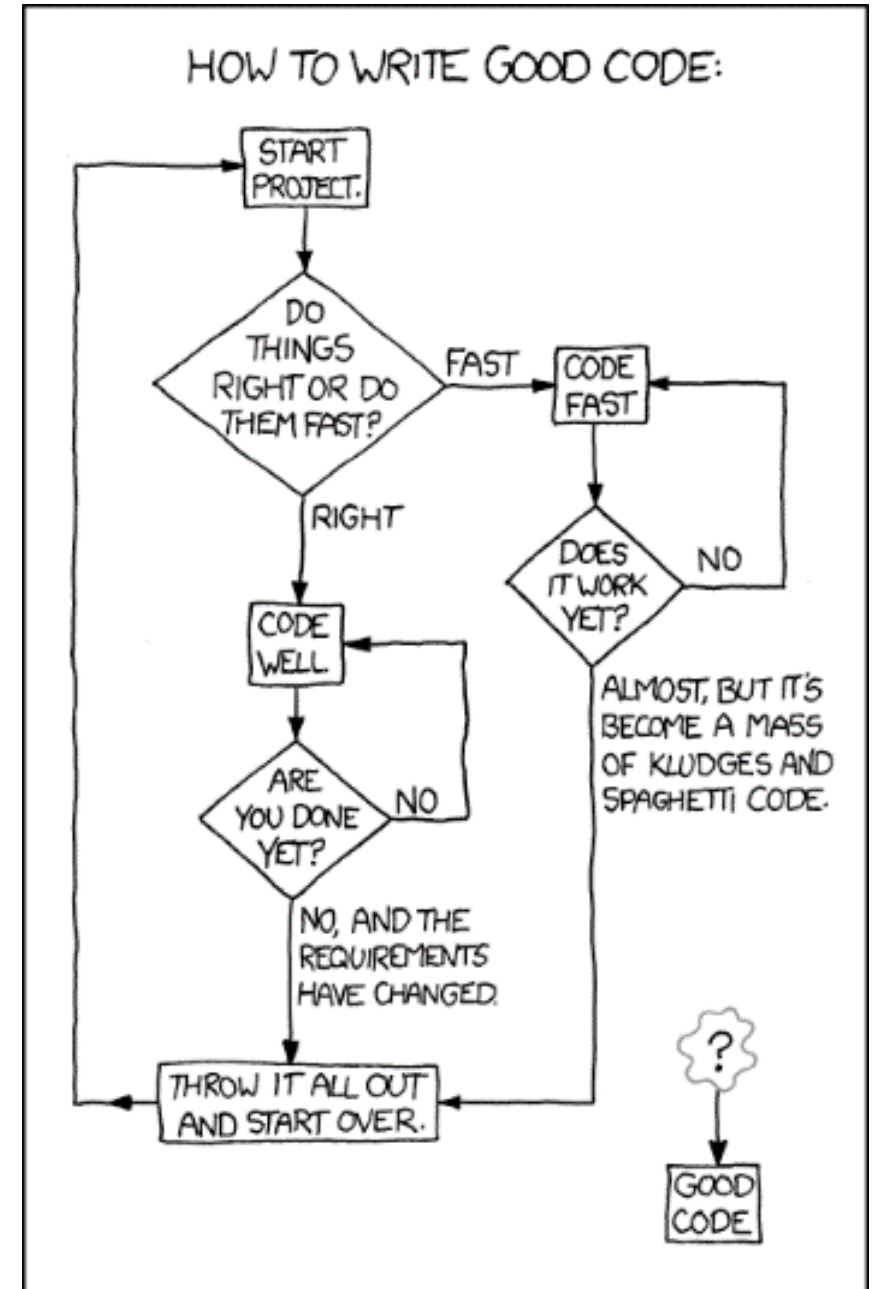
# Classification of Software

- **Engineering and scientific software:** This class of software has emerged as a powerful tool in the research and development of next generation technology. Applications such as the study of celestial bodies, under-surface activities, and programming of an orbital path for space shuttles are heavily dependent on engineering and scientific software. This software is designed to perform precise calculations on complex numerical data that are obtained during real time environment.
- **Artificial intelligence (AI) software:** This class of software is used where the problem-solving technique is non-algorithmic in nature. The solutions of such problems are generally non-agreeable to computation or straightforward analysis. Instead, these problems require specific problem-solving strategies that include expert system, pattern recognition, and game-playing techniques. In addition, they involve different kinds of search techniques which include the use of heuristics. The role of artificial intelligence software is to add certain degrees of intelligence to the mechanical hardware in order to get the desired work done in an agile manner.
- **Web-based software:** This class of software acts as an interface between the user and the Internet. Data on the Internet is in the form of text, audio, or video format, linked with hyperlinks. Web browser is a software that retrieves web pages from the Internet. The software incorporates executable instructions written in special scripting languages such as CGI or ASP. Apart from providing navigation on the Web, this software also supports additional features that are useful while surfing the Internet.
- **Personal computer (PC) software:** This class of software is used for both official and personal use. The personal computer software market has grown over in the last two decades from normal text editor to word processor and from simple paintbrush to advanced image-editing software. This software is used predominantly in almost every field, whether it is database management system, financial accounting package, or multimedia-based software. It has emerged as a versatile tool for routine applications.

# Definition: Legacy code

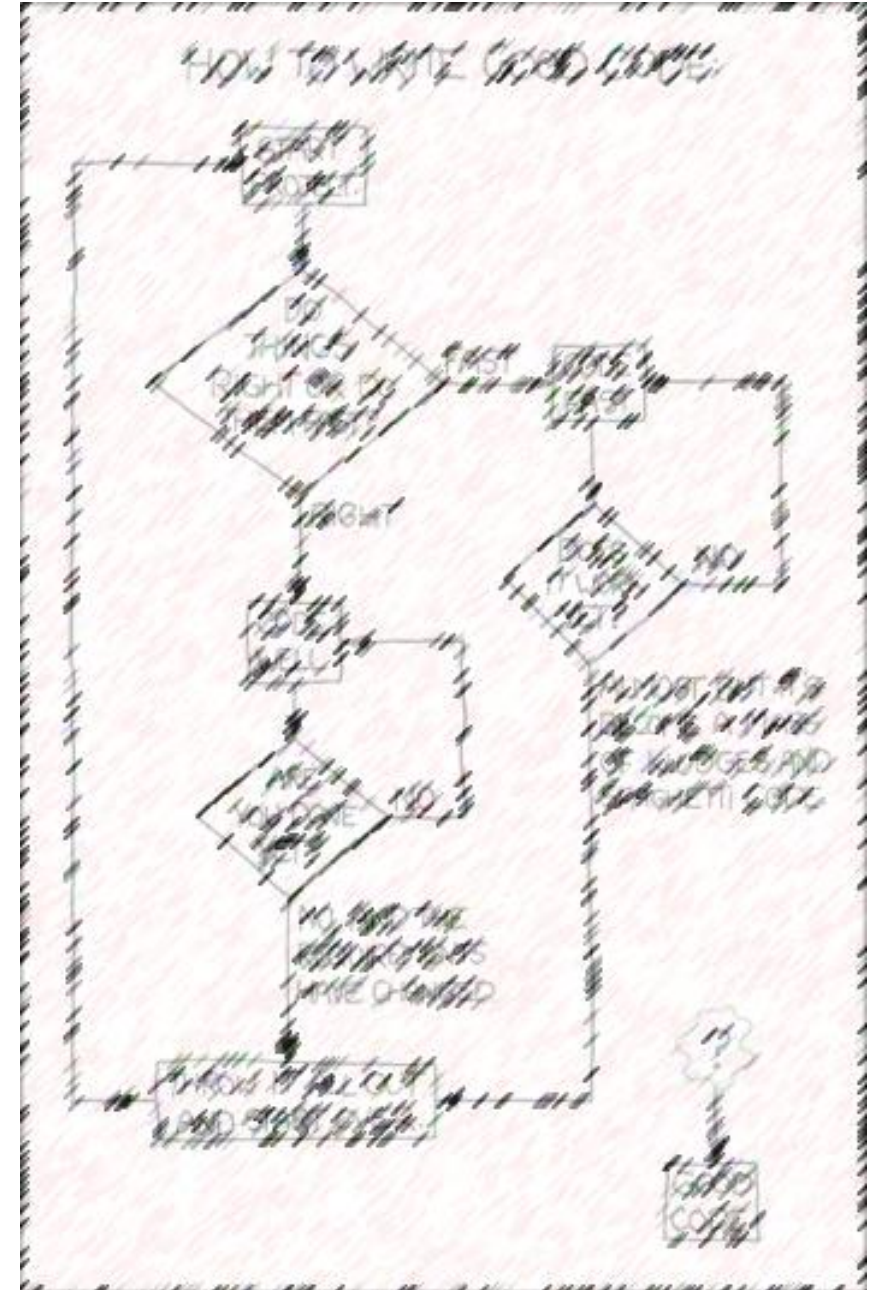
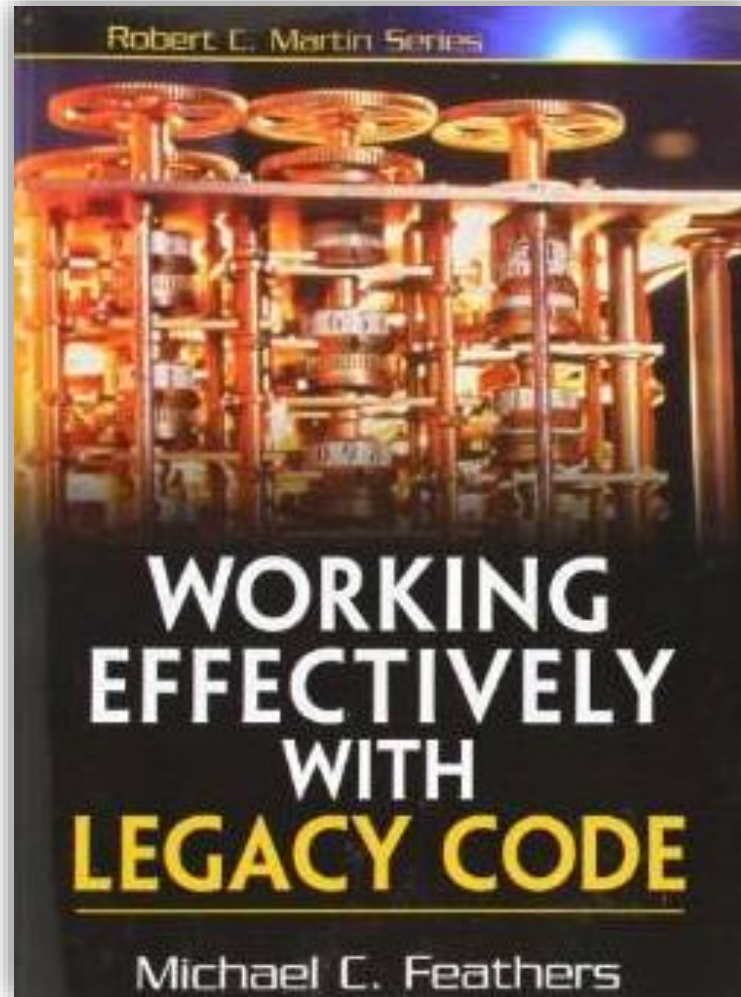
Source code that relates to a no longer supported computer technology or software (source: Wikipedia).

- Many jobs in CS involve Legacy Code.
- This was the prime problem for Y2K.





# Definition: Legacy code



# Software Products

- Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

# Product Specification

- Generic products
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.
- Customized products
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.



# What Constitutes Good Software?



# Definition: Elegant Code

Good code should be clean, simple and easy to understand first of all. The simpler and cleaner it is, the less the chance of bugs slipping in. As Saint-Exupery coined,

*"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."*

..  
- *SWE stackexchange*

∴

.



<http://softwareengineering.stackexchange.com/>

# Elegant Code



**Benedict Raymond Gershom**, Thinker. Over-thinker.

Written Mar 31, 2016

These 11 lines of code almost broke the internet

```
module.exports = leftpad;
function leftpad (str, len, ch) {
  str = String(str);
  var i = -1;
  if (!ch && ch !== 0) ch = ' ';
  len = len - str.length;
  while (++i < len) {
    str = ch + str;
  }
  return str;
}
```

**Quora**

<https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/>

# Elegant Code

## Stuxnet Malware Analysis Paper



AmrThabet, 9 Sep 2011

CPOL

★★★★★ 4.94 (44 votes)

HookSomeAPIs

```
proc near                                ; CODE XREF: StartAddress+38↓p
push edi
push offset dword_1000617C
push offset FindFirstW_Hooker
push offset aKernel32_dll ; "KERNEL32.DLL"
mov edi, offset aFindfirstfilew ; "FindFirstFileW"
call HookAPI
push offset dword_10006180
push offset FindNext_Hooker
push offset aKernel32_dll ; "KERNEL32.DLL"
mov edi, offset aFindnextfilew ; "FindNextFileW"
call HookAPI
push offset dword_10006184
push offset FindFirstExW_Hooker
push offset aKernel32_dll ; "KERNEL32.DLL"
mov edi, offset aFindfirstfilee ; "FindFirstFileExW"
call HookAPI
push offset dword_10006178
push offset QueryDirectory_Hooker
push offset aNtdll_dll_0 ; "NTDLL.DLL"
mov edi, offset aNtquerydirecto ; "NtQueryDirectoryFile"
call HookAPI
push offset dword_10006178
push offset QueryDirectory_Hooker
push offset aNtdll_dll_0 ; "NTDLL.DLL"
mov edi, offset aZwquerydirecto ; "ZwQueryDirectoryFile"
call HookAPI
pop edi
jmp sub_10001790
endp
```

HookSomeAPIs





# Essential Attributes of Good Software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

# Another View on Attributes 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:

- Operational
- Transitional
- Maintenance



# Another View on Attributes of Good Software

## Operational

This tells us how well the software works in operations. It can be measured on:

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





# Another View on Attributes of Good Software

## Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability



# Another View on Attributes of Good Software

## Maintenance

This aspect briefs about how well the software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability



# ISO9126 – Software Quality Characteristics

**The ISO 9126-1 software quality model identifies 6 main quality characteristics, namely:**

- Functionality.
- Reliability.
- Usability.
- Efficiency.
- Maintainability.
- Portability.

ISO9126 - Software Quality Characteristics

[www.sqa.net/iso9126.html](http://www.sqa.net/iso9126.html)

# General Issues That Affect Software

- Heterogeneity
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- Business and social change
  - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.

# General Issues That Affect Software

- Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.
- Scale
  - Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

# Software Engineering Diversity

- There are many different types of software system and *there is no universal set of software techniques that is applicable to all of these.*
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

# Application Types from Sommerville

- Stand-alone applications
  - These are 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.
- Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.



# Application Types from Sommerville

- Batch processing systems
  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- Entertainment systems
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- Systems for modelling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

# Application Types from Sommerville

- Data collection systems
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- Systems of systems
  - These are systems that are composed of a number of other software systems.

# Software Engineering Fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software (don't keep re-inventing the wheel).

# Web-Based Software Engineering



# Web-Based Software Engineering

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Web services (discussed in Chapter 19) allow application functionality to be accessed over the web.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the 'cloud'.
  - Users do not buy software but pay according to use.

# Web-Based Software Engineering

- Web-based systems are complex distributed systems but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.
- The fundamental ideas of software engineering apply to web-based software in the same way that they apply to other types of software system.

# Web-Based Software Engineering

- Software reuse
  - Software reuse is 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.
- Incremental and agile development
  - Web-based systems should be developed and delivered incrementally. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.

# Web-Based Software Engineering

- Service-oriented systems
  - Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.
- Rich interfaces
  - Interface development technologies such as AJAX and HTML5 have emerged that support the creation of rich interfaces within a web browser.



# Summary

- To understand that the development of different types of S/W Systems requires different SWE techniques.
- Specifically, S/W may be classified beyond just system versus application software, including Legacy Code.
- What constitutes “elegant code” and, for that matter, GOOD CODE? See (ISO9126 – Software Quality Characteristics)
- Web-based SWE (nothing new).

# Presentation Terminated

