INF3705

Activity 1.3

As an information system developer, explain the following process activities to your co-workers: specification, development, validation and evolution

Specification: Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

Development: The implementation stage of software development is the process of converting a system specification into an executable system.

Validation: Verification and validation (V&V) is intended to show that a system both conforms to its specification and that it meets the expectations of the system customer.

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.

The principles of agile methods include the following:

Principle

Customer involvement.
Incremental delivery.
People not process.
Maintain simplicity.

2.2.2 What are the differences between functional and non-functional software requirements?

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations
  - May state what the system should not do
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, and so forth
  - Often apply to the system as a whole rather than individual features or services
- Domain requirements
  - Constraints on the system from the domain of operation

Example of Functional requirements for the MHc-PMS

- A user shall be able to search the appointment lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments on the specific day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Metrics for specifying non-functional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Development testing

Development testing includes all testing activities that are carried out by the team developing the system.

- **Unit testing**– where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
- **Component testing** – where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
- **System testing** – where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

Importance of dependability

- System failures may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high if the failure leads to economic losses or physical damage.
- Undependable systems may cause information loss with a high consequent recovery cost.

End of chapter review questions

Chapter 1

1. What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?

Answer:
The essential difference is that in generic software product development, the specification is owned by the product developer. For custom product development, the specification is owned and controlled by the customer. The implications of this are significant – the developer can quickly decide to change the specification in response to some external change (e.g. a competing product) but, when the customer owns the specification, changes have to be negotiated between the customer and the developer and may have contractual implications. For users of generic products, this means they have no control over the software specification so cannot control the evolution of the product. The developer may decide to include/exclude features and change the user interface. This could have implications for the user's business processes and add extra training costs when new versions of the system are installed. It also may limit the customer's flexibility to change their own business processes.

2. Based on your own knowledge of some of the application types discussed, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.

Answer:
Different application types require the use of different development techniques for a number of reasons:

1. Costs and frequency of change. Some systems (such as embedded systems in consumer devices) are extremely expensive to change; others must change frequently in response to changing requirements (e.g. business systems). Systems which are very expensive to change need extensive upfront analysis to ensure that the requirements are consistent and extensive validation to ensure that the system meets its specification. This is not cost effective for systems that change very rapidly.

2. The most important 'non-functional' requirements. Different systems have different priorities for non-functional requirements. For example, a real-time control system in an aircraft has safety as its principal priority; an interactive game has responsiveness and usability as its priority. The techniques used to achieve safety are not required for interactive gaming; the extensive UI design required for games is not needed in safety-critical control systems.

3. The software lifetime and delivery schedule. Some software systems have a relatively short lifetime (many web-based systems), others have a lifetime of tens of years (large command and control systems). Some systems have to be delivered quickly if they are to be useful. The techniques used to develop short-lifetime, rapid delivery systems (e.g. use of scripting languages, prototyping, etc.) are inappropriate for long-lifetime systems which require techniques that allow for long-term support such as design modelling.

3. Apart from the challenges of heterogeneity, business and social change and trust and security, identify other problems and challenges that software engineering is likely to face in the 21st century (hint: think about the environment).

Answer:
Problems and challenges for software engineering

- Developing systems for multicultural use
- Developing systems that can be adapted quickly to new business needs (or market requirement)
- Designing systems for outsourced development
- Developing systems that are resistant to attack (information security)
- Developing systems that can be adapted and configured by end-users (user can design or assemble what they want)
- Developing systems that are energy-efficient.

1. Explain why there are fundamental ideas of software engineering that apply to all types of software systems.

Because of all software systems have common quality attributes, including availability, modifiability, performance, security and safety, testability and usability, the fundamental software ideas provides common solutions or tactics to support those qualities.

What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

What is a software process?

A set of activities whose goal is the development or evolution of software.
Generic/Basic activities in all software processes are:

- Specification - what the system should do and its development constraints
- Development - production of the software system
- Validation - checking that the software is what the customer wants
- Evolution - changing the software in response to changing demands.

What is a software process model?

A simplified representation of a software process, presented from a specific perspective.
Examples of process perspectives are

- Workflow perspective - sequence of activities;
- Data-flow perspective - information flow;
- Role/action perspective - who does what.

Generic process models

- Waterfall;
- Iterative development;
- Component-based software engineering

What are software engineering methods?

Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
Model descriptions

- Descriptions of graphical models which should be produced;

Rules

- Constraints applied to system models;

Recommendations

- Advice on good design practice;

Process guidance

- What activities to follow.

What is CASE (Computer-Aided Software Engineering)

Software systems that are intended to provide automated support for software process activities.

What are the attributes of good software?

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable. Essential product attributes are:

- Maintainability -Software must evolve to meet changing needs;
- Dependability - Software must be trustworthy;
- Efficiency - Software should not make wasteful use of system resources;
- Acceptability - Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

Other attributes of good software

- reusability (can it be reused in other applications),
- distributability (can it be distributed over a network of processors),
- portability (can it operate on multiple platforms, ie different operation system)
- inter-operability (can it work with a wide range of other software systems, ie different program language).

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.

What are methods?

Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

Please give three type of software process model? And what the software development model that most software process models are based on?

1. Three types: Workflow model, dataflow/activity model and role/action model.
2. Waterfall approach, Iterative development, and Component-based software engineering.

Chapter 2

2.1. Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:
A system to control anti-lock braking in a car
A virtual reality system to support software maintenance
A university accounting system that replaces an existing system
An interactive travel planning system that helps users plan journeys with the lowest environmental impact

Answer:

a) *Anti-lock braking system* - This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analysed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.

b) *Virtual reality system* - This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.

c) *University accounting system* - This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.

d) *Interactive travel planning system* - System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

2.2. Explain why incremental development is the most effective approach for developing business software systems. Why is this model less appropriate for real-time systems engineering?

Answer:
Business software systems are usually complex, software intensive, and frequently being changed when business goals or processes are changed. So incremental development is better.
Real-time systems usually involve many hardware components which are not easy to change and cannot be incremental. Also real-time systems usually safety critical which needed be built based on well planned process.

2.3. Explain why it is essential to have two separate requirements engineering activities in the process.

Answer:
In a reuse based process, you need two requirements engineering activities because it is essential to adapt the system requirements according to the capabilities of the system/components to be reused.
These activities are:

- An initial activity where you understand the function of the system and set out broad requirements for what the system should do. These should be expressed in sufficient detail that you can use them as a basis for deciding if a system/component satisfies some of the requirements and so can be reused.
- Once systems/components have been selected, you need a more detailed requirements engineering activity to check that the features of the reused software meet the business needs and to identify changes and additions that are required.

2.4. Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

Answer:
There is a fundamental difference between the user and the system requirements that mean they should be considered separately.

- The user requirements are intended to describe the system's functions and features from a user perspective and it is essential that users understand these requirements. They should be expressed in natural language and may not be expressed in great detail, to allow some implementation flexibility. The people involved in the process must be able to understand the user's environment and application domain.

- The system requirements are much more detailed than the user requirements and are intended to be a precise specification of the system that may be part of a system contract. They may also be used in situations where development is outsourced and the development team need a complete specification of what should be developed. The system requirements are developed after user requirements have been established.

2.5. Describe the main activities in the software design process and the outputs of these activities. Using a diagram, show possible relationships between the outputs of these activities.

Answer:

Main activities:
Architectural design – Where you identify the overall structure of the system
Interface design – Where you define the interfaces between system components
Component design – Where you take each system component and design how it will operate
Database design – Where you design the system data structures and how these are to be represented in a database.

Outputs:
System architecture
Database specification
Interface specification
Component specification

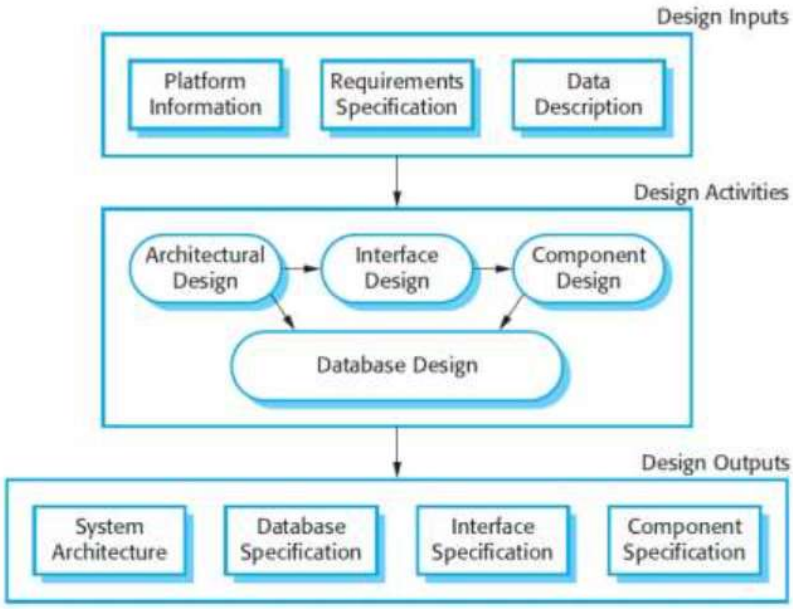

Figure 2.5 A general model of the design process

2.6. Explain why change is inevitable in complex systems and give examples (apart from prototyping and incremental delivery) of software process activities that help predict changes and make the software being developed more resilient to change.

Answer:

Systems must change because as they are installed in an environment the environment adapts to them and this adaptation naturally generates new/different system requirements. Furthermore, the system's environment is dynamic and constantly generates new requirements as a consequence of changes to the business, business goals and business policies. Unless the system is adapted to reflect these requirements, its facilities will become out-of-step with the facilities needed to support the business and, hence, it will become less useful.

Examples of process activities that support change are:

1. Recording of requirements rationale so that the reason why a requirement is included is known. This helps with future change.
2. Requirements traceability that shows dependencies between requirements and between the requirements and the design/code of the system.
3. Design modelling where the design model documents the structure of the software.
4. Code refactoring that improves code quality and so makes it more amenable to change.

2.7. Explain why systems developed as prototypes should not normally be used as production systems.

Answer:

Prototypes should not normally be used as production systems for the following reasons:

a) The user interface may be minimal and not intuitive.
b) There may be no error detecting or handling code.
c) Such error messages as there are will likely be vague.
d) Generally, prototypes are not viewed as high quality products, but just tools to aid the development process.

2.8. Explain why Boehm's spiral model is an adaptable model that can support both change avoidance and change tolerance activities. In practice, this model has not been widely used. Suggest why this might be the case.

Answer:
Boehm's spiral model combines change avoidance with change tolerance. It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks.
It is not widely used as it might be difficult to apply in some contexts.

2.9. What are the advantages of providing static and dynamic views of the software process as in the Rational Unified Process?

Answer:
An approach to process modeling which is simply based on static activities, such as requirements, implementation, etc. forces these activities to be set out in a sequence which may not reflect the actual way that these are enacted in any one organization.

In most cases, the static activities are actually interleaved so a sequential process model does not accurately describe the process used. By separating these from the dynamic perspective i.e. the phases of development, you can then discuss how each of these static activities may be used at each phase of the process.

Furthermore, some of the activities that are required during some of the system phases are in addition to the central static activities. These vary from one organization to another and it is not appropriate to impose a particular process in the model.

2.10. Historically, the introduction of technology has caused profound changes in the labour market and, temporarily at least, displaced people from jobs. Discuss whether the introduction of extensive process automation is likely to have the same consequences for software engineers. If you don't think it will, explain why not. If you think that it will reduce job opportunities, is it ethical for the engineers affected to passively or actively resist the introduction of this technology?

Answer:
There are obviously different views here and a lot depends on the development of CASE technology in the future. A major difference between the introduction of CASE technology and, for example, the introduction of CAD technology which made draftsmen redundant, is that the routine elements in the design and development of software are relatively minor parts of the whole development process. Therefore, savings are not that large. However, if AI technology develops so that truly intelligent tools can be developed than, obviously, this situation will change. Technology will make some jobs redundant but will also create job opportunities.

Chapter 3

3.1. Explain why the rapid delivery and deployment of new systems is often more important to businesses than the detailed functionality of these systems.

Answer:
A conventional waterfall or specification-based process is usually prolonged and the final software is delivered to the customer long after it was originally specified. In a fast-moving business environment, this can cause real problems. By the time the software is available for use, the original reason for its procurement may have changed so radically that the software is effectively useless. Therefore, for business systems in particular, development processes that focus on rapid software development and delivery are essential.

3.2. Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

a) *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team knows what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.

b) *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.

c) *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.

d) *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

3.3. When would you recommend *against* the use of an agile method for developing a software system?

Answer:
Agile methods should probably not be used when the software is being developed by teams who are not co-located. If any of the individual teams use agile methods, it is very difficult to coordinate their work with other teams. Furthermore, the informal communication which is an essential part of agile methods is practically impossible to maintain. Agile methods should probably also be avoided for critical systems where the consequences of a specification error are serious. In those circumstances, a system specification that is available before development starts makes a detailed specification analysis possible. However, some ideas from agile approaches such as test first development are certainly applicable to critical systems.

3.4. Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Answer:
Advantages of stories:

- They represent real situations that commonly arise so the system will support the most common user operations.
- It is easy for users to understand and critique the stories.
- They represent increments of functionality – implementing a story delivers some value to the user.

Disadvantages of stories:

- They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
- They focus on functional requirements rather than non-functional requirements. Representing cross-cutting system requirements such as performance and reliability is impossible when stories are used.
- The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

3.5. Explain why test-first development helps the programmer to develop a better understanding of the system requirements. What are the potential difficulties with test-first development?

Answer:
Test-first development helps with understanding the requirements because, in order to write a test, you have to analyse the requirements in detail to discover what is intended. In many cases, you may find that writing a test is impossible because the requirements are incomplete. The problem with test-first development is that some tests are very difficult to write because they require a system infrastructure to be in place before anything can be tested.

3.6. Suggest four reasons why the productivity rate of programmers working as a pair might be more than half that of two programmers working individually.

Answer:
Reasons why pair programming may be more efficient as the same number programmers working individually:
1. Pair programming leads to continuous informal reviewing. This discovers bugs more quickly than individual testing.
2. Information sharing in pair programming is implicit – it happens during the process. This reduces the need for documentation and the time required if one programmer has to pick up another's work. Individual programmers have to spend time explicitly sharing information and they are not being productive when doing so.
3. Pair programming encourages refactoring (the code must be understandable to another person). This reduces the costs of subsequent development and change and means that future changes can be made more quickly. Hence, efficiency is increased.
4. In pair programming, people are likely to spend less time in fine-grain optimization as this does not benefit the other programmer. This means that the pair focuses on the essential features of the system which they can then produce more quickly.

3.7. Compare and contrast the Scrum approach to project management with conventional plan-based approaches, as discussed in Chapter 23. The comparisons should be based on the effectiveness of each approach for planning the allocation of people to projects, estimating the cost of projects, maintaining team cohesion, and managing changes in project team membership.

3.8. You are a software manager in a company that develops critical control software for aircraft. You are responsible for the development of a software design support system that supports the translation of software requirements to a formal software specification (discussed in Chapter 13). Comment on the advantages and disadvantages of the following development strategies:
a. Collect the requirements for such a system from software engineers and external stakeholders (such as the regulatory certification authority) and develop the system using a plan-driven approach.
b. Develop a prototype using a scripting language, such as Ruby or Python, evaluate this prototype with software engineers and other stakeholders, then review the system requirements. Redevelop the final system using Java.
c. Develop the system in Java using an agile approach with a user involved in the development team.

3.9. It has been suggested that one of the problems of having a user closely involved with a software development team is that they 'go native'; that is, they adopt the outlook of the development team and lose sight of the needs of their user colleagues. Suggest three ways how you might avoid this problem and discuss the advantages and disadvantages of each approach.

Answer:
1. Involve multiple users in the development team. Advantages are you get multiple perspectives on the problem, better coverage of user tasks and hence requirements and less likelihood of having an atypical user. Disadvantages are cost, difficulties of getting user engagement and possible user conflicts.
2. Change the user who is involved with the team. Advantages are, again, multiple perspectives. Disadvantages are each user takes time to be productive and possible conflicting requirements from different users.
3. Validate user suggestions with other user representatives. Advantages are independent check on suggestions; disadvantage is that this slows down the development process as it takes time to do the checks.

3.10. To reduce costs and the environmental impact of commuting, your company decides to close a number of offices and to provide support for staff to work from home. However, x the senior management who introduce the policy are unaware that software is developed using agile methods, which rely on close team working and pair programming. Discuss the difficulties that this new policy might cause and how you might get around these problems.

Answer:
The **difficulties that may arise** with this policy if making employees work from home are:
The benefits obtained through agile methods will be less effective.
Communication gap between the members of a team
The benefit of error detection and evaluation through pair programming is lost.
Pair programming is not possible.
Due to sudden changes in the teams, the project development may be slowed down.

The **measures that can be taken** to get around such difficulties are:
Rather than completely closing some offices and asking people to work from home, employees can be moved to some offices and accommodated
Communication within the members of the team must be improved and must communicate regularly.
Information regarding project should be shared and communicated.

Chapter 4

4.1. Identify and briefly describe four types of requirement that may be defined for a computer based system.

Answer:
Functional requirements: Those specify some services or functionality to be provided by the system.
Non-functional requirements: Those define operational constraints on the behaviour of the system
User requirements: The requirements are the statements in a natural language plus diagrams of the services the system provides and its operational constraints.
System requirements: A structured document setting out detailed description of the system's functions, services and operational constraints. Define what should be implemented. It may be part of a contract between client and contactor.
Design requirements: Those define constraints on the system design and implementation
Process requirements: Those define constraints on the system development process

4.2. Discover ambiguities or omissions in the following statement of requirements for part of a ticket-issuing system: An automated ticket-issuing system sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination. Once a destination has been selected, users are requested to input their credit card. Its validity is checked and the user is then requested to input a personal identifier. When the credit transaction has been validated, the ticket is issued.

Answer:
Ambiguities and omissions include:
• Can a customer buy several tickets for the same destination together or must they be bought one at a time?
• Can customers cancel a request if a mistake has been made?
• How should the system respond if an invalid card is input?
• What happens if customers try to put their card in before selecting a destination (as they would in ATM machines)?
• Must the user press the start button again if they wish to buy another ticket to a different destination?
• Should the system only sell tickets between the station where the machine is situated and direct connections or should it include all possible destinations?

4.3. Rewrite the above description using the structured approach described in this chapter. Resolve the identified ambiguities in an appropriate way.

Function – Give customer a rail ticket, and charge credit account accordingly
Description – Determine customer's destination, calculate the charge for the trip, and charge the customer the appropriate amount. If charge is complete, print the ticket, otherwise, print no ticket, and report error to customer.
Inputs – Destination, credit card number, personal ID number
Outputs – Tickets, error messages
Action – Ask the customer for their destination, when input, calculate the total, and prompt for swiping of a credit card, prompt customer for PIN, prompt customer that the transaction is taking place, if successful print the ticket and return to start state, if unsuccessful, ask customer to swipe their card again and re-input the PIN. If unsuccessful again, prompt that the transaction has failed, and return to start state.
Requires – Destination, credit card number, and PIN
Pre-condition – None
Post-condition – None
Side effects – Charge to the customer's credit account

4.4. Write a set of non-functional requirements for the ticket-issuing system, setting out its expected reliability and response time.

Answer:
Possible non-functional requirements for the ticket issuing system include:
1. Between 0600 and 2300 in any one day, the total system down time should not exceed 5 minutes.
2. Between 0600 and 2300 in any one day, the recovery time after a system failure should not exceed 2 minutes.
3. Between 2300 and 0600 in any one day, the total system down time should not exceed 20 minutes.
All these are availability requirements – note that these vary according to the time of day.
Failures when most people are traveling are less acceptable than failures when there are few customers.
4. After the customer presses a button on the machine, the display should be updated within 0.5 seconds.
5. The ticket issuing time after credit card validation has been received should not exceed 10 seconds.
6. When validating credit cards, the display should provide a status message for customers indicating that activity is taking place.
This tells customer that the potentially time consuming activity of validation is still in progress and that the system has not simply failed.
7. The maximum acceptable failure rate for ticket issue requests is 1: 10000.
Note that this is really ROCOF. I have not specified the acceptable number of incorrect tickets as this depends on whether or not the system includes trace facilities that allow customer requests to be logged. If so, a relatively high failure rate is acceptable as customers can complain and get refunds. If not, only a very low failure rate is acceptable.

4.5. Using the technique suggested here, where natural language descriptions are presented in a standard format, write plausible user requirements for the following functions:

1. An unattended petrol (gas) pump system that includes a credit card reader. The customer swipes the card through the reader then specifies the amount of fuel required. The fuel is delivered and the customer's account debited.
2. The cash-dispensing function in a bank ATM.
3. The spelling-check and correcting function in a word processor.

Answer:
1. Fuel delivery system
1.1 The system should provide an unattended fuel delivery service where a specified amount of fuel is delivered to customers; The cost is deducted from the customer's credit card account.
1.2 The sequence of actions to dispense fuel should be:

1. The customer selects the type of fuel to be delivered.
2. The customer inputs either a cash limit or a maximum amount of fuel to be delivered
3. The customer validates the transaction by providing credit card account details.

Rationale: The amount of fuel allowed depends on the credit limit but customers may wish to 'fill up' rather than have a specified amount of fuel. By specifying a maximum, the system can check if credit is available. Note that the definition does not set out how credit card details should be provided.

4. The pump is activated and fuel is delivered, under customer control.
5. The transaction is terminated either when the pump nozzle is returned to its holster for 15 seconds or when the customers fuel or cash limit is reached.

Rationale: Termination should not be immediate when the nozzle is returned as the customer may wish to restart the transaction e.g. to fill a fuel can as well as the car fuel tank. If a pump display is
available, it may be appropriate to issue a 'Please wait for your receipt' message.

6. A receipt is printed for the customer.
7. The fuel stock is updated.

Specification: PUMP_SYS/FS. Section 1

2. Dispensing cash

2.1 The system must provide a facility which allows a specified amount to cash to be issued to customers. The amount is requested by the customer but the system may reduce this amount if the customer's daily limit or overdraft limit is reached.

2.1.1 The sequence of actions to dispense cash should be:

1. The customer inputs the amount of cash required
2. The system checks this against daily card limits and the customer's overdraft limit.
3. If the amount breaches either of these limits, then a message is issued which tells the customer of the maximum amount allowed and the transaction is cancelled.
4. If the amount is within limits, the requested cash should be dispensed
5. The customer's account balance and daily card limit should be reduced by the amount of cash dispensed.

Specification: ATM/Customer functionality/FS. Section 2.1

3. Spell checking

3.1 The system shall provide a user-activated facility which checks the spelling of words in the document against spellings in the system dictionary and user-supplied dictionaries.

3.2 When a word is found in the document which is not in any dictionary, a user query should be issued with the following options:

1. Ignore this instance of the word
2. Ignore all instances of the word
3. Replace the word with a suggested word from the dictionary
4. Replace the word with user-supplied text
5. Ignore this instance and add the word to a specified dictionary

3.3 When a word is discovered which is not in the dictionary, the system should propose 10 alternative words based on a match between the word found and those in the dictionaries.

Specification: NewWP/Tools/FS. Section 7.2

4.6. Suggest how an engineer responsible for drawing up a system requirements specification might keep track of the relationships between functional and non-functional requirements.

Answer:

Keeping track of the relationships between functional and non-functional requirements is difficult because non-functional requirements are sometimes system level requirements rather than requirements which are specific to a single function or group of functions.

One approach that can be used is to explicitly identify system-level non-functional requirements that are associated with a functional requirement and list them separately. All system requirements that are relevant for each functional requirement should be listed. They can also be related by including them in a table to explicitly identify the relationship.

4.7. Using your knowledge of how an ATM is used, develop a set of use cases that could serve as a basis for understanding the requirements for an ATM system.

Answer:
There are a variety of different types of ATM so, obviously, there is not a definitive set of use cases that could be produced. However, I would expect to see use cases covering the principal functions such as withdraw cash, display balance, print statement, change PIN and deposit cash. The use case description should describe the actors involved, the inputs and outputs, normal operation and exceptions.

Withdraw cash:

- Actors: Customer, ATM, Accounting system
- Inputs: Customer's card, PIN, Bank Account details
- Outputs: Customer's card, Receipt, Bank account details

Normal operation:

- The customer inputs his/her card into the machine. He/she s promoted for a PIN which is entered on the keypad. If correct, he/she is presented with a menu of options. The Withdraw cash option is selected. The customer is promoted with a request for the amount of cash required and inputs the amount. If there are sufficient funds in his/her account, the cash is dispensed, a receipt if printed and the account balance is updated. Before the cash is dispensed, the card is returned to the customer who is prompted by the machine to take their card.

Exception:

- Invalid card. Card is retained by machine; Customer advised to seek advice.
- Incorrect PIN. Customer is request to rekey PIN. If incorrect after 3 attempts, card is retained by machine and customer advised to seek advice.
- Insufficient balance Transaction terminated. Card returned to customer.

4.8. Who should be involved in a requirements review? Draw a process model showing how a requirements review might be organized.

Answer:

A requirements review is a process where a group of people from the system customer and the system developer read the requirements document in detail and check for errors, anomalies, and inconsistencies.

A requirements review is a manual process that involves people from both client and contractor organisations. They check the requirements document for anomalies and omissions. The review process may be managed in the same way as program inspections. Alternatively, it may be organised as a broader activity with different people checking different parts of the document. Requirements reviews can be informal or formal. Informal reviews simply involve contractors discussing requirements with as many system stakeholders as possible. In a formal requirements review, the development team should 'walk' the client through the system requirements explaining the implications of each requirement.

4.9. When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications that will ensure that the requirements document and the system implementation do not become inconsistent.

Answer:
Figure below shows a change process which may be used to maintain consistency between the requirements document and the system. The process should assign a priority to changes so that emergency changes are made but these changes should then be given priority when it comes to making modifications to the system requirements. The changed code should be an input to the final change process but it may be the case that a better way of making the change can be found when more time is available for analysis.

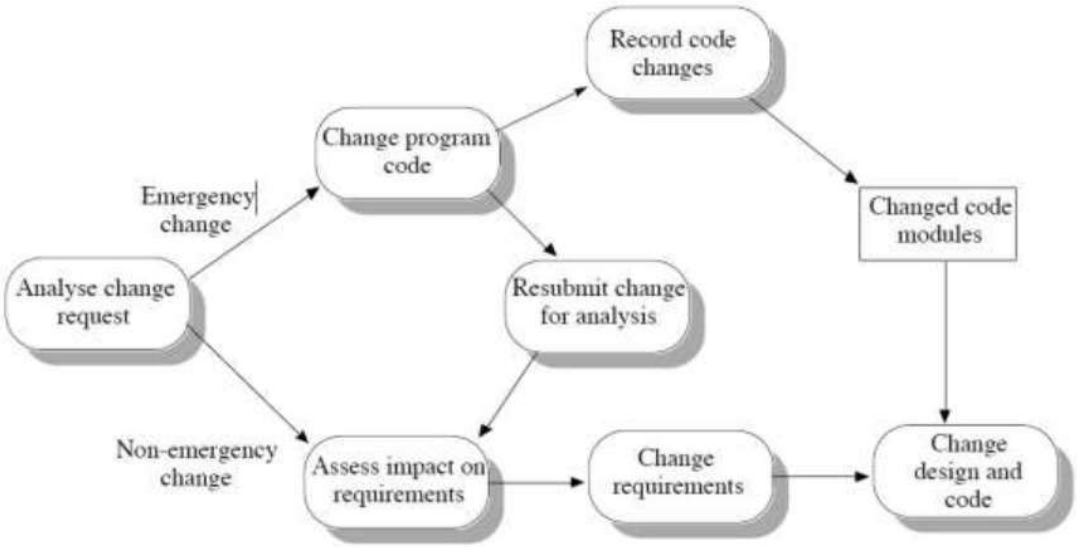

Figure 7.2 A change process for emergency changes

4.10. You have taken a job with a software user who has contracted your previous employer to develop a system for them. You discover that your company's interpretation of the requirements is different from the interpretation taken by your previous employer. Discuss what you should do in such a situation. You know that the costs to your current employer will increase if the ambiguities are not resolved. However, you have also a responsibility of confidentiality to your previous employer.

Answer:

The key here is the ambiguities....there is nothing illegal about resolving the interpretation of ambiguities. I would discuss the ambiguities, and email a bullet pointed list of the specific ambiguities and recommendations to my current employer specific decision maker.

As a system developer, explain to your team members types of non-functional requirements?

Answer:
Non-functional requirements are those which describe aspects of the system that are concerned with how well it provides the functional requirements. These include the following:

- Performance criteria such as desired response times for updating data in the system or retrieving data from the system.
- Anticipated volumes of data, either in terms of throughput or of what must be stored.
- Response time, memory utilisation, understandability of the program code, security, reliability, robustness, performance, timing constraints, constraints on the development process, constraints, constraints imposed by standards, availability, speed, size, ease of use, portability.

Give 5 reasons why eliciting requirements is difficult?

Answer:
Eliciting and understanding requirements from system stakeholders is a difficult process for several reasons:

1. Stakeholders often don't know what they want from a computer system except in the most general terms; they may find it difficult to articulate what they want the system to do; they may make unrealistic demands because they don't know what is and isn't feasible.
2. Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, may not understand these requirements.
3. Different stakeholders have different requirements and they may express these in different ways. Requirements engineers have to discover all potential sources of requirements and discover commonalities and conflict.
4. Political factors may influence the requirements of a system. Managers may demand specific system requirements because these will allow them to increase their influence in the organization.
5. The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. The importance of particular requirements may change. New requirements may emerge from new stakeholders who were not originally consulted.

What is the distinction between functional and non-functional requirements?

Answer:

Functional requirements define what the system should do. Non-functional requirements are not directly concerned with specific system functions but specify required system properties or place constraints on the system or its development process.

What are the main advantages of using a standard format to specify requirements?

Answer:

- All requirements have the same format so are easier to read
- The definition of form fields means that writers are less likely to forget to include information
- Some automated processing is possible.

What checks should be applied during requirements validation?

Answer:
1. Validity checks
2. Consistency checks
3. Completeness checks
4. Realism checks
5. The verifiability of the requirements should be assessed

Chapter 5

5.1. Explain why it is important to model the context of a system that is being developed. Give two examples of possible errors that could arise if software engineers do not understand the system context.

Answer:
At early stage in the requirements elicitation and analysis process software engineers should decide on the boundaries of the system. This involves working with system stakeholders to distinguish what is the system and what is the system's environment. Software engineers should make these decisions early in the process to limit the system costs and the time needed for analysis.

The definition of a system boundary is not a value-free judgement. The position of the system boundary has profound effect on the system requirements. There may be pressure to develop system boundaries that increase/decrease the influence or workload of different parts of an organisation. Social and organisational concerns may mean that the position of a system boundary may be determined by non-technical factors.

Two Examples:
1. Software engineers may give an incorrect system environment in requirements documentation
2. System cost and time of system analysis may increase in the future because the wrong system boundary was defined in the previous process.

5.2. How might you use a model of a system that already exists? Explain why it is not always necessary for such a system model to be complete and correct. Would the same be true if you were developing a model of a new system?

Answer:

a) Using an existing model during requirements engineering will decrease the system cost and time when setting up the corresponding system model. It is difficult to determine whether the chosen model will fulfil the requirements of the current proposed system.
b) There is no absolutely true or false for a certain system model. System modelling is a process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It only helps the analyst to understand the functionality of the system and models are used to communicate with customers. System models are determined by the functional and non-functional requirements, therefore different systems have different system models to fulfil the system requirements and there is no complete or correct system.
c) 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.

5.3. You have been asked to develop a system that will help with planning large-scale events and parties such as weddings, graduation celebrations, birthday parties, etc. Using an activity diagram, model the process context for such a system that shows the activities involved in planning a party (booking a venue, organizing invitations, etc.) and the system elements that may be used at each stage.

5.4. For the MHC-PMS, propose a set of use cases that illustrates the interactions between a doctor, who sees patients and prescribes medicine and treatments, and the MHC-PMS.

5.5. Develop a sequence diagram showing the interactions involved when a student registers for a course in a university. Courses may have limited enrollment, so the registration process must include checks that places are available. Assume that the student accesses an electronic course catalog to find out about available courses.

Answer:

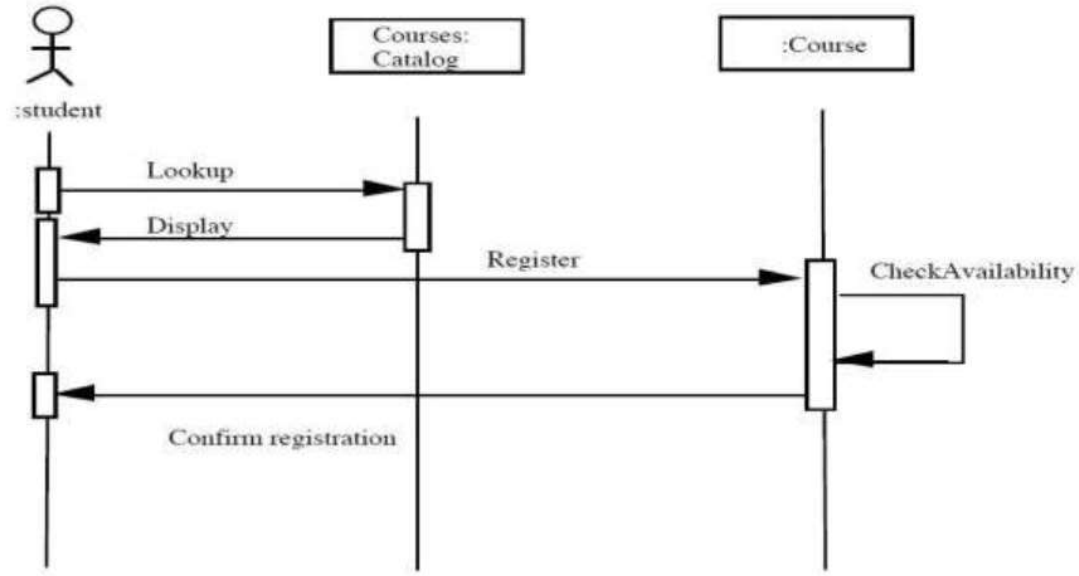

Figure 8.6 Sequence diagram of course registration

5.6. Look carefully at how messages and mailboxes are represented in the e-mail system that you use. Model the object classes that might be used in the system implementation to represent a mailbox and an e-mail message.

5.7. Based on your experience with a bank ATM, draw an activity diagram that models the data processing involved when a customer withdraws cash from the machine.

5.8. Draw a sequence diagram for the same system. Explain why you might want to develop both activity and sequence diagrams when modeling the behavior of a system.

5.9. Draw state diagrams of the control software for:
_ An automatic washing machine that has different programs for different types of clothes.
_ The software for a DVD player.
_ A telephone answering system that records incoming messages and displays the number of accepted messages on an LED. The system should allow the telephone customer to dial in from any location, type a sequence of numbers (identified as tones), and play any recorded messages.

5.10. You are a software engineering manager and your team proposes that model-driven engineering should be used to develop a new system. What factors should you take into account when deciding whether or not to introduce this new approach to software development?

Answer:
The factors that you have to consider when making this decision include:
1. The expertise of the team in using UML and MDA. (Is expertise already available or will extensive training be required.)
2. The costs and functionality of the tools available to support MDA. (Are tools available in house or will they have to be purchased. Are they good enough for the type of software being developed)
3. The likely lifetime of the software that you are developing. (MDA is most suitable for long-lifetime systems)
4. Requirements for high performance or throughput (MDA relies on code generation that creates code which may be less efficient than hand written code)
5. The long term benefits of using MDA (are there real cost savings from this approach)
6. The enthusiasm of the software developers. (are all team members committed to this new approach)

What perspectives may be used for system modelling?

Answer:
An external perspective
An interaction perspective
A behavioural perspective
A structural perspective.

Your manager expects you to inform him how an already existing system might be used. Elaborate on how you might use a model of a system that already exists? Explain why it is not always necessary for such a system model to be complete and correct. Would the same be true if you were developing a model of a new system?

See above 5.1.

Give a short explanation of each of the following
- **Activity diagrams**
- **Use case diagrams**
- **Sequence diagrams**
- **Class diagrams**
- **State diagrams**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Use case were developed originally to support requirements elicitation and now incorporated into the UML. Each use case diagram represents a discrete task that involves external interaction with a system. Actors in a use case diagram may be people or other systems.

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 sequence diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.

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.

State diagrams State diagrams are used to model the behavior of the system in response to external and internal events. They show the system's responses to stimuli so are often used for modeling real-time systems. State diagrams show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves form one state to another.

Chapter 7

7.1. Using the structured notation shown in Figure 7.3, specify the weather station use cases for Report status and Reconfigure. You should make reasonable assumptions about the functionality that is required here.

7.2. Assume that the MHC-PMS is being developed using an object-oriented approach. Draw a use case diagram showing at least six possible use cases for this system.

7.3. Using the UML graphical notation for object classes, design the following object classes, identifying attributes and operations. Use your own experience to decide on the attributes and operations that should be associated with these objects.

_ a telephone
_ a printer for a personal computer
_ a personal stereo system
_ a bank account
_ a library catalog

7.4. Using the weather station objects identified in Figure 7.6 as a starting point, identify further objects that may be used in this system. Design an inheritance hierarchy for the objects that you have identified.

7.5. Develop the design of the weather station to show the interaction between the data collection subsystem and the instruments that collect weather data. Use sequence diagrams to show this interaction.

7.6. Identify possible objects in the following systems and develop an object-oriented design for them. You may make any reasonable assumptions about the systems when deriving the design.

_ A group diary and time management system is intended to support the timetabling of meetings and appointments across a group of co-workers. When an appointment is to be made that involves a number of people, the system finds a common slot in each of their diaries and arranges the appointment for that time. If no common slots are available, it interacts with the user to rearrange his or her personal diary to make room for the appointment.

_ A filling station (gas station) is to be set up for fully automated operation. Drivers swipe their credit card through a reader connected to the pump; the card is verified by communication with a credit company computer, and a fuel limit is established. The driver may then take the fuel required. When fuel delivery is complete and the pump hose is returned to its holster, the driver's credit card account is debited with the cost of the fuel taken. The credit card is returned after debiting. If the card is invalid, the pump returns it before fuel is dispensed.

7.7. Draw a sequence diagram showing the interactions of objects in a group diary system when a group of people are arranging a meeting.

7.8. Draw a UML state diagram showing the possible state changes in either the group diary or the filling station system.

7.9. Using examples, explain why configuration management is important when a team of people are developing a software product.

Edition 8 Answer:

In software development, change happens all the time, so change management is absolutely essential. When a team of people are developing software, you have to make sure that team members don't interfere with each other's work. That is, if two people are working on a component, their changes have to be coordinated. Otherwise, one programmer may make changes and overwrite the other's work. You also have to ensure that everyone can access the most up-to-date versions of software components, otherwise developers may redo work that has already been done. When something goes wrong with a new version of a system, you have to be able to go back to a working version of the system or component.

Edition 9 answer:

The aims of configuration management is to ensure that (a) changes made by different system developers do not interfere with each other and (b) it is always possible to create a specific version of a system. Without configuration management it is easy to lose track of the changes that each developer makes to code and for changes made by one programmer to overwrite changes made by another programmer. For example, one programmer may change a component to improve its performance whilst another may correct a bug in the functionality of the component. Without CM, whoever writes the component last to the shared component store will overwrite and so lose the previous component changes. Furthermore, systems are usually composed of multiple components, each of which exists in multiple versions, where each version as a specific purpose. For example, there may be a version of a system for different platforms such as Windows, Linux and MacOS. These versions have some specific components and some shared components and it is potentially error prone if these versions are assembled without CM tool support. It is very easy to include the wrong component in a version and this is likely to lead to subsequent software failure.

7.10. A small company has developed a specialized product that it configures specially for each customer. New customers usually have specific requirements to be incorporated into their system, and they pay for these to be developed. The company has an opportunity to bid for a new contract, which would more than double its customer base. The new customer also wishes to have some involvement in the configuration of the system. Explain why, in these circumstances, it might be a good idea for the company owning the software to make it open source.

Answer:

The key benefits of open source are is that it opens up development to a wide range of developers and so accelerates the development and debugging of the product. Doubling the customer base places immense strains on a small company of they have to take on a lot of new staff and so going open source means that the costs of expansion are reduced.

In this case, because the product is specialized to the needs of different users, the company that own the software can still charge these users to make the changes to the system. Hence the loss in revenue from selling the software is compensated by the additional effort available to service more customers.

Furthermore, large companies are often reluctant to buy from small companies who may go out of business, To some extent, open source provides reassurance to customers that, even of the original owners of the software are unavailable, they can get access to the source code and hence continue to maintain their system.

Finally, open source may increase knowledge of the company's product and so attract more customers.

Chapter 8

8.1. Explain why it is not necessary for a program to be completely free of defects before it is delivered to its customers.

Answer:
A program need not be completely free of defects before delivery if:
1. Remaining defects are minor defects that do not cause system corruption and which are transient i.e. which can be cleared when new data is input.
2. Remaining defects are such that they are recoverable and a recovery function that causes minimum user disruption is available.
3. The benefits to the customer's business from the system exceed the problems that might be caused by remaining system defects.
Testing cannot completely validate that a system is fit for its intended purpose as this requires a detailed knowledge of what that purpose will be and exactly how the system will be used. As these details inevitably change between deciding to procure a system and deploying that system, the testing will be necessarily incomplete. In addition, it is practically impossible for all except trivial system to have a complete test set that covers all possible ways that the system is likely to be used.

8.2. Explain why testing can only detect the presence of errors, not their absence.

Answer:
Assume that exhaustive testing of a program, where every possible valid input is checked, is impossible (true for all but trivial programs). Test cases either do not reveal a fault in the program or reveal a program fault. If they reveal a program fault then they demonstrate the presence of an error. If they do not reveal a fault, however, this simply means that they have executed a code sequence that – for the inputs chosen – is not faulty. The next test of the same code sequence – with different inputs – could reveal a fault.

8.3. Some people argue that developers should not be involved in testing their own code but that all testing should be the responsibility of a separate team. Give arguments for and against testing by the developers themselves.

Answer:

For	Against

If the developer is executing tests during the 'Build' phase, he could identify and rectify defects before passing the code to testers for testing	Many if not most developers believe their code is 'pure quality' and will not test thoroughly enough
Developers need to be measured on delivery on-time, in budget **and** of the appropriate quality which should motivate them to want to test their code properly.	Developers are not always of the mindset that will enable them to test properly as a good tester needs a more enquiring mind than a creative mind which is often the developers mindset.
If an organisation is looking to reduce costs, having a developer that can double as a tester is a way of saving money. This just needs the checks and balances to ensure all testing is appropriate.	An independent tester will have no emotional connection with any code and will have no doubts about testing application code thoroughly.
	The characteristics of a good tester are different from the characteristics of a good developer and the tester is considered good due to his characteristics driving him/her to do a good job of testing.

8.4. You have been asked to test a method called 'catWhiteSpace' in a 'Paragraph' object that, within the paragraph, replaces sequences of blank characters with a single blank character. Identify testing partitions for this example and derive a set of tests for the 'catWhiteSpace' method.

8.5. What is regression testing? Explain how the use of automated tests and a testing framework such as JUnit simplifies regression testing.

Answer:
Regression testing is the process of running tests for functionality that has already been implemented when new functionality is developed or the system is changed. Regression tests check that the system changes have not introduced problems into the previously implemented code.

Automated tests and a testing framework, such as JUnit, radically simplify regression testing as the entire test set can be run automatically each time a change is made. The automated tests include their own checks that the test has been successful or otherwise so the costs of checking the success or otherwise of regression tests is low.

8.6. The MHC-PMS is constructed by adapting an off-the-shelf information system. What do you think are the differences between testing such a system and testing software that is developed using an object-oriented language such as Java?

8.7. Write a scenario that could be used to help design tests for the wilderness weather station system.

Answer:
John is a meteorologist responsible for producing weather maps for the state of Minnesota. These maps are produced from automatically collected data using a weather mapping system and they show different data about the weather in Minnesota. John selects the area for which the map is to be produced, the time period of the map and requests that the map should be generated. While the map is being created, John runs a weather station check that examines all remotely collected weather station data and looks for gaps in that data – this would imply a problem with the remote weather station.

8.8. What do you understand by the term 'stress testing'? Suggest how you might stress test the MHC-PMS.

Answer:
In performance testing, this means stressing the system by making demands that are outside the design limits of the software.

8.9. What are the benefits of involving users in release testing at an early stage in the testing process? Are there disadvantages in user involvement?

Answer:
Early involvement is usually an advantage. The earlier the testers know what it is that the users want to see demonstrated in order to satisfy themselves that the system is acceptable, the earlier the testers can confirm that these tests are included in their test packs. This could only be a disadvantage where the users negatively interfere with test efforts that the testers are trying to implement.

8.10. A common approach to system testing is to test the system until the testing budget is exhausted and then deliver the system to customers. Discuss the ethics of this approach for systems that are delivered to external customers.

Answer:
This is unethical from a business perspective. In theory the budget could be exhausted after an insignificant amount of testing has been conducted and the risk to the solution is effectively unknown. Often testing budgets are consumed due to reasons beyond the control of the test team. This could include scope changes, additional functionality added shortened test times due to development over-runs etc. In my opinion, there needs to be a core of priority tests that need to be successfully completed irrespective of the costs and timelines and if these get past excessive levels, maybe the system should be shelved. Very few companies could afford to take a risk that if realised the consequences are unknown. Deliver these systems to external customers could lead to expensive lawsuits against the developers and if the customer is successful in the lawsuit the developer could be put out of business.

Explain why interface testing is necessary even when individual components have been extensively validated through component testing and program inspections

Answer:
Interface testing can observe the interaction between components in a system.
There is a necessity to observe the error in a particular critical area such as parameter interfaces, shared memory interfaces, procedural interfaces, and message passing interfaces which component testing can't detect these errors.

There are several reasons why interface testing is a necessary stage after unit testing:

• The interface to the module may have been incorrectly specified. The validation process is based on this specification rather than actual usage of the module or sub-system.
• The assumptions made by other modules about the behaviour of a given module (A say) in response to particular interface stimuli may be incorrect. That is, these modules expect A to behave in a way in which it was never designed to operate.
• Interface testing can reveal omissions in the interface design. It may be discovered, when integrated with other modules, that the interface must be augmented in some way.

Explain the differences between verification and validation, and explain why validation is a particularly difficult process

Verification is checking whether software is behaving according to its requirements.
Validation is confirming the software to meet the client's expectation.
In software engineering, validating software might be harder since client's expectation may be vague or unclear. Client may use adjectives for their expectation which may result in confusion or misunderstanding.

Explain why it may be cost-effective to use formal methods in the development of safety-critical software system. Why do you think that some developers of this type of system are against the sue of formal methods?

Formal methods focus on the design phase of the system. Any error that might occur in the system is detected in the design phase before going to the development phase.

This is an effective method since the cost of system failure might be huge and involves people lives and safety.
Developers are against the formal methods since:

- It is highly possible to misunderstand the specification that is announced by the system users which may result to a design malfunction.
- Program proofs are large and complex. It is prone to have errors.
- The system may not be used as anticipated and lead to a program ineffectiveness.
- Highly chance of failure since system is time-consuming.

Explain the advantages of formal specification to practising engineers

To explain the advantages of formal specification to practising engineers, it is important to focus on what it brings to the practice of software development rather than on more abstract advantages such as the ability to mathematically analyse the specification. Advantages that might be stressed are:

- The detailed analysis of the requirements that is necessary to produce a formal specification. This results in the discovery and resolution of ambiguities and errors at an early stage in the process.
- The unambiguous specification of interfaces. Interface problems are one of the major problems in system integration and a reduction in such problems can significantly reduce software costs.
- The ability to mix formal and informal specifications. The whole system need not be formally specified but only those parts where most benefit can be gained.

Chapter 10

10.1. Give two examples of government functions that are supported by complex sociotechnical systems and explain why, in the foreseeable future, these functions cannot be completely automated.

Answer:
Two examples of government functions are Health related services and Department of home affairs. As long as such systems provide services to different types of human users with backgrounds, capabilities, and personalities, these functions cannot be completely automated.

10.2. Explain why the environment in which a computer-based system is installed may have unanticipated effects on the system that lead to system failure. Illustrate your answer with a different example from that used in this chapter.

Answer:
Other systems in the system's environment can have unanticipated effects because they have relationships with the system over and above whatever formal relationships (e.g. data exchange) are defined in the system specification. For example, the system may

share an electrical power supply and air conditioning unit, they may be located in the same room (so if there is a fire in one system then the other will be affected) etc.

10.3. Why is it **impossible** to infer the emergent properties of a complex system from the properties of the system components?

Answer:
For a complex system, integration of large number of components is also complex, which may result in emergent properties of the integrated system own, even if individual component has satisfactory emergent properties.

10.4. Why is it sometimes difficult to decide whether or not there has been a failure in a sociotechnical system? Illustrate your answer by using examples from the MHC-PMS that has been discussed in earlier chapters.

Answer:
The notion of a system failure is a judgment on the part of the observer of the failure, depending on their experience and expectations. Users of a system never read the specification so it is pointless to define failures as a deviation from a specification.

For example, consider two users of the MHC-PMS from different backgrounds:
a) User 1 is a doctor who has extensive experience of mental health care. When selecting a menu of options to identify the patient's condition, he or she will expect to see in this menu the conditions with which they are familiar. If these conditions do not appear in the menu then he or she may consider this to be a system failure.

b) User 2 is a doctor who has recently graduated and has only limited experience of mental health care. When selecting the menu of options, they assume that these reflect the conditions which the system can handle so they classify the patient according to these conditions. They do not observe a system failure.

10.5. What is a 'wicked problem'? Explain why the development of a national medical records system should be considered a 'wicked problem'.

Answer:
This is an inherently wicked problem because of the uncertainties associated with the problem. It is impossible to anticipate exactly when and where a disaster will occur, the numbers of people involved, the effects on the environment, the technology available to the emergency services, etc. Planning can only be in very general terms and detailed software specifications to cope with specific situations are almost impossible to write.

10.6. A multimedia virtual museum system offering virtual experiences of ancient Greece is to be developed for a consortium of European museums. The system should provide users with the facility to view 3-D models of ancient Greece through a standard web browser and should also support an immersive virtual reality experience. What political and organizational difficulties might arise when the system is installed in the museums that make up the consortium?

Answer:

Museums are conservative places and some staff may resent the introduction of new technology.
Existing museum staff may be asked to deal with problems of the equipment not working and may not wish to appear unable to deal with this.
Other areas of the museum may oppose the system because they see it as diverting resources from their work.
Different museums may have different preferred suppliers for the equipment so that all equipment used is not identical thus causing support problems.
The new displays take up a lot of space and this displaces other displays.
The maintainers of these displays may oppose the introduction of the system.
Some museums may have no mechanism for providing technical support for the system.

10.7. Why is system integration a particularly critical part of the systems development process? Suggest three sociotechnical issues that may cause difficulties in the system integration process.

Answer:
System integration is particularly critical because it is at the integration stage that incompatibilities between the different sub-systems or components may come to light. Generally, the first view that a customer has of a system is after integration. Sociotechnical difficulties that may arise are:
1. Refusal of parts of the team to recognise problems. Some developers may refuse to recognise that their software is faulty and may try to pass the blame for integration problems to people in different organisations. Different organizations in the integration team are, essentially, trying to transfer the costs to other organizations.
2. Cultural problems due to different organizational approaches to integration. Integration is perhaps the first time that teams have had to work closely together and their organizations may use different processes for system integration. Reconciling these processes can be difficult.
3. Organizations may be at different stages in their project involvement. For some organizations, integration may be their last project activity and their objective is simply to complete and sign off the process as quickly as possible. For other organizations, there may be later work to be done so they may have a longer-term perspective and wish to spend more time on the integration process.

10.8. Explain why legacy systems may be critical to the operation of a business.

Answer:
Legacy systems may be critical for the successful operation of a business for two basic reasons:
• They may be an intrinsic part of one or more processes which are fundamental to the operation of a business. For example, a university has a student admissions process and systems which support this are critical. They must be maintained.
• They may incorporate organisational and business knowledge which is simply not documented elsewhere. For example, exceptions on student admissions may simply have been coded directly into the system with no paper record of these. Without this system, the organisation loses valuable knowledge.

10.9. What are the arguments for and against considering system engineering as a profession in its own right, like electrical engineering or software engineering?

10.10. You are an engineer involved in the development of a financial system. During installation, you discover that this system will make a significant number of people redundant. The people in the environment deny you access to essential information to complete the system installation. To what extent should you, as a systems engineer, become involved in this situation? Is it your professional responsibility to complete the installation as contracted? Should you simply abandon the work until the procuring organization has sorted out the problem?

Answer:

As a systems engineer you should attempt to gain the information you require to do the job you were contracted to do. If it is being kept from you by others, you should avoid direct conflict (getting in arguments or threatening those barring your way), and have the procuring organization gain and provide the data. At no point should you do any illegal activity to gain access to the data you need.

What is a system?

A purposeful collection of inter-related components working together to achieve some common objective.

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. The system is not self-aware.

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 organisational policies and rules.

Socio-technical system characteristics

Emergent properties

- Properties of the system of a whole that depend on the system components and their relationships.

Non-deterministic

- They do not always produce the same output when presented with the same input because the systems's behaviour is partially dependent on human operators.

Complex relationships with organisational objectives

- The extent to which the system supports organisational objectives does not just depend on the system itself.

What are emergent properties?

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

Examples of emergent properties : Volume, Reliability, Security, Repairability, Usability

Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in

	safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating environment.

Influences on reliability

Hardware reliability
- What is the probability of a hardware component failing and how long does it take to repair that component?

Software reliability
- How likely is it that a software component will produce an incorrect output. Software failure is usually distinct from hardware failure in that software does not wear out.

Operator reliability
- How likely is it that the operator of a system will make an error?

Systems engineering

Concerned with specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.

The system engineering process

Requirements definition
System design
Subsystem development
System Integration
System Installation
System Evolution
System Decommissioning

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

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

System requirements definition

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

Should also define overall organisational objectives for the system.

System objectives

Should define why a system is being procured for a particular environment. Includes all Functional objectives and Organisational objectives.

The system design process

Partition requirements
- Organise requirements into related groups.

Identify sub-systems
- Identify a set of sub-systems which collectively can meet the system requirements.

Assign requirements to sub-systems
- Causes particular problems when COTS are integrated.

Specify sub-system functionality.

Define sub-system interfaces
- Critical activity for parallel sub-system development.

System integration

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

Sub-system development

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

Human and organisational factors

Process changes
- Does the system require changes to the work processes in the environment?

Job changes
- Does the system de-skill the users in an environment or cause them to change the way they work?

Organisational changes
- Does the system change the political power structure in an organisation?

Chapter 11

11.1. Suggest six reasons why software dependability is important in most sociotechnical systems.

Answer:
Six reasons why dependability is important are:
a) Users may not use the system if they don't trust it.
b) System failure may lead to a loss of business.
c) An undependable system may lose or damage valuable data.
d) An undependable system may damage its external environment.
d) The reputation of the company who produced the system may be damaged hence affecting other systems.
e) The system may be in breach of laws on consumer protection and the fitness of goods for purpose.

11.2. What are the most important dimensions of system dependability?

Answer:

1. *Availability* - Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services to users at any given time.
2. *Reliability* - Informally, the reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.
3. *Safety* - Informally, the safety of a system is a judgment of how likely it is that the system will cause damage to people or its environment.
4. *Security* - Informally, the security of a system is a judgment of how likely it is that the system can resist accidental or deliberate intrusions.

11.3. Why do the costs of assuring dependability increase exponentially as the reliability requirement increases?

There are two reasons for this:

- The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability
- The increased testing and system validation that is required to convince the system client that the required levels of dependability have been achieved

Alt answer: (from textbook)
If your software is not very dependable, you can get significant improvements at relatively low costs by using better software engineering. However, if you are already using good practice, the costs of improvement are much greater and the benefits from that improvement are less. There is also the problem of testing your software to demonstrate that it is dependable. This relies on running many tests and looking at the number of failures that occur. As your software becomes more dependable, you see fewer and fewer failures. Consequently, more and more tests are needed to try and assess how many problems remain in the software. As testing is very expensive, this dramatically increases the cost of high-dependability systems.

11.4. Giving reasons for your answer, suggest which dependability attributes are likely to be most critical for the following systems:
An Internet server provided by an ISP with thousands of customers
A computer-controlled scalpel used in keyhole surgery
A directional control system used in a satellite launch vehicle
An Internet-based personal finance management system

Answer:

- An internet service : availability, because the ISP must serve many customers at once, so it must be able to be accessed by all customers
- A computer controlled scalpel: safety, because the system will have to perform a critical task to do a surgery, it must concern for patient's safety
- A directional control system: reliability, because the satellite must not miss the target, if not it the satellite can collide with other satellites

- An Internet-based personal finance management system: security, because many credit cards fraught and identity theft so the system must be as secure as possible to protect customers data

11.5. Identify six consumer products that are likely to be controlled by safety-critical software systems.

Answer:
Microwave oven
Power tools such as a drill or electric saw
Lawnmower
Central heating furnace
Garbage disposal unit
Food processor or blender

11.6. Reliability and safety are related but distinct dependability attributes. Describe the most important distinction between these attributes and explain why it is possible for a reliable system to be unsafe and vice versa.

Answer:
System reliability is concerned whether a system is delivering to its requirements. System safety is concerned whether a system is safe from dangerous behaviour.

Ensuring system reliability does not necessarily lead to system safety as reliability is concerned with meeting the system specification (the system 'shall') whereas safety is concerned with excluding the possibility of dangerous behaviour (the system 'shall not'). If the specification does not explicitly exclude dangerous behaviour then a system can be reliable but unsafe.

Reliability is a measure of how well a system meets its specification. That is, it measures how well the system does what it is supposed to do. Safety is often concerned with what the system must not do and this may be impossible to express completely in a system specification. Furthermore, specifications may be in error and these errors may result in safety-related failures.

11.7. In a medical system that is designed to deliver radiation to treat tumors, suggest one hazard that may arise and propose one software feature that may be used to ensure that the identified hazard does not result in an accident.

Answer:
Possible hazard is delivery of too much radiation to a patient. This can arise because of a system failure where a dose greater than the specified dose is delivered or an operator failure where the dose to be delivered is wrongly input.
Possible software features to guard against system failure are the delivery of radiation in increments with an operator display showing the dose delivered and the requirement that the operator confirm the delivery of the next increment. To reduce the probability of operator error, there could be a feature that requires confirmation of the dose to be delivered and that compares this to previous doses delivered to that patient. Alternatively, two different operators could be required to independently input the dose before the machine could operate.

11.8. In computer security terms, explain the differences between an attack and a threat.

Answer:
An attack is an exploitation of a system vulnerability. A threat is a circumstance that has the potential to cause loss or harm. An attack can lead to a threat if the exploitation of the vulnerability leads to a threat. However, some attacks can be successful but do not lead to threats as other system features protect the system.

11.9. Using the MHC-PMS as an example, identify three threats to this system (in addition to the threat shown in Figure 11.8). Suggest controls that might be put in place to reduce the chances of a successful attack based on these threats.

11.10. As an expert in computer security, you have been approached by an organization that campaigns for the rights of torture victims and have been asked to help the organization gain unauthorized access to the computer systems of an American company. This will help them confirm or deny that this company is selling equipment that is used directly in the torture of political prisoners. Discuss the ethical dilemmas that this request raises and how you would react to this request.

Answer:
There is no definitive answer for this question. However, I would expect students to demonstrate that they understand this by putting forward coherent arguments for and against whatever position that they choose to adopt. They should show that they understand that hacking is unlawful whatever position that they adopt but in some cases it can be ethically justified. They should also show that they understand in this case that there could be negative as well as positive results of unauthorised access. For example, acceptance of the request. Hacking is unlawful but, in this case, the benefits which might accrue from accessing the data justify the breaking of the law. Accepting the request does not presuppose guilt on the part of the company - if the request is not accepted, they may be unjustly accused of supplying torture equipment if, in fact, they do not do so. However, using unauthorised means to access data might damage the standing of the campaign especially if the company is not guilty and access is detected. Therefore, it may do more harm than good.

Chapter 14

14.1. Explain the important differences between application security engineering and infrastructure security engineering.

Answer:
Application security engineering is the responsibility of system designers who have to design security into the system that reflects the security requirements and policies of the system procurer. Infrastructure security engineering is the responsibility of system managers or administrators whose job is to configure the existing infrastructure software (operating systems, databases, middleware, etc.) to ensure that it conforms to the security policies of the organisation that uses the infrastructure.

14.2. For the MHC-PMS, suggest an example of an asset, exposure, vulnerability, attack, threat, and control.

14.3. Explain why there is a need for risk assessment to be a continuing process from the early stages of requirements engineering through to the operational use of a system.

Answer:
As more and more systems were connected to the Internet, a variety of different external attacks were devised to threaten these systems. The problems of producing dependable systems were hugely increased. Systems engineers have to consider threats from malicious and technically skilled attackers as well as problems resulting from accidental mistakes in the development process.

Risk assessment starts before the decision to acquire the system has been made and should continue throughout the system development process and after the system has gone into use. Risk assessment is a staged process consisting of Preliminary risk assessment, Life-cycle risk assessment and Operational risk assessment. These phases are necessary because it is impossible to make all dependability and security decisions without complete information about the system implementation.

14.4. Using your answers to question 2 about the MHC-PMS, assess the risks associated with that system and propose two system requirements that might reduce these risks.
14.5. Explain, using an analogy drawn from a non-software engineering context, why a layered approach to asset protection should be used.

Answer:
In designing a system architecture that maintains security, you need to consider two fundamental issues:
1. Protection—how should the system be organized so that critical assets can be protected against external attack?
2. Distribution—how should system assets be distributed so that the effects of a successful attack are minimized?
These issues are potentially conflicting. If you put all your assets in one place, then you can build layers of protection around them. As you only have to build a single protection system, you may be able to afford a strong system with several protection layers. However, if that protection fails, then all your assets are compromised. Adding several layers of protection also affects the usability of a system so it may mean that it is more difficult to meet system usability and performance requirements.
On the other hand, if you distribute assets, they are more expensive to protect because protection systems have to be implemented for each copy. Typically, then, you cannot afford as many protection layers. The chances are greater that the protection will be breached. However, if this happens, you don't suffer a total loss. It may be possible to duplicate and distribute information assets so that if one copy is corrupted or inaccessible, then the other copy can be used. However, if the information is confidential, keeping additional copies increases the risk that an intruder will gain access to this information.

14.6. Explain why it is important to use diverse technologies to support distributed systems in situations where system availability is critical.

Answer:
The use of diverse technologies provides some protection against common vulnerabilities in different elements of the distributed system. Availability is enhanced by distributing assets so that attacks on one element do not disable the entire system. If diverse technologies are used, it reduces the chances that an attack on all elements of the system will be successful.

14.7. What is social engineering? Why is it difficult to protect against it in large organizations?

Answer:
Social engineering occurs where accredited users of a system are fooled into giving away secret information (such as passwords) to potential attackers. It is difficult to protect against this in large organisations because these have a hierarchical structure and people are used to obeying instructions from their managers. Also, because of the size of the organisation,

there is less chance that a manager's manager (say) will be known personally so it is therefore easier for an attacker to impersonate someone in authority.

14.8. For any off-the-shelf software system that you use (e.g., Microsoft Word), analyze the configuration facilities included and discuss any problems that you find.

Include support for viewing and analyzing configurations You should always include facilities in a system that allow administrators or permitted users to examine the current configuration of the system. This facility is, surprisingly, lacking from most software systems and users are frustrated by the difficulties of finding configuration settings. For example, in the version of the word processor that
I used to write this chapter, it is impossible to see or print the settings of all system preferences on a single screen. However, if an administrator can get a complete picture of a configuration, they are more likely to spot errors and omissions. Ideally, a configuration display should also highlight aspects of the configuration that are potentially unsafe—for example, if a password has not been set up.

14.9. Explain how the complementary strategies of resistance, recognition, and recovery may be used to enhance the survivability of a system.

Answer:
Resistance: Built-in mechanisms to resist attacks (such as the use of firewalls) means that many attacks on the system that may threaten its survivability are unsuccessful.
Recognition: This is the process of recognising that an attack is underway. Early recognition means that counter-measures can be quickly deployed and that extra protection can be applied to critical assets, thus increasing the overall chances of survival.
Recovery: If the system has built-in features to support recovery, then normal system service can be resumed more quickly after a successful attack. The overall availability of the system is therefore increased.

14.10. For the equity trading system discussed in Section 14.2.1, whose architecture is shown in Figure 14.5, suggest two further plausible attacks on the system and propose possible strategies that could counter these attacks.

Answer:
Attack 1: Unauthorised orders are inserted into the system between the system and the external computer system of the stock buyer or seller. That is, the communications link between the system and the external world is compromised.
Counter-strategies: Ensure that all orders are encrypted using a key that is known only to the ordering system and the stock buyer/seller. Thus, additional orders introduced into the system can be detected. Monitor all orders transmitted on communication link and ensure that the number of transmitted orders matches the number of placed orders.
Attack 2: Authorised insider places orders that could result in unacceptable losses for the company (this has occurred in several real systems).
Counter-strategies: Ensure that authorised users have an order limit and this can only be exceeded with approval from their manager. Monitor transactions of all insiders to ensure that losses do not exceed limit. Provide daily lists of insider transactions for checking.

Chapter 19

19.1. What are the most important distinctions between services and software components?

Answer:
Software components are under the control of the buyer of the component whereas services are controlled by the service provider. Thus, changes can be made to services without the consent of the service user.

Services are stand-alone entities that do not normally require users to make any other services available. Components may have a 'requires' interface that defines what other components are required to be present in the system.

19.2. Explain why SOAs should be based on standards.

Answer:
Services are intended to be independent and usable in different contexts. A service defines what it needs from another service by setting out its requirements in a message and sending it to that service. If the building of applications based on services is based on standards it allows companies and other organizations to cooperate and make use of each other's business functions. Service providers can also develop specialised services and offer these to a range of service users from different organizations. If SOA is based on standards software systems can be constructed by composing local services and external services from different providers, with seamless interaction between the services in the system. SOA will also not suffer from the incompatibilities that normally arise with technical innovations, where different suppliers maintain their proprietary version of the technology.

19.3. Using the same notation, extend Figure 19.5 to include definitions for MaxMinType and InDataFault. The temperatures should be represented as integers with an additional field indicating whether the temperature is in degrees Fahrenheit or degrees Celsius. InDataFault should be a simple type consisting of an error code.
19.4. Define an interface specification for the Currency Converter and Check credit rating services shown in Figure 19.7.
19.5. Design possible input and output messages for the services shown in Figure 19.11. You may specify these in the UML or in XML.

19.6. Giving reasons for your answer, suggest two important types of applications where you would *not* recommend the use of service-oriented architecture.

Answer:
- Embedded applications in devices where a network connection cannot be guaranteed. These are unlikely to make use of services as there is no guarantee that these services will be available when required.
- Real-time applications with stringent deadlines, especially those with lots of user interaction e.g. computer games. In these applications, the performance overhead in coding and decoding XML messages is likely to be unacceptable.

19.7. In Section 19.2.1, I introduced an example of a company that has developed a catalog service that is used by customers' web-based procurement systems. Using BPMN, design a workflow that uses this catalog service to look up and place orders for computer equipment.

19.8. Explain what is meant by a 'compensation action' and, using an example, show why these actions may have to be included in workflows.

Answer:
A compensating action is an action that is included in a workflow to 'undo' a transaction that has been completed earlier in the workflow. Compensating actions may have to be included in workflows because the success of the entire workflow may rely on all included workflows successfully completing. If some of these included workflows are successful but some aren't, then the compensating actions have to be executed to ensure that the overall system is left in a consistent state. Compensating actions are required when dependent services, offered by different suppliers, are composed to create an integrated service. For example, say a meeting organiser has to book a room for a meeting then organise catering for the people attending the meeting. It may not be possible to do this concurrently as the numbers attending the meeting may not be known. If the room is booked but, subsequently, it transpires that no catering is available that day, then the booking must be cancelled by a compensating action.

19.9. For the example of the vacation package reservation service, design a workflow that will book ground transportation for a group of passengers arriving at an airport. They should be given the option of booking either a taxi or renting a car. You may assume that the taxi and car rental companies offer web services to make a reservation.

19.10. Using an example, explain in detail why the thorough testing of services that include compensation actions is difficult.

Answer:

There is a problem in testing such actions as they may depend on the failure of other services. Ensuring that these services actually fail during the testing process may be very difficult. When services are offered by an external provider, source code of the service implementation is not available. Service-based system testing cannot therefore use proven source code–based techniques. Therefore thorough testing of services including compensation actions will be difficult. The long-term vision of SOAs is for services to be bound dynamically to service-oriented applications. This means that an application may not always use the same service each time that it is executed. This makes it difficult to ensure that thorough testing of services can be done as well as its related compensation actions.

An example of this is an in-car information system that provides drivers with information on weather, local information, road traffic, etc. Different providers in different places offer these services, and the in-car system uses a discovery service to locate appropriate information services and bind to them.
The compensation action for each of these modules will need to be tested thoroughly.

Chapter 20

Explain why an object-oriented approach to software development may not be suitable for real-time systems

Answer:

An object-oriented approach may result in unacceptable timing delays because structuring a system into objects probably means that there will be a large number of small tasks currently active in a system. The overhead of task communications will slow down the system and may cause timing problems. A further problem with object orientation is unpredictable timing as the time to call a method depends on the inheritance hierarchy.

Circumstances where software reuse is not recommended:

Answer:

- If the business status of the code provider is dubious. If the provider goes out of business, then no support for the reused code may be available.
- In critical applications where source code is not available. Testing the code to the required standards may be very difficult.
- In small systems where the costs of reuse are comparable to the savings that result if code is reused.
- In systems where performance is a critical requirement – specially developed code can usually be made more efficient.

Patterns are an effective form of design reuse because the reflect accumulated wisdom that has been collected over several applications rather than a single application (By definition, a pattern is something that should appear in more than one application). There are two problems with patterns for reuse. The first is knowing which patterns actually have been documented and then finding these patterns - the time taken to do this can be significant. The second is that patterns are by their nature generalisations so their performance is likely to be limited. If performance is critical, then a special-purpose tailored approach to a problem will almost always be more effective.

Risks that can arise when systems are constructed using COTS include:

Answer:
Vendor risks:

- Failure of vendor to provide support when required
- Vendor goes out of business or drops product from its portfolio

Product risks:

- Incompatible event/data model with other systems
- Inadequate performance when integrated with other systems
- Product is undependable in intended operating environment

Process risk:

- Time required to understand how to integrate product is higher than expected.

The risks can be addressed by only dealing with vendors that use an escrow system so that source code is available if they go out of business, by extensive research and testing of product capabilities before use, discussion with other users etc. In general though, because COTS are provided by external vendors, risk reduction is difficult.

Suggest why the savings in cost from reusing existing software is not simply proportional to the size of the components that are reused.

Answer:
Reusing components of existing software may bring several problems such as:

- Compatibility of the reused components may not conform to the new developed system.
- Additional bugs can occurred in the passing of messages between components' interactions.

The cost savings from reusing components are the cost savings from not having to write that component. Therefore, the larger the component, the greater the cost saving. However, there are costs of reuse from finding and understanding components to changing other parts of the system to accommodate the reused components. For small components, these reuse costs may actually be greater than the costs of rewriting the component so the cost savings are relatively small unless the component is very complex and difficult to write. As the component size increases, the overhead of finding and understanding the component becomes relatively less so cost savings increase. Furthermore, the larger the component, the cost of changing the system to accommodate it also becomes relatively less so cost savings are greater. Therefore, reuse benefits are not simply proportional to size but increase as the reused component becomes larger.

Explain in detail why a software system that is used in a real-world environment must change or become progressively less useful.

Answer:

- The number of users may increases the burden of the system, requiring it to expand its hardware capability to handle several connections.
- The business model of the company may change so the system become obsolete and need for a change to cope its requirements.

- The law in the particular country may impose a particular standard to conform a legal usable software.
- The expansion of requirements which requires the software to enable additional features to cope users' requests.

What are essential conditions for software re-engineering to be successful?

Answer:
- The quality of the software to be reengineered is good to the extent of minimum errors occurrence and complete documentation of the system.
- The tools for reengineering is available. Automated tools such as CASE can be factor to decide a successful reengineering.
- The expert staff is available and understand the system works. This can be a potential problem if the system were an old system and no longer used.

What are the strategic options for legacy system evolution? When would you normally replace all or part of a system rather than continue maintenance of the software (with or without re-engineering)?

Answer:
Strategy options for legacy system evolution:
- *Scrap the system completely*. This option is performed when the system is ineffective to the business process or the business process has changed, leaving the system to be obsolete.
- *Leave the system unchange and continue regular maintenance*. This option is chosen when the system is still required but is fairly stable and change requests is rarely happen.
- *Reengineer system to improve maintainability*. This is performed when the system maintenance cost exceeds the cost of reengineering the system.
- *Replace all or part of the system with a new system*. This is to be chosen when old system can't continue operation or where off-the-shelf systems would allow the new system to be developed at a reasonable cost.

Discuss whether it is ethical to instrument software to monitor its use without telling end users that their work is being monitored.

2 answers – see below:

To design a software to monitor the work of individual is ethical since critical-system is concerned about failures. Every action of individual have to observed to detect any anomaly or error and it is functional as a black box to investigate accident causes.
However, not telling the workers that they are being watched is unethical since they deserve to know what's going on in the company. It violates the human rights to obtain information they deserve to know.

Sommerville : In general, I do not think that it is ethical to monitor users without telling them that they are being monitored and without telling them the purpose of the monitoring. Depending on the users, it is arguable whether monitoring is ethical at all in that individuals have a right to privacy and this includes how they use systems. If the users are members of the general public then I believe that this should hold; if, however, the users are all employees of the same organisation then the situation is more difficult and monitoring may be permitted (see cases where employees who have downloaded

pornography from the web have been sacked). The argument for monitoring, of course, is that it allows behaviour to be tracked and the system improved for users. Without monitoring, any improvements are simply guesses. Furthermore, if users know of the monitoring they may change their behaviour so therefore secret monitoring is justified. It can also be argued that it is the system and not the users that is being monitored.

Explain why the rapid delivery and development of new system is often more important to business than the detailed functionality of these systems

Answer:
Rapid delivery focuses on the delivery of the system. It is good for a system that is required to show the result of the system.
It is good for business since the system can be used early if the essential functionality is available and be later improved as the user requirements change. Rapid delivery can make profit swiftly.
However, rapid delivery is not good for critical-system development since the delivery of the system needs to be perfect and without failures.

To help counter terrorism, many countries are planning the development of computer systems that track large numbers of their citizens and their actions. Clearly this has privacy implications. Discuss the ethics of developing this type of system.

Answer:
- The system will violate the privacy of the people which is the based of human rights. In the past years, American citizens has already complained about illegal wiretapping in their house performed by the FBI.
- Even the most recent Patriot Acts which give permission for federal officer to eavesdrop on telephone conversation brings controversy to the community even though it is for the sake of countering terrorism.
- Hence, this system might be oppressed by the citizens and clearly unethical to the privacy of humankind.

In which case might software be scrapped or rewritten?

Answer:
Examples of where software might be scrapped and rewritten are:
• When the cost of maintenance is high and the organisation has decided to invest in new hardware. This will involve significant conversion costs anyway so the opportunity might be taken to rewrite the software.
• When a business process is changed and new software is required to support the process.
• When support for the tools and language used to develop the software is unavailable. This is a particular problem with early 4GLs where, in many cases, the vendors are no longer in business.

What are the strategic options for the evolution of legacy systems?

Answer:
The strategic options for legacy system evolution are:
a. Abandon maintenance of the system and replace it with a new system.
b. Continue maintaining the system as it is.
c. Perform some re-engineering (system improvement) that makes the system easier to maintain and continue maintenance.
d. Encapsulate the existing functionality of the system in a wrapper and add new functionality by writing new code which calls on the existing system as a component.
e. Decompose the system into separate units and wrap them as components. This is similar to the solution above but gives more flexibility in how the system is used.

You would normally choose the replacement option in situations where the hardware platform for the system is being replaced, where the company wishes to standardize on some approach to development that is not consistent with the current system, where some major sub-system is being replaced (e.g. a database system) or where the technical quality of the existing system is low and there are no current tools for re-engineering.