FRESNO STATE
Discovery. Diversity. Distinction.

CSCI 150
Intro to Software Engineering

Sep 6., 2018

Shih-Hsi “Alex” Liu

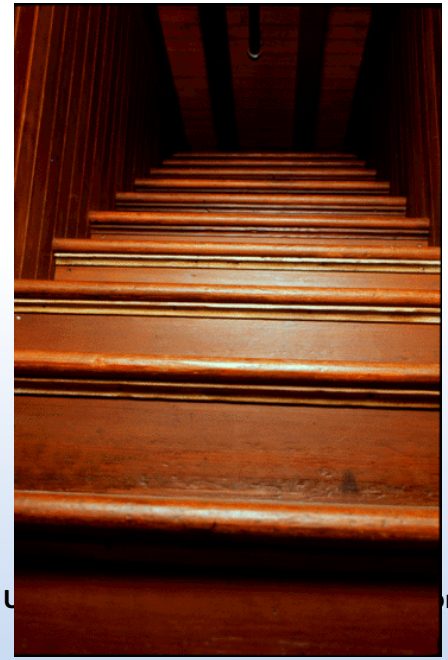
Chapter 1- Introduction

Topics covered

- Professional software development
  - What is meant by software engineering.
- Software engineering ethics
  - A brief introduction to ethical issues that affect software engineering.

Motivation: Poor Engineering leads to ad-hoc structure!

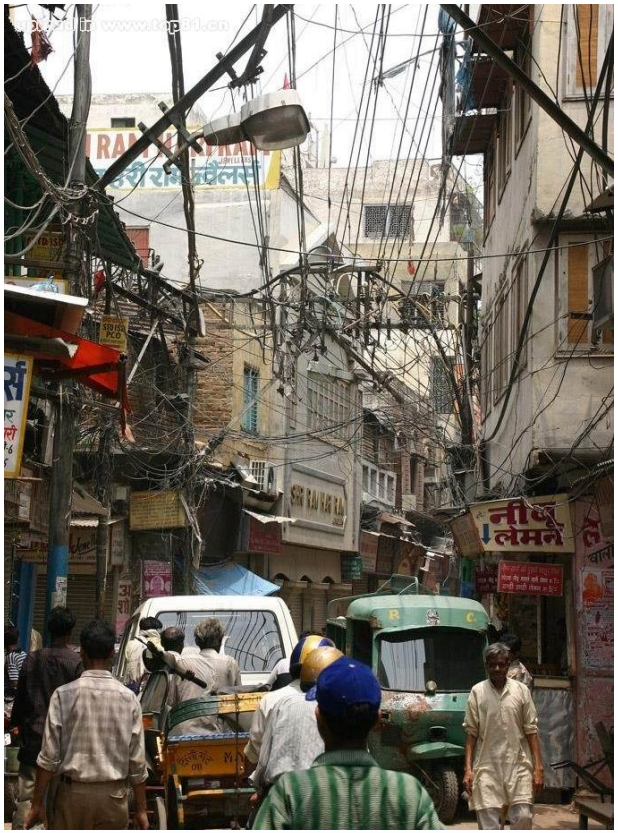
The result of continuous building without any thought toward design.



Result:

Stairs leading to ceiling;
Windows in the middle of room;
Doors opening to wall;
Non-intuitive floor plan!.

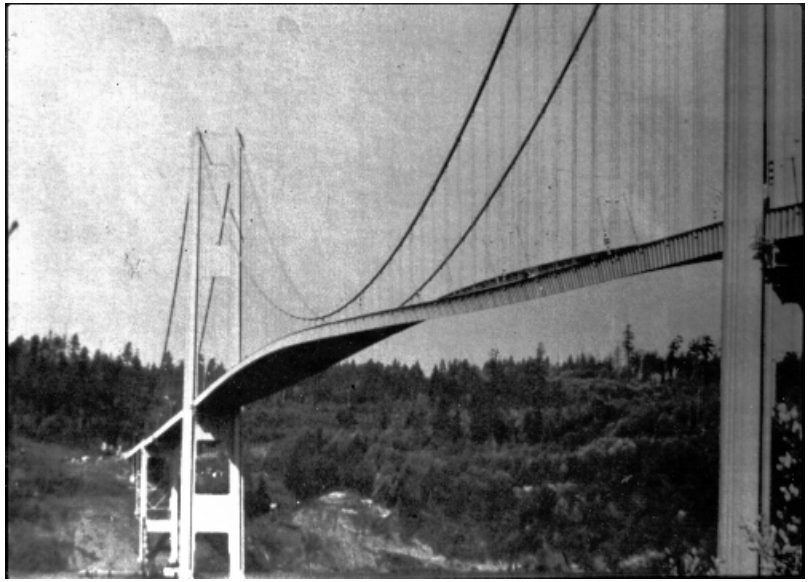
Poor Engineering leads to ad-hoc structure!



The result of continuous building without any thought toward design.
Problems:

- How would you maintain this if something went wrong?
- How would you extend this to add more connections or features?

Poor Engineering Has Disastrous Consequences!

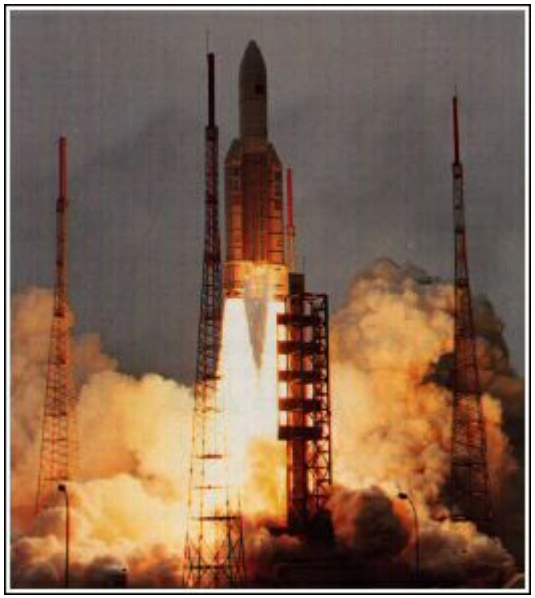


Aerodynamic phenomena in suspension bridges were not adequately understood in the profession nor had they been addressed in this design. New research was necessary to understand and predict these forces.

The remains, located on the bottom of the sound, are a permanent record of man's capacity to build structures without fully understanding the implications of the design.

http://www.nwrain.net/~newtsuit/recoveries/narrows/narrows.htm

Poor SOFTWARE Engineering Has Disastrous Consequences!
$7 Billion Fire Works – One Bug, One Crash



On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift- off). This loss of information was due to specification and design errors in the software of the inertial reference system.

The launcher started to disintegrate at about H0 + 39 seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the
This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcan main engine.
These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the active Inertial Reference System (SRI 2). Part of these data at that ti
The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.
The OBC could not switch to the back-up SRI 1 because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI 2.
The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a
The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the la
The alignment function is operative for 50 seconds after starting of the Flight Mode of the SRIs which occurs at H0 - 3 seconds for Ariane 5. Consequently, when lift-off occurs, the function continue
The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value

The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.

http://java.sun.com/people/jag/Ariane5.html
http://www.around.com/ariane.html
http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html

Poor Engineering Has Disastrous Consequences!

- Software Runaways: Lessons Learned from Massive Software Failures, Robert Glass
  - Denver Airport
  - (http://calleam.com/WTPF/?page_id=2086)
    - Airport idled for 16 months and cost $560M. Ultimately become $1M cost a day. finally determined back to manual system in 2005.
- Safeware: System Safety and Computers, Nancy Leveson
  - Therac-25 (between 1985-1987)
  - http://sunnyday.mit.edu/papers/therac.pdf
- ACM FORUM ON RISKS TO THE PUBLIC IN COMPUTERS AND RELATED SYSTEMS (comp.risks) by Peter G. Neumann, moderator, chmn ACM Committee on Computers and Public Policy
  - http://www.csl.sri.com/users/risko/risks.txt

Poor Engineering Has Disastrous Consequences!

- Virtual case file (VCF)
  - By FBI between 2000-2005
  - Replace Automated Case Support (ACS) software system. ACS had been developed in-house by the bureau and was used to manage all documents relating to cases being investigated by the FBI, enabling agents to search and analyze evidence between different cases.
  - A $170 million dollars lesson
  - https://en.wikipedia.org/wiki/Virtual_Case_File

Poor Engineering Has Disastrous Consequences!

- Virtual case file (VCF): Reasons of failure (continued)
  - Lack of a strong blueprint led to poor architectural decisions.
  - Repeated changes in specification.
  - Repeated turnover of management, which contributed to the specification problem.
  - Micromanagement of software developers.
  - The inclusion of many FBI Personnel who had little or no formal training in Csci as managers and engineers on the project.
  - Scope creep as the requirements were continually added to the system even as it was falling behind schedule.
  - Code bloat due to changing specifications and scope creep. Over 700,000 lines of code.
  - Planned use of a flash cutover deployment, which made it difficult to adopt the system until it was perfected.

Poor Engineering Has Disastrous Consequences!

- Other examples
  - http://www.cs.toronto.edu/~sme/CSC444F/slides/L01-SoftwareFailure.pdf
    - Shuttle Flight 51-L (Challenger) 18
    - The Chinook Helicopter Disaster
      - Bugs in Engine Control Computer https://en.wikipedia.org/wiki/1994_Scotland_RAF_Chinook_crash
- Software Errors Cost in 2017
  - Research by tricentis.com found that in 2017, software failures cost the economy **USD$1.7 trillion in assets**. In total, software failures at 314 companies affected 3.6 billion customers and caused more than 268 years of lost time. https://raygun.com/blog/cost-of-software-errors/
  - *It was 59 billion in 2002, 312 billion in 2012, 1.1 trillion in 2016*

Poor Engineering Has Disastrous Consequences!

- A collection of well known software failures http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html
  - Economic Cost of Software Bugs
  - Microsoft Zune's New Year Crash
  - Air-Traffic Control System in LA Airport
  - Northeast Blackout
  - NASA Mars Climate Orbiter
  - USS Yorktown Incident
  - A List of Security Bugs
  - iCloud? (not sure yet)
  - Knight Capital's $440 million loss (2012)
- https://www.worksoft.com/top-software-failures-of-2017-so-far

Poor Engineering Has Disastrous Consequences!

- Can you think of other examples….?
  - 20 famous software disasters
    - http://www.devtopics.com/20-famous-software-disasters/
  - Top 15 Worst Computer Software Blunders
    - https://www.intertech.com/Blog/15-worst-computer-software-blunders/
  - Software Engineering Disaster Hall of Fame
    - http://www.parseerror.com/bugs/

What Software Can Contribute?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
  - E.g., agriculture, weather systems
    - One of my students collected and analyzed Fresno's weather records for the past decade and use the computational results to control the timing of irrigation system of Fresno State farm.
    - IoT in HML
    - Smart farm

What Software Can Contribute?

- Expenditure on software represents a significant fraction of Gross National Products (GNP) in all developed countries.
- In 2009, global software production reached $985.70 billion, growing at 6.8% annually from 1999-2009.
- The Bureau of Labor Statistics shows that from 2010 to 2020 the total number of jobs in application development software engineer and system analyst positions is expected to increase from 520,800 to 664,500 (27.6%) and from 544,400 to 664,800 (22.1%), respectively

What Software Can Contribute?

- Expenditure on software represents a significant fraction of Gross National Products (GNP) in all developed countries.
- In 2009, global software production reached $985.70 billion, growing at 6.8% annually from 1999-2009.
- The Bureau of Labor Statistics shows that from 2010 to 2020 the total number of jobs in application development **software engineer** and system analyst positions is expected to increase from 520,800 to 664,500 (27.6%) and from 544,400 to 664,800 (22.1%), respectively

What Software Can Contribute?

Summary

Quick Facts: Software Developers	
2016 Median Pay	$102,280 per year $49.17 per hour
Typical Entry-Level Education	Bachelor's degree
Work Experience in a Related Occupation	None
On-the-job Training	None
Number of Jobs, 2014	1,114,000
Job Outlook, 2014-24	17% (Much faster than average)
Employment Change, 2014-24	186,600

- The above figure is from US Dept. of Labor
- https://www.comptia.org/resources/it-industry-outlook-2016-final

Software Engineer Salaries in Fresno, CA Area

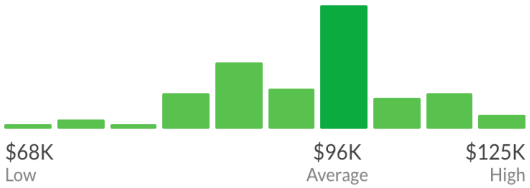
58 Salaries Updated Jul 21, 2018

Industries | Company Sizes | Years of Experience

Average Base Pay

$95,766 /yr

17% below national average



Additional Cash Compensation

Average	$8,016
Range	$1,843 - $20,905

How much does a Software Engineer make in Fresno, CA?
The average salary for a Software Engineer is $95,766 in Fresno, CA. Salaries estimates are based on 58... More

Software Engineer Salaries in Los Angeles, CA Area

8,731 Salaries Updated Sep 4, 2018

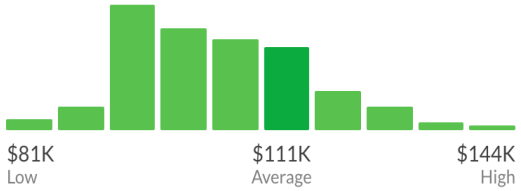
Industries | Company Sizes | Years of Experience

To filter salaries for Software Engineer in Los Angeles, CA Area, Sign In or Register.

Average Base Pay

$110,706 /yr

4% below national average



Additional Cash Compensation

Software Engineer Salaries in San Francisco, CA Area

23,680 Salaries Updated Sep 4, 2018

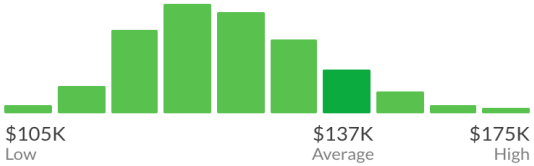
Industries | Company Sizes | Years of Experience

To filter salaries for Software Engineer in San Francisco, CA Area, Sign In or Register.

Average Base Pay

$137,000 /yr

19% above national average



Additional Cash Compensation

Average	$xx,xxx
Range	$xx,xxx

How much does a Software Engineer make in San Francisco, CA?
The average salary for a Software Engineer is $137,000 in San Francisco, CA. Salaries estimates are based on... More

Software Engineer Salaries in Austin, TX Area

4,080 Salaries Updated Sep 3, 2018

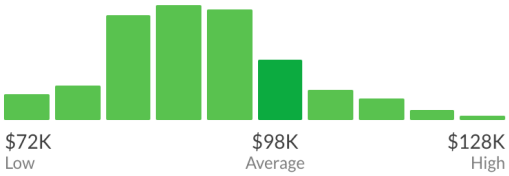
Industries | Company Sizes | Years of Experience

To filter salaries for Software Engineer in Austin, TX Area, Sign In or Register.

Average Base Pay

$97,880 /yr

15% below national average



Additional Cash Compensation

Average	$xx,xxx
Range	$xx,xxx

How much does a Software Engineer make in Austin, TX?
The average salary for a Software Engineer is $97,880 in Austin, TX. Salaries estimates are based on 4,080... More

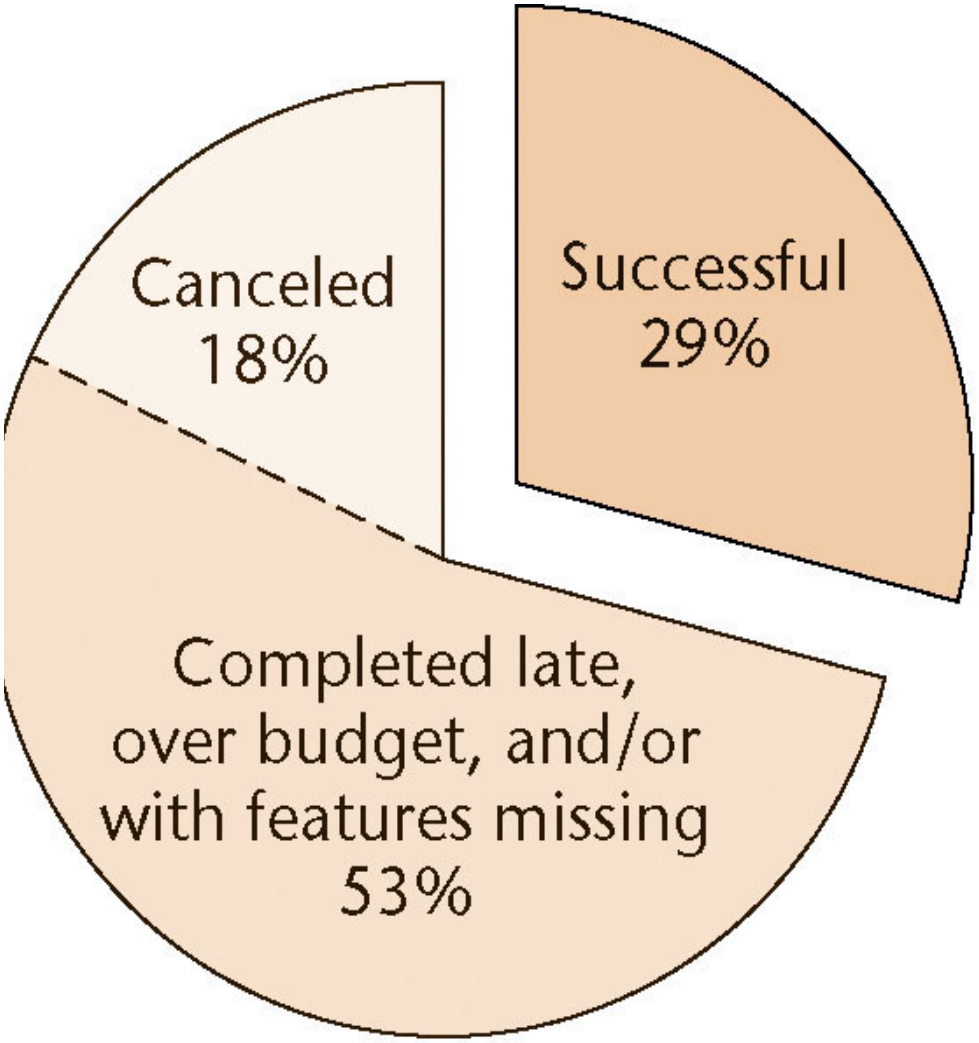
- https://www.stilt.com/blog/2018/03/top-companies-pay-software-engineers/

But…. (Based on Cutter Consortium Data)

- 2002 survey of information technology organizations
  - 78% have been involved in **disputes** ending in litigation
- For the organizations that entered into litigation:
  - In 67% of the disputes, the functionality of the information system as delivered did not meet up to the claims of the developers
  - In 56% of the disputes, the promised delivery date slipped several times
  - In 45% of the disputes, the defects were so severe that the information system was unusable

Standish Group Data

- Data on 9236 projects completed in 2004
- https://blog.capterra.com/surprising-project-management-statistics/
- https://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf



Nature of Software in terms of Costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- **Software costs more to maintain than it does to develop**. For systems with a long life, maintenance costs may be several times development costs.
  - Bad software is discarded
  - Good software is maintained, for 10, 20 years or more
    - E.g., OS X 10.12 (Sierra)
  - Software is a model of reality, which is *constantly changing*
- **Software engineering** is concerned with cost-effective software development.

Time (= Cost) of Postdelivery Maintenance

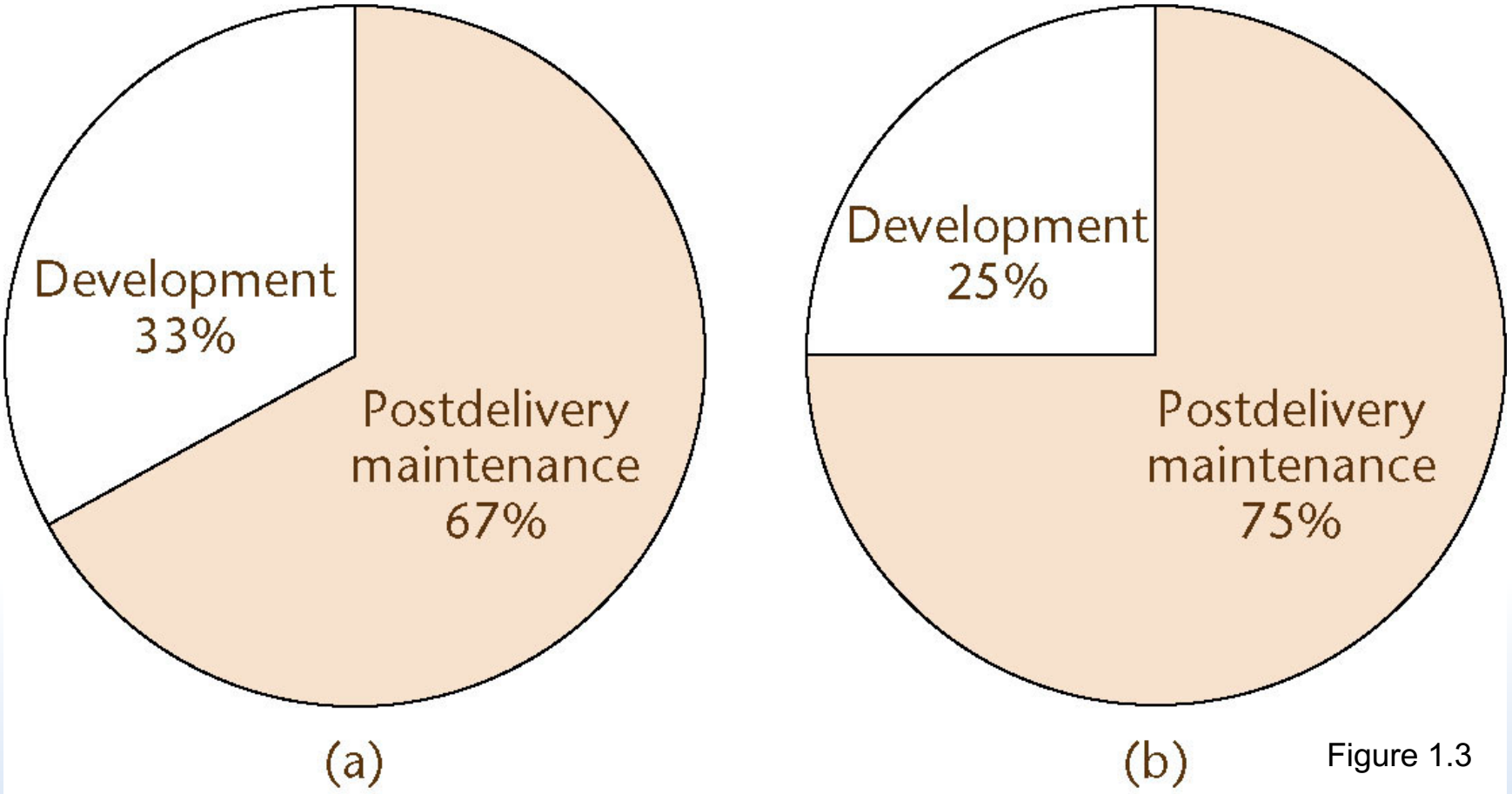


Figure 1.3

(a) Between 1976 and 1981
(b) Between 1992 and 1998

General issues that affect most 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.
- Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Software engineering diversity

- There are many different types of software systems 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**.

Software engineering diversity: Application types

- ✧ 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 (e.g., anti-lock, SW for cell phone). Numerically, there are probably more embedded systems than any other type of system.

Software engineering diversity: Application types

- 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 (e.g., periodic billing systems).
- Entertainment systems
  - These are systems that are primarily for personal use (no longer the case) and which are intended to entertain the user (e.g., Kinect, PS4, Nintendo Switch).
- Systems for modeling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects (e.g., Matlab, CSCI 154).

Software engineering diversity: Application types

- 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 (sensor network).
- Systems of systems
  - These are systems that are composed of a number of other software systems. (e.g., Web service, cloud)

So what can Software Engineering do?

- Much of our software is
  - Delivered late
  - Over budget
  - Residual faults
  - Doesn't meet client's needs
- How did they happen?
  - Increasing demands: As new SE tech help us to build *larger*, more *complex* systems, the demands change. Even *quicker*, much *larger*, and much *more complex* systems are required.
  - Low expectations (or wrong impression): It's easy to write programs **without** using SE methods and tech. Most companies are drifted to SW development.

Why Software Engineering?

1. Software is expanding into all sectors of our society:

- Companies rely on software to run and expand their businesses.
- Software systems are getting larger and more complex – millions of lines of code.
- Software costs are 90 – 95% of total system costs (software costs were only 5 – 10% of total system costs two decades ago).
- Embedded systems contain application specific integrated circuits (ASIC), which are costly to replace – software quality is critical.

We need an engineering approach to software development.

Why Software Engineering?

2. Large software systems development requires teamwork and software engineering supports teamwork.
   - A typical software engineer produces an average 50–100 lines of source code per day.
   - A *small* system of 10,000 lines of code requires one software engineer to work between 100 and 200 days or 5 to 10 months.
   - A medium-size system of 500,000 lines of code requires a software engineer to work 5,000 to 10,000 days or 20 to 40 years.

$\Rightarrow$ Real-world software systems require many software engineers to work together to jointly develop a software system.

Why Software Engineering?

To work together, the software engineers must overcome three challenges, among others:

Conceptualization

Communication

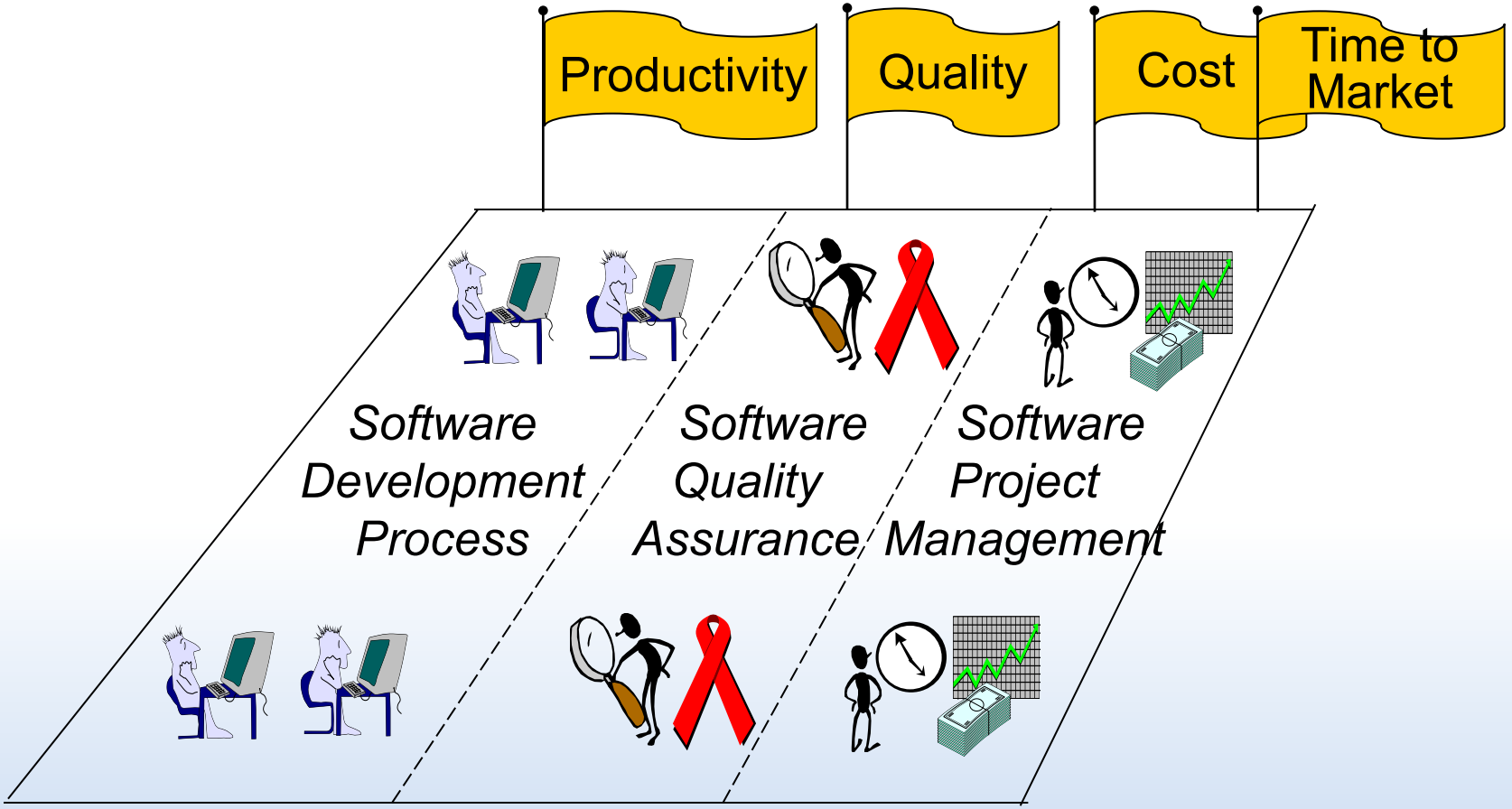
Coordination

Solution:

- Processes and methodologies for analysis and design
- UML for communication and coordination
- Tools that automate or support methodology steps.

Software Life Cycle Activities

- Software processes and methodologies consist of life cycle activities:



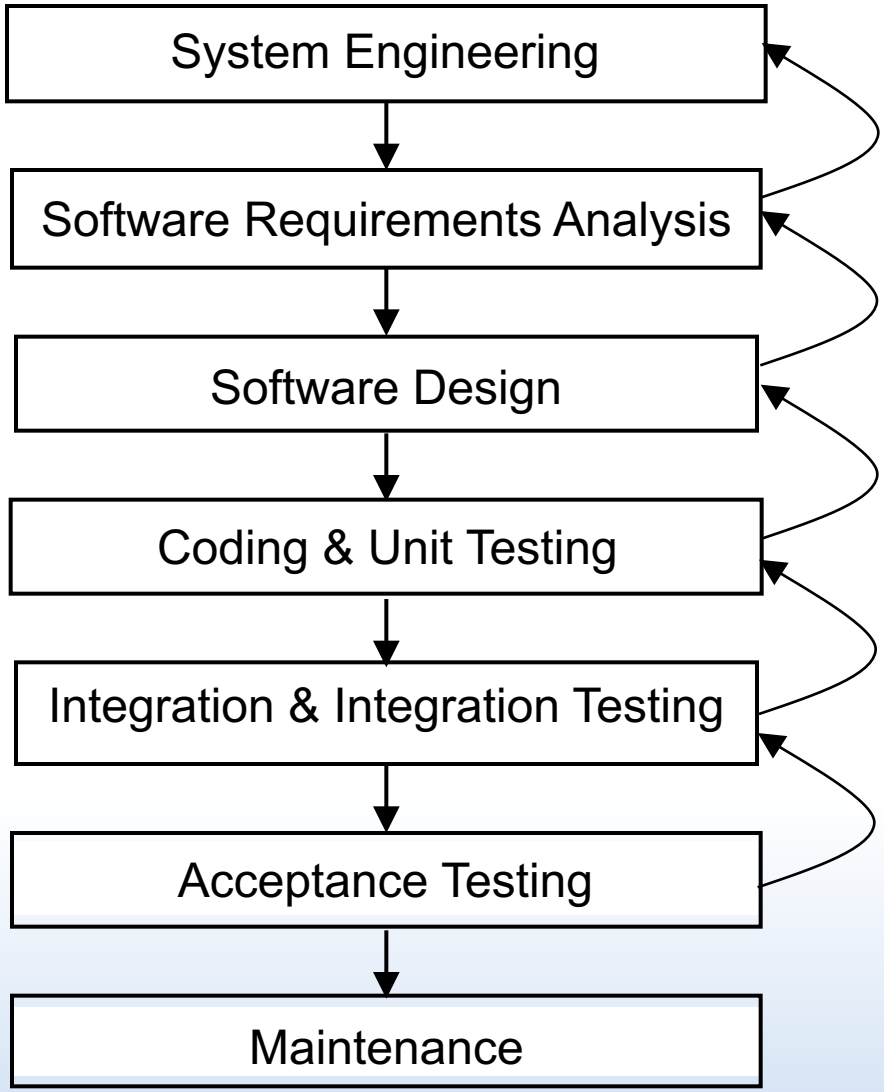
Software Development Process

- A software development process transforms the initial system concept into the operational system running in the target environment.
- It identifies the business needs, conducts a feasibility study, and formulates the requirements or capabilities that the system must deliver.
- It also designs, implements, tests, and deploys the system to the target environment.

Software process activities

- Software process (chapters 2+3) is a sequence of activities that leads to the production of a software product.
- Software specification, where customers and engineers ***define*** the software that is to be produced and the constraints on its operation.
  - Explore the concept, elicit the client's requirements, analyze the client's requirements, draw up the specification document, draw up the software project management plan
  - "What the product is supposed to do"
- Software development, where the software is designed and programmed.
  - Architectural design, detailed design, coding, testing*
  - "How the product does it"

The Waterfall Process (covered in Ch 2)



Documenting a software project

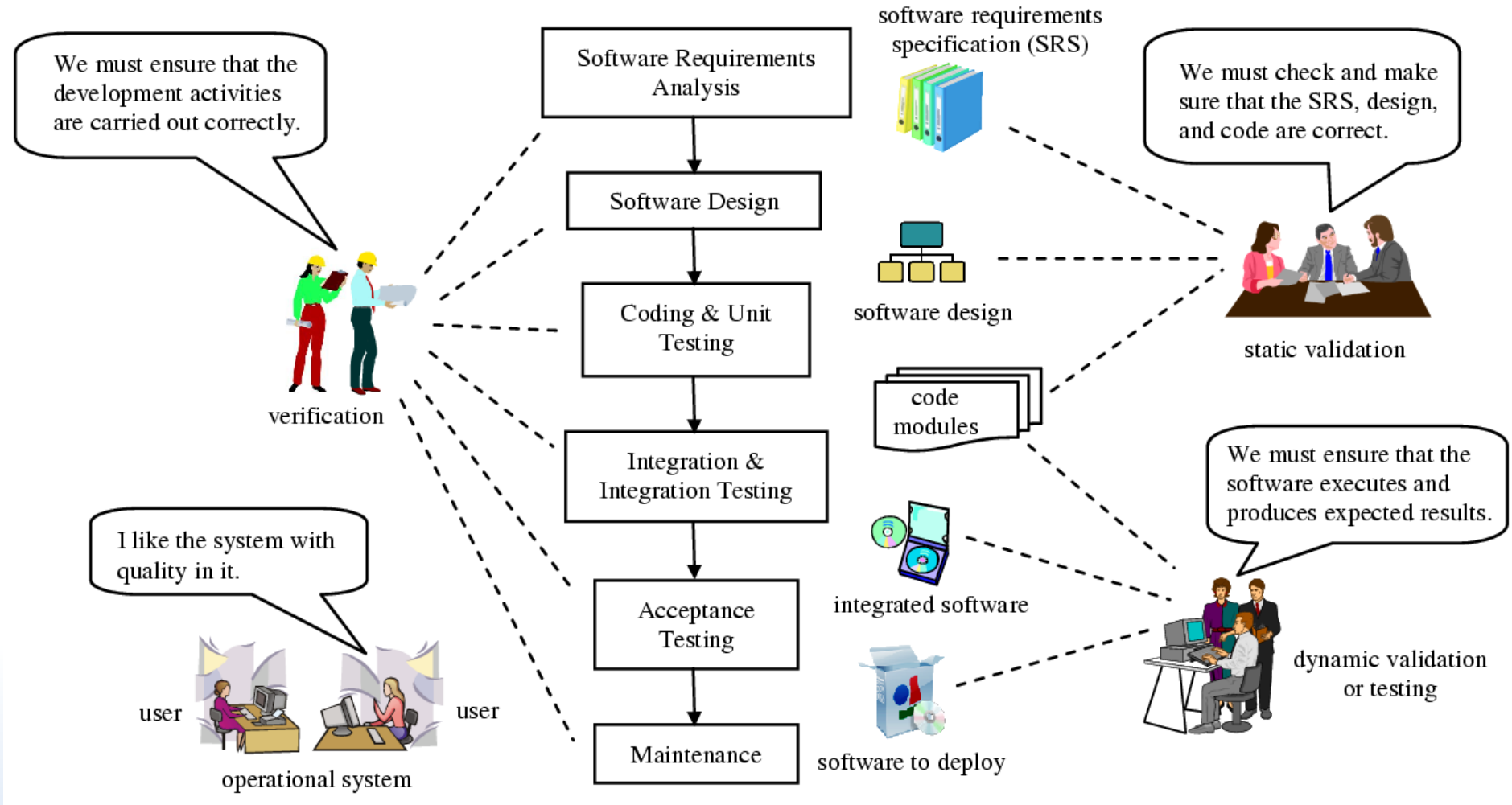
- It is far too late to document after development and before delivery
- Documentation Must Always be Current*
  - Key individuals may leave before the documentation is complete
  - We cannot perform a phase without having the documentation of the previous phase
  - We cannot test without documentation
  - We cannot maintain without documentation
- Documentation activities must be performed in parallel with all other development and maintenance activities. There is no separate documentation phase
- *Test-driven software development

Software Quality Assurance

Software quality assurance (SQA) ensures that

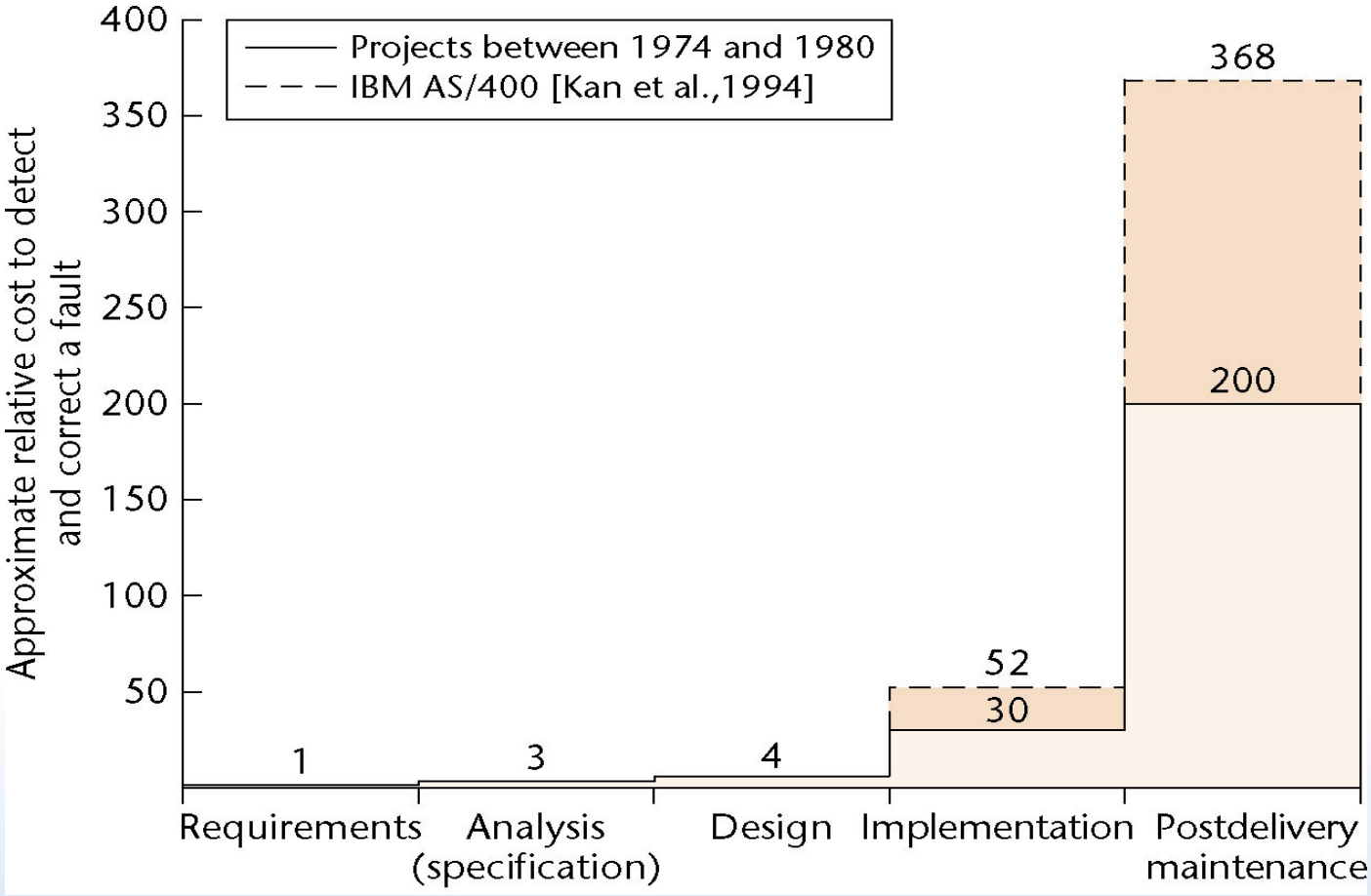
- the development activities are performed properly, and
- the software artifacts produced by the development activities meet the software requirements and desired quality standards.

SQA Activities



"Are you building it right?" vs. "Are you building the right thing?"

Relative Costs to detect and correct a fault



Software Project Management

- Software project management oversees the control and administration of the development and SQA activities.
- Project management activities include
  - effort estimation
  - project planning and scheduling
  - risk management
  - project administration, and
  - others.

These activities ensure that the software system is delivered on time and within budget.

Frequently asked questions about software engineering

Question	Answer
What is software?	Computer programs **and** associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development (design, implementation), software quality assurance, and software evolution (and don't forget management and documenting)
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Frequently asked questions about software engineering

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity (Heterogeneity) and changes, demands for reduced delivery times and developing trustworthy software (security and trust).
What are the costs of software engineering? (maintenance cost not yet included)	Roughly 60% of software costs are development costs, 40% are testing costs (maintenance is not included yet). For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, **different techniques are appropriate for different types of system**. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

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.

Importance of software engineering

- 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.
- It is usually cheaper, in the long run, to use software **engineering methods and techniques** for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the **majority of costs** are the **costs of changing** the software after it has gone into use (i.e., maintenance cost).

Chapter 1- Introduction

- Part 2

Software engineering myths (Management)

- If we get behind schedule, we can add more programmers and catch up (Mongolian horde concept)
  - Actually it will make it **later**
- If I decide to outsource the software to a third party, I can just relax and let the firm build it.
  - If an org. does not understand how to manage and control software projects internally, it will invariably struggle.

Software engineering myths (Customers)

- A general statement of objective is sufficient to begin writing programs – will can fill in the details later.
  - Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is a recipe for disaster.
- Software Req. continually change, but change can be easily accommodated because software is flexible.
  - The impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has started), the cost impact is relatively small. The cost impact grows rapidly.

Software engineering myths (Practitioner)

- Once we write the program and get it to work, our job is done.
  - Industry data indicates that between 60%-80% of all effort expended on software will be expended after it is delivered to customer for the first time.
- Until I get the program running, I have no way of assessing its quality
  - One of the most effective software quality assurance mechanisms can be applied from the inception of a project – technical review, which is more effective than testing.

Software engineering myths (Practitioner)

- The only deliverable work product for a successful project is the working program.
  - A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, docs, plans) provide a foundation for successful engineering and guidance for software support.
- SE will make us create voluminous and unnecessary doc and will invariably slow us down
  - SE is not about creating doc. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Chapter 1- Introduction

- Part 3

Software engineering ethics

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside their competence.
    - What happens if you accept work like that?

Issues of professional responsibility

- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Rationale for the code of ethics

- *Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.*
- *Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.*

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Ethical principles

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Ethical dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.

backup

Software Engineering is to but not limited to

- concerned with theories, methods and tools for **professional software development**.
- Solve the aforementioned problems
- Easy to modify when client's needs change
- Support professional SW TEAM development, rather than individual programming.
- Support program specification, design, implementation, integration, testing, and evolution (maintenance), documentation, configuration, planning and management, etc.
- aims to significantly improve **software productivity** and **software quality** while reducing software costs and time to market.

Software engineering

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- All aspects of software production
  - Not just **technical** process of development. Also project **management** and the development of tools,

Conclusion in short

- The software crisis has not been solved
  - A software engineer requires a broad range of *technical* and *managerial* skills
  - Skills applied to not just *programming* but also from *requirements* to *post-delivery maintenance*

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.
- However, the difference between generic and customized products is getting blurred. (E.g., Enterprise Resource Planning (ERP) systems – SAP.). These products start with generic and then customized based o customers' need.

Software engineering and the web

- 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 software engineering

- 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.
- 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.
- User interfaces are constrained by the capabilities of web browsers.

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, discussed in the previous section, apply to web-based software in the same way that they apply to other types of software system.

Key points

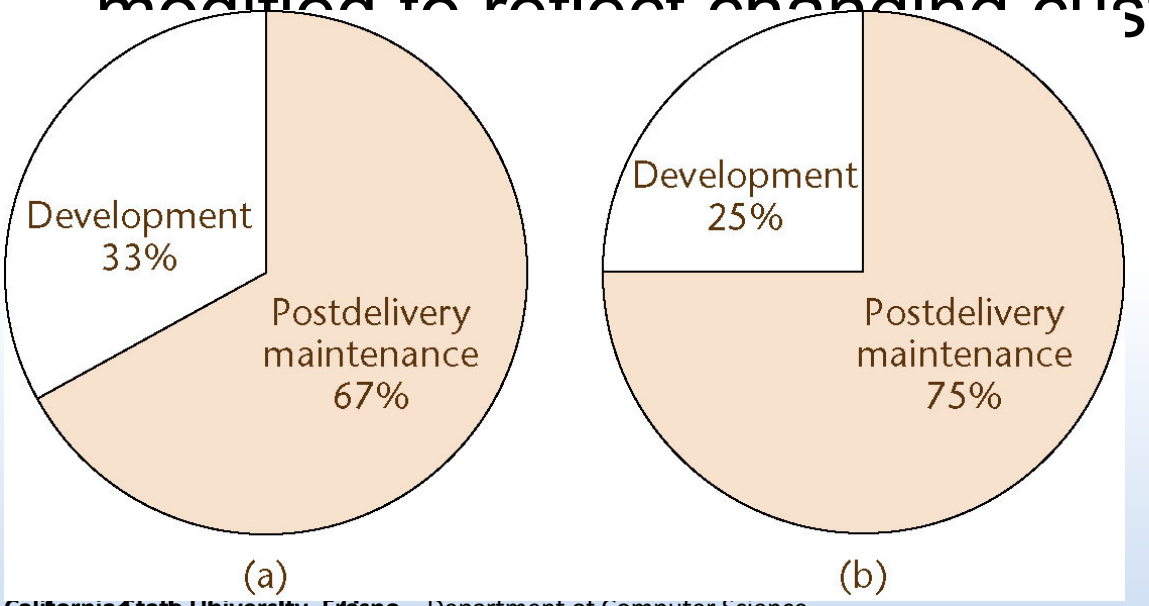
- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Essential software product attributes are maintainability, dependability and security, efficiency and acceptability.
- The high-level activities of specification, development, validation and evolution are part of all software processes.
- The fundamental notions of software engineering are universally applicable to all types of system development

Key points

- There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
- The fundamental ideas of software engineering are applicable to all types of software system.

Software process activities

- Software validation*, where the software is checked to ensure that it is what the customer requires.
- Software evolution, where the software is modified to reflect changing customer and



(a) Between 1976 and 1981
(b) Between 1992 and 1998

Consequence of Relative Costs of Activities/Phases

- Coding method CM_{new} is 10% faster than currently used method CM_{old}. Should it be used?
- Common sense answer
  - Of course!
- **Software Engineering answer**
  - Consider the cost of training
  - Consider the impact of introducing a new technology
  - Consider the effect of CM_{new} on maintenance
- Reducing the coding cost by 10% yields at most a 0.85% (=34%*25%) reduction in total costs (based on the previous pie chart and next table)
  - Consider the expenses and disruption incurred
- Reducing postdelivery maintenance cost by 10% yields a 7.5% reduction in overall costs

The Costs of Different Activities/Phases of Software Development

- Surprisingly, the costs of the classical phases have hardly changed

	Various Projects between 1976 and 1981	132 More Recent Hewlett-Packard Projects
Requirements and analysis (specification) phases	21%	18%
Design phase	18	19
Implementation phase		
Coding (including unit testing)	36	34
Integration	24	29

Relative Costs to detect and correct a fault

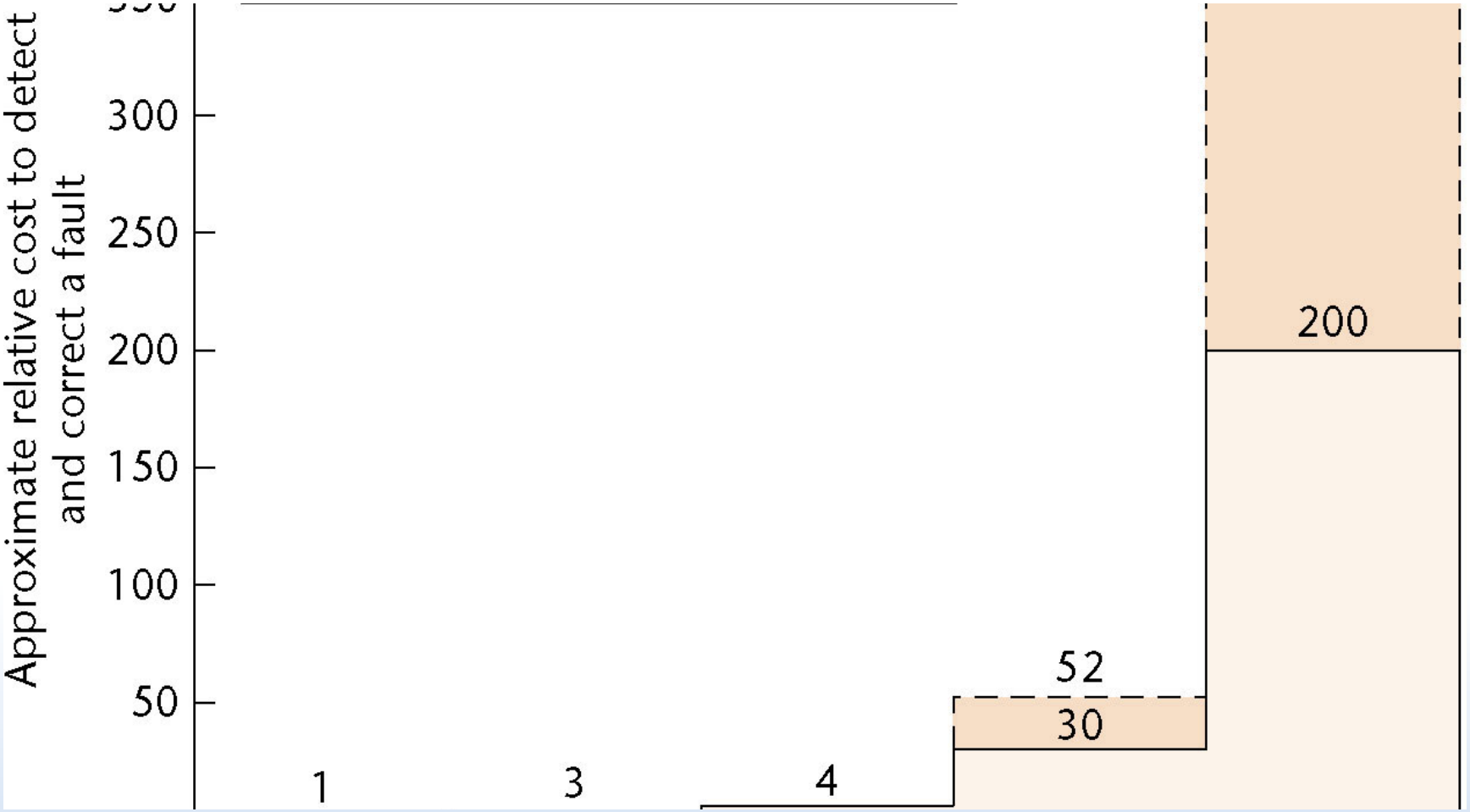


Figure 1.6

Relative Costs to detect and correct a fault

- To correct a fault **early** in the life cycle
  - Usually just a **document** needs to be changed
- To correct a fault late in the life cycle
  - Change the code and the documentation
  - Test the change itself
  - Perform **regression testing (what is this?)**
  - Reinstall the product on the client's computer(s)

Relative Costs to detect and correct a fault

- Between 60 and 70% of all faults in large-scale products are requirements, analysis, and design faults

- Example: Jet Propulsion Laboratory inspections
  - 1.9 faults per page of specifications
  - 0.9 per page of design
  - 0.3 per page of code

Other activities in software process?

- Planning?
- Testing?
- Documenting?

Planning for a software project

- We cannot plan at the beginning of the project —we do not yet know exactly what is to be built
- *Preliminary* planning of the requirements+analysis phases at the start of the project
- The software project management plan is drawn up when the specifications have been signed off by the client (waterfall model)
- Management needs to monitor the SPMP throughout the rest of the project
- Planning activities are carried out **throughout** the life cycle. There is no separate planning

Testing for a software project

- It is far **too late** to test after development and before delivery
- Verification
  - Testing at the end of each phase (too late)
- Validation
  - Testing at the end of the project (far too late)

Testing for a software project

- **Continual** testing activities must be carried out throughout the life cycle
- This testing is the responsibility of
  - Every software professional, and
  - The software quality assurance group
- There is no separate testing phase
- Test-driven software development!

Software engineering fundamentals

- Some fundamental principles apply to all types of software systems, 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

Case studies

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
  - An information system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
  - A data collection system that collects data about weather conditions in remote areas.

Insulin pump control system

- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Figure 1.4 Insulin pump hardware architecture

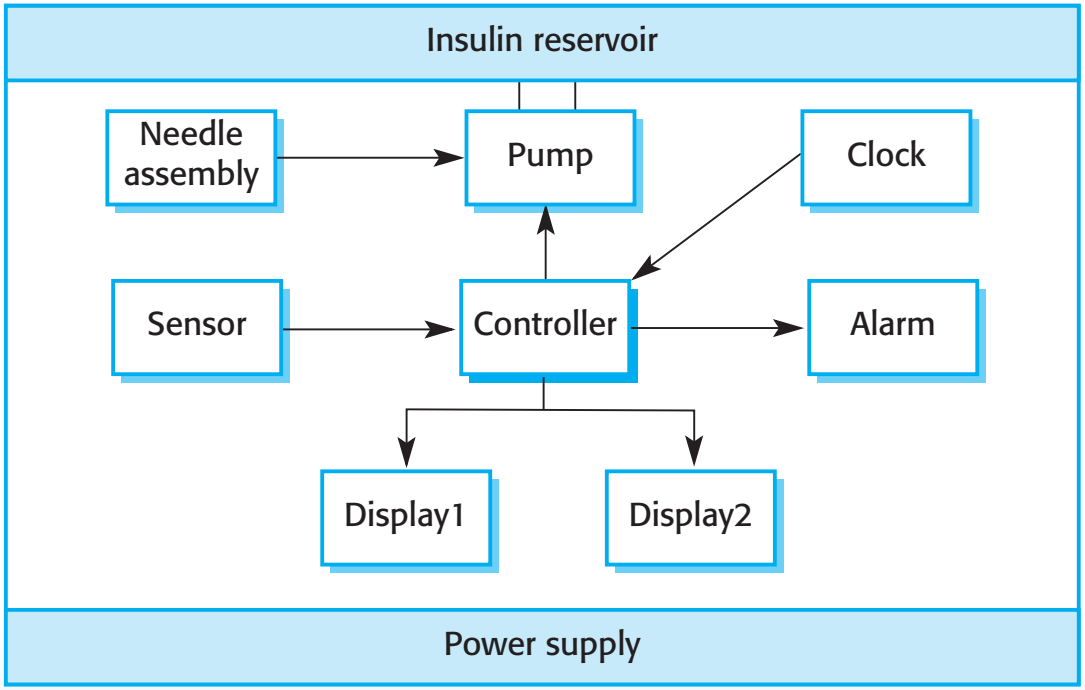
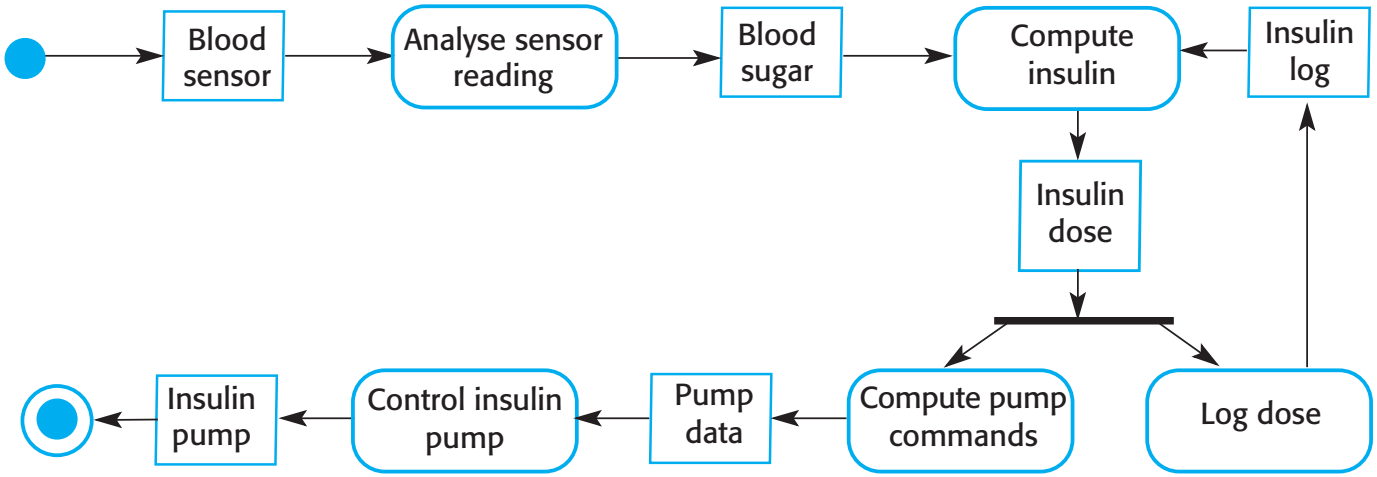


Figure 1.5 Activity model of the insulin pump



Essential high-level requirements

- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may

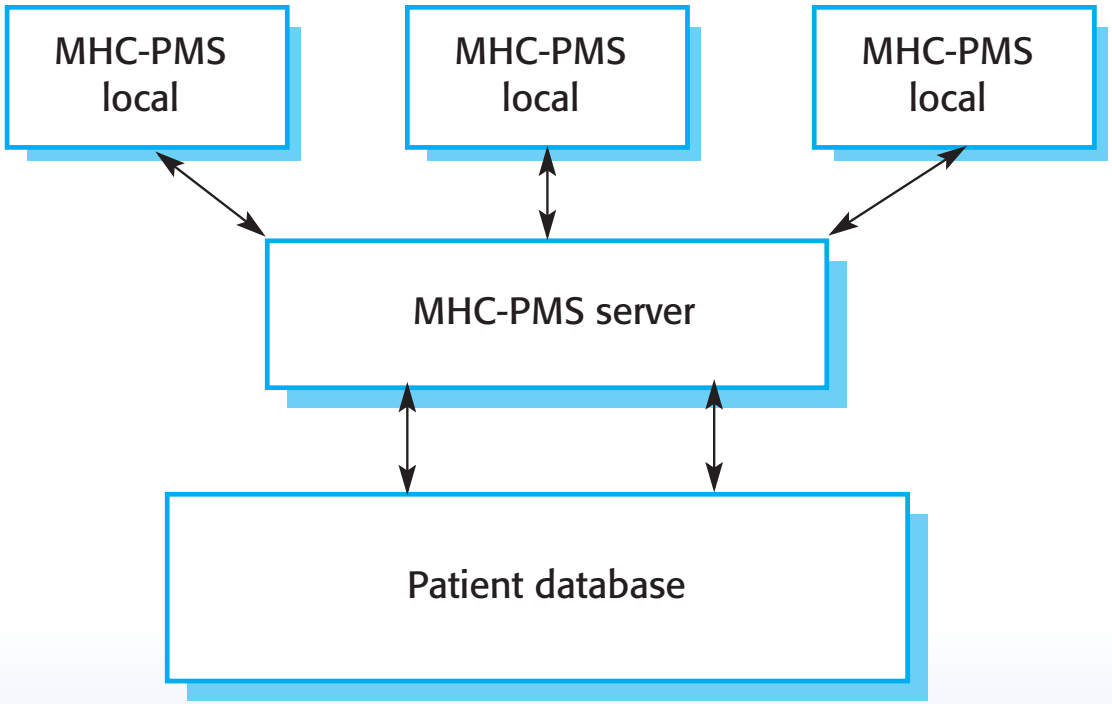
MHC-PMS

- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are

MHC-PMS goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

Figure 1.6 The organization of the MHC-PMS



MHC-PMS key features

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic

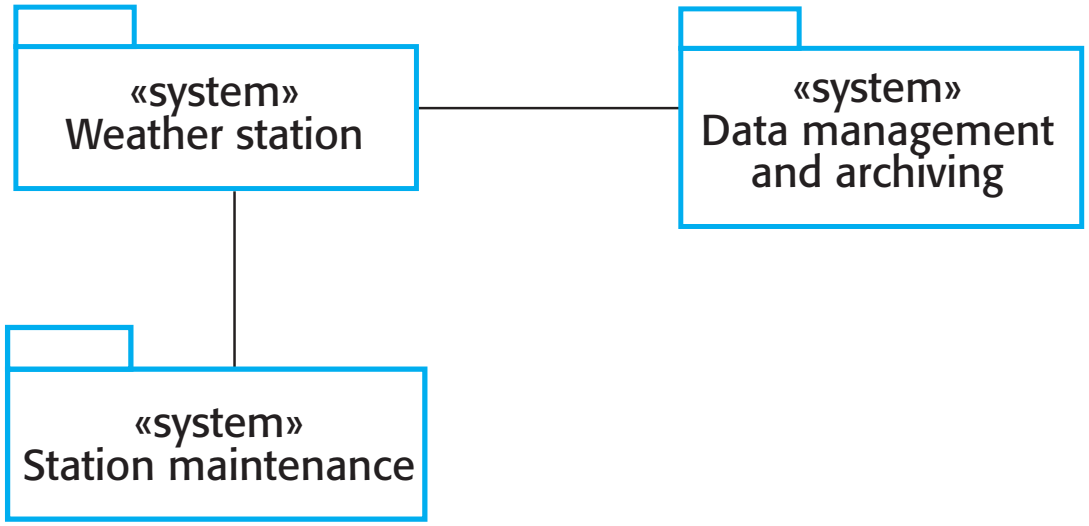
MHC-PMS concerns

- Privacy
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- Safety
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Wilderness weather station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings

Figure 1.7 The weather station's environment



Weather information system

- The weather station system
  - This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system
  - This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
  - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Additional software functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the

Key points

- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.
- Three case studies are used in the book:
  - An embedded insulin pump control system
  - A system for mental health care patient management
  - A wilderness weather station