**Chapter 1 - Introduction**

**Topics covered**

* Professional software development
  + What is meant by software engineering.
* Software engineering ethics
  + A brief introduction to ethical issues that affect software engineering.
* Case studies
  + An introduction to three examples that are used in later chapters in the book.

**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.
* Expenditure on software represents a   
  significant fraction of GNP in all developed countries.

**Software costs**

* Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
* Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
* Software engineering is concerned with cost-effective software development.

**Software project failure**

* *Increasing system complexity*
  + As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.
* *Failure to use software engineering methods*
  + It is fairly easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

**Professional software development**

**Frequently asked questions about software engineering**

|  |  |
| --- | --- |
| **Question** | **Answer** |
| What is software? | Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market. |
| What are the attributes of good software? | Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable. |
| What is software engineering? | Software engineering is an engineering discipline that is concerned with all aspects of software production. |
| What are the fundamental software engineering activities? | Software specification, software development, software validation and software evolution. |
| What is the difference between software engineering and computer science? | Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software. |
| What is the difference between software engineering and system engineering? | System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process. |
| What are the key challenges facing software engineering? | Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software. |
| What are the costs of software engineering? | Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs. |
| What are the best software engineering techniques and methods? | While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another. |
| What differences has the web made to software engineering? | The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse. |

**Software products**

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

**Product specification**

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

**Essential attributes of good software**

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

**Software engineering**

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

**Importance of software engineering**

* More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
* It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

**Software process activities**

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

**General issues that affect software**

* Heterogeneity
  + Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
* Business and social change
  + Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
* Security and trust
  + As software is intertwined with all aspects of our lives, it is essential that we can trust that software.
* Scale
  + Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

**Software engineering diversity**

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

**Application types**

* Stand-alone applications
  + These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
* Interactive transaction-based applications
  + Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
* Embedded control systems
  + These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
* Batch processing systems
  + These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
* Entertainment systems
  + These are systems that are primarily for personal use and which are intended to entertain the user.
* Systems for modeling and simulation
  + These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
* Data collection systems 
  + These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
* Systems of systems
  + These are systems that are composed of a number of other software systems.

**Software engineering fundamentals**

* Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  + Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  + Dependability and performance are important for all types of system.
  + Understanding and managing the software specification and requirements (what the software should do) are important.
  + Where appropriate, you should reuse software that has already been developed rather than write new software.

**Internet software engineering**

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

**Web-based software engineering**

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

**Web software engineering**

* Software reuse
  + Software reuse is the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.
* Incremental and agile development
  + Web-based systems should be developed and delivered incrementally. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.
* Service-oriented systems
  + Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.
* Rich interfaces
  + Interface development technologies such as AJAX and HTML5 have emerged that support the creation of rich interfaces within a web browser.

**Software engineering ethics**

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

**Issues of professional responsibility**

* Confidentiality
  + Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
* Competence
  + Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.
* Intellectual property rights
  + Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
* Computer misuse
  + Software engineers should not use their technical skills to misuse other people’s computers. Computer misuse ranges from relatively trivial (game playing on an employer’s machine, say) to extremely serious (dissemination of viruses).

**ACM/IEEE Code of Ethics**

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

**Rationale for the code of ethics**

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

**The ACM/IEEE Code of Ethics**

**Software Engineering Code of Ethics and Professional Practice**

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

**PREAMBLE**

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

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

**Ethical principles**

1. PUBLIC - Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

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

6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

**Case studies**

**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.
* A personal insulin pump
  + An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
* A mental health case patient management system
  + Mentcare. A system used to maintain records of people receiving care for mental health problems.
* A wilderness weather station
  + A data collection system that collects data about weather conditions in remote areas.
* iLearn: a digital learning environment
  + A system to support learning in schools

**Insulin pump control system**

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

**Insulin pump hardware architecture**

1.4 InsulinPumpHW.eps

**Activity model of the insulin pump**

1.5 InsulinPumpActDiag.eps

**Essential high-level requirements**

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

**Mentcare: A patient information system for mental health care**

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

**Mentcare**

* Mentcare is an information system that is intended for use in clinics.
* It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
* When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

**Mentcare goals**

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

**The organization of the Mentcare system**

1.6 MHC-PMS.eps

**Key features of the Mentcare system**

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

**Mentcare system concerns**

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

**Wilderness weather station**

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

**The weather station’s environment**

1.7 WeatherStationEnv.eps

**Weather information system**

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

**Additional software functionality**

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

**iLearn: A digital learning environment**

* A digital learning environment is a framework in which a set of general-purpose and specially designed tools for learning may be embedded plus a set of applications that are geared to the needs of the learners using the system.
* The tools included in each version of the environment are chosen by teachers and learners to suit their specific needs.
  + These can be general applications such as spreadsheets, learning management applications such as a Virtual Learning Environment (VLE) to manage homework submission and assessment, games and simulations.

**Service-oriented systems**

* The system is a service-oriented system with all system components considered to be a replaceable service.
* This allows the system to be updated incrementally as new services become available.
* It also makes it possible to rapidly configure the system to create versions of the environment for different groups such as very young children who cannot read, senior students, etc.

**iLearn services**

* *Utility services* that provide basic application-independent functionality and which may be used by other services in the system.
* *Application services* that provide specific applications such as email, conferencing, photo sharing etc. and access to specific educational content such as scientific films or historical resources.
* *Configuration services* that are used to adapt the environment with a specific set of application services and do define how services are shared between students, teachers and their parents.

**iLearn architecture**

1.8 iLearn architecture.eps

**iLearn service integration**

* *Integrated services* are services which offer an API (application programming interface) and which can be accessed by other services through that API.  Direct service-to-service communication is therefore possible.
* *Independent services* are services which are simply accessed through a browser interface and which operate independently of other services. Information can only be shared with other services through explicit user actions such as copy and paste; re-authentication may be required for each independent service.

**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 standards of behaviour expected of their members.

**Chapter 2 – Software Processes**

**Topics covered**

* Software process models
* Process activities
* Coping with change
* Process improvement

**The software process**

* A structured set of activities required to develop a   
  software system.
* 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.
* A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

**Software process descriptions**

* When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
* 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**

* Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
* In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
* In practice, most practical processes include elements of both plan-driven and agile approaches.
* There are no right or wrong software processes.

**Software process models**

* The waterfall model
  + Plan-driven model. Separate and distinct phases of specification and development.
* Incremental development
  + Specification, development and validation are interleaved. May be plan-driven or agile.
* Integration and configuration
  + The system is assembled from existing configurable components. May be plan-driven or agile.
* In practice, most large systems are developed using a process that incorporates elements from all of these models.

**The waterfall model**

2.1.Waterfall-model.eps

**Waterfall model phases**

* 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
* The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

**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.
* The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  + In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

**Incremental development**

2.2 Incremental-dev.eps

**Incremental development benefits**

* The cost of accommodating changing customer requirements is reduced.
  + The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
* It is easier to get customer feedback on the development work that has been done.
  + Customers can comment on demonstrations of the software and see how much has been implemented.
* More rapid delivery and deployment of useful software to the customer is possible.
  + Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

**Incremental development problems**

* The process is not visible.
  + Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
* System structure tends to degrade as new increments are added*.* 
  + Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

**Integration and configuration**

* Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems).
* Reused elements may be configured to adapt their behaviour and functionality to a user’s requirements
* Reuse is now the standard approach for building many types of business system
  + Reuse covered in more depth in Chapter 15.

**Types of reusable software**

* Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
* Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
* Web services that are developed according to service standards and which are available for remote invocation.

**Reuse-oriented software engineering**

2.3 Reuse oriented SE.eps

**Key process stages**

* Requirements specification
* Software discovery and evaluation
* Requirements refinement
* Application system configuration
* Component adaptation and integration

**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

**Process activities**

* Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
* The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
* For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

**The requirements engineering process**

2.4 RE-process.eps

**Software specification**

* The process of establishing what services are required and the constraints on the system’s operation and development.
* Requirements engineering process
  + 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

**Software design and implementation.**

* The process of converting the system specification into an executable system.
* Software design
  + Design a software structure that realises 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**

2.5 Design-process.eps

**Design activities**

* *Architectural design,* where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
* *Database design,* where you design the system data structures and how these are to be represented in a database.
* *Interface design,* where you define the interfaces between system components.
* *Component selection and design,* where you search for reusable components. If unavailable, you design how it will operate.

**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.

**Stages of testing**

2.6 Testing-process.eps

**Testing stages**

* Component 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.
* Customer 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)**

2.7 Testing-phases.eps

**Software evolution**

* 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.

**System evolution**

2.8 System evolution.eps

**Coping 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-analysing requirements) as well as the costs of implementing new functionality

**Reducing the costs of rework**

* Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.
  + For example, a prototype system may be developed to show some key features of the system to customers.
* Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  + This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

**Coping with changing requirements**

* System prototyping, where a version of the system or part of the system is developed quickly to check the customer’s requirements and the feasibility of design decisions. This approach supports change anticipation.
* Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

**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.

**Benefits of prototyping**

* Improved system usability.
* A closer match to users’ real needs.
* Improved design quality.
* Improved maintainability.
* Reduced development effort.

**The process of prototype development**

2.9 PrototypeProcess.eps

**Prototype development**

* May be based on rapid prototyping languages or tools
* May involve leaving out functionality
  + Prototype should focus on areas of the product that are not well-understood;
  + Error checking and recovery may not be included in the prototype;
  + Focus on functional rather than non-functional requirements such as reliability and security

**Throw-away prototypes**

* Prototypes should be discarded after development as they are not a good basis for a production system:
  + It may be impossible to tune the system to meet non-functional requirements;
  + Prototypes are normally undocumented;
  + The prototype structure is usually degraded through rapid change;
  + The prototype probably will not meet normal organisational quality standards.

**Incremental delivery**

* 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 prioritised 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
  + Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  + Normal approach used in agile methods;
  + Evaluation done by user/customer proxy.
* Incremental delivery
  + Deploy an increment for use by end-users;
  + More realistic evaluation about practical use of software;
  + Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

**Incremental delivery**

2.10 Incremental-delivery.eps

**Incremental delivery advantages**

* Customer value can be delivered with each increment so system functionality is available earlier.
* Early increments act as a prototype to help elicit requirements for later increments.
* Lower risk of overall project failure.
* The highest priority system services tend to receive the most testing.

**Incremental delivery problems**

* Most systems require a set of basic facilities that are used by different parts of the system.
  + As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
* The essence of iterative processes is that the specification is developed in conjunction with the software.
  + However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

**Process improvement**

* Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
* Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

**Approaches to improvement**

* The process maturity approach, which focuses on improving process and project management and introducing good software engineering practice.
  + The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.
* The agile approach, which focuses on iterative development and the reduction of overheads in the software process.
  + The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

**The process improvement cycle**

26.3 Process improvement.eps

**Process improvement activities**

* *Process measurement* 
  + You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.
* *Process analysis*
  + The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed.
* *Process change* 
  + Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

**Process measurement**

* Wherever possible, quantitative process data   
  should be collected
  + However, where organisations 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**

26.10 StagesCMMI.eps

**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
* Optimising
  + 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.