Software Engineering 9

Solutions Manual

IAN SOMMERVILLE

Preface

This solutions manual is intended to help teachers of software engineering courses in marking homework questions for students. Each chapter in the book has 10 exercises of different types, which you may set for students either as is or in a modified form. I have supplied answers to 50% of the exercises in this manual.

The exercises for which answers have not been supplied are, generally, of one of three types:

1. Simple exercises whose answers can be found in the text of the chapter. There are typically one or two of these questions in each chapter and they are intended to stimulate students to read the chapter.
2. Design problems for which there is a range of solutions and you have to use your judgment to decide if the solution is appropriate. Supplying a solution here would imply that there is only one right answer to the question.
3. Ethics-related questions as the aim of these questions is to encourage students to think about the ethics issues involved. The notion of a right and wrong answer does not apply in this case as the student's response to the question depends both on their cultural background and on their particular views on a topic. I suggest that these questions should be used to stimulate class discussions rather than as part of class tests.

It is important when marking the student's answers to exercises to see the supplied solutions as a guide only rather than a definitive statement of the only possible answer to the question. It is generally good educational practice to give students credit for what they know and if they produce credible answers that reveal they have thought about the exercise and have some knowledge of the topic, then this should be rewarded.

This solutions manual may be used in conjunction with the associated quiz book, which lists short questions and answers for each chapter in the book. These can be used for short class tests to assess if students have read the material or as self-assessment tests which the students complete in their own time.

If you think that I have made a mistake in some of these answers (quite possible), please let me know. In some cases, there are obviously several possible answers and you may disagree with my solutions. I'd be delighted to consider including your alternative solutions but I do not have time to engage in detailed email discussions about the exercises in the book.

Ian Sommerville
January 2010

1 Introduction

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

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.

1.3 What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant.

Four important attributes are maintainability, dependability, performance and usability. Other attributes that may be significant could be 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 e.g laptop and mobile platforms) and inter-operability (can it work with a wide range of other software systems).

Decompositions of the 4 key attributes e.g. dependability decomposes to security, safety, availability, etc. is also a valid answer to this question.

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

Problems and challenges for software engineering

There are many possible challenges that could be identified. These include:

1. Developing systems that are energy-efficient. This makes them more usable on low power mobile devices and helps reduce the overall carbon footprint of IT equipment.
2. Developing validation techniques for simulation systems (which will be essential in predicting the extent and planning for climate change).
3. Developing systems for multicultural use
4. Developing systems that can be adapted quickly to new business needs
5. Designing systems for outsourced development
6. Developing systems that are resistant to attack
7. Developing systems that can be adapted and configured by end-users
8. Finding ways of testing, validating and maintaining end-user developed systems

1.5 Based on your own knowledge of some of the application types discussed in section 1.1.2, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.

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 up-front 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.

1.8 Discuss whether professional engineers should be certified in the same way as doctors or lawyers.

These are possible discussion points - any discussion on this will tend to be wide ranging and touch on other issues such as the nature of professionalism, etc.

Advantages of certification

- Certification is a signal to employers of some minimum level of competence.
- Certification improves the public image of the profession.
- Certification generally means establishing and checking educational standards and is therefore a mechanism for ensuring course quality.
- Certification implies responsibility in the event of disputes. Certifying body is likely to be accepted at a national and international level as ‘speaking for the profession’.
- Certification may increase the status of software engineers and attract particularly able people into the profession.

Disadvantages of certification

- Certification tends to lead to protectionism where certified members tend not to protect others from criticism.
- Certification does not guarantee competence merely that a minimum standard was reached at the time of certification.
- Certification is expensive and will increase costs to individuals and organisations.
- Certification tends to stultify change. This is a particular problem in an area where technology developments are very rapid.

2 Software Processes

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

---

1. *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.
2. *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.
3. *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.
4. *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.

2.3 Consider the reuse-based process model shown in Figure 2.3. Explain why it is essential to have two separate requirements engineering activities in the process.

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:

1. 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 of a system/component satisfies some of the requirements and so can be reused.
2. 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.

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

1. 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.
2. 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.

2.6 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.

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 modeling 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.

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

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 shown in Figure 2.13 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 shown in Figure 2.13. These vary from one organization to another and it is not appropriate to impose a particular process in the model.

3 Agile Software Development

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

The principles underlying agile development are:

1. *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team know 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.
2. *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.
3. *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.
4. *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?

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.

Advantages of stories:

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

Disadvantages of stories

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

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.

Reasons why pair programming may be more efficient as the same number of 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..
4. 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.
5. 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 focus on the essential features of the system which they can then produce more quickly.

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.

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.

4 Requirements Engineering

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.

Ambiguities and omissions include:

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

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

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 the 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.

Obviously, these requirements are arbitrary and there are many other possible answers. You simply have to examine their credibility.

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.

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 be related by including them in a table as shown below.

Functional requirement	Related non-functional system requirements	Non-functional requirements
The system shall provide an operation which allows operators to open the release valve to vent steam into the atmosphere.	Safety requirement: No release of steam shall be permitted if maintenance work is being carried out on any steam generation plant.	Timing requirement: The valve must open completely within 2 seconds of the operator initiating the action.

Notice that in this example, the system non-functional requirement would normally take precedence over the timing requirement, which applied to the specific operation.

Obviously, any sensible answer that provides a way of linking functional and non-functional requirements is acceptable here.

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.

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.

Display balance:

Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details
Outputs: Customer's card
Normal operation: The customer authenticates using card and PIN as in Withdraw cash and selects the Display Balance option. The current balance of their account is displayed on the screen. The card is returned to the customer.
Exception: Invalid card. As in Withdraw cash
Incorrect PIN. As in Withdraw cash

Print statement:

Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details
Outputs: Customer's card, Printed statement
Normal operation: The customer authenticates using card and PIN as in Withdraw cash and selects the Print statement option. The last five transactions on their account is printed. The card is returned to the customer.
Exception: Invalid card. As in Withdraw cash
Incorrect PIN. As in Withdraw cash

Change PIN:

Actors: Customer, ATM
Inputs: Customer's card, PIN
Outputs: Customer's card

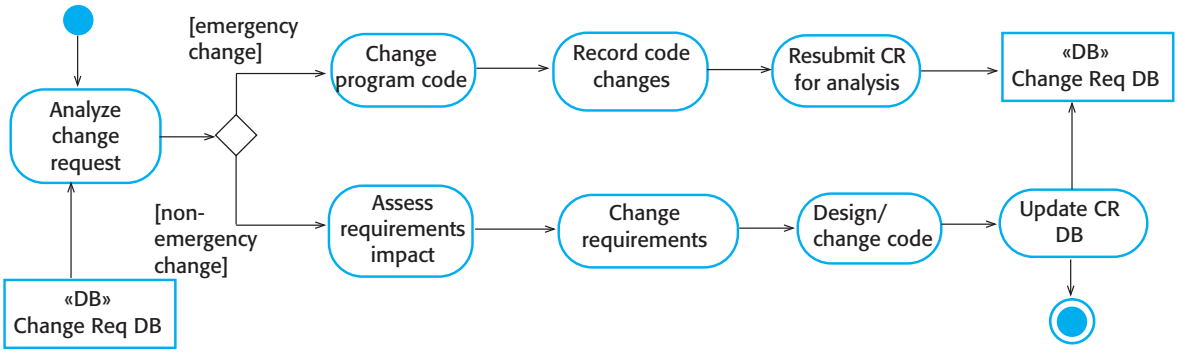
Normal operation: The customer authenticates as in Withdraw cash and selects the Change PIN option. He/she is prompted twice to input the new PIN. The PINS input should be the same. The customer's PIN is encrypted and stored on the card. Card returned to customer.
Exception: Invalid card. As in Withdraw cash.
Incorrect PIN. As in Withdraw cash.
PINS do not match. The customer is invited to repeat the process to reset his/her PIN.

Deposit cash:

Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details, Cash to be deposited
Outputs: Customer's card, Receipt
Normal operation: The customer authenticates as in Withdraw cash and selects the Deposit option. The customer is promoted with a request for the amount of cash to be deposited and inputs the amount. He or she is then issued with a deposit envelope in which they should put the cash then return it to the machine. The customer's account balance is updated with the amount deposited but this is marked as uncleared funds and is not cleared until checked. A receipt is issued and the customer's card is returned.
Exception: Invalid card. As in Withdraw cash.
Incorrect PIN. As in Withdraw cash.
No cash deposited within 1 minute of envelope being issued. Transaction terminated. Card returned to customer.

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.

The following diagram shows a change process that 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.


[emergency change]
Change program code
Record code changes
Resubmit CR for analysis
«DB» Change Req DB
Analyze change request
[non-emergency change]
Assess requirements impact
Change requirements
Design/ change code
Update CR DB
«DB» Change Req DB

5 System Modeling

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?

You might create and use a model of a system that already exists for the following reasons:

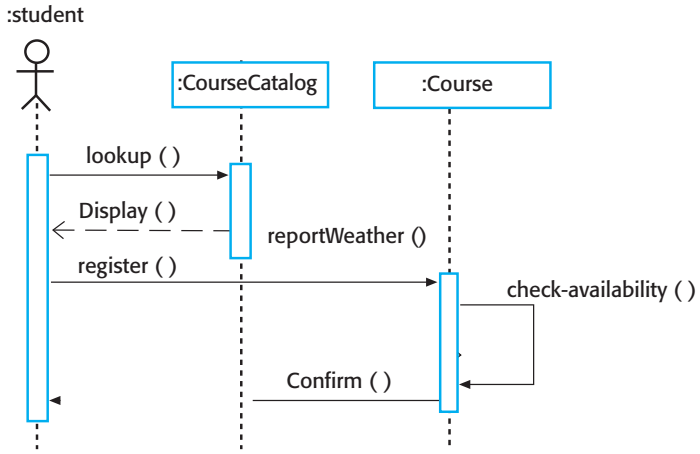
1. To understand and document the architecture and operation of the existing system.
2. To act as the focus of discussion about possible changes to that system.
3. To inform the re-implementation of the system.

You do not need a complete model unless the intention is to completely document the operation of the existing system. The aim of the model in such cases is usually to help you work on parts of the system so only these need to be modelled. Furthermore, if the model is used as a discussion focus, you are unlikely to ne interested in details and so can ignore parts of the system in the model.

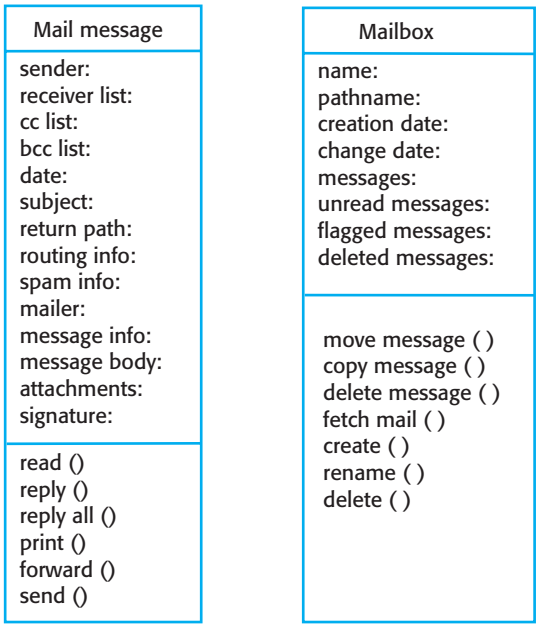
This is true, in general, for models of new systems unless a model-based approach to development is taking place in which case a complete model is required. The other circumstances where you may need a complete model is when there is a contractual requirement for such a model to be produced as part of the system documentation.

5.5 Develop a sequence diagram showing the interactions involved when a student registers for a course in a university. Courses may have limited enrolment, 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.

A relatively simple diagram is all that is needed here. It is best not to be too fussy about things like UML arrow styles as hardly anyone can remember the differences between them.

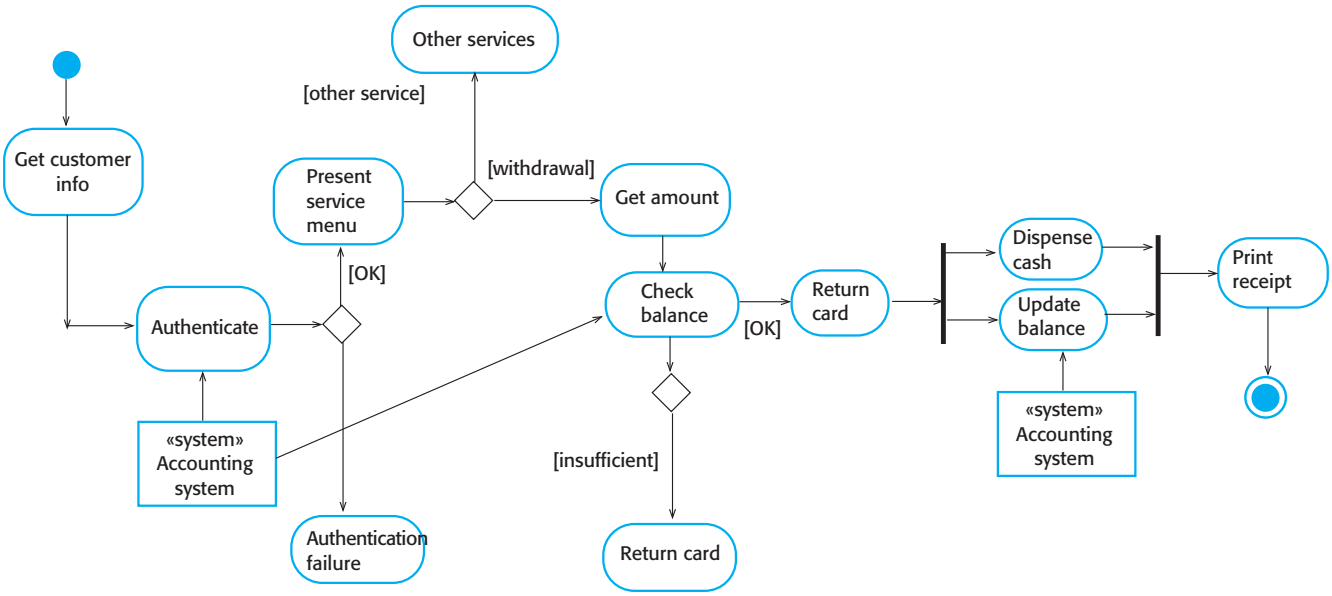


5.6 Look carefully at how messages and mailboxes are represented in the email system that you use. Model the object classes that might be used in the system implementation to represent a mailbox and an e-mail message.



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.

Notice that I have not developed the activities representing other services or failed authentication.



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?

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)

6 Architectural Design

6.1 When describing a system, explain why you may have to design the system architecture before the requirements specification is complete.

The architecture may have to be designed before specifications are written to provide a means of structuring the specification and developing different sub-system specifications concurrently, to allow manufacture of hardware by sub-contractors and to provide a model for system costing.

6.3 Explain why design conflicts might arise when designing an architecture for which both availability and security requirements are the most important non-functional requirements.

Fundamentally, to provide availability, you need to have (a) replicated components in the architecture so that in the event of one component failing, you can switch immediately to a backup component. You also need to have several copies of the data that is being processed. Security requires minimizing the number of copies of the data and, wherever possible, adopting an architecture where each component only knows as much as it needs to, to do its job. This reduces the chance of intruders accessing the data.

Therefore, there is a fundamental architectural conflict between availability (replication, several copies) and security (specialization, minimal copies). The system architect has to find the best compromise between these fundamentally opposing requirements.

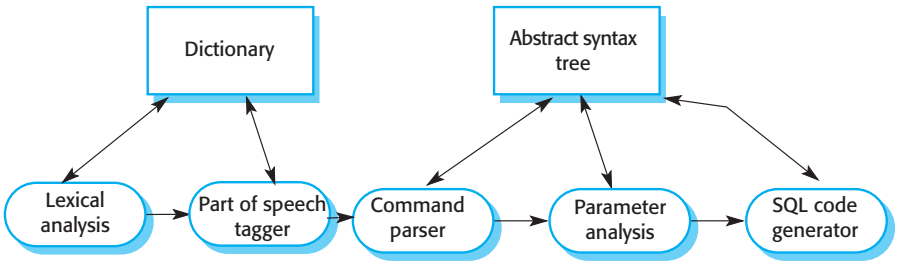
6.7 Explain how you would use the reference model of CASE environments (available on the book's web pages) to compare the IDEs offered by different vendors of a programming language such as Java.

You can make the comparison between the IDEs by taking the different components of the reference model in turn than assess how well the IDE toolset

being studied provides these services. You also have to look at how these services are used in particular toolsets. Generally, IDEs are tightly integrated systems and all parts of the reference model may not be applicable. In this case, comparisons would be drawn using:

1. Data repository services. What kind of data management is supported?
2. Data integration services. How well can data be interchanged with other tools and what support is provided for configuration management?
3. User interface services. What facilities are supported to allow presentation integration? How well integrated at the user interface level are different parts of the systems?
4. Task management services. This is really for general purpose environments so is probably inapplicable to Java IDEs.
5. Message services. How do different components of the IDE communicate?

6.8 **Using the generic model of a language processing system presented here, design the architecture of a system that accepts natural language commands and translates these into database queries in a language such as SQL.**



6.9 Using the basic model of an information system as presented in Figure 6.16, suggest the components that might be part of an information system that allows users to view information about flights arriving and departing from a particular airport.

Students should consider the levels in the information system and should identify components that might be included at each level. Examples of these components might be:

Level 1 (Database level)

Flight database; Flight status database; Airport information;

Level 2: (Information retrieval level)

Status management; Flight management; Search;

Level 3: (User interaction level)

Authentication; session management; forms processing ()

Level 4 (User interface)

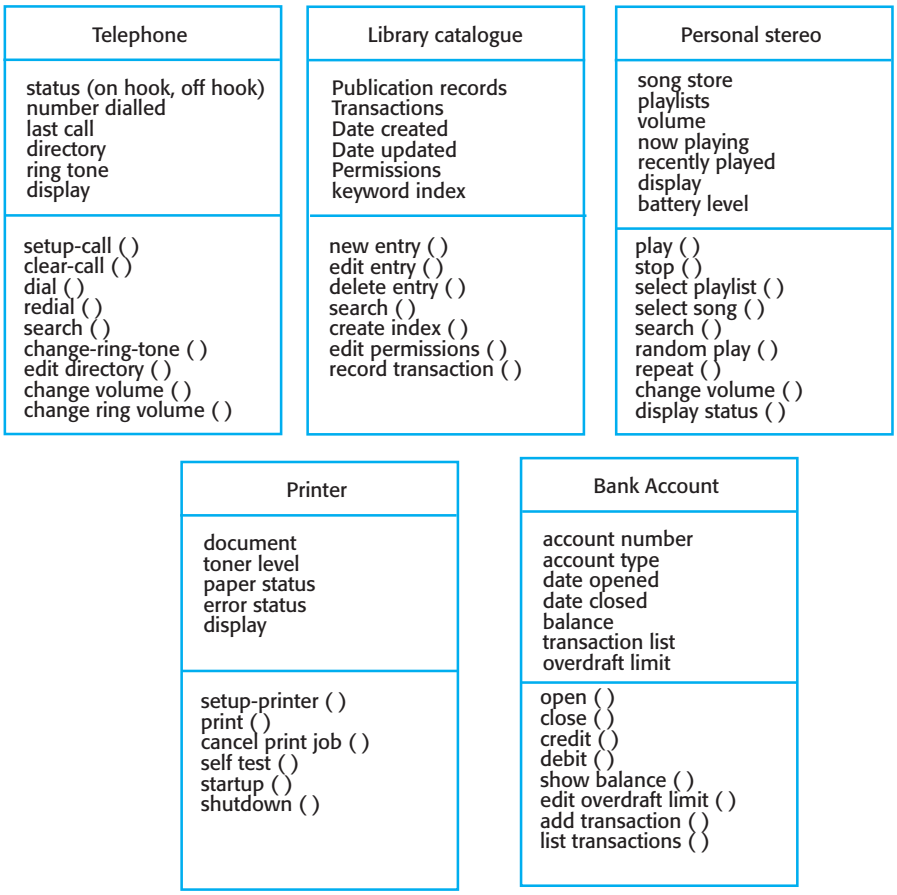
Input checking (Javascript), browser

7 Design and Implementation

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.

System: Weather station
Use case: Report status
Actors: Weather information system, weather station
Data: The weather station sends a status update to the weather information system giving information about the status of its instruments, computers and power supply.
Stimulus: The weather information system establishes a satellite link with the weather station and requests status information.
Response: A status summary is uploaded to the weather information system
Comments: System status is usually requested at the same time as the weather report.

System: Weather station
Use case: Reconfigure
Actors: Weather information system, weather station
Data: The weather information station sends a reconfiguration command to the weather station. This places it into remote control mode where further commands may be sent from the remote system to update the weather station software.
Stimulus: A command from the weather information system.
Response: Confirmation that the system is in remote control mode
Comments: Used occasionally when software updates have to be installed.

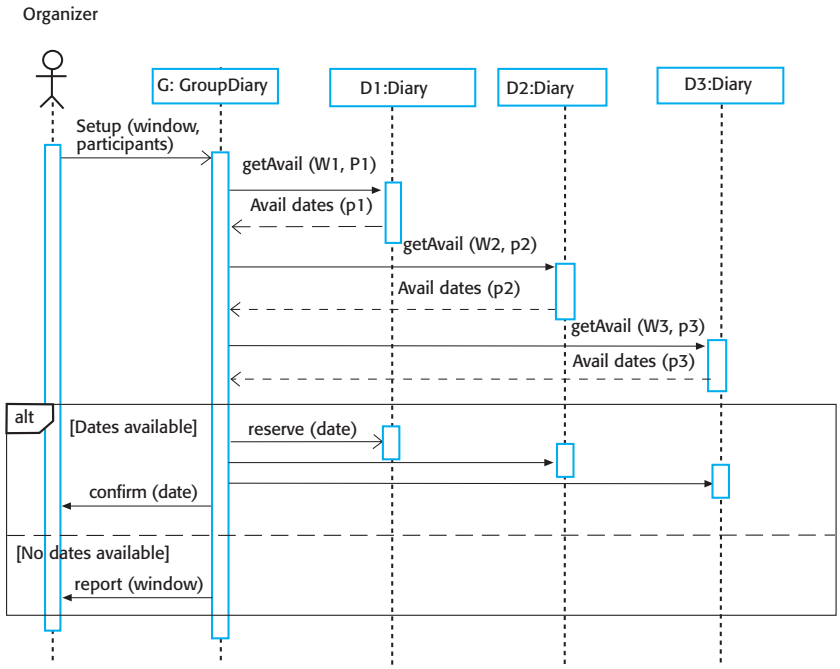


7.3 Using the UML graphical notation for object classes, design the following object classes, identifying attributes and operations. Use your own experience to decide on the attributes and operations that should be associated with these objects.

- a telephone
- a printer for a personal computer
- a personal stereo system
- a bank account
- a library catalogue

There are many possible designs here and a great deal of complexity can be added to the objects. However, I am only really looking for simple objects which encapsulate the principal requirements of these artefacts. Possible designs are shown in the above diagram.

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.



The above diagram assumes there are 3 participants in the meeting, one of whom is the meeting organizer. The organizer suggests a 'window' in which the meeting should take place and the participants involved. The group diary communicates with the diaries of the participants in turn, modifying the window accordingly as there availability is known. So, if the organizer suggests a window of 18th-19th June, the group diary consults the organizer's diary (D1) and finds availability on these days. D2 is then contacted with that availability, not the original window.

If there are no mutually available dates in the window, the system reports this to the organizer. Otherwise, a date is selected, entered in all diaries and confirmed to the organizer.

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

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 versions 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.

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.

8 Software Testing

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

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.

8.4 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.

Testing partitions are:

Strings with only single blank characters

Strings with sequences of blank characters in the middle of the string

Strings with sequences of blank characters at the beginning/end of string

Examples of tests:

The quick brown fox jumped over the lazy dog (only single blanks)

The quick brown fox jumped over the lazy dog (different numbers of blanks in the sequence)

The quick brown fox jumped over the lazy dog (1st blank is a sequence)

The quick brown fox jumped over the lazy dog (Last blank is a sequence)

The quick brown fox jumped over the lazy dog (2 blanks at beginning)

The quick brown fox jumped over the lazy dog (several blanks at beginning)

The quick brown fox jumped over the lazy dog (2 blanks at end)

The quick brown fox jumped over the lazy dog (several blanks at end)

Etc.

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

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.

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

A possible scenario for high-level testing of the weather station system is:

> 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.

There are many possible alternative scenarios here. They should identify the role of the actors involved and should discuss a typical task that might be carried out by that role.

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

Stress testing is where you deliberately increase the load on a system beyond its design limit to see how it copes with high loads. The system should degrade gracefully rather than collapse.

The MHC-PMS has been designed as a client-server system with the possibility of downloading to a client. To stress test the system, you need to arrange for (a) many different clinics to try and access the system at the same time and (b) Large numbers of records to be added to the system. This may involve using a simulation system to simulate multiple users.

9 Software Evolution

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

Systems must change or become progressively less useful for a number of reasons:

1. The presence of the system changes the ways of working in its environment and this generates new requirements. If these are not satisfied, the usefulness of the system declines.
2. The business in which the system is used changes in response to market forces and this also generates new system requirements.
3. The external legal and political environment for the system changes and generates new requirements.
4. New technologies become available that offer significant benefits and the system must change to take advantage of them.

9.4 As a software project manager in a company that specializes in the development of software for the offshore oil industry, you have been given the task of discovering the factors that affect the maintainability of the systems developed by your company. Suggest how you might set up a program to analyze the maintenance process and determine appropriate maintainability metrics for the company.

This is a very open question, where there are many possible answers.

Basically, the students should identify factors which affect maintainability such as (program and data complexity, use of meaningful identifiers, programming language, program documentation etc.). They should then suggest how these can be evaluated in existing systems whose maintenance cost is known and discuss problems of interaction. The approach should be to discover those program units which have particularly high maintenance costs and to evaluate the cost factors for these components and for other components. Then check for correlations.

Other factors may account for anomalies so these should be looked for in the problem components.

9.5 Briefly describe the three main types of software maintenance. Why is it sometimes difficult to distinguish between them?

The three main types of software maintenance are:

1. Corrective maintenance or fault repair. The changes made to the system are to repair reported faults which may be program bugs or specification errors or omissions.
2. Adaptive maintenance or environmental adaptation. Changing the software to adapt it to changes in its environment e.g. changes to other software systems.
3. Perfective maintenance or functionality addition. This involves adding new functionality or features to the system.

They are sometimes difficult to distinguish because the same set of changes may cover all three types of maintenance. For example, a reported fault in the system may be repaired by upgrading some other software and then adapting the system to use this new version (corrective + adaptive). The new software may have additional functionality and as part of the adaptive maintenance, new features may be added to take advantage of this.

9.7 Under what circumstances might an organization decide to scrap a system when the system assessment suggests that it is of high quality and high business value?

Examples of where software might be scrapped and rewritten are:

1. 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.
2. When a business process is changed and new software is required to support the process.
3. 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.

There are other reasons why software may be scrapped, depending on local cirumstances.

9.8 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?

The strategic options for legacy system evolution are:

1. Abandon maintenance of the system and replace it with a new system.
2. Continue maintaining the system as it is.
3. Perform some re-engineering (system improvement) that makes the system easier to maintain and continue maintenance.
4. 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.
5. 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.

10 Sociotechnical Systems

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.

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.

10.4 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.

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:

1. 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.
2. 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.

10.6 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?

A range of answers is possible here. Possible issues covered in the solution might be:

1. Museums are conservative places and some staff may resent the introduction of new technology.
2. 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.
3. Other areas of the museum may oppose the system because they see it as diverting resources from their work.
4. Different museums may have different preferred suppliers for the equipment so that all equipment used is not identical thus causing support problems.
5. 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.
6. 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.

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.

Legacy systems may be critical for the successful operation of a business for two basic reasons

1. 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 that support this are critical. They must be maintained.

2. They may incorporate organizational 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 organization loses valuable knowledge.

11 Security and Dependability

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

Six reasons why dependability is important are:

1. Users may not use the system if they don't trust it.
2. System failure may lead to a loss of business.
3. An undependable system may lose or damage valuable data.
4. An undependable system may damage its external environment.
5. The reputation of the company who produced the system may be damaged hence affecting other systems.
6. The system may be in breach of laws on consumer protection and the fitness of goods for purpose.

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

Internet server: **Availability** as failure of availability affects a large number of people, the reputation of the supplier and hence its current and future income.

A computer-controlled scalpel: **Safety** as safety-related failures can cause harm to the patient.

A directional control system: **Reliability** as mission failure could result from failure of the system to perform to specification.

An personal finance management system: **Security** because of potential losses to users.

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

Possible domestic appliances that may include safety-critical software include:

Microwave oven

Power tools such as a drill or electric saw

Lawnmower

Central heating furnace

Garbage disposal unit

Vacuum cleaner

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.

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 behavior (the system 'shall not'). If the specification does not explicitly exclude dangerous behavior then a system can be reliable but unsafe.

11.7 In a medical system that is designed to deliver radiation to treat tumours, 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.

A 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.

Software features that may be included to guard against system failure are the delivery of radiation in increments with a 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.

12 Dependability and Security Specification

12.3 In the insulin pump system, the user has to change the needle and insulin supply at regular intervals and may also change the maximum single dose and the maximum daily dose that may be administered. Suggest three user errors that might occur and propose safety requirements that would avoid these errors resulting in an accident.

Possible user errors are:

1. Maximum daily dose set wrongly
2. Maximum single dose set wrongly
3. Failure to replace empty insulin reservoir
4. Insulin reservoir improperly fitted
5. Needle improperly fitted

Examples of safety requirements to avoid these errors are:

1. When the maximum dose and the maximum daily dose is changed, the user should be asked to input the changed values twice.
2. If the maximum daily dose has already been set by the user then the new daily dose should be no more than 1.25 and no less than 0.75 of the previous maximum daily dose.
3. The insulin reservoir case should be designed so that it is only possible to fit the insulin bottle the right way and the case should not close unless the bottle is properly seated.
4. If the back pressure from the needle assembly is more than XX then the system should shut down and issue an audible and text warning. This allows for blocked needles as well as improperly fitted needles.

12.4 A safety-critical software system for treating cancer patients has two principal components:

- A radiation therapy machine that delivers controlled doses of radiation to tumor sites. This machine is controlled by an embedded software system.
- A treatment database that includes details of the treatment given to each patient. Treatment requirements are entered in this database and are automatically downloaded to the radiation therapy machine.

Identify three hazards that may arise in this system. For each hazard, suggest a defensive requirement that will reduce the probability that these hazards will result in an accident. Explain why your suggested defense is likely to reduce the risk associated with the hazard.

Hazards:

1. Incorrect dosage of radiation computed
2. Radiation delivered to the wrong site on patient's body
3. Data for wrong patient used to control machine
4. Data transfer failure between database and therapy machine

Software protection:

1. Comparison with previous doses delivered. Establishment of a maximum monthly dose which may never be exceeded. Feasibility checks (e.g. for negative dosages). Confirmation of dose to be delivered by operator. Continuous visual display of dose being delivered.
2. Comparison with delivery site in previous treatment. Light used to illuminate site of radiation delivery. Operator confirmation of site before machine can operate.
3. Patient asked to verify name, address and age before machine starts by pressing button. Issue patient with a personal treatment card which is handed over to identify patient. Maintain separate list of patients to be treated each day and correlate with patient databases. Force machine operator to verify list and database consistency before starting machine.
4. Dual display of information in therapy machine and database. Highlighting of differences in operator display. Locking of machine until information is consistent. Use of check digits and other error checking codes in the data. Duplicate communication channels between machine and database.

12.5 Suggest appropriate reliability metrics for the classes of software system below. Give reasons for your choice of metric. Predict the usage of these systems and suggest appropriate values for the reliability metrics.

- a system that monitors patients in a hospital intensive care unit
- a word processor
- an automated vending machine control system
- a system to control braking in a car
- a system to control a refrigeration unit
- a management report generator

See following table. Note that the values in this table are really quite arbitrary and you need to know more about the domain to set accurate values. Any values which take into account the type of system involved are equally good.

System	Reliability metric	Suggested value	Rationale
Patient monitoring system	Availability	System should be unavailable for less than 20 minutes per month.	The system needs to be continuously available as patients may be admitted or discharged at any time. The chosen figure is acceptable because, if necessary, critical system functions can be taken over manually.
Word processor	ROCOF	Failures resulting in loss of data should not occur more than once per 1200 hours of use.	
Vending machine controller	POFOD (Probability of failure on demand)	Failure acceptable in 1:5000 demands	Not a critical system so relatively high failure rate is OK.
Braking system controller	POFOD	The software should never fail within the predicted lifetime of the system.	Very critical system. Failure is unacceptable at any time.
Refrigeration unit control	Availability	20 minutes per month	Non-stop system but not critical. Short periods of failure are not a real problem as temperature takes some time to rise.
Management report generator	ROCOF	1 fault/100 hours of use	Not a critical system. Faults are unlikely to cause severe disruption.

12.6 A train protection system automatically applies the brakes of a train if the speed limit for a segment of track is exceeded, or if the train enters a track segment that is currently signaled with a red light (i.e. the segment should not be entered). Giving reasons for your answer, chose a reliability metric that might be used to specify the required reliability for such a system.

The most appropriate reliability metric is Probability of Failure on demand (POFOD). This is the probability that the system will respond correctly when a request is made for service at a given point in time. This metric is used for protection systems where demands for service are intermittent and relatively infrequent over the lifetime of the system.

12.7 There are two essential safety requirements for the train protection system:

- The train shall not enter a segment of track that is signaled with a red light.
- The train shall not exceed the specified speed limit for a section of track.

Assuming that the signal status and the speed limit for the track segment are transmitted to on-board software on the train before it enters the track segment, propose five possible functional system requirements for the onboard software that may be generated from the system safety requirements.

There are several different possibilities here. Some examples:

1. The system shall ensure that the train brakes are applied when a 'red signal' is received.
2. The system shall sound an alarm in the driver's cabin when a 'red signal' is received.
3. The system shall compare the train speed with the segment speed limit once per second.
4. If the train speed exceeds the segment speed limit and the train throttle position is not zero then the throttle position should be reset to zero.
5. If the train speed exceeds the segment speed limit and the train deceleration is less than the comfortable decleration limit then the train brakes should be applied.

13 Dependability Engineering

13.4 What is the common characteristic of all architectural styles that are geared to supporting software fault tolerance?

The common characteristics of all styles to support fault tolerance is that there are multiple separate implementations of system functionality and some error detection mechanism that can detect possible software failures.

13.6 You are responsible for the design of a communications switch that has to provide 24/7 availability, but which is not safety-critical. Giving reasons for your answer, suggest an architectural style that might be used for this system.

The clue here is in the question – the system is not safety critical so eliminates protection systems. However, there is a need for availability so the most appropriate architectural pattern is an N-version programming architecture or a replicated server architecture with each server running a different OS.

13.7 It has been suggested that the control software for a radiation therapy machine, used to treat patients with cancer, should be implemented using N-version programming. Comment on whether or not you think this is a good suggestion.

Advantages of N-version programming

1. Increases design diversity so probability of faults that result in failures should be reduced
2. Increases availability of the system

Disadvantages

1. Increased cost because of the need to use independent development teams
2. Increased software complexity because of the need for a fault tolerant controller. Increased complexity increases the probability of error
3. Improvement in reliability in practice is limited because of the possibility of common errors made by different development teams.

N-version programming would not be a good design strategy for this type of software. There is no need for high availability and the increased complexity and cost would make the overall cost of the machine too high.

13.8 Give two reasons why different versions of a system based around software diversity may fail in a similar way.

---

1. There may be a specification error that is reflected in both versions.
2. The problem may be a numeric error that has not been explicitly trapped.
3. The specification may be ambiguous and may be misunderstood in the same way by both teams.

13.9 Explain why you should explicitly handle all exceptions in a system that is intended to have a high level of availability.

You should handle all exceptions explicitly because the default exception handler in most systems causes the application system to stop executing. Obviously, this is unacceptable in systems that have to have a high level of availability. Even where some other exception handling strategy is used, it is unlikely that a single strategy is appropriate for all different types of exception and the strategy used may lead to a loss of services in the application system.

14 Security Engineering

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

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.

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

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?

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.

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

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.

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.

15 Dependability and Security Assurance

15.1 Explain when it may be cost-effective to use formal specification and verification in the development of safety-critical software systems. Why do you think that critical systems engineers are against the use of formal methods?

Formal methods can be cost-effective in the development of safety-critical software systems because the costs of system failure are very high and so additional cost in the development process is justified. Most safety-critical systems have to gain regulatory approval before they are used and it is a very expensive process to convince a regulator that a system is safe. The use of a formal specification and associated correctness argument may be less than the costs e.g. of additional testing to convince the regulator of the safety of the system.

Some developers of systems are against the use of formal methods because they are unfamiliar with the technology and unconvinced that a formal specification can be complete representation of the system. Furthermore, the problem with formal specifications are that they cannot be understood by system customers so they may conceal errors and give a false picture of the correctness of the system.

15.3 Explain why it is practically impossible to validate reliability specifications when these are expressed in terms of a very small number of failures over the total lifetime of a system.

To measure reliability you need to have statistically valid failure data for the system so you need to induce more failures than are specified in the given time period. However, because the number of failures is so low, this will take an unrealistically large amount of time.

15.6 Suggest how you would go about validating a password protection system for an application that you have developed. Explain the function of any tools that you think may be useful.

Validating a password protection system involves:

1. Identifying possible threats. The principal threats are

   a. Attacker gains access without a password

   b. Attacker guesses a password of an authorised user

   c. Attacker uses a password cracking tool to discover passwords of authorised users

   d. Users make passwords available to attackers

   e. Attacker gains access to an unencrypted password file

2. Developing tests that cover each of these threats

   a. Test system for all authorised used to check that they have set a password.

   b. Test system heuristically for commonly used passwords such as names of users, festivals, other proper names, strings such as '12345' etc.

   c. Check that all user passwords are not words that are in a dictionary. A password cracking tool usually checks encrypted passwords against the same encryptions of words in a dictionary.

   d. This is very hard to check. To stop users writing down passwords you need to allow words that are in the dictionary and are hence easy to remember.

   e. Check that access to the password file is very limited. Check that all copy actions on the password file are logged.

15.8 List four types of systems that may require software safety cases, explaining why safety cases are required.

Safety cases would normally be required for any system that needs to be certified by a regulator before it is used e.g.:

1. Systems used to control equipment in the nuclear industry where there is a possibility of the release of radioactivity.
2. Air traffic control software

3. Signalling and control systems in the railway industry.

4. Software for critical aircraft functions such as flight control systems.

15.9 The door lock control mechanism in a nuclear waste storage facility is designed for safe operation. It ensures that entry to the storeroom is only permitted when radiation shields are in place or when the radiation level in the room falls below some given value (dangerLevel). So:

(i) If remotely controlled radiation shields are in place within a room, an authorized operator may open the door.

(ii) If the radiation level in a room is below a specified value, an authorized operator may open the door.

(iii) An authorized operator is identified by the input of an authorized door entry code.

The code shown in Figure 15.12 controls the door-locking mechanism. Note that the safe state is that entry should not be permitted. Using the approach discussed in section 15.5.2, develop a safety argument for this code. Use the line numbers to refer to specific statements. If you find that the code is unsafe, suggest how it should be modified to make it safe.

There are two potential safety problems with this code:

1. Say the door was unlocked when the door entry code was entered. Line 13 checks if the state is safe and, if it is safe then unlocks the door. However, if the door was unlocked to begin with, there is no locking action if the state is unsafe so therefore a potential safety loop hole exists.

2. If the radiation level is less than the danger level then line 8 sets the state to be safe. However, line 10 checks the shields to see if they are in place. If they are not in place, the state is unchanged although, in fact, the system is unsafe if the shields are down. Therefore, the door can be opened with the shields down and a safety loophole exists.

There are two changes which should be made to ensure that the code is safe:

• An initial statement which locks the door and sets door locked to be true.

• The if statement if shield-status == Shield.inPlace then should be changed to:

```
if (shield_status  == Shield.inPlace())
   state  := safe;
else
   state  := unsafe;
```

There are other ways to do this with nested if statements.

16 Software Reuse

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

If savings from reuse were proportional to the amount of code reused, then reusing 2000 lines of code would save twice as much as reusing 1000 lines of code. However, to reuse 2000 lines of code, that code must be understood and the costs of program understanding are not linear – the more code to be understood, the more effort it takes to understand it. Furthermore, more changes may be required, the larger the amount of code reused so this also adds to the costs of reusing more code.

Of course, all this is only true if the code has to be understood before it is reused. If it can be reused without change, then savings from reusing large chunks of code tend to be proportionally greater than savings from reusing small code fragments.

16.3 Give four circumstances where you might recommend against software reuse.

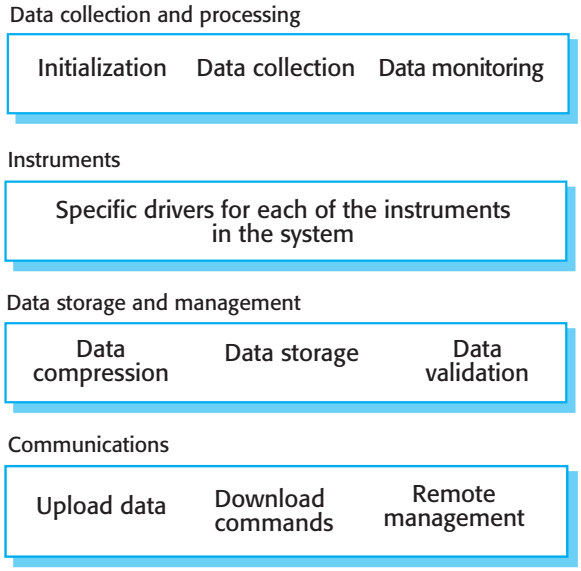
Circumstances where software reuse is not recommended:

1. 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.
2. In critical applications where source code is not available. Testing the code to the required standards may be very difficult.
3. In small systems where the costs of reuse are comparable to the savings that result if code is reused.
4. In systems where performance is a critical requirement – specially developed code can usually be made more efficient.

16.5 Using the example of the weather station system described in Chapters 1and 7, suggest a product line architecture for a family of applications that are concerned with remote monitoring and data collection. You should present your architecture as a layered model, showing the components that might be included at each level.

There are various options here. The important characteristic of the solution is to have clear functional separation between the layers. I have suggested a 4-layer system as shown below:

1. Communications (the lowest layer). All components that are concerned with interaction with a remote system.
2. Data storage and management. All components that are concerned with looking after the data that has been collected.
3. Instruments. All components that are concerned with managing the specific instruments in the system.
4. Data collection and processing. All components that control the collection, processing and monitoring of the collected data.



16.8 Identify six possible risks that can arise when systems are constructed using COTS. What steps can a company take to reduce these risks?

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

1. Vendor risks: Failure of vendor to provide support when required

   Vendor goes out of business or drops product from its portfolio

2. Product risks: Incompatible event/data model with other systems

   Inadequate performance when integrated with other systems

   Product is undependable in intended operating environment

3. 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.

16.9 Explain why adaptors are usually needed when systems are constructed by integrating COTS products. Suggest three practical problems that might arise in writing adaptor software to link two COTS application products.

Adaptors are usually required because the COTS products are independently developed at different times and, therefore, they are not designed for integration. There is no reason why the database organisation, user interfaces, APIs, etc. should take into account the integration with other components.

Three practical problems that may arise when integrating COTS application systems are:

1. *Missing information*. Application A may require information from application B to work properly. However, application B may not need this information (or it may be optional) and so it cannot guarantee that it can be made available to application A.

2. *Control incompatibilities*. Application A and application B may have different control philosophies. For example, application A may be reactive, depending on the user inputs whereas application B may be proactive and use workflow-based, pre-defined interaction.

3. *Semantic mismatches.* This occurs when different applications use the same name for some kind of information but that actually means different things.

For example, a system that manages project budgets may record ‘start date’ of the project as the date after which funding may be spent whereas the project management system considers start date to the date that work actually started.

17 Component-based Software Engineering

17.1 Why is it important that all component interactions are defined through 'requires' and 'provides' interfaces?

It is important to define all interactions through requires and provides interfaces so that the use of the component is completely independent of its implementation. If component interactions use some knowledge of the components that is not defined in the requires/provides interfaces then the coupling between the components is increased and it is harder to interchange one component for an equivalent component with the same interfaces.

17.3 What are the fundamental differences between components as program elements and components as services?

Differences between components and web services:

1. Once a component is purchased, it is owned by the user whereas the web service is always owned by the provider. This is significant because it means that the owner has no control over changes to the service – if it changes (or disappears) then this may have adverse consequences for the user. With components, however, the user decides when newer versions are to be used.
2. Payment for services is by utilization so that users don't have to buy an expensive component that is only occasionally used.
3. Component interactions can use much more efficient protocols than web services so components are better suited to high throughput/performance applications.
4. There are widely-accepted standards for services against several competing standards for components so service inter-operability is (should be) much better.

17.5 Using an example of a component that implements an abstract data type such as a stack or a list, show why it is usually necessary to extend and adapt components for reuse.

Take, for example, a stack component. This will provide basic operations that are common to all stacks such as Initialise (Create a stack), Push (an item onto the stack), Pop (an item from the stack), Size (the number of items currently on the stack), and perhaps others. However, each application will use stacks in different ways and so may require different versions of these operations and additional operations.

For example, consider a graphical browsing operation that allows users to browse a digital library. The library is divided into areas and the identifier for each area points to the books in that area. When the user enters an area, its identifier is pushed onto a stack and popped from the stack when he or she leaves that area and goes back to the previous area. Thus, the top of the stack always refers to the books in the current area. However, there is a requirement to provide a facility where the user can view all areas visited and this requires an additional stack operation that provides access to all stack elements. This then has to be added to the stack component. It also requires the Pop operation to be modified so that, when an item is popped from the stack it is added to a 'visited areas' list that can be displayed in conjunction with the current stack elements.

17.6 Explain why it is difficult to validate a reusable component without the component source code. In what ways would a formal component specification simplify the problems of validation?

Component validation without source code is very difficult because there is no way of assessing how the component handles exceptions (and this is rarely defined in a component specification). The only validation method that can be used is black-box testing so static techniques cannot be used. Component specifications are rarely complete and this increases the problems of black-box testing. Formal specifications would help because they would precisely define what the component was supposed to do and its actual behaviour could be compared to the specification. However, formal specification rarely cover all exceptions and they do not help with testing performance, dependability or other non-functional characteristics.

17.8 Using examples, illustrate the different types of adaptor needed to support sequential composition, hierarchical composition and additive composition.

1. In sequential composition, Component C is created by composing A and B in sequence i.e. A; B. For example, in an object oriented system, the code of C would be implemented as a call to method A in object class X followed by a call to method B in object class Y.
2. In hierarchical composition, Component C is created from Component A calling component B. In an object oriented program, component C could be a method that calls a method X.A. Within X.B, there is a call to Y.B.
3. In additive composition, Component C is created by integrating the interfaces of component A and component B to create the interface of component C. In an object oriented program, this would be implemented by creating a new class C, which includes the interfaces of classes A and B.

18 Distributed Software Engineering

18.3 Using an example of a remote procedure call, explain how middleware coordinates the interaction of computers in a distributed system.

In a remote procedure call, an executing component on one computer (A) calls a procedure or method, which is part of a component that is executing on a different computer (B). The role of the middleware is to coordinate this interaction. There are several steps involved in this:

1. The provision of a stub procedure with the same interface as the called component. Calling this stub procedure initiates a call to the system middleware.
2. The middleware running on computer A accepts the call and discovers the location of the called component.
3. It translates the parameters into a standard format and sends these to computer B along with a request to call the required component.
4. The middleware on computer B converts the parameters into the appropriate format for the language of the called component and then calls that component.
5. After execution, the called component returns the result to the middleware on computer B which then translates this into the middleware standard format.
6. The result is transmitted to the middleware on computer A, which then translates that into the appropriate language format and returns it to the original calling component.

18.4 What is the fundamental difference between a fat-client and a thin-client approach to client–server systems architectures?

In a fat-client system, some of the application processing is carried out on the client whereas in a thin client system only the user interface is displayed on the client and all of the application processing is carried out on the server. However, modern web browsers are all javascript enabled which means that code can be downloaded from the web page on the server and executed within the client browser. This means that some of the functionality of fat clients can be replicated without the need to install software on the client system.

18.6 Your customer wants to develop a system for stock information where dealers can access information about companies and evaluate various investment scenarios using a simulation system. Each dealer uses this simulation in a different way, according to his or her experience and the type of stocks in question. Suggest a client–server architecture for this system that shows where functionality is located. Justify the client–server system model that you have chosen.

In this case, I would chose a fat client model with company information located on a central server (this is critical information and its important that it is consistent for all dealers). Simulations would run on the dealer's computer as these are used in different ways depending on the individual dealers. A fat client architecture is required because simulations require considerable processing and it would place an unacceptable load on the server if several dealers started simulations at the same time.

18.8 Give two advantages and two disadvantages of decentralized and semi-centralized peer-to-peer architectures.

Advantages of decentralized p2p architectures:

Less vulnerable to denial of service attack

Scaleable – system performance should not be adversely affected by adding peers.

Disadvantages of decentralized p2p architectures:

Peer communications more difficult to organize.

More expensive to discover what peers are available on the network.

Advantages of semi-centralised p2p architectures:

Easy to keep track of what peers are available.

Peer data exchange is simplified – it takes place via the server.

Disadvantages of semi-centralised p2p architectures:

Failure of server results in system unavailability.

Vulnerable to denial of service attack on server

18.9 Explain why deploying software as a service can reduce the IT support costs for a company. What additional costs might arise if this deployment model is used?

Deploying software as a service has the potential for reducing the IT support costs as there is no need to install and support separate software on each client. Rather, all software is hosted on a server and when e.g. upgrades are required, only the server (or servers) need be upgraded. There are no support problems with different computers in an organization running different software versions. General help support is provided by the service provider rather than the local IT staff.

The additional costs that can arise from this model are:

1. Network costs, as obviously there is a considerable increase in network traffic. Service providers (such as Amazon) may charge for data uploads and downloads. This is only applicable of the service is provided by a 3rd party rather than in-house.
2. Server costs, as the servers are responsible for all computation and so must either be more powerful or more numerous. This is most significant if the service is provided in-house.
3. There may in fact be additional support costs from this model in the short-term if it requires users to change the software that they normally use. This is likely to lead to additional demands for help.

19 Service-oriented architecture

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

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.

19.3 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.

```
<xs: complexType name = "MaxMinType"
        <xs: sequence>
                <xs:element name = "maxtemp" type = "xs:integer"/>
                <xs:element name = "mintemp" type = "xs:integer"/>
 <!--  Assume that an enumerated type called tempscale with values Celsius and Fahrenheit has been defined →
                <xs:element name = "scale" type = "xs:tempscale" />
        </xs:complexType>
```

```
        <xs: complexType name = "InDataFault"
         <xs: sequence>
                <xs:element name = "errorcode" type = "xs:integer"/>
                <xs:element name = "severity" type = "xs:integer"/>
        </xs:complexType>
```

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

1. 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.
2. 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.

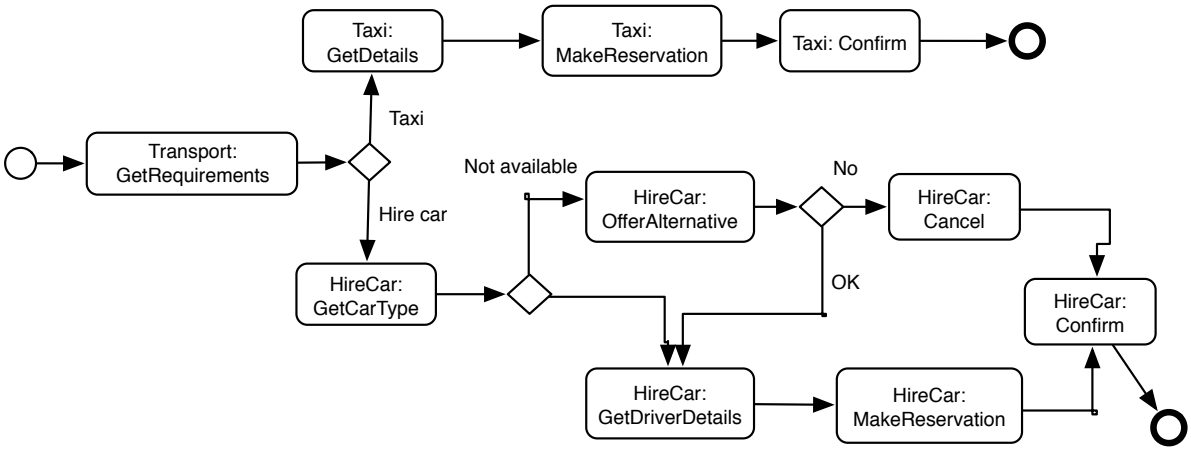
19.8 Explain what is meant by a 'compensation action' and, using an example, show why these actions may have to be included in workflows.

A compensation action is an action that is included in a workflow to 'undo' a transaction that has been completed earlier in the workflow.

Compensation 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.

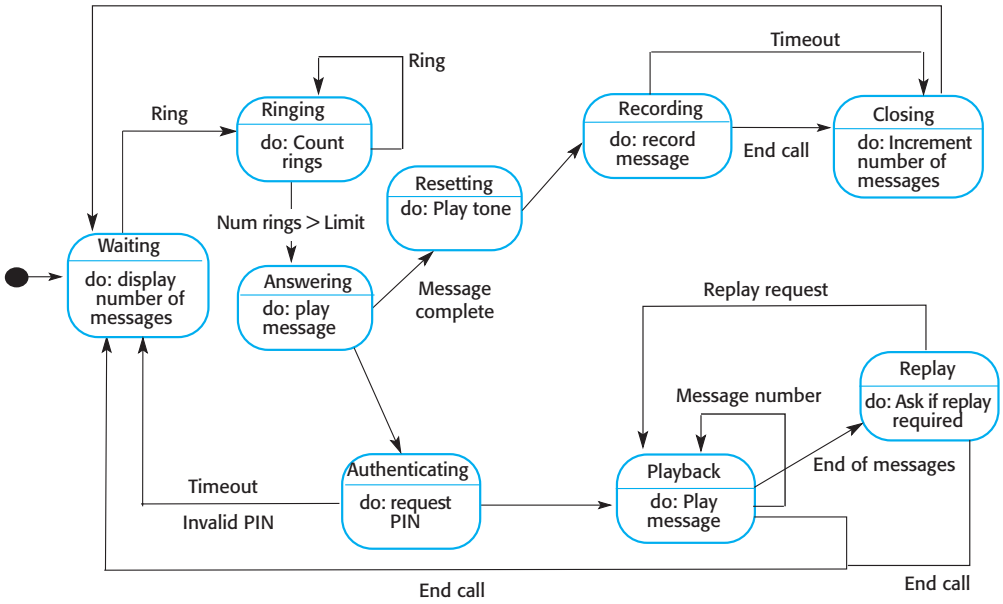
Compensation actions are required when dependent services, offered by different suppliers, are composed to create an integrated service. For example, say a meeting organiser has to book a room for a meeting then organise catering for the people attending the meeting. It may not be possible to do this 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 a hire car. You may assume that the taxi and car hire companies offer web services to make a reservation.



20 Embedded Software

20.3 Using the state-based approach to modeling, as discussed in section 20.1.1, model the operation of an embedded software system for a voicemail system included in a landline phone. This should display the number of recorded messages on an LED display and should allow the user to dial-in and listen to the recorded messages.



There are many possible solutions to this question depending on how detailed a model is produced. The diagram shows one possible state model. Notice this does not allow for message deletion (as this was not asked for in the question). Adding deletion after playback is simple with an additional state that deletes the message and reduces the message count.

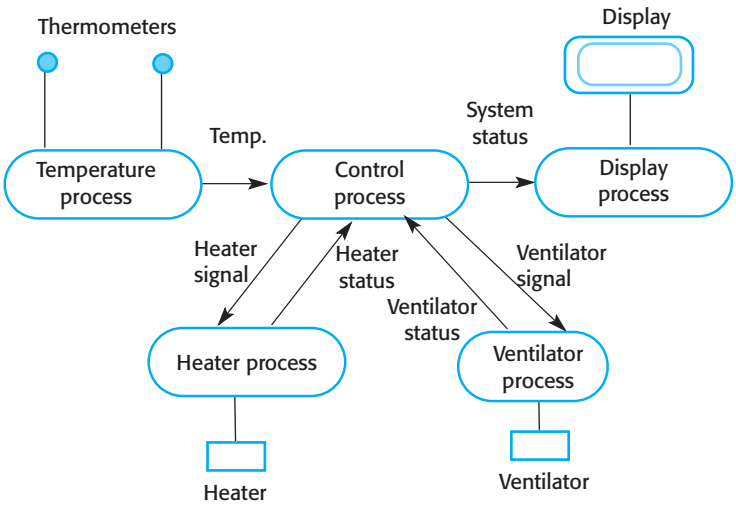
20.4 Explain why an object-oriented approach to software development may not be suitable for real-time systems.

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.

The student may answer this in terms of problems with Java e.g. unpredictable timing due to garbage collection. I was really looking for a more general discussion but you may think this is OK.

20.5 Show how the Environmental Control pattern could be used as the basis of the design of a system to control the temperature in a greenhouse. The temperature should be between 10 and 30 degrees Celsius. If it falls below 10 degrees, the heating system should be switched on; if it goes above 30, the windows should be automatically opened.



Again there are several possible answers here. Students will often over-complicate the diagram by considering situations such as too cold and ventilator open. It is best to have a simple diagram as shown here + a separate table showing the signals to be produced.

	Heater status	**Ventilator status**	**Signal**
Temp < 10	On	Closed	
Temp < 10	Off	Open	Ventilator close Heater on
Temp < 10	Off	Closed	Heater on
Temp < 10	On	Open	*Should never arise*
Temp > 10, < 30	On	Closed	Heater off
Temp > 10, < 30	Off	Closed	
Temp >10, <30	On	Open	*Should never arise*
Temp > 10., < 30	Off	Open	Ventilator close
Temp > 30	On	Closed	Heater off Ventilator open
Temp > 30	Off	Closed	Ventilator open
Temp> 30	On	Open	*Should never arise*
Temp > 30	Off	Open	

20.7 A train protection system automatically applies the brakes of a train if the speed limit for a segment of track is exceeded or if the train enters a track segment that is currently signalled with a red light (i.e. the segment should not be entered). Details are shown in Figure 20.19. Identify the stimuli that must be processed by the on-board train control system and the associated responses to these stimuli.

Stimulus	**Associated response**
Train speed information	Check that the speed is within the speed limit for the current segment. If speed limit is exceeded, sound an audible warning in driver's cabin
Brake status information	If the train speed is within the speed limit, no action; if the train speed exceeds the speed limit then the brakes should be applied
Trackside transmitter signal	Set speed limit for current segment. Set status of current segment. If status is 'red', apply the train brakes

20.9 If a periodic process in the on-board train protection system is used to collect data from the trackside transmitter, how often must it be scheduled to ensure that the system is guaranteed to collect information from the transmitter? Explain how you arrived at your answer.

To estimate the period of the process for trackside information collection, we must be able to ensure that the process will always be scheduled during the period can access the trackside information and collect information before it passes out of range of the transmitter. Here we use the maximum train speed (50 m/sec), the transmission time (50 ms) and the fact that the train must be within 10 m of the transmitter before it can collect information.

Simplistically, the train is within 10 m of a transmitter for 400 ms assuming that the distance between the train and the transmitter is negligible. It takes 50 ms to collect the information so collection must start within 350 ms of entering a segment. Therefore, if the process collecting data from the transmitter is scheduled 3 times/second, it ought to always be able to collect transmitter data.

21 Aspect-oriented Software Engineering

21.1 What are the different types of stakeholder concern that may arise in a large system? How can aspects support the implementation of each of these types of concern?

1. Functional concerns which reflect specific functionality required. The base system should implement core functionality and extensions, implemented as aspects, can implement secondary functionality.
2. Quality of service concerns related to non-functional behaviour of the system. Aspects may be used to implement cross-cutting functionality, such as a cache, which helps these requirements to be met.
3. Policy concerns relating to the overall policies of use of the system. These are inevitably cross-cutting. Aspects may be used to implement these concerns.
4. System concerns which relate to the attributes of the system as a whole. Aspects may be used to implement monitoring that checks the system attributes.
5. Organisational concerns that are related to organisational goals and priorities such as maintaining reputation. Aspects have limited usefulness in implementing this type of concern.

21.2 Summarize what is meant by tangling and scattering. Using examples, explain why tangling and scattering can cause problems when system requirements change.

Tangling arises when one module implements (part of) several requirements; scattering arises where the implementation of a single requirement is spread across several modules.

Say several requirements in a system have a shared component that implements the issuing of a confirmation to a user that some action has occurred. This is tangled system. If one of these changes so that some change to the confirmation is required, all of the other requirements that issue confirmations are affected by this.

If the implementation of each requirement which issues a confirmation is scattered across several modules, then all of these have to be checked that the change will not affect their operation.

22.5 What viewpoints should be considered when developing a requirements specification for the MHC-PMS? What are likely to be the most important cross-cutting concerns?

- Viewpoints: Clinical, Administrative, Patients, Carers, Legal
- Cross-cutting concerns: Privacy and data protection, Safety, Security

---

21.8 Explain how aspect interference can arise and suggest what should be done during the system design process to reduce the problems of aspect interference.

Aspect interference can arise where two or more aspects specify that advice should be woven into the system at the same pointcut. The effect of this is that the system has to decide which aspect should have priority over the other and how the aspects should be composed.

- Use of a consistent approach to naming means that the probability of accidental interference can be reduced.
- Pointcuts specified as patterns are also best avoided as these again may result in accidental interference.

21.9 Explain why expressing pointcut specifications as patterns increases the problems of testing aspect-oriented programs. To answer this, think about how program testing normally involves comparing the expected output to the actual output produced by a program.

If a mistake if made in specifying a poincut pattern specification, the advice will be woven into the system in the wrong place. The tester may assume that the intended

weaving will have taken place and so will expect particular outputs from that; however, if an alternative weaving has occurred, what is actually being tested is a different program.

22 Project Management

22.2 Explain why the best programmers do not always make the best software managers. You may find it helpful to base your answer on the list of management activities in Section 22.1.

Management activities such as proposal writing, project planning and personnel selection require a set of skills including presentation and communication skills, organisational skills and the ability to communicate with other project team members.

Programming skills are distinct from these (indeed, it is a common criticism of programmers that they lack human communication skills) so it does not follow that good programmers can re-orient their abilities to be good managers.

22.4 In addition to the risks shown in Figure 22.1, identify at least six other possible risks that could arise in software projects.

Other possible risks are:

Technology: Communications network saturates before expected transaction limit is reached.

People: Level of skill of available people is lower than expected.

Organizational: Organizational changes mean that the project schedule is accelerated.

Tools: Software tools cannot handle the volume of data available for large systems.

Requirements: New non-functional requirements are introduced that require changes to the system architecture. Estimation: The difficult of the software is underestimated.

22.5 Fixed-price contracts, where the contractor bids a fixed price to complete a system development, may be used to move project risk from client to contractor. If anything goes wrong, the contractor has to pay. Suggest how the use of such contracts may increase the likelihood that product risks will arise.

Fixed price contracts increase the chances of product risks because they remove options from the development process. Because the contract is fixed-price, the contractor is naturally reluctant to increase the effort or time expended on the project as this will reduce their profits on the work. Therefore, if problems arise they will look for ways to reduce the scope of the product or to reduce the costs of product development (e.g. by reducing the effort devoted to testing). Both of these factors can lead to products that are not as expected by the customer.

22.7 What problems do you think might arise in extreme programming teams where many management decisions are devolved to the team members?

While the notion of devolving management decisions to the team is attractive in terms of motivation, there are two types of problem that can arise:

1. Decisions are liable to be primarily influenced by technical considerations rather than business decisions. This is natural given the type of people on an XP team – it is difficult for them to take a business perspective.
2. Because of the focus on rapid iteration, management decisions tend to be short-term and pay insufficient attention to long-term issues. While this is in keeping with the XP philosophy, there is sometimes a need for a more detached, longer-term perspective which can be taken by a manager.

I assume here that management decisions on e.g. the performance of team members are not taken by the team. Given the close knit nature of XP teams, it is difficult for the team to take decisions that censure individual team members.

22.8 Write a case study in the style used here to illustrate the importance of communications in a project team. Assume that some team members work remotely and it is not possible to get the whole team together at short notice.

There are obviously a range of possible answers here. The following is one possibility.

Alice is an experienced project manager working in a company developing assistive technology products. Her project is going well but in the course of the project, the personal circumstances of two team members changes.

1. Carol's oldest child starts at school, which finishes at a different time from the kindergarten of her youngest child. This makes her current working arrangements very difficult so she asks if she can work from home most days, spending only 1 day a week in the office.

2. Ed, the user interface designer, decides to relocate to a different part of the country with his partner who has a new job there. He offers to resign but Alice is keen to maintain the team so she offers him the opportunity to work remotely.

Alice understands the importance of maintaining team communications and conventional techniques such as daily face to face meetings are not impossible. She therefore decides to use a range of technologies to facilitate communications in the team:

(a) The team have a weekly skype conference early on Friday morning. This is simply to catch up on news and work done. Team members who are travelling on business as well as those working remotely are expected to attend. More frequent skype and phone discussions between team members are encouraged.

(b) She sets up a project blog and every team member, every day is expected to write a short report (5/6 lines) about what they have done that day.

(c) A project Twitter account is set up for team members to pass snippets of information to each other.

(d) Every 2 months, all team members have to attend a ½ day face to face meeting where issues are discussed.

There are obviously many different possibilities here. What is most important is that a range of communication possibilities are identified. Communication should not just depend on a single channel.

23 Project planning

23.1 Under what circumstances might a company justifiably charge a much higher price for a software system than the software cost estimate plus a reasonable profit margin?

Circumstances where a high price might be charged:

1. Where a customer expects the developer to take on a considerable amount of project risk.
2. Where the customer has special requirements e.g. for very rapid delivery.
3. When the work is not central to the companies business and so diverts people from other more business-focused activities. The high price is intended to compensate for this.
4. When the customer has no alternative! Think about the ethics of excessive pricing in this situation.

23.2 Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.

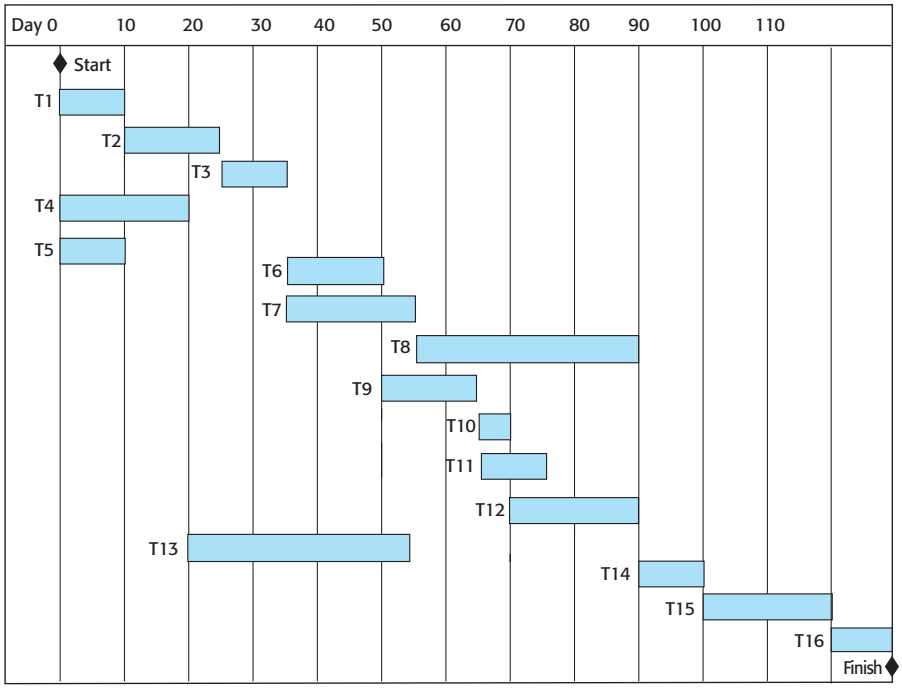
Project planning can only be based on available information. At the beginning of a project, there are many uncertainties in the available information and some information about the project and the product may not be available. As the project develops, more and more information becomes available and uncertainties are resolved. The project plan therefore must be reviewed and updated regularly to reflect this changing information environment.

23.4 Cost estimates are inherently risky, irrespective of the estimation technique used. Suggest four ways in which the risk in a cost estimate can be reduced.

Possible techniques of risk reduction include:

1. Obtain a number of independent estimates using different estimation techniques. If these are widely divergent, generate more costing information iterate until the estimates converge.
2. For those parts of the system which are hard to estimate, develop a prototype to find out what problems are likely to arise.
3. Reuse software to reduce the amount of estimation required and to reduce overall costs.
4. Adopt a design to cost approach to development where the system functionality is adapted to a fixed cost.
5. Partition software requirements into critical, desirable and 'gold plating'. Eliminate 'gold-plating' if necessary.

23.5 Figure 23.14 sets out a number of tasks, their durations and their dependencies. Draw a bar chart showing the project schedule.



23.8 A software manager is in charge of the development of a safety-critical software system, which is designed to control a radiotherapy machine to treat patients suffering from cancer. This system is embedded in the machine and must run on a special-purpose processor with a fixed amount of memory (256 Mbytes). The machine communicates with a patient database system to obtain the details of the patient and, after treatment, automatically records the radiation dose delivered and other treatment details in the database.

The COCOMO method is used to estimate the effort required to develop this system and an estimate of 26 person-months is computed. All cost driver multipliers were set to 1 when making this estimate.

Explain why this estimate should be adjusted to take project, personnel, product and organizational factors into account. Suggest four factors that might have significant effects on the initial COCOMO estimate and propose possible values for these factors. Justify why you have included each factor.

It is adjusted because the time and effort required to complete a project depends on various factors such as the experience of the development team, the development schedule, the support facilities etc.

In this case, they have to go back to the system description and recognise the factors which might add to the difficulty (and hence the cost) of implementing the system. For example:

- Safety-criticality
- Memory limitations on system
- External interfaces with DBMS
- Unusual hardware (‘special-purpose processor)

24 Quality management

24.2 Explain how standards may be used to capture organizational wisdom about effective methods of software development. Suggest four types of knowledge that might be captured in organizational standards.

Standards encapsulate organizational wisdom because they capture good practice that have evolved over the years. Knowledge that might be captured in organizational standards include:

1. Knowledge of specific types of fault that commonly occur in the type of software developed by an organization. This might be encapsulated in a standard review checklist.
2. Knowledge of the types of system model that have proved useful for software maintenance. This can be encapsulated in design documentation standards.
3. Knowledge of tool support that has been useful for a range of projects. This can be encapsulated in a standard for a development environment to be used by all projec ts.
4. Knowledge of the type of information that is useful to include as comments in code. This can be encapsulated in a code commenting standard.

24.4 Design an electronic form that may be used to record review comments and which could be used to electronically mail comments to reviewers.

Any form design that includes the following fields is acceptable:

1. Name of person raising review comment
2. Date comment raised
3. Contact phone number or e-mail address
4. The review comment itself
5. Name of comment assessor
6. Date of comment assessment

7. Action taken from comment (Return for clarification, invalid comment, accepted comment)
8. System change proposed
9. Person responsible for change
10. Date change made
11. Date change checked

24.6 Assume you work for an organization that develops database products for individuals and small businesses. This organization is interested in quantifying its software development. Write a report suggesting appropriate metrics and suggest how these can be collected.

No absolute right or wrong answer to this question – I am looking for evidence that the student has thought about the problem.

The organisation is interested in quantifying its software development so may collect metrics about its products and about its processes. The type of software which is developed is important as the metrics should take into account its characteristics. In this case, the company is developing database products for microcomputers so:

- As they are shrink-wrapped products, they will run on many different system configurations. Configuration dependent problems may occur. It is important that the system should not hang the machine on which it is running.
- As they are database products, it is important that the system does not corrupt the database

Product metrics

1. Product metrics should be used to judge the quality and efficiency of the software.
2. Total number of measured faults detected by testing
3. Total number of faults which resulted in database corruption
4. Total number of system failures which forced a system restart
5. Number of database transactions processed per unit time.
6. Time to read/write large DB records

Process metrics

1. Number of different configurations used for system testing
2. Number of fault reports submitted
3. Average time required to clear fault after it is reported
4. Time required to run system regression tests

24.7 Explain why program inspections are an effective technique for discovering errors in a program. What types of error are unlikely to be discovered through inspections?

Program inspections are effective for the following reasons:

1. They can find several faults in one pass without being concerned about interference between program faults.
2. They bring a number of people with different experience of different types of errors. Hence, the team approach offers greater coverage than any individual can bring.
3. They force the program author to re-examine the program in detail - this often reveals errors or misunderstandings.

The types of errors that inspections are unlikely to find are specification errors or errors that are based on a misunderstanding of the application domain (unless there are domain experts in the team).

24.9 Explain why it is difficult to validate the relationships between internal product attributes, such as cyclomatic complexity and external attributes, such as maintainability.

The basic difficulty arises because the external attributes such as maintainability are not just dependent on a small number of internal product attributes. While the complexity of a system influences its maintainability, other issues such as the use of variable names, the system documentation and, particularly, the skills of the people doing the maintenance have such a large effect on the process that they may mask any maintainability differences arising from different levels of complexity. This does not contradict experiments where a relationship between maintainability and complexity was discovered – however, we don't have enough evidence at the moment to generalize this.

25 Configuration Management

25.3 Describe 6 essential features that should be included in a tool to support change management processes.

The idea here is that the student should look at the change management process and propose tool features to support system activities. Examples include:

1. Change request form definition.
2. Change request validation.
3. Workflow definition – showing the flow of the change request form.
4. Change costing support.
5. Change request traceability – ie maintaining information about the changes made to the software to implement the change request.
6. Change notification and reminders – notification to people involved of work done and reminders to team members who have work to do.

25.5 Imagine a situation where 2 developers are simultaneously modifying 3 different software components. What difficulties might arise when they try to merge the changes that they have made?

There is the usual problem where developers each make changes to the same component and these changes are, in some way, incompatible. However, where several components are being changed at the same time, the problems are exacerbated because there may be dependencies between the components that are affected by the changes. For example, say developer A checks out components X and Y and decides to implement a change by changing Y, which depends on a particular feature of X. Developer A checks X and Y back in with no changes recorded as being made to X. Developer B also is working on X and Y and changes both X and Y. However, the changes made to X mean that the assumptions made

by Developer A no longer hold. However, incompatibility is not detected as there has only been a single change made to component X.

With more than 2 components, the problem becomes even worse because of the chains of dependencies that can be introduced. These can be very difficult or impossible to detect automatically.

25.7 Describe the difficulties that may arise when building a system from its components. What particular problems might occur when a system is built on a host computer for some target machine?

---

1. Have all components been included?
2. Is the right version of all components been included?
3. Are all configuration/data files included?
4. Is the right version of the system building tools used?
5. Are there any problems with full path name references?

25.8 With reference to system building, explain why you may sometimes have to maintain obsolete computers on which large software systems were developed.

Obsolete computers may have to be maintained if the software used to build the system (compiler, linker, etc.) is not available for newer hardware. This situation can arise of the compiler vendor has gone out of business and no-one else is supporting their system. It may not be possible to use a compiler on a newer system as the code produced may be different. It can also arise when programs are developed in a programming language that is no longer used – it may be cheaper to maintain the obsolete hardware to run the compiler than to buy a new compiler for occasional use.

25.10 Describe five factors that should be taken into account by engineers during the process of building a release of a large software system.

---

1. Have all components been included in the build instructions;
2. Has the right version of each component been specified;

3. Are all data files available;
4. Are all data files properly referenced;
5. Are the correct versions of the compiler or other tools available and specified.

26 Process Improvement

26.1 What are the important differences between the agile approach and the process maturity approach to software process improvement?

The fundamental difference in these approaches is that the agile approach is people-centric and the process maturity approach is process centric. In the agile approach, practices are introduced that are geared to supporting communication between people, making it easier for them to make changes to the software and minimizing the time that they need to spend doing things apart from software production (e.g. documentation).

The process maturity approach is based on defined processes that incorporate good practice and in ensuring that these processes are followed by all of the teams in an organization. They assume that by defining process and good practice, all engineers involved in development can perform in a comparable way. That is, they do not focus on the capabilities of the individual engineers but rather on being able to maintain consistent practice even although the team changes.

26.4 Assume that the goal of process improvement in an organization is to increase the number of reusable components that are produced during development. Suggest three questions in the GQM paradigm that this might lead to.

Goal: Increase the number of reusable components

Q1: How many components are currently reused in our development?

Q2: Are there activities in our 'standard' processes that are geared to discovering reusable components? How much time/effort do engineers expend looking for reusable components.

Q3: What percentage of developed components are candidates for reuse?

26.5 Describe three types of software process metric that may be collected as part of a process improvement process. Give one example of each type of metric.

1. *Elapsed time*. How long it takes to do something. Many possible examples e.g. time taken to carry out design review.
2. *Resource utilisation*. The amount of resources used. E.g. the effort required to test a module.
3. *Events that occur*. E.g. The number of defects discovered after a system has been delivered.

26.7 Give two advantages and two disadvantages of the approach to process assessment and improvement that is embodied in the process improvement frameworks such as the CMMI.

Advantages of process improvement frameworks

1. Provides a means of measuring the state of a process and a structured approach to introducing process improvements.
2. Useful as a way of building on the experience of others in process improvement.

Disadvantages of process improvement frameworks

1. Like any measurement system, there is a tendency to introduce improvements to improve the measured rating rather than concentrate on improvements that meet real business goals.
2. Expensive and bureaucratic to operate. Not suitable for agile approaches.

26.8 Under what circumstances would you recommend the use of the staged representation of the CMMI?

The staged version of the CMMI could be used when an organization has experience of and has been assessed using the earlier, staged Capability Maturity Model. The staged CMMI is compatible with this and its use would be a means of continuing improvement according the the CMM approach. You might also use it in organizations that are very immature and, for the earliest activities at least, would like a package of improvements to introduce.

27 Formal Specification

27.2 You have been given the task of 'selling' formal specification techniques to a software development organization. Outline how you would go about explaining the advantages of formal specifications to sceptical, practising software engineers.

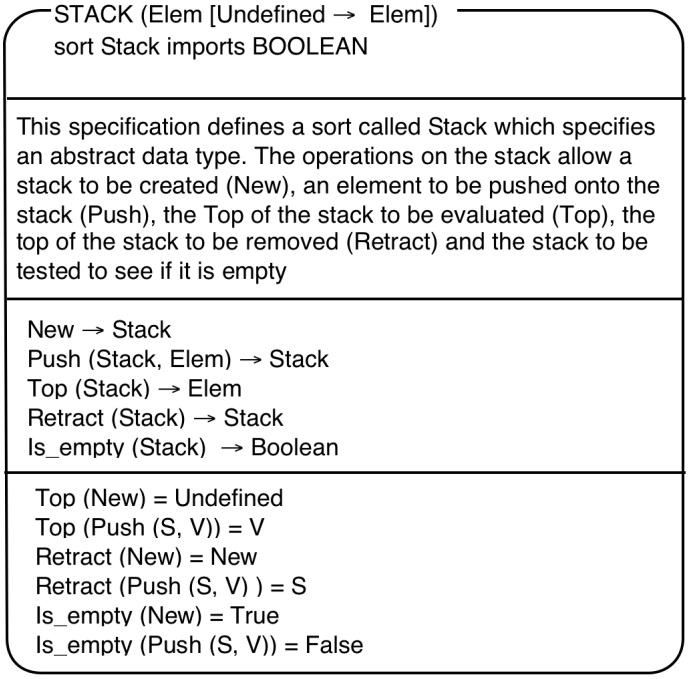
To explain the advantages of formal specification to practising engineers, it is important to focus on what it brings to the practice of software development rather than on more abstract advantages such as the ability to mathematically analyse the specification. Advantages that might be stressed are:

1. 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.
2. 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.
3. 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.

27.4 An abstract data type representing a stack has the following operations associated with it:

New:	Bring a stack into existence
Push:	Add an element to the top of the stack
Top:	Evaluate the element on top of the stack
Retract:	Remove the top element from the stack and return the modified stack
Empty:	True if there are no elements on the stack

Define this abstract data type using an algebraic specification.



27.6 Bank teller machines rely on using information on the user's card giving the bank identifier, the account number and the user's personal identifier. They also derive account information from a central database and update that database on completion of a transaction. Using your knowledge of ATM operation, write Z schemas defining the state of the system, card validation (where the user's identifier is checked) and cash withdrawal.

The Z schemas are shown below. There are many different possibilities here depending on how much information is maintained. This is one of the simplest which assumes that customers may not withdraw money if there is an insufficient cleared balance in their account (a cleared balance is where all cheques paid into cleared balance is where all cheques paid in to the account have been cleared for payment).

BankAccount

CustomerNumber: ℕ
AccountNumber: ℕ
DateOpened: Date
DateClosed: Date
CustomerPIN: ℕ
Balance: Money
ClearedBalance:Money
AvailabletoWithdraw: Money

DateOpened < DateClosed
ClearedBalance ≤ Balance
AvailabletoWithdraw ≤ ClearedBalance

Validate

Ξ BankAccount
Account? ℕ
PIN?: ℕ
TransactionOK!: Boolean

Account? = AccountNumber
PIN? = CustomerPIN
TransactionOK!'

Withdraw

ΔBankAccount

Amount?: Money
Amount? <= ClearedBalance
Amount? <= AvailabletoWithdraw
Balance' = Balance - Amount?

27.7 Modify the insulin pump schema, shown in Figure 27.10, to add a further safety condition that the ManualDeliveryButton? can only have a non-zero value if the pump switch is in the manual position.

To specify that the manual delivery button can only have a non-zero value if the switch is in the manual position, you should add the following invariant to the state schema.

$$switch? \neq manual \Rightarrow ManualDeliveryButton = 0$$

27.8 Write a Z schema called SELF_TEST that tests the hardware components of the insulin pump and sets the value of the state variable HardwareTest?. Then modify the RUN schema to check that the hardware is operating successfully before any insulin is delivered. If not, the dose delivered should be zero and an error should be indicated on the insulin pump display.

SELF_TEST

Δ INSULIN_PUMP_STATE

(HardwareTest? = OK $\wedge$ Needle? = present $\wedge$ InsulinReservoir? = present $\Rightarrow$
status' = running $\wedge$ alarm! = off $\wedge$ display1!= "") $\vee$
(
status' = error
alarm! = on
Needle? = notpresent $\Rightarrow$ display1! = display1! $\cup$ "No needle unit" $\vee$
(InsulinReservoir? = notpresent $\vee$ insulin_available < max_single_dose)
$\Rightarrow$ display1! = display1! $\cup$ "No insulin" $\vee$
HardwareTest? = batterylow $\Rightarrow$ display1! = display1! $\cup$ "Battery low" $\vee$
HardwareTest? = pumpfail $\Rightarrow$ display1! = display1! $\cup$ "Pump failure" $\vee$
HardwareTest? = sensorfail $\Rightarrow$ display1! = display1! $\cup$ "Sensor failure" $\vee$
HardwareTest? = deliveryfail $\Rightarrow$ display1! = display1! $\cup$ "Needle failure"
)

The RUN schema should be modified to check that HardwareTest? is true before continuing operation.

28 Application Architectures

28.2 Using the four basic application types introduced in this chapter, classify the following systems and explain your classification:

A point-of-sale system in a supermarket

A system that sends out reminders that magazine subscriptions are due to be paid

A photo album system that provides some facilities for restoring old photographs

A system that reads web pages to visually disabled users

An interactive game in which characters move around, cross obstacles and collect treasure

An inventory control system that keeps track of what items are in stock and automatically generates orders for new stock then the level falls below a certain value.

1. Point of sale system: Transaction processing
2. Subscription reminder system: Batch processing
3. Photo album system: Event processing
4. Web page reader: Language processing
5. Interactive game: Event-processing
6. Inventory control system: Transaction processing

28.4 Explain why transaction management is necessary in systems where user inputs can result in database changes.

A transaction mechanism is necessary to ensure database consistency. As a transaction is an atomic operation, all of the changes in the transaction are not committed until it is complete. If these changes were made in sequence without a transaction mechanism, a system failure could mean that the update of the database was unfinished and the data left in an inconsistent state.

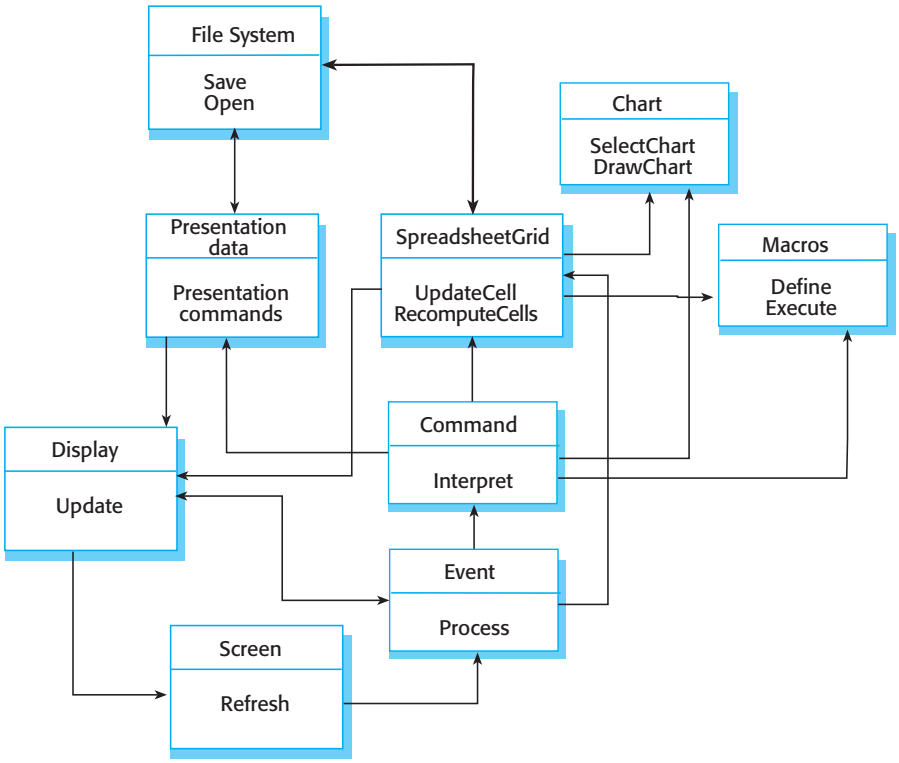
28.6 In an editing system, all user interface events can be translated into implicit or explicit commands. Explain why, in Figure 28.10, the Event object therefore communicates directly with the editor data structure as well as the Command object.

The Event object communicates directly with the editor data structure to allow more efficient operation. Some commands that are implicit, such as inserting a character by pressing a key on a keyboard, require very fast response and, rather than lookup the command in a separate object, the event processing object interprets these directly and makes changes to the data structure.

28.7 Modify Figure 28.10 to show the generic architecture of a spreadsheet system. Base your design on the features of any spreadsheet system that you have used

---

Many different possibilities here. The key is to identify the principal spreadsheet components. A possible example of an architecture is shown below.



28.9 What is the function of the syntax tree component in a language processing system?

The syntax tree represents the structure of the ‘sentences’ in the language that is being processed. A parser analyzes text strings written in that language and creates the syntax tree, which shows the different parts of the sentence and its type e.g. in a programming language, it would tag tree entries as keyword, identifier, etc.

29 Interaction Design

29.1 I suggested in Section 29.1 that the objects manipulated by users should be drawn from their domain rather than from a computer domain. Suggest appropriate objects for the following users and systems.

- A warehouse assistant using an automated parts catalogue
- An airline pilot using an aircraft safety monitoring system
- A manager manipulating a financial database
- A policeman using a patrol car control system

(a) Warehouse assistant using a parts catalogue. Objects are parts, part numbers, stock, etc.

(b) Airline pilot using a safety monitoring system. Objects are aircraft sub-systems e.g. engine sub-system and their performance parameters.

(c) Manager manipulating a financial database. Objects are stocks, bonds, investments, interest rates, etc.

(d) Policeman using a patrol car system. Objects are other cars, locations, types of incident, etc.

29.2 Suggest situations where it is unwise or impossible to provide a consistent user interface.

A consistent user interface may be impossible to produce for complex systems with a large number of interface options. In such systems, there is a wide imbalance between the extent of usage of different commands so for frequently used commands, it is desirable to have short cuts. Unless all commands have short cuts, then consistency is impossible.

It may also be the case in complex systems that the entities manipulated are of quite different types and it is inappropriate to have consistent operations on each of these types.

An example of such a system is an operating system interface. Even MacOS which has attempted to be as consistent as possible has inconsistent operations that are liked by users. For example, to delete a file it is dragged to the trash but dragging a disk image to the trash does not delete it but unmounts that disk.

29.3 What factors have to be taken into account in the design of a menu-based interface for 'walk-up' systems such as bank ATMs? Write a critical commentary on the interface of an ATM that you use.

Factors to be taken into account when designing 'walk up and use' systems are:

- System users may be infirm, or disabled so will not be able to respond quickly to requests.
- Users may not be able to speak the native language of the country where the machine is installed.
- System users may be completely unfamiliar with technology and may make almost any kind of error in using the machine. The interface must minimise the number of possible errors and must be resilient to any possible error.
- Some system users are likely to be intimidated by many options. On the other hand, as users gain familiarity with the system, they may expect to use it for a wider range of banking services.
- Different people may understand the meaning of icons in different ways.
- If the system has navigation options, users are almost certain to become lost.
- Most users will want to use the system for very simple functions (e.g. withdraw cash from an ATM) and will want to do this as quickly as possible.

There are many different ATM interfaces so each must be considered separately. Problems that I have found are:

When is it possible to cancel a transaction?

What happens when I do so?

What will I have to re-input if I restart the transaction?

There is not usually any way of saying give me the maximum amount of money I may withdraw today.

Some machines only support single transactions - there is no way of saying I will be making several transactions and the same validation process is applicable to all of them.

29.5 Discuss the advantages of graphical information display and suggest four applications where it would be more appropriate to use graphical rather than digital displays of numeric information.

Advantages are 'at a glance' magnitude indication and relative magnitude indication. Any applications where these are important might be mentioned:

- Temperature control
- Speed indicators
- Weather statistics
- Relative comparisons of cars, etc.

29.9 Design a questionnaire to gather information about the user interface of some tool (such as a word processor) with which you are familiar. If possible, distribute this questionnaire to a number of users and try to evaluate the results. What do these tell you about the user interface design?

I assume that the object of this exercise is to discover problems rather than objectively compare different systems. The questionnaire should include questions which cover:

- The experience of the word processor user.
- The types of documents he/she produces. Are these all text, include text and graphics, use mathematical symbols, etc.
- Whether the user is familiar with and uses other word processors. What they like/dislike about them.
- The principal problems perceived by the word processor user.
- What facilities they used most. What they felt about the way in which the menus (if any) were arranged.
- Where they felt that they made mistakes when using the system.
- Whether they used the mouse or the keyboard to issue commands. If they used keyboard shortcuts, which did they use? Did they have problems with these?