Introduction pag 4

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:

1. understand what software engineering is and why it is important;
2. understand that the development of different types of software systems may require different software engineering techniques;
3. understand some ethical and professional issues that are important for software engineers;

1. We can't run the modern world without software.
2. Software systems are abstract and intangible. They are not constrained by the properties of materials, governed by physical laws, or by manufacturing processes.
3. There are many different types of software systems, from simple embedded systems to complex, worldwide information systems.
4. There are still many reports of software projects going wrong and 'software failures'. Software **engineering is criticized as inadequate for modern software development**.

However, **many of these so-called software failures are a consequence of two factors**:

1. **Increasing demands** As new software engineering techniques help us to build larger, more complex systems, the demands change**. Systems have to be built and delivered more quickly; larger, even more complex systems are required;** systems have to have new capabilities that were previously thought to be impossible. Existing software engineering methods cannot cope and new software engineering techniques have to be developed to meet new these new demands.
2. **Low expectations It** is relatively easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved. **They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be**. **We need better software engineering education and training to address this problem**.

History of software engineering

The notion of 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called the 'software crisis' (Naur and Randell, 1969).

It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late.

Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding and object-oriented development. Tools and standard notations were developed and are now extensively used.

http://www.SoftwareEngineering-9.com/Web/History/

Software engineering is intended to support professional software development, rather than individual programming.

It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development. To help you to get a broad view of what software engineering is about

Many people think that software is simply another word for computer programs.
However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs operate correctly.
A professionally developed software system is often more than a single program. **The system usually consists of a number of separate programs and configuration files that are used to set up these programs. It may include system documentation, which describes the structure of the system; user documentation, which explains how to use the system, and websites for users to download recent product information.**

This is one of the **important differences between professional and amateur software development**. If you are writing a program for yourself, no one else will use it and you don't have to worry about writing program guides, documenting the program design, etc. **However, if you are writing software that other people will use and other engineers will change then you usually have to provide additional information as well as the code of the program.**

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, software validation, and software evolution.
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.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.

What are the costs of software engineering?	Roughly 60% of software costs are development costs; 40% are testing costs. 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 professionallymanaged 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.

Ce este software-ul?	Programe de calculator și documentație asociată. Produsele software pot fi dezvoltate pentru un anumit client sau pot fi dezvoltate pentru o piață generală.
Care sunt atributele unui software bun?	Un software bun ar trebui să ofere utilizatorului funcționalitatea și performanțele necesare și ar trebui să fie întreținibil, fiabil și utilizabil. Ingineria software este o disciplină de inginerie care se ocupă de toate aspectele producției de software.
Care sunt activitățile fundamentale ale ingineriei software?	Specificații software, dezvoltare software, validare software și evoluție software.
Care este diferența dintre software inginerie și informatică?	Informatica se concentrează pe teorie și elemente fundamentale; ingineria software este preocupată de practicile dezvoltării și livrării de software utile.
Care este diferența dintre software inginerie și inginerie de sisteme?	Ingineria de sistem este preocupată de toate aspectele dezvoltării sistemelor bazate pe computer, inclusiv hardware, software și inginerie de procese. Ingineria de software face parte din acest proces mai general.
Care sunt provocările cheie cu care se confruntă Inginerie software-ul ?	Faceți față diversității în creștere, cerințelor de timpi de livrare reduse și dezvoltării unui software de încredere.
Care sunt costurile ingineriei software?	Aproximativ 60% din costurile software sunt costuri de dezvoltare; 40% sunt costuri de testare. Pentru software personalizat, costurile de evoluție depășesc adesea costurile de dezvoltare.
Care sunt cele mai bune tehnici de inginerie și metode software?	În timp ce toate proiectele software trebuie gestionate și dezvoltate profesional, diferite tehnici sunt adecvate pentru diferite tipuri de sisteme. De exemplu, jocurile ar trebui dezvoltate

	întotdeauna folosind o serie de prototipuri, în special sistemele de control critice în materie de siguranță necesită o specificație completă și analizabilă. Prin urmare, nu pot spune că o metodă este mai bună decât alta
Ce diferențe a făcut Web-ul față de software Inginerie?	Web-ul a condus la disponibilitatea serviciilor software și la posibilitatea dezvoltării sistemelor bazate pe servicii foarte distribuite. Dezvoltarea sistemelor bazate pe web a dus la progrese importante în limbaje de programare și reutilizarea software-ului.

There are two kinds of software products:

1. ***Generic products*** These are stand-alone systems that are produced by a development organization and **sold on the open market to any customer who is able to buy them**. Examples of this type of product include software for PCs such as databases, word processors, drawing packages, and project-management tools. It also includes so-called vertical applications designed for some specific purpose such as library information systems, accounting systems, or systems for maintaining dental records.
2. ***Customized (or bespoke) products*** These are systems that are commissioned by a particular customer. A software contractor develops the **software especially for that customer**. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

An important difference between these types of software is that,

- **In generic products**, the organization that develops the software controls the software specification.
- **For custom products**, the specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification.

However, the distinction between these system product types is becoming increasingly blurred. **More and more systems are now being built with a generic product as a base, which is then adapted to suit the requirements of a customer**. **Enterprise Resource Planning (ERP) systems**, such as the SAP system, are the best examples of this approach. Here, a large and complex system is adapted for a company by incorporating information about business rules and processes, reports required, and so on.

The specific set of attributes that you might expect from a software system obviously depends on its application.
Therefore, a banking system must be secure, an interactive game must be responsive, a telephone switching system must be reliable, and so on**.**
These can be generalized into the set of attributes shown in the table below, which I believe are the essential characteristics of a professional software system.

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 utilization, 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.

Mentenabilitate	**Software-ul trebuie scris astfel încât să poată evolua pentru a satisface nevoile în schimbare ale clienților. Acesta este un atribut critic, deoarece schimbarea software-ului este o cerință inevitabilă a unui mediu de afaceri în schimbare.**
Fiabilitate și securitate	**Fiabilitatea software-ului include o serie de caracteristici, inclusiv fiabilitate, securitate și siguranță. Software-ul de încredere nu ar trebui să provoace daune fizice sau economice în caz de defecțiune a sistemului. Utilizatorii rău intenționați nu ar trebui să poată accesa sau deteriora sistemul.**
Eficiența	**Software-ului nu ar trebui să utilizeze risipă resursele sistemului, cum ar fi ciclurile de memorie și procesor. Prin urmare, eficiența include reacția, timpul de procesare, utilizarea memoriei etc.**
Acceptabilitate	**Software-ul trebuie să fie acceptabil pentru tipul de utilizatori pentru care este proiectat. Aceasta înseamnă că trebuie să fie de înțeles, utilizabil și compatibil cu alte sisteme pe care le utilizează.**

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. In this definition, there are two key phrases:

1. ***Engineering discipline*** Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints.
2. ***All aspects of software production*** Software engineering i***s not just concerned with the technical processes*** of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production.

Ingineria software este o disciplină de inginerie care se ocupă de toate aspectele producției de software, de la primele etape ale specificațiilor sistemului până la menținerea sistemului după ce a intrat în uz.

În această definiție, există două fraze cheie:
1**. Disciplina de inginerie Inginerii fac lucrurile să funcționeze**.
Aplică teorii, metode și instrumente acolo unde acestea sunt adecvate. Cu toate acestea, le folosesc selectiv și încearcă întotdeauna să descopere soluții la probleme chiar și atunci când nu există teorii și metode aplicabile. Inginerii recunosc, de asemenea, că trebuie să lucreze la constrângerile organizaționale și financiare, astfel încât să caute soluții în cadrul acestor constrângeri.
2. **Toate aspectele producției de software** Ingineria software-ului nu se referă doar la procesele tehnice de dezvoltare a software-ului, ci include activități precum gestionarea proiectelor software și dezvoltarea de instrumente, metode și teorii pentru a sprijini producția de software.

Software engineering is important for two reasons:

1. More and more, individuals and society rely on **advanced** software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
2. It is usually cheaper, in the long run, to use 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 systems, the majority of costs are the costs of changing the software after it has gone into use**!!!!!!!!!

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

1. **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. **Software development,** where the software is designed and programmed.
3. **Software validation,** where the software is checked to ensure that it is what the customer requires.
4. **Software evolution,** where the software is modified to reflect changing customer and market requirements.

There are many different types of software. **There is no universal software engineering method or technique that is applicable for all of these.**
However, there are three general issues that affect many different types of software:

1. ***Heterogeneity (nivel inalt de diversitate)*** Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. Aswell as running on general-purpose computers, software may also have to executeon mobile phones. You often have to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope withthis heterogeneity.
2. ***Business and social change*** Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. Theyneed to be able to change their existing software and to rapidly develop new software. Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
3. ***Security and trust*** As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot attack our software and that information security is maintained.

Perhaps the most significant factor in determining which software engineering methods and techniques are most important is the type of application that is being developed.

There are many different types of application including:

1. ***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. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.
2. **Interactive transaction-based applications** These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. Obviously, these include web applications such as e-commerce applications where you can interact with a remote system to buy goods and services. This class of application also includes business systems, where a business provides access to its systems through a web browser or special-purpose client program and cloud-based services, such as mail and photo sharing. Interactive applications often incorporate a large data store that is accessed and updated in each transaction.
3. **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. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.
4. **Batch processing systems** These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.
5. **Entertainment systems** These are systems that are primarily for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.
6. **Systems for modeling and simulation** These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.
7. **Data collection systems** These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing. The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location.
8. **Systems of systems** These are systems that are composed of a number of other software systems. Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

Nevertheless, **there are software engineering fundamentals that apply to all types of software system:**

1. They should be developed using a **managed and understood development process**. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, different processes are used for different types of software.
2. **Dependability and performance** are important for all types of systems. Software should behave as expected, without failures and should be available for use when it is required. It should be safe in its operation and, as far as possible, should be secure against external attack. The system should perform efficiently and should not waste resources.
3. **Understanding and managing the software specification** and requirements (what the software should do) are important. You have to know what different customers and users of the system expect from it and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.

4. You should make **as effective use as possible of existing resources**. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

The advent of the web, therefore, has led to a significant change in the way that business software is organized. Before the web, business applications were mostly monolithic, single programs running on single computers or computer clusters.
Communications were local, within an organization. Now, software is highly distributed, sometimes across the world. Business applications are not programmed from scratch but involve extensive reuse of components and programs.
This radical change in software organization has, obviously, led to changes in the ways that web-based systems are engineered.

For example:

1. Software reuse has become the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.

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

User interfaces are constrained by the capabilities of web browsers. Although technologies such as AJAX (Holdener, 2008) mean that rich interfaces can be created within a web browser, these technologies are still difficult to use. Web forms with local scripting are more commonly used. **Application interfaces on web-based systems are often poorer than the specially designed user interfaces on PC system products**.

Software engineering ethics

1. ***Confidentiality*** You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
2. ***Competence*** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. ***Intellectual property rights*** You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. ***Computer misuse*** You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses or other malware).

Case studies

To illustrate software engineering concepts,we can use examples from three different types of systems. The reason why we have not used a single case study is that one of the key messages in this book is that software engineering practice depends on the type of systems being produced. I therefore choose an appropriate example when discussing concepts such as safety and dependability, system modeling, reuse, etc.

Three types of systems that can use as case studies are:

1. ***An embedded system*** This is a system where the software controls a hardwaredevice and is embedded in that device. Issues in embedded systems typically include physical size,

responsiveness, power management, etc. The example of an embedded system that I use is a software system to control a medical device.

2. ***An information system*** This is a system whose primary purpose is to manage and provide access to a database of information. Issues in information systems include security, usability, privacy, and maintaining data integrity. The example of an information system that I use is a medical records system.
3. ***A sensor-based data collection system*** This is a system whose primary purpose is to collect data from a set of sensors and process that data in some way. The key requirements of such systems are reliability, even in hostile environmental conditions, and maintainability. The example of a data collection system that I use is a wilderness weather station.

1.3.1 *An insulin pump control system*
1.3.2 *A patient information system for mental health care*
1.3.3 *A wilderness weather station*

EXERCISES

1.1. Explain why professional software is not just the programs that are developed for a customer.
1.2. What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?
1.3. What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant.
1.4. Apart from the challenges of heterogeneity, business and social change, and trust and security, identify other problems and challenges that software engineering is likely to face in the 21st century (Hint: think about the environment).
1.5. Based on your own knowledge of some of the application types discussed in section 1.1.2, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.
1.6. Explain why there are fundamental ideas of software engineering that apply to all types of software systems.
1.7. Explain how the universal use of the Web has changed software systems.
1.8. Discuss whether professional engineers should be certified in the same way as doctors or lawyers.
1.9. For each of the clauses in the ACM/IEEE Code of Ethics shown in Figure 1.3, suggest an appropriate example that illustrates that clause.
1.10. To help counter terrorism, many countries are planning or have developed computer systems that track large numbers of their citizens and their actions. Clearly this has privacy implications. Discuss the ethics of working on the development of this type of system.