Cmpt 275：

**Introduction to software engineering**

* Introduction
  + How is software engineering similar to other engineering?
    - Abstraction and Modularity
  + Where does the software engineer fit in?
    - Computer science: focusing on
      * Computer hardware
      * compilers
      * operating systems
      * programming languages
    - Software engineering:
      * A discipline that uses computer and software technologies as a problem-solving tools
* FAQs about software engineering
  + What is software?
    - Software is:
      * Instructions (computer programs) that when executed provide desired features, functions, and performance
      * Data structures that enable the programs to adequately manipulate information
      * Documentation that describe the operation and use of the programs
    - Software is developed or engineered, it is not manufactured in the classical sense
    - Software doesn’t “wear out”
    - Although the industry is moving toward component-based construction, most software continues to be custom-built
    - Generic vs. Custom
    - Characteristics of Software
    - Legacy Software
  + What are the essential attributes of good software?
    - Maintainability
    - Dependability and security
    - Efficiency
    - Acceptability
  + What is software engineering?
    - The IEEE Computer Society defines software engineering as “the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e. the application of engineering to software”
    - Relationship between computer science and software engineering: Fig. 1
    - Software engineering body of knowledge: Fig. 2
      * Computing Fundamentals
        + Algorithms and data structures
        + Computer architecture
        + Mathematical fundamentals
        + Operating systems
        + Programming languages
      * Software Product Engineering
        + Requirements engineering
        + Software design
        + Software coding
        + Software testing
        + Software Ops. & Maint.
      * Software management
        + Project process management
        + Risk management
        + Quality management
        + Configuration management
        + Dev. Process management
        + Acquisition management
      * Software domains
        + Artificial intelligence
        + Database systems
        + Human-computer interaction
        + Numerical & symbolic comp.
        + Computer simulation
        + Real-time systems
  + Why many software development projects fail?
    - Software project failure
      * Increasing system complexity
      * Failure to use software engineering methods
    - Software size categories: Fig. 3
    - Why software engineering?
      * ???
    - Controlling complexity
      * When a large system is being developed, it is essential to design (engineer) the software
      * The project must be divided into manageable units
        + Clear interfaces between the units must be defied and adhered to
      * Clear interfaces
        + Each of the unit will be assigned to a programmer or group of programmers. This group may have no knowledge of what other units are and how their unit fits into the larger project
        + To assure that all units will integrate smoothly into the final large system design is essential, and clear interfaces between units must be defined early in the project

A small change by one group to “improve” the interface will cause disaster at integration time if it is not communicated to and agreed upon by all groups involved

* + - * Communication Breakdown
        + ???
  + What is the difference between software engineering and computer science?
    - Computing science is concerned with 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 process concerned with developing the software infrastructure, control, applications and database in the system.
    - System engineers are involved in system specification, architectural design, integration and deployment
  + What is a software development process?
    - Process framework
      * A process framework: Fig. 4
      * Framework Activities
        + ???
    - Software engineering 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 a software system (designed and implemented)

* + - * + Validation

Checking that the software is what the customer wants

* + - * + Evolution

Changing the software in response to changing demands

* + - Software process models
  + What is a software process model?
    - A simplified representation of a software process, presented from a specific perspective
    - Generic process models
      * Waterfall : Fig. 5
      * Iterative development
        + Incremental / evolutionary development: Fig. 6
      * Component-based software engineering
        + Spiral model: Fig. 7
  + What are the costs of software engineering?
    - Software costs
      * ???
    - 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
    - Activity cost distribution: Fig. 8
  + What are the software engineering methods?
    - ???
  + What are the key challenges facing software engineering?
    - Heterogeneity, delivery and trust
    - Heterogeneity
      * Developing techniques for building software that can cope with heterogeneous platforms and execution environments
    - Delivery
      * Developing techniques that lead to faster delivery of software
    - Trust
      * Developing techniques that demonstrate that software can be trusted by its users
* 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 behavior 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 out with their competence
      * Issue: should software engineers be licensed? What does competence really mean here?
        + ???
    - 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 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 organizations sign up to the code of practice when they join
  + The code contains eight principles related to the behavior 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 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 8 principles:
        + Public

Software engineers shall act consistently with the public interest

* + - * + 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

* + - * + 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

* + - * + Management

Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance

* + - * + Profession

Software engineers shall advance the integrity and reputation of the profession consistent with the public interest

* + - * + Colleagues

Software engineers shall be fair to and supportive of their colleagues

* + - * + 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

* 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
  + 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 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 standard of behavior expected of their members

**Project management**

* Software project management
  + A manager can control 4 things
    - Resources: can get more dollars, facilities, personnel
    - Time: can increase schedule, delay milestones, etc.
    - Product: can reduce functionality
      * E.g. scrub requirements
    - Risk: can decide which risks are acceptable
* Management activities
  + Proposal writing
  + Project planning and scheduling
  + Project costing
  + Project monitoring and reviews
  + Personnel selection and evaluation
  + Report writing and presentations
* Activity organization
  + Activities in a project should be organized to produce tangible outputs for management to judge progress
  + Milestones are the end-point of a process activity
  + Deliverables are project results delivered to customers
  + The warerfall process allows for the straightforward definition of progress milestones
* Project planning
* Project scheduling
  + Split project into tasks and estimate time and resources required to complete each task
  + Organize tasks concurrently to make optimal use of workforce
  + Minimize task dependencies to avoid delays caused by one task waiting for another to complete
  + Dependent on project managers intuition and experience
  + The project scheduling process: Fig. 9
  + Scheduling problems
    - Estimating the difficulty of problems and hence the cost of developing a solution is hard
    - Productivity is not proportional to the number of people working on the task
    - Adding people to a late project makes it later because of communication overheads
    - The unexpected always happens, always allow contingency in planning
  + Bar charts and activity networks
    - Graphical notations used to illustrate the project schedule
    - Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two
    - Activity charts show task dependencies and the critical path
    - Bar charts show schedule against calendar time
  + Task durations and dependencies: Fig. 10
  + Activity network: Fig. 11
  + Activity timeline: Fig. 12
  + Staff allocation: Fig. 13
  + Work breakdown structure: Fig. 14
  + PERT charts: Fig. 15
  + Gantt charts: Fig. 16
* Risk management
  + Definitions
    - Risk: an undesirable outcome that poses a threat to the achievement of some objective. A process risk threatens the schedule or cost of a process; a product risk is a risk that may mean that some of the system requirements may not be achieved
    - Risk management: the process of identifying risks, assessing their severity, planning measures to put in place if the risks arise, and monitoring the software and the software process for risks
  + Risk management is concerned with identifying risks and drawing up plans to minimize their effect on a project
    - Risk assessment
    - Risk control
  + A risk is a probability that some adverse circumstance will occur
    - Project risks affect schedule or resources
    - Project risks affect the quality or performance of the software being developed
    - Business risks affect the organization developing or procuring the software
  + Risk classification
    - There are 2 dimensions of risk classification
      * The type of risk (technical, organizational, ..)
      * What is affected by the risk
        + Project risks affect schedule or resources
        + Project risks affect the quality or performance of the software being developed
        + Business risks affect the organization developing or procuring the software
  + Example of project, product, and business risks: Fig. 17
  + The risk management process: Fig. 18
    - Risk identification: Identify project, product and business risks
      * May be a team activities or based on the individual project manager’s experience.
      * A checklist of common risks may be used to identify risks in a project
        + Technology risks
        + People risks
        + Organizational risks
        + Requirements risks
        + Estimation risks
      * Examples of different risk types: Fig. 19
    - Risk analysis: Assess the likelihood and consequences of these risks
      * Assess probability and seriousness of each risk.
      * Probability may be very low, low, moderate, high or very high.
      * Risk effects might be catastrophic, serious, tolerable or insignificant.
      * Risk types and examples: Fig. 20
    - Risk planning: Draw up plans to Draw up plans to avoid or minimize the effects of the risk
      * Consider each risk and develop a strategy to manage that risk.
      * Avoidance strategies
        + The probability that the risk will arise is reduced;
      * Minimization strategies
        + The impact of the risk on the project or product will be reduced;
      * Contingency plans
        + If the risk arises, contingency plans are plans to deal with that risk;
      * What-if questions
        + ???
      * Risk management strategies: Fig. 21
    - Risk monitoring: Monitor the risks throughout the project
      * Assess each identified risk regularly to decide whether or not it is becoming less or more probable.
      * Also assess whether the effects of the risk have changed.
      * Each key risk should be discussed at management progress meetings.
      * Risk indicators: Fig. 22
* Forming team
  + Managing people
    - People are an organization’s most important assets.
    - The tasks of a manager are essentially people oriented. Unless there is some understanding of people, management will be unsuccessful.
    - Poor people management is an important contributor to project failure.
  + Software development life cycle oversight: Fig. 23
  + People management factors
    - Consistency
      * Team members should all be treated in a comparable way without favorites or discrimination.
    - Respect
      * Different team members have different skills and these differences should be respected.
    - Inclusion
      * Involve all team members and make sure that people’s views are considered.
    - Honesty
      * You should always be honest about what is going well and what is going badly in a project.
  + Motivating people
    - An important role of a manager is to motivate the people working on a project
    - Motivation means organizing the work and the working environment to encourage people to work effectively.
    - If people are not motivated, they will not be interested in the work they are doing. They will work slowly, be more likely to make mistakes and will not contribute to the broader goals of the team or the organization.
    - Motivation is a complex issue but it appears that there are different types of motivation based on:
      * Basic needs (e.g. food, sleep, etc.);
      * Personal needs (e.g. respect, self-esteem);
      * Social needs (e.g. to be accepted as part of a group).
  + Human needs hierarchy: Fig. 24
  + Need satisfaction
    - In software development groups, basic physiological and safety needs are not an issue.
    - Social
      * Provide communal facilities;
      * Allow informal communications e.g. via social networking
    - Esteem
      * Recognition of achievements;
      * Appropriate rewards.
    - Self-realization
      * Training - people want to learn more;
      * Responsibility.
  + Group communications
    - Good communications are essential for effective group working.
    - Information must be exchanged on the status of work, design decisions and changes to previous decisions.
    - Good communications also strengthen group cohesion as it promotes understanding.
    - Group size
      * The larger the group, the harder it is for people to communicate with other group members.
    - Group structure
      * Communication is better in informally structured groups than in hierarchically structured groups.
    - Group composition
      * Communication is better when there are different personality types in a group and when groups are mixed rather than single sex.
    - The physical work environment
      * Good workplace organization can help encourage communications.
* Key points
  + Good project management is essential for project success.
  + The intangible nature of software causes problems for management.
  + Managers have diverse roles, but their most significant activities are planning, estimating and scheduling.
  + Planning and estimating are iterative processes which continue throughout the course of a project.
  + A project milestone is a predictable state where a formal report of progress is presented to management.
  + Project scheduling involves preparing various graphical representations showing project activities, their durations and staffing.
  + Risk management is concerned with identifying risks which may affect the project and planning to ensure that these risks do not develop into major threats.

**Software development processes**

* The software process
  + Many different software processes but all involve:
    - Specification defining what the system should do;
    - Design and implementation defining the organization of the system and implementing the system;
    - Validation checking that it does what the customer wants;
    - Evolution changing the system in response to changing customer needs.
* Software process descriptions
  + Process descriptions may also include:
    - Products, which are the outcomes of a process activity;
    - Roles, which reflect the responsibilities of the people involved in the process;
    - Pre and post conditions, which are statements that are true before and after a process activity has been enacted or a product produced.
* Plan-driven and agile processes
  + ???
* Generic software process models
  + The waterfall model
  + Incremental development (Evolutionary development)
  + Reuse-oriented software engineering (Component-based software engineering)
  + There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.
* Waterfall process model: Fig. 5
* Waterfall model phases
  + Waterfall model describes a process of stepwise refinement
    - Based on hardware engineering models
    - Widely used in defense and aerospace industries
  + There are separate identified phases in the waterfall model:
    - Requirements analysis and definition
    - System and software design
    - Implementation and unit testing
    - Integration and system testing
    - Operation and maintenance
* Waterfall model problems
  + Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements
    - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process
    - Few business systems have stable requirements
    - Unrealistic separation of specification from design
    - Doesn’t accommodate prototyping, reuse, etc.
  + The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. (Widely used in defense and aerospace industries)
* Incremental development: Fig. 6
  + Exploratory development
    - Objective is to work with customers and to evolve a final system from an initial outline specification
    - Should start with well-understood requirements and add new features as proposed by the customer.
  + Throw-away prototyping
    - Objective is to understand the system requirements
    - Should start with poorly understood requirements to clarify what is really needed.
* Incremental development benefits
  + The cost of accommodating changing customer requirements is reduced.
  + It is easier to get customer feedback on the development work that has been done.
  + More rapid delivery and deployment of useful software to the customer is possible.
* Evolutionary development
  + Problems
    - Lack of process visibility
    - Systems are often poorly structured
    - Special skills (e.g. in languages for rapid prototyping) may be required.
  + Applicability
    - For small or medium-size interactive systems
    - For parts of large systems (e.g. the user interface)
    - For short-lifetime systems.
* Boehm’s spiral model: Fig. 7
* Reuse-oriented / Component-based software engineering
  + Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
  + Process stages
    - Component analysis
    - Requirements modification
    - System design with reuse
    - Development and integration
  + Reuse is now the standard approach for building many types of business system
    - Reuse covered in more depth in Chapter 15.
* Reuse-oriented development: Fig. 26
* Types of software component
  + Web services that are developed according to service standards and which are available for remote invocation
  + Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE
  + Stand-alone software systems (COTS) that are configured for use in a particular environment.
* Advantages and disadvantages
  + Reduced costs and risks as less software is developed from scratch
  + Faster delivery and deployment of system
  + But requirements compromises are inevitable so system may not meet real needs of users
  + Loss of control over evolution of reused system elements
* Hybrid development process: Fig. 27
* Process activities:
  + Software specification
  + Software design and implementation
  + Software validation
  + Software evolution
* Software specification
  + The process of establishing what services are required and the constraints on the system’s operation and development
  + Requirements engineering process
    - Feasibility study
      * Is it technically and financially feasible to build the system?
    - Requirements elicitation and analysis
      * What do the system stakeholders require or expect from the system?
    - Requirements specification
      * Defining the requirements in detail
    - Requirements validation
      * Checking the validity of the requirements
* The requirements engineering process: Fig. 28
* Software design and implementation
  + The process of converting the system specification into an executable system
  + Software design
    - Design a software structure that realizes the specification
  + Implementation
    - Translate this structure into an executable program
  + The activities of design and implementation are closely related and may be inter-leaved.
* A general model of the design process: Fig. 29
* Structured methods
  + Systematic approaches to developing a software design
  + The design is usually documented as a set of graphical models
  + Possible models
    - Object model
    - Sequence model
    - State transition model
    - Structural model
    - Data-flow model
* System implementation
  + The software is implemented either by developing a program or programs or by configuring an application system
  + Design and implementation are interleaved activities for most types of software system
  + Programming is an individual activity with no standard process
  + Debugging is the activity of finding program faults and correcting these faults
* Software validation
  + Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer
  + Involves checking and review processes and system testing
  + System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system
  + Testing is the most commonly used V & V activity.
* The stages of testing: Fig. 30
* Testing stages:
  + Component or unit testing
    - Individual components are tested independently
    - Components may be functions or objects or coherent groupings of these entities
  + System testing
    - Testing of the system as a whole. Testing of emergent properties is particularly important
  + Acceptance testing
    - Testing with customer data to check that the system meets the customer’s needs
* Testing phases in a plan-driven software process (V-model): Fig. 31
* Software evolution: Fig. 32
  + Software is inherently flexible and can change.
  + As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
  + Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.
* Systems engineering vs. software engineering
* What is a system?
  + A purposeful collection of inter-related components working together to achieve 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 behavior of system components are inextricably inter-mingled
* System categories
  + Technical computer-based systems
    - Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system.
  + Socio-technical systems
    - Systems that include technical systems but also operational processes and people who use and interact with the technical system. Socio- technical systems are governed by organizational policies and rules
* 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
* Definitions
  + Reliability is
    - “the ability of a system to deliver services as specified. Reliability can be specified quantitatively as a probability of failure on demand or as the rate of occurrence of failure.”
* Systems engineering
  + Specifying, designing, implementing, validating, deploying and maintaining socio-technical systems
  + Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used
  + 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.
* System requirements problems
  + Complex systems are usually developed to address wicked problems
    - Problems that are not fully understood
    - 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 knowing the component structure of the system
* Coding with change
  + Change is inevitable in all large software projects
    - Business changes lead to new and changed system requirements
    - New technologies open up new possibilities for improving implementations
    - Changing platforms require application changes
  + Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality
  + Reducing the costs of rework
    - Change anticipation
    - Change tolerance
  + Coping with changing requirements
    - System prototyping
    - Incremental delivery
  + Benefits of prototyping
* Software prototyping
  + A prototype is an initial version of a system used to demonstrate concepts and try out design options
  + A prototype can be used in:
    - The requirements engineering process to help with requirements elicitation and validation
    - In design processes to explore options and develop a UI design
    - In the testing process to run back-to-back tests.
* Incremental delivery: Fig. 33
  + Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
  + User requirements are prioritized and the highest priority requirements are included in early increments.
  + Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
* Incremental development and delivery
  + Incremental development
  + Incremental delivery
* Boehm’s spiral model
  + Process is represented as a spiral rather than as a sequence of activities with backtracking
  + Each loop in the spiral represents a phase in the process
  + No fixed phases such as specification or design
    - loops in the spiral are chosen depending on what is required
  + Risks are explicitly assessed and resolved throughout the process
* Spiral process model: Fig. 34
* Spiral model sectors
  + Objective setting
    - Specific objectives for the phase are identified
  + Risk assessment and reduction
    - Risks are assessed and activities put in place to reduce the key risks
  + Development and validation
    - A development model for the system is chosen which can be any of the generic models
  + Planning
    - The project is reviewed and the next phase of the spiral is planned
* Spiral model usage
  + Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk driven approach to development
  + In practice, however, the model is rarely used as published for practical software development.
* Process improvement
  + Goals of Process Improvement
    - ???
* Approaches to improvement
  + The process maturity approach
  + The agile approach
* The process improvement cycle: Fig. 35
* Process measurement
  + Wherever possible, quantitative process data should be collected
    - However, where organizations do not have clearly defined process standards this is very difficult as you don’t know what to measure. A process may have to be defined before any measurement is possible.
  + Process measurements should be used to assess process improvements
    - But this does not mean that measurements should drive the improvements. The improvement driver should be the organizational objectives.
* Process metrics
  + Time taken for process activities to be completed
    - E.g. Calendar time or effort to complete an activity or process
  + Resources required for processes or activities
    - E.g. Total effort in person-days
  + Number of occurrences of a particular event
    - E.g. Number of defects discovered.
* Capability maturity levels: Fig. 36
* The SEI capability maturity model
  + Initial
    - Essentially uncontrolled
  + Repeatable
    - Product management procedures defined and used
  + Defined
    - Process management procedures and strategies defined and used
  + Managed
    - Quality management strategies defined and used
  + Optimizing
    - Process improvement strategies defined and used
* Key points
  + Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
  + General process models describe the organization of software processes.
    - Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
  + Requirements engineering is the process of developing a software specification.
  + Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
  + Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
  + Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.
  + Processes should include activities such as prototyping and incremental delivery to cope with change.
  + Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
  + The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
  + The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good software engineering practice.
  + Hybrid development process: Fig. 27

**Requirements Engineering**

* What is requirement?
  + It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
  + This is inevitable as requirements may serve a dual function
    - May be the basis for a bid for a contract - therefore must be open to interpretation
    - May be the basis for the contract itself - therefore must be defined in detail;
    - Both these statements may be called requirements.
  + A requirement is a statement of one of the following:
    - What a system must do
    - A known limitation or constraint on resources or design
    - How well the system must do what it does
  + The first definition is for Functional Requirements
  + The second and third definitions are for Nonfunctional Requirements (NFRs)
* Types of requirement
  + User requirements
  + System requirements
* User and system requirements
  + User requirements definition
    - The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month
  + System requirements specification
    - On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated
    - The system shall generate the report for printing after 17.30 on the last working day of the month
    - A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs
    - If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separated reports shall be created for each dose unit
    - Access to drug cost reports shall be restricted to authorized users as listed on a management access control list
* Readers of different types of requirements specification: Fig. 37
* System stakeholders
  + Any person or organization who is affected by the system in some way and so who has a legitimate interest
  + Stakeholder types
    - End users
    - System managers
    - System owners
    - External stakeholders
* Requirements engineering processes
  + The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements
  + However, there are a number of generic activities common to all processes
    - Requirements elicitation
    - Requirements analysis
    - Requirements validation
    - Requirements management
  + In practice, RE is an iterative activity in which these processes are interleaved
* A spiral view of the requirements engineering process: Fig. 38
* Requirements engineering
  + Inception - ask a set of questions that establish …
    - basic understanding of the problem
    - the people who want a solution
    - the nature of the solution that is desired, and
    - the effectiveness of preliminary communication and collaboration between the customer and the developer
  + Elicitation - elicit requirements from all stakeholders
  + Elaboration - create an analysis model that identifies data, function and behavioral requirements
  + Negotiation - agree on a deliverable system that is realistic for developers and customers
  + Specification - can be any one (or more) of the following:
    - A written document
    - A set of models
    - A formal mathematical
    - A collection of user scenarios (use-cases)
    - A prototype
  + Validation - a review mechanism that looks for
    - errors in content or interpretation
    - areas where clarification may be required
    - missing information – inconsistencies (a major problem when large products or systems are engineered)
    - conflicting or unrealistic (unachievable) requirements
  + Requirements management
* The requirements engineering process: Fig. 28
* Requirements elicitation and analysis
  + Sometimes called requirements elicitation or requirements discovery
  + Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints
  + May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.
* Requirements elicitation
  + Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc
  + Stages include:
    - Requirements discovery
    - Requirements classification and organization
    - Requirements prioritization and negotiation
    - Requirements specification.
* Problems of requirements elicitation
  + ???
* The requirements elicitation and analysis process: Fig. 39
* Users of a requirements document: Fig. 40
* Requirements discovery
  + The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
  + Interaction is with system stakeholders from managers to external regulators.
  + Systems normally have a range of stakeholders.
* Interviewing
  + Formal or informal interviews with stakeholders are part of most RE processes.
  + Types of interview
    - Closed interviews based on pre-determined list of questions
    - Open interviews where various issues are explored with stakeholders.
  + Effective interviewing
    - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
    - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.
* Interviews in practice
  + Normally a mix of closed and open-ended interviewing.
  + Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
  + Interviewers need to be open-minded without pre-conceived ideas of what the system should do
  + You need to prompt the use to talk about the system by suggesting requirements rather than simply asking them what they want.
* Problems with interviews
  + Application specialists may use language to describe their work that isn’t easy for the requirements engineer to understand.
  + Interviews are not good for understanding domain requirements
* Ethnography
  + A social scientist spends a considerable time observing and analyzing how people actually work.
  + People do not have to explain or articulate their work
  + Social and organizational factors of importance may be observed.
  + Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
* Scope of ethnography
  + ???
* Focused ethnography
  + Developed in a project studying the air traffic control process
  + Combines ethnography with prototyping
  + Prototype development results in unanswered questions which focus the ethnographic analysis.
  + The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.
* Ethnography and prototyping for requirements analysis: Fig. 41
* Stories and scenarios
  + Scenarios and user stories are real-life examples of how a system can be used.
  + Stories and scenarios are a description of how a system may be used for a particular task.
  + Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.
* Photo sharing in the classroom (iLearn)
  + Jack is a primary school teacher in Ullapool (a village in northern Scotland). He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing. As part of this, pupils are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history resources site) to access newspaper archives and photographs. However, Jack also needs a photo sharing site as he wants pupils to take and comment on each others’ photos and to upload scans of old photographs that they may have in their families.
  + Jack sends an email to a primary school teachers group, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account. He uses the iLearn setup service to add KidsTakePics to the services seen by the pupils in his class so that when they log in, they can immediately use the system to upload photos from their mobile devices and class computers.
* Scenarios
  + A structured form of user story
  + Scenarios should include
    - A description of the starting situation
    - A description of the normal flow of events
    - A description of what can go wrong
    - Information about other concurrent activities
    - A description of the state when the scenario finishes.
* Uploading photos (iLearn)
  + Initial assumption: A user or a group of users have one or more digital photographs to be uploaded to the picture sharing site. These are saved on either a tablet or laptop computer. They have successfully logged on to KidsTakePics.
  + Normal: The user chooses upload photos and they are prompted to select the photos to be uploaded on their computer and to select the project name under which the photos will be stored. They should also be given the option of inputting keywords that should be associated with each uploaded photo. Uploaded photos are named by creating a conjunction of the user name with the filename of the photo on the local computer
  + On completion of the upload, the system automatically sends an email to the project moderator asking them to check new content and generates an on-screen message to the user that this has been done.
* Uploading photos
  + What can go wrong:
  + No moderator is associated with the selected project. An email is automatically generated to the school administrator asking them to nominate a project moderator. Users should be informed that there could be a delay in making their photos visible.
  + Photos with the same name have already been uploaded by the same user. The user should be asked if they wish to re-upload the photos with the same name, rename the photos or cancel the upload. If they chose to re-upload the photos, the originals are overwritten. If they chose to rename the photos, a new name is automatically generated by adding a number to the existing file name.
  + Other activities: The moderator may be logged on to the system and may approve photos as they are uploaded.
  + System state on completion: User is logged on. The selected photos have been uploaded and assigned a status ‘awaiting moderation’. Photos are visible to the moderator and to the user who uploaded them.
* Feasibility studies
  + A feasibility study decides whether or not the proposed system is worthwhile.
  + A short focused study that checks
    - If the system contributes to organizational objectives
    - If the system can be engineered using current technology and within budget
    - If the system can be integrated with other systems that are used.
* Feasibility study implementation
  + Based on information assessment (what is required), information collection and report writing.
  + Questions for people in the organization
    - What if the system wasn’t implemented?
    - What are current process problems?
    - How will the proposed system help?
    - What will be the integration problems?
    - Is new technology needed? What skills?
    - What facilities must be supported by the proposed system?
* Requirements specification
  + The process of writing down the user and system requirements in a requirements document.
  + User requirements have to be understandable by end-users and customers who do not have a technical background.
  + System requirements are more detailed requirements and may include more technical information.
  + The requirements may be part of a contract for the system development
    - It is therefore important that these are as complete as possible.
* Ways of writing a system requirements specification: Fig. 42
* Requirements and design
  + In principle, requirements should state what the system should do and the design should describe how it does this.
  + In practice, requirements and design are inseparable
    - A system architecture may be designed to structure the requirements
    - The system may inter-operate with other systems that generate design requirements
    - The use of a specific architecture to satisfy nonfunctional requirements may be a domain requirement.
    - This may be the consequence of a regulatory requirement.
* Natural language specification
  + Requirements are written as natural language sentences supplemented by diagrams and tables.
  + Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.
* Guidelines for writing requirements
  + Invent a standard format and use it for all requirements.
  + Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
  + Use text highlighting to identify key parts of the requirement.
  + Avoid the use of computer jargon.
  + Include an explanation (rationale) of why a requirement is necessary.
* Problems with natural language
  + Lack of clarity
    - Precision is difficult without making the document difficult to read
  + Requirements confusion
    - Functional and non-functional requirements tend to be mixed-up.
  + Requirements amalgamation
    - Several different requirements may be expressed together
* Example requirements for the insulin pump software system: Fig. 43
* Structured specifications
  + An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
  + This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.
* Form-based specifications
  + Definition of the function or entity.
  + Description of inputs and where they come from.
  + Description of outputs and where they go to.
  + Information about the information needed for the computation and other entities used.
  + Description of the action to be taken.
  + Pre and post conditions (if appropriate).
  + The side effects (if any) of the function.
* A structured specification of a requirement for an insulin pump: Fig. 44
* Tabular specification
  + Used to supplement natural language.
  + Particularly useful when you have to define a number of possible alternative courses of action.
  + For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.
* Tabular specification of computation for an insulin pump: Fig. 45
* Use cases
  + Use-cases are a kind of scenario that are included in the UML.
  + Use cases identify the actors in an interaction and which describe the interaction itself.
  + A set of use cases should describe all possible interactions with the system.
  + High-level graphical model supplemented by more detailed tabular description (see Chapter 5).
  + UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.
* Use cases for the Mentcare system: Fig. 46
* Types of requirements
  + Functional requirements
  + Non-functional requirements
  + Domain requirements
* Making requirements testable
  + ???
* Functional requirements
  + Describe functionality or system services.
  + Depend on the type of software, expected users and the type of system where the software is used.
  + Functional user requirements may be high-level statements of what the system should do
  + Functional system requirements should describe the system services in detail.
  + A Functional Requirement:
    - is a statement of what a system must do (#1)
    - is measured in “yes” or “no” terms – usually employs the word “shall”
  + Examples:
    - Add Participant
      * “The software shall display an option to add a participant”
    - Summon Operator
      * “The software shall summon the operator if the participant clicks the Operator Help icon.
* The LIBSYS system
  + A library system that provides a single interface to a number of databases of articles in different libraries.
  + Users can search for, download and print these articles for personal study.
* Examples of functional requirements
  + The user shall be able to search either all of the initial set of databases or select a subset from it.
  + The system shall provide appropriate viewers for the user to read documents in the document store.
  + Every order shall be allocated a unique identifier (ORDER\_ID) which the user shall be able to copy to the account’s permanent storage area.
  + Customer searches should take < 3 seconds to complete under heavy loading (up to 20 simultaneous customer connections allowed)
  + Up to 10MB video; 10 MB rental data to be kept at any one time
* Mentcare system: functional requirements
  + A user shall be able to search the appointments lists for all clinics.
  + The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
  + Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.
* Requirements imprecision
  + Problems arise when requirements are not precisely stated.
  + Ambiguous requirements may be interpreted in different ways by developers and users.
  + Consider the term ‘search’ in requirement 1
    - User intention – search for a patient name across all appointments in all clinics;
    - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.
* Requirements imprecision
  + Example
    - Consider the term ‘appropriate viewers’
      * User intention - special purpose viewer for each different document type;
      * Developer interpretation - Provide a text viewer that shows the contents of the document.
* Requirements completeness and consistency
  + In principle, requirements should be both complete and consistent.
    - Complete
      * They should include descriptions of all facilities required.
    - Consistent
      * There should be no conflicts or contradictions in the descriptions of the system facilities.
  + In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.
* Non-functional requirements
  + These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
  + Process requirements may also be specified mandating a particular IDE, programming language or development method.
  + Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.
  + A Non-Functional Requirement:
    - is a known limitation or constraint on resources or design (#2)
    - usually measured in yes/no terms
    - can include documentation, marketing collateral, product localization, legal compliance restrictions
    - typically employs the word “must”
  + Examples:
    - Cost
      * “The retail cost of the software must be between $175 and $199.”
    - Localization
      * “The help file must be released in English, French and Spanish.”
  + A Non-Functional Requirement:
    - is a measure of how well the system must do what it does (#3)
    - Is measured over an Interval or range
    - usually employs the word “must”
    - includes the “ilities” (e.g., quality, reliability, scalability, availability)
* “Non-Functional Requirements (NFRs)”
  + constraints/obligations (non-negotiable)
    - E.g. compatibility with (and reuse of) legacy systems
    - E.g. compliance with interface standards, data formats, communications protocols quality requirements (soft goals)
    - E.g. security, safety, availability, usability, performance, portability,…must be specified
* Types of nonfunctional requirement; Fig. 47
* Non-functional requirements implementation
  + Non-functional requirements may affect the overall architecture of a system rather than the individual components.
    - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
  + A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
    - It may also generate requirements that restrict existing requirements.
* Non-functional classifications
  + Product requirements
    - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
  + Organizational requirements
    - Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.
  + External requirements
    - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.
* Examples: Non-functional requirements
  + Product requirement
    - 8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.
  + Organizational requirement
    - 9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.
  + External requirement
    - 7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.
* Examples of nonfunctional requirements in the Mentcare system: Fig. 48
* Non-functional requirements Goals and requirements
  + Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
  + Goal – A general intention of the user such as ease of use.
  + Verifiable non-functional requirement – A statement using some measure that can be objectively tested.
  + Goals are helpful to developers as they convey the intentions of the system users.
* Examples – Usability Requirements
  + A system goal
    - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.
  + A verifiable non-functional requirement
    - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.
* Metrics for specifying nonfunctional requirements: Fig. 49
* Requirements interaction
  + Conflicts between different non-functional requirements are common in complex systems.
  + Spacecraft system
    - To minimize weight, the number of separate chips in the system should be minimized.
    - To minimize power consumption, lower power chips should be used.
    - However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?
* Domain requirements
  + The system’s operational domain imposes requirements on the system.
    - For example, a train control system has to take into account the braking characteristics in different weather conditions.
  + Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
  + If domain requirements are not satisfied, the system may be unworkable.
* Domain requirements problems
  + Understandability
    - Requirements are expressed in the language of the application domain
    - This is often not understood by software engineers developing the system
  + Implicitness
    - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.
* Requirements validation
  + Concerned with demonstrating that the requirements define the system that the customer really wants.
  + Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
* Requirements checking
  + Validity. Does the system provide the functions which best support the customer’s needs?
  + Consistency. Are there any requirements conflicts?
  + Completeness. Are all functions required by the customer included?
  + Realism. Can the requirements be implemented given available budget and technology
  + Verifiability. Can the requirements be checked?
* Requirements validation techniques
  + Requirements reviews
  + Prototyping
  + Test-case generation
* Requirements reviews
  + Regular reviews should be held while the requirements definition is being formulated.
  + Both client and contractor staff should be involved in reviews.
  + Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.
* Review checks
  + Verifiability. Is the requirement realistically testable?
  + Comprehensibility. Is the requirement properly understood?
  + Traceability. Is the origin of the requirement clearly stated?
  + Adaptability. Can the requirement be changed without a large impact on other requirements?
* Requirements Examples
* Reviewing Requirements for Quality
* Example #1: "The product shall provide status messages at regular intervals not less than every 60 seconds."
  + Incomplete: what are the status messages and how are they supposed to be displayed to the user?
  + Ambiguities: What part of "the product" are we talking about? Is the interval between status messages really supposed to be at least 60 seconds, so showing a new message every 10 years is okay?
  + The word "every" just confuses the issue.
  + So, the requirement is not verifiable.
  + 1. Status Messages.
    - 1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.
    - 1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed.
    - 1.3. A message shall be displayed when the background task is completed
    - 1.4. An error message shall be displayed if the background task has stalled."
* Example 2: "The product shall switch between displaying and hiding non-printing characters instantaneously."
  + Not feasible: Computers cannot do anything instantaneously.
  + Incomplete: because it does not state the conditions that trigger the state switch.
  + Ambiguous: what is the scope of the display change within the document: selected text, the entire document, or something else?
  + Ambiguous: Are "non-printing" characters the same as hidden text, or are they attribute tags or control characters of some kind?
  + As a result of these problems this requirement cannot be verified.
* "The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition."
* Example #3: "The HTML Parser shall produce an HTML markup error report which allows quick resolution of errors when used by HTML novices."
* Activities: Requirements Analysis
  + Requirements Gathering (Elicitation)
    - Understand what the product must do, what the requirements of the product are
  + Requirements Specification
    - Enumerate (list) each function the product must do and the constraints under which the function must be done
  + Requirements Analysis
    - Analyze the requirements of the system and verify that the list is complete, unambiguous, consistent and correct
* Requirements Analysis
  + A requirement is a feature (ability) of the system that client requires
  + Requirements analysis seeks to assess (analyze) and specify the behavior of the final system as a set of requirements
  + Requirements analysis results in a complete, consistent, correct, and unambiguous specification of the requirements
  + Requirements analysis can be iteratively carried out or done in a top-down fashion
* Definitions
  + Complete: All features of interest to the client are described by the requirements
  + Consistent: No requirement contradicts any other requirement in the specification
  + Unambiguous: The requirements cannot be interpreted in multiple ways.
  + Correct: The requirements describe all features of interest to the client, but not extra or superfluous features
* Purpose of Requirements Analysis
  + Tool used by software developers to understand
    - The functions of the application to be developed
    - What the client/user expects the application to do
    - The importance of each function to the client/user
    - Necessary details of the application domain
  + Tool also helps the clients/users to understand
    - What their own requirements are
    - The feasibility and costs of some features that may be difficult to implement (this may change the user’s prioritization of features)
* Importance of Requirements Analysis
  + Frederick Brooks: “The hardest single part of building a software system is deciding precisely what to build”
  + Barry Boehm: by investing more up-front effort in verifying and validating the software requirements, software developers will see reduced integration and test costs, as well as higher software reliability and maintainability
* Why requirements analysis? Fig. 50
  + Early software engineering studies tried to identify the sources of problems and errors in large software development projects
* Requirements management
  + Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
  + Requirements are inevitably incomplete and inconsistent
    - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
    - Different viewpoints have different requirements and these are often contradictory.
  + New requirements emerge as a system is being developed and after it has gone into use.
  + You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.
* Changing requirements
  + Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
    - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.
* Requirement Change
  + The business and technical environment of the system always changes after installation.
    - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
  + The people who pay for a system and the users of that system are rarely the same people.
    - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.
  + The priority of requirements from different viewpoints changes during the development process.
  + System customers may specify requirements from a business perspective that conflict with end-user requirements.
  + The business and technical environment of the system changes during its development.
* Requirements change management; Fig. 52
  + Should apply to all proposed changes to the requirements.
  + Principal stages
    - Problem analysis. Discuss requirements problem and propose change
    - Change analysis and costing. Assess effects of change on other requirements
    - Change implementation. Modify requirements document and other documents to reflect change.
  + Deciding if a requirements change should be accepted
    - Problem analysis and change specification
      * During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
    - Change analysis and costing
      * The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
    - Change implementation
      * The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.
* Requirements evolution: Fig. 51
* Requirements management
  + Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
  + New requirements emerge as a system is being developed and after it has gone into use.
  + You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.
* Requirements management planning
  + Establishes the level of requirements management detail that is required.
  + Requirements management decisions:
    - Requirements identification. Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
    - A change management process. This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
    - Traceability policies. These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
    - Tool support Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.
  + During the requirements engineering process, you have to plan:
    - Requirements identification
      * How requirements are individually identified
    - A change management process
      * The process followed when analyzing a requirements change
    - Traceability policies
      * The amount of information about requirements relationships that is maintained;
    - CASE tool support
      * The tool support required to help manage requirements change;
* Enduring and volatile requirements
* Enduring requirements. Stable requirements derived from the core activity of the customer organization. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models
* Volatile requirements. Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy
* Traceability
  + Traceability is concerned with the relationships between requirements, their sources and the system design
  + Source traceability
    - Links from requirements to stakeholders who proposed these requirements;
  + Requirements traceability
    - Links between dependent requirements
  + Design traceability
    - Links from the requirements to the design;
* A traceability matrix: Fig. 53
* The requirements document
  + The requirements document is the official statement of what is required of the system developers.
  + Should include both a definition of user requirements and a specification of the system requirements.
  + It is NOT a design document. As much as possible, it should set of WHAT the system should do rather than HOW it should do it
* Requirements document variability
  + Information in requirements document depends on type of system and the approach to development used.
  + Systems developed incrementally will, typically, have less detail in the requirements document.
  + Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.
* The structure of a requirements document: Fig. 54
* A structured specification of a requirement for an insulin pump: Fig. 44
* Tabular specification
  + Used to supplement natural language.
  + Particularly useful when you have to define a number of possible alternative courses of action.
  + For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.
* Tabular specification of computation for an insulin pump: Fig. 45
* Two requirements prioritization scales: Fig. 55
* Alternative wording
  + Critical
  + Important
  + Useful
* Example: Requirement Prioritization: Fig. 56
* Example: Functional Requirements
  + 1. Update Riding Information
    - An election official can update any or all of the following information for any chosen riding:
      * The names of a candidate
        + The party affiliations of that candidate
      * A map of the riding
      * The number of eligible voters in the riding
      * The number of representatives for the riding
    - An elections official will update one riding at a time
    - An elections official may preview her changes before saving them in the riding repository
* Clarifying the Requirements:
  + Other questions need to be answered to specify detailed requirements. For example:
    - What sort of Human computer interface is needed?
    - Are there any other pieces of data that need to be recorded or updated?
    - What is the maximum number of candidates in a riding?
    - Can the same candidate run in more than one riding?
    - And so on
  + Answers to such questions may raise additional questions so this is by nature an iterative process
* Requirements and design
  + In principle, requirements should state what the system should do and the design should describe how it does this.
  + In practice, requirements and design are inseparable
    - A system architecture may be designed to structure the requirements;
    - The system may inter-operate with other systems that generate design requirements;
    - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
    - This may be the consequence of a regulatory requirement.
* Problem statement format: Fig. 57
* Five-Step Problem Analysis Technique
  + 1. Gain agreement on the problem definition.
  + 2. Understand the root causes of the problem.
  + 3. Identify the stakeholders and the users whose collective judgment will ultimately determine the success or failure of your system.
  + 4. Determine where the boundaries of the solution are likely to be found.
  + 5. Understand the constraints that will be imposed on your team and on the solution.
* Availability: Fig. 58 Fig. 59
  + Total down time = maintenance + upgrades + failure
* Fundamental characteristics of a quality SRS: Fig. 60
* Also…
  + Writing Software Requirements Specifications by Donn Le Vie, Jr. – http://www.techwrl.com/techwhirl/magazine/writing/softwarerequir ementspecs.html
  + OpenSeminar in Software Engineering – <http://openseminar.org/se/screen.do>
* IEEE Standard SW Requirements Document
* IEEE requirements standard
* Defines a generic structure for a requirements document that must be instantiated for each specific system.
  + Introduction.
  + General description.
  + Specific requirements.
  + Appendices.
  + Index.
* Requirements document structure
  + Preface
  + Introduction
  + Glossary
  + User requirements definition
  + System architecture
  + System requirements specification
  + System models
  + System evolution
  + Appendices
  + Index
* IEEE 830 Std Software Requirement Specification Section Outline (simplified)
  + Introduction
    - Purpose
    - Scope
    - Definitions, Acronyms, and Abbreviations
    - References
    - Overview
  + General Description
    - Product Perspective
    - Product Functions
    - User Characteristics
    - General Constraints
    - Assumptions and Dependencies
  + Specific Requirements
    - Functional Requirements
    - External Interfaces Requirements
    - Performance Requirements
    - Design Constraints
    - Attributes
    - Other requirements
  + Appendixes
  + Index
* Software Requirements Specification (SRS)
  + Several standards organizations (including the IEEE) have identified nine topics that must be addressed when designing and writing an SRS:
    - Interfaces
    - Functional Capabilities
    - Performance Levels
    - Data Structures/Elements
    - Safety
    - Reliability
    - Security/Privacy
    - Quality
    - Constraints and Limitations
* SRS Topics (from text)
  + 1. Introduction, purpose/scope, overview
  + 2. Current system
  + 3. Proposed system
    - 3.1 Overview
    - 3.2 Functional Requirements
    - 3.3 Non-Functional Requirements
    - 3.4 System Models
      * 3.4.1 Scenarios
      * 3.4.2 Use Case Model
      * 3.4.3 Object Model
        + 3.4.3.1 Data Dictionary
        + 3.4.3.2 Class Diagram
      * 3.4.4 Dynamic Model
      * 3.4.5 UI mockups and quick reference
  + 4. Glossary
* IEEE SRS Standard
  + 0. Table of Contents
  + 1. Introduction
    - Purpose and scope of document
    - glossary, references, overview
  + 2. General Description (introduction to product)
    - Product functionality, Comparison to similar products
    - User Characteristics, system context
    - External Interface (UI, HW, SW, CMNS)
  + 3. Specific Requirements
    - Functional Requirements
    - Non-Functional Requirements
* Software interfaces: Fig. 61

**Design concepts and system modeling**

“Software design is an iterative process

through which requirements are translated

into a ‘blueprint’ for constructing the

software.” --Roger Pressman

* Why is Design so Difficult?
  + *Analysis:* Focuses on the application domain
  + *Design:* Focuses on the solution domain
    - Design knowledge is a moving target
    - The reasons for design decisions are changing very rapidly
  + Halftime knowledge in software engineering: About 3-5 years • Cost of hardware rapidly sinking
  + “Design window”:
    - Time in which design decisions have to be made
  + Technique
    - Time-boxed prototyping
* The purpose of system design: Fig. 62, 63, 64
  + Bridging the gap between desired and existing system in a manageable way
  + Use Divide and Conquer
    - We model the new system to be developed as a set of subsystems
* System modeling
  + System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
  + Detailed requirements are often best represented as “models” instead of natural language
    - More precise
    - Often graphical
    - May have CASE tool support
  + Importantly, models are abstractions of a system, and so leave out non-essential details
* Existing and planned system models
  + Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
  + Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
* System perspectives
  + An external perspective , where you model the context or environment of the system.
  + An interaction perspective , where you model the interactions between a system and its environment, or between the components of a system.
  + A structural perspective , where you model the organization of a system or the structure of the data that is processed by the system.
  + A behavioral perspective , where you model the dynamic behavior of the system and how it responds to events.
* UML diagram types
  + Activity diagrams , which show the activities involved in a process or in data processing .
  + Use case diagrams , which show the interactions between a system and its environment.
  + Sequence diagrams , which show interactions between actors and the system and between system components.
  + Class diagrams , which show the object classes in the system and the associations between these classes.
  + State diagrams , which show how the system reacts to internal and external events.
  + Introduction to the Diagrams of UML 2.0
    - <http://www.agilemodeling.com/essays/umlDiagrams.htm>
* Use of graphical models
  + As a means of facilitating discussion about an existing or proposed system
    - Incomplete and incorrect models are OK as their role is to support discussion.
  + As a way of documenting an existing system
    - Models should be an accurate representation of the system but need not be complete.
  + As a detailed system description that can be used to generate a system implementation
    - Models have to be both correct and complete.
* High Level Architecture
  + Such “lines and boxes” diagrams show the basic system context and relations
  + Not usually directly useful to programming, but good as a system overview and as an introduction to what you are doing
  + E.g. figure 8.1 in the textbook
* Fig 8.1: The context of an ATM system: Fig. 65
* Context models
  + Context models are used to illustrate the operational context of a system they show what lies outside the system boundaries.
  + Social and organizational concerns may affect the decision on where to position system boundaries.
  + Architectural models show the system and its relationship with other systems.
* System boundaries
  + System boundaries are established to define what is inside and what is outside the system.
    - They show other systems that are used or depend on the system being developed.
  + The position of the system boundary has a profound effect on the system requirements.
  + Defining a system boundary is a political judgment
    - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.
* Process perspective
  + Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
  + Process perspective show the overall process and the processes that are supported by the system.
  + Data flow models may be used to show the processes and the flow of information from one process to another.
* Process model of involuntary detention: Fig. 66, 67
* Interaction models
  + Modeling user interaction is important as it helps to identify user requirements.
  + Modeling system to system interaction highlights the communication problems that may arise.
  + Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
  + Use case diagrams and sequence diagrams may be used for interaction modeling.
* Two related approaches to interaction modeling
  + Use case modeling, which is mostly used to model interactions between a system and external actors (users or other systems).
  + Sequence diagrams, which are used to model interactions between system components, although external agents may also be included.
* Use case modeling
  + Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
  + Each use case represents a discrete task that involves external interaction with a system.
  + Actors in a use case may be people or other systems.
  + Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.
* Transfer data use case: Fig. 68
  + A use case in the MHC PMS
* Tabular description of the “transfer data” use-case: Fig. 69
* Use cases in the MHC-PMS involving the role “Medical Receptionist”: Fig. 70
* Sequence diagrams
  + Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
  + A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
  + The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
  + Interactions between objects are indicated by annotated arrows.
* Sequence diagram for View patient information: Fig. 71
* Sequence diagram for Transfer Data: Fig. 72
* 5.3 Structural models
  + Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
  + Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
  + You create structural models of a system when you are discussing and designing the system architecture.
* Class diagrams
  + Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
  + An object class can be thought of as a general definition of one kind of system object.
  + An association is a link between classes that indicates that there is some relationship between these classes.
  + When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.
* UML classes and association: Fig. 73
* P190