In my class, I’d expect:

- You will be an Active Learner.
- You will be an Active Participant in all group or team activities.
- You will be Positive in your study.
- You will treat all your classmates and lecturer with respect.
- You will react with grace and dignity should you receive criticisms or adverse comments.
- You will do your BEST!

In return, I promise

- You will receive a challenging but rewarding learning experience
- You will receive fair assessment
- You are treated as an equal partner in your journey of study
- You may build up networks and contacts
- You may even gain friendships for life

Topics

- Unit Outline :ICT381_04_Outline.pdf
- Tutorial and Reading for Week 1
- Introduction (Ch 1 Sommerville)
- Computer Based Software Engineering (Ch 2 Sommerville)

Tutorial & Reading for Week 1

- IT Predictions 2004 (Ref: Australian, IT Section 10th February, 2004)
  - Comment on these articles and provide a summary on the predictions in the IT industry for 2004 and beyond.
  - What will be the implications on Software Engineering?
- Additional Exercise
  - Find out the Code of Ethnic for the Australian Computer Society (ACS). Compare it to the code from ACM/IEEE.
Introduction

• Getting started with software engineering


Objectives

• To introduce software engineering and to explain its importance
• To set out the answers to key questions about software engineering
• To introduce ethical and professional issues and to explain why they are of concern to software engineers

Topics covered

• FAQs about software engineering
• Professional and ethical responsibility

Software engineering

• The economies of ALL developed nations are dependent on software
• More and more systems are software controlled
• Software engineering is concerned with theories, methods and tools for professional software development
• Software engineering expenditure represents a significant fraction of GNP in all developed countries

Software costs

• Software costs often dominate 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
• Software engineering is concerned with cost-effective software development

FAQs about software engineering

• What is software?
• What is software engineering?
• What is the difference between software engineering and computer science?
• What is the difference between software engineering and system engineering?
• What is a software process?
• What is a software process model?
FAQs about software engineering
• What are the costs of software engineering?
• What are software engineering methods?
• What is CASE (Computer-Aided Software Engineering)
• What are the attributes of good software?
• What are the key challenges facing software engineering?

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
• Software products may be
  – Generic - developed to be sold to a range of different customers
  – Bespoke (custom) - developed for a single customer according to their specification

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

What is the difference between software engineering and computer science?
• Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
• Computer science theories are currently insufficient to act as a complete underpinning for software engineering

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 process
• System engineers are involved in system specification, architectural design, integration and deployment

What is a software process?
• A set of activities whose goal is the development or evolution of software
• Generic activities in all software processes are:
  – Specification - what the system should do and its development constraints
  – Development - production of the software system
  – Validation - checking that the software is what the customer wants
  – Evolution - changing the software in response to changing demands
What is a software process model?

• A simplified representation of a software process, presented from a specific perspective
• Examples of process perspectives are
  – Workflow perspective - sequence of activities
  – Data-flow perspective - information flow
  – Role/action perspective - who does what

What is a software process model?

• Generic process models
  – Waterfall
  – Evolutionary development
  – Formal transformation
  – Integration from reusable components

What are the costs of software engineering?

• Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs
• Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability
• Distribution of costs depends on the development model that is used

What are software engineering methods?

• Structured approaches to software development which include system models, notations, rules, design advice and process guidance
• Model descriptions
  – Descriptions of graphical models which should be produced
• Rules
  – Constraints applied to system models
• Recommendations
  – Advice on good design practice
• Process guidance
  – What activities to follow

What is CASE (Computer-Aided Software Engineering)

• Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support
• Upper-CASE
  – Tools to support the early process activities of requirements and design
• Lower-CASE
  – Tools to support later activities such as programming, debugging and testing

What are the attributes of good software?

• The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable
• Maintainability - Software must evolve to meet changing needs
• Dependability - Software must be trustworthy
• Efficiency - Software should not make wasteful use of system resources
• Usability - Software must be usable by the users for which it was designed
What are the key challenges facing software engineering?

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times.
- Legacy systems - Old, valuable systems must be maintained and updated.
- Heterogeneity - Systems are distributed and include a mix of hardware and software.
- Delivery - There is increasing pressure for faster delivery of software.

Stopped here: Professional and ethical responsibility

- 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.

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 outwith their competence.

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.

Code of ethics - preamble

- 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.
Code of ethics - preamble

– 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:

Code of ethics - 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.
• 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

Key points

• Software engineering is an engineering discipline which is concerned with all aspects of software production.
• Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
Key points

• The software process consists of activities which are involved in developing software products. Basic activities are software specification, development, validation and evolution.
• Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

Key points

• CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.
• 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.

Chapter 2

Computer-Based System Engineering


Objectives

• To explain why system software is affected by broader system engineering issues
• To introduce the concept of emergent system properties such as reliability and security
• To explain why the systems environment must be considered in the system design process
• To explain system engineering and system procurement processes

Topics covered

• Emergent system properties
• Systems and their environment
• System modelling
• The system engineering process
• System procurement
What is a system?
• A purposeful collection of inter-related components working together towards some common objective.
• A system may include software, mechanical, electrical and electronic hardware and be operated by people.
• System components are dependent on other system components
• The properties and behaviour of system components are inextricably inter-mingled

Problems of systems engineering
• Large systems are usually designed to solve ‘wicked’ problems
• Systems engineering requires a great deal of co-ordination across disciplines
  – Almost infinite possibilities for design trade-offs across components
  – Mutual distrust and lack of understanding across engineering disciplines
• Systems must be designed to last many years in a changing environment

Software and systems engineering
• The proportion of software in systems is increasing. Software-driven general purpose electronics is replacing special-purpose systems
• Problems of systems engineering are similar to problems of software engineering
• Software is (unfortunately) seen as a problem in systems engineering. Many large system projects have been delayed because of software problems

Emergent properties
• Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
• Emergent properties are a consequence of the relationships between system components
• They can therefore only be assessed and measured once the components have been integrated into a system

Examples of emergent properties
• The overall weight of the system
  – This is an example of an emergent property that can be computed from individual component properties.
• The reliability of the system
  – This depends on the reliability of system components and the relationships between the components.
• The usability of a system
  – This is a complex property which is not simply dependent on the system hardware and software but also depends on the system operators and the environment where it is used.

Types of emergent property
• Functional properties
  – These appear when all the parts of a system work together to achieve some objective. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
• Non-functional emergent properties
  – Examples are reliability, performance, safety, and security. These relate to the behaviour of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.
System reliability engineering

- Because of component inter-dependencies, faults can be propagated through the system
- System failures often occur because of unforeseen inter-relationships between components
- It is probably impossible to anticipate all possible component relationships
- Software reliability measures may give a false picture of the system reliability

Influences on reliability

- Hardware reliability
  - What is the probability of a hardware component failing and how long does it take to repair that component?
- Software reliability
  - How likely is it that a software component will produce an incorrect output. Software failure is usually distinct from hardware failure in that software does not wear out.
- Operator reliability
  - How likely is it that the operator of a system will make an error?

Reliability relationships

- Hardware failure can generate spurious signals that are outside the range of inputs expected by the software
- Software errors can cause alarms to be activated which cause operator stress and lead to operator errors
- The environment in which a system is installed can affect its reliability

The ‘shall-not’ properties

- Properties such as performance and reliability can be measured
- However, some properties are properties that the system should not exhibit
  - Safety - the system should not behave in an unsafe way
  - Security - the system should not permit unauthorised use
- Measuring or assessing these properties is very hard

Systems and their environment

- Systems are not independent but exist in an environment
- System’s function may be to change its environment
- Environment affects the functioning of the system e.g. system may require electrical supply from its environment
- The organizational as well as the physical environment may be important
Human and organisational factors

- Process changes
  - Does the system require changes to the work processes in the environment?
- Job changes
  - Does the system de-skill the users in an environment or cause them to change the way they work?
- Organisational changes
  - Does the system change the political power structure in an organisation?

System architecture modelling

- An architectural model presents an abstract view of the sub-systems making up a system
- May include major information flows between sub-systems
- Usually presented as a block diagram
- May identify different types of functional component in the model

Intruder alarm system

- Movement sensors
- Door sensors
- Alarm controller
- Siren
- Voice synthesizer
- Telephone caller
- External control centre

Component types in alarm system

- Sensor
  - Movement sensor, door sensor
- Actuator
  - Siren
- Communication
  - Telephone caller
- Co-ordination
  - Alarm controller
- Interface
  - Voice synthesizer

Functional system components

- Sensor components
- Actuator components
- Computation components
- Communication components
- Co-ordination components
- Interface components
System components

- Sensor components
  - Collect information from the system’s environment e.g. radars in an air traffic control system
- Actuator components
  - Cause some change in the system’s environment e.g. valves in a process control system which increase or decrease material flow in a pipe
- Computation components
  - Carry out some computations on an input to produce an output e.g. a floating point processor in a computer system

Component types in alarm system

- Sensor
  - Movement sensor, Door sensor
- Actuator
  - Siren
- Communication
  - Telephone caller
- Coordination
  - Alarm controller
- Interface
  - Voice synthesizer

The system engineering process

- Usually follows a ‘waterfall’ model because of the need for parallel development of different parts of the system
  - Little scope for iteration between phases because hardware changes are very expensive. Software may have to compensate for hardware problems

Inevitably involves engineers from different disciplines who must work together
- Much scope for misunderstanding here. Different disciplines use a different vocabulary and much negotiation is required. Engineers may have personal agendas to fulfil
Inter-disciplinary involvement

- Software engineering
- Electronic engineering
- Mechanical engineering
- Structural engineering
- Civil engineering
- Electrical engineering
- Architecture
- User interface design
- ATC systems engineering

System requirements definition

- Three types of requirement defined at this stage
  - Abstract functional requirements. System functions are defined in an abstract way
  - System properties. Non-functional requirements for the system in general are defined
  - Undesirable characteristics. Unacceptable system behaviour is specified
- Should also define overall organisational objectives for the system

System objectives

- Functional objectives
  - To provide a fire and intruder alarm system for the building which will provide internal and external warning of fire or unauthorized intrusion
- Organisational objectives
  - To ensure that the normal functioning of work carried out in the building is not seriously disrupted by events such as fire and unauthorized intrusion

System requirements problems

- Changing as the system is being specified
- Must anticipate hardware/communications developments over the lifetime of the system
- Hard to define non-functional requirements (particularly) without an impression of component structure of the system.

The system design process

- Partition requirements
  - Organise requirements into related groups
- Identify sub-systems
  - Identify a set of sub-systems which collectively can meet the system requirements
- Assign requirements to sub-systems
  - Causes particular problems when COTS are integrated
- Specify sub-system functionality
- Define sub-system interfaces
  - Critical activity for parallel sub-system development

The system design process

- Partition requirements
- Identify sub-systems
- Specify sub-system functionality
- Assign requirements to sub-systems
- Define sub-system interfaces
System design problems

- Requirements partitioning to hardware, software and human components may involve a lot of negotiation
- Difficult design problems are often assumed to be readily solved using software
- Hardware platforms may be inappropriate for software requirements so software must compensate for this

Sub-system development

- Typically parallel projects developing the hardware, software and communications
- May involve some COTS (Commercial Off-the-Shelf) systems procurement
- Lack of communication across implementation teams
- Bureaucratic and slow mechanism for proposing system changes means that the development schedule may be extended because of the need for rework

System integration

- The process of putting hardware, software and people together to make a system
- Should be tackled incrementally so that sub-systems are integrated one at a time
- Interface problems between sub-systems are usually found at this stage
- May be problems with uncoordinated deliveries of system components

System installation

- Environmental assumptions may be incorrect
- May be human resistance to the introduction of a new system
- System may have to coexist with alternative systems for some time
- May be physical installation problems (e.g. cabling problems)
- Operator training has to be identified

System operation

- Will bring unforeseen requirements to light
- Users may use the system in a way which is not anticipated by system designers
- May reveal problems in the interaction with other systems
  - Physical problems of incompatibility
  - Data conversion problems
  - Increased operator error rate because of inconsistent interfaces

System evolution

- Large systems have a long lifetime. They must evolve to meet changing requirements
- Evolution is inherently costly
  - Changes must be analysed from a technical and business perspective
  - Sub-systems interact so unanticipated problems can arise
  - There is rarely a rationale for original design decisions
  - System structure is corrupted as changes are made to it
- Existing systems which must be maintained are sometimes called legacy systems
System decommissioning

- Taking the system out of service after its useful lifetime
- May require removal of materials (e.g., dangerous chemicals) which pollute the environment
  - Should be planned for in the system design by encapsulation
- May require data to be restructured and converted to be used in some other system

System procurement

- Acquiring a system for an organization to meet some need
- Some system specification and architectural design is usually necessary before procurement
  - You need a specification to let a contract for system development
  - The specification may allow you to buy a commercial off-the-shelf (COTS) system. Almost always cheaper than developing a system from scratch

The system procurement process

- Align requirements
- Choose system
- Issue request for bid
- Choose supplier
- Survey market for existing systems
- Let contract for development
- Negotiate contract
- Select tender
- Issue request to tender
- Off-the-shelf system available
- Bespoke system required

Procurement issues

- Requirements may have to be modified to match the capabilities of off-the-shelf components
- The requirements specification may be part of the contract for the development of the system
- There is usually a contract negotiation period to agree changes after the contractor to build a system has been selected

Contractors and sub-contractors

- The procurement of large hardware/software systems is usually based around some principal contractor
- Sub-contracts are issued to other suppliers to supply parts of the system
- Customer liaises with the principal contractor and does not deal directly with sub-contractors

Contractor/Sub-contractor model

- System customer
- Principal contractor
- Sub-contractor 1
- Sub-contractor 2
- Sub-contractor 3
Key points

- System engineering involves input from a range of disciplines
- Emergent properties are properties that are characteristic of the system as a whole and not its component parts
- System architectural models show major sub-systems and inter-connections. They are usually described using block diagrams

Key points

- System component types are sensor, actuator, computation, co-ordination, communication and interface
- The systems engineering process is usually a waterfall model and includes specification, design, development and integration.
- System procurement is concerned with deciding which system to buy and who to buy it from

Conclusion

- Systems engineering is hard! There will never be an easy answer to the problems of complex system development
- Software engineers do not have all the answers but may be better at taking a systems viewpoint
- Disciplines need to recognise each others strengths and actively rather than reluctantly cooperate in the systems engineering process