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.

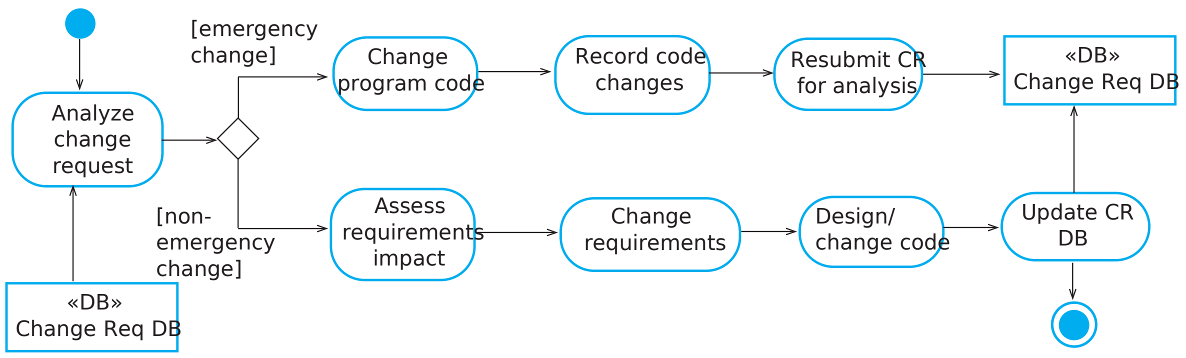
Incorrect PIN. As in Withdraw cash.

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:

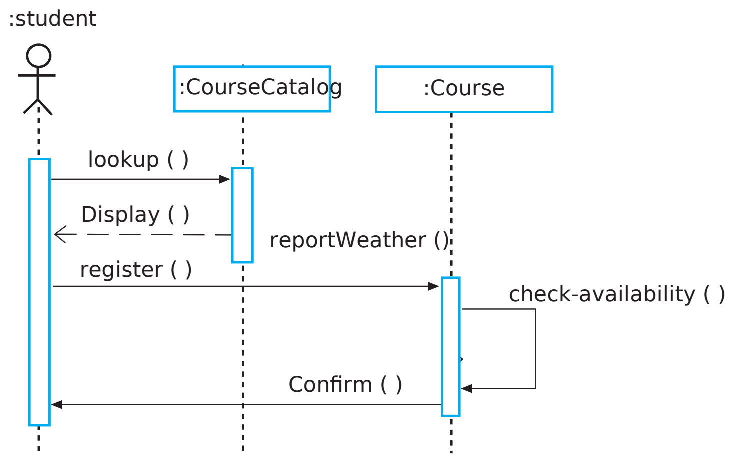
1. To understand and document the architecture and operation of the existing system.
2. To act as the focus of discussion about possible changes to that system.
3. To inform the re-implementation of the system.

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.

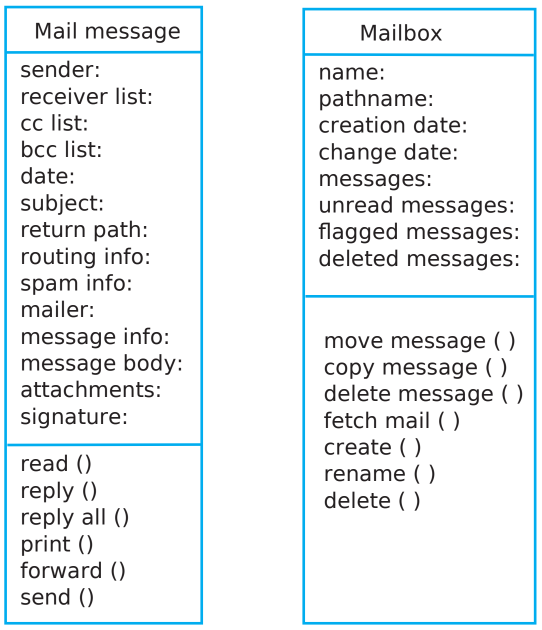
This is true, in general, for models of new systems unless a model-based approach to development is taking place in which case a complete model is required. The other circumstances where you may need a complete model is when there is a contractual requirement for such a model to be produced as part of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

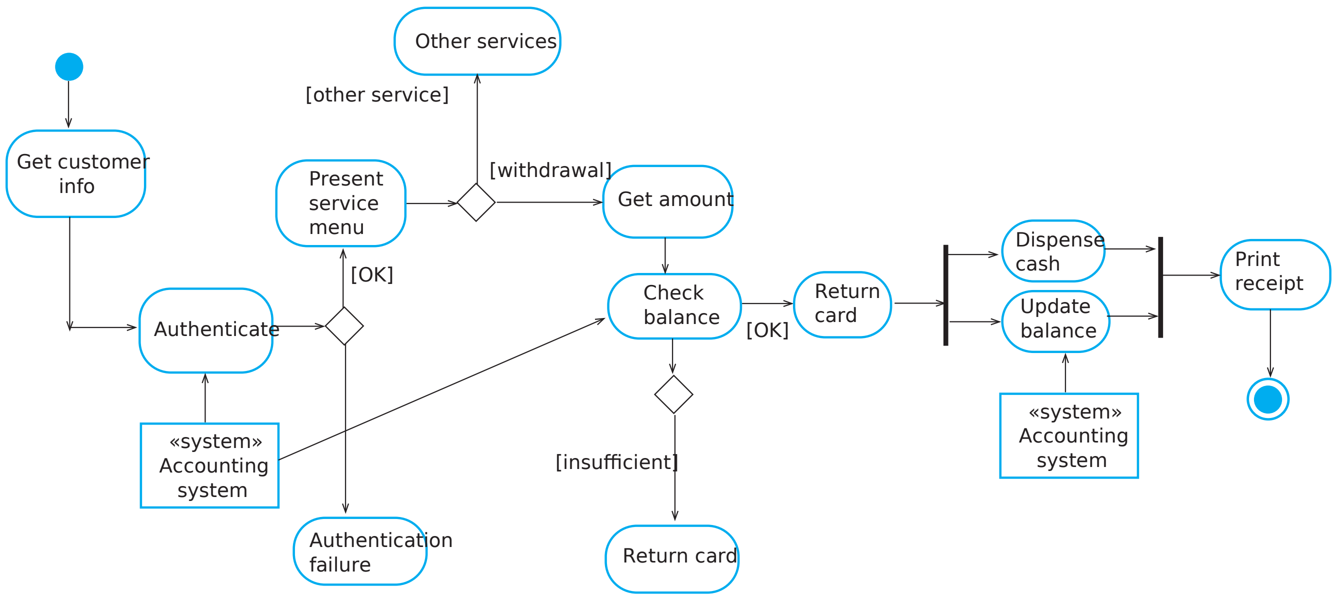


5.6 Look carefully at how messages and mailboxes are represented in the email system that you use. Model the object classes that might be used in the system implementation to represent a mailbox and an e-mail message.



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.

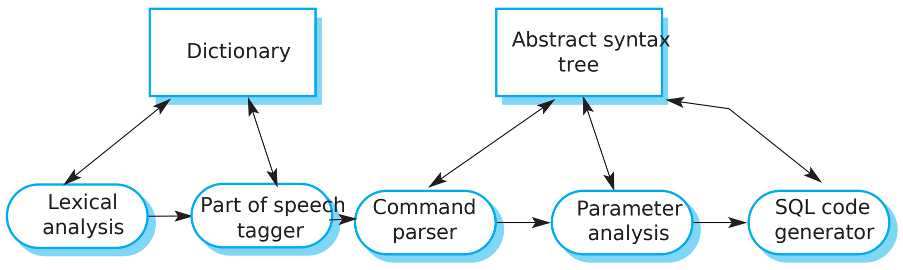
Therefore, there is a fundamental architectural conflict between availability (replication, several copies) and security (specialization, minimal copies). The system architect has to find the best compromise between these fundamentally opposing requirements.

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)

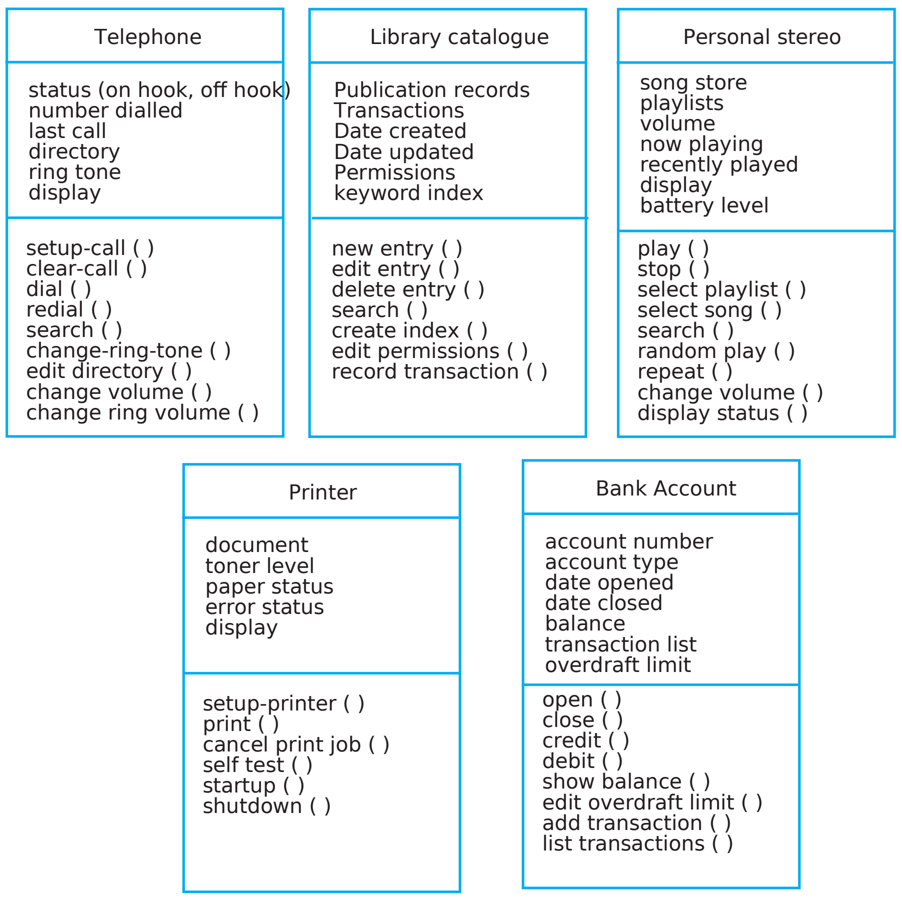
Input checking (Javascript), browser

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.

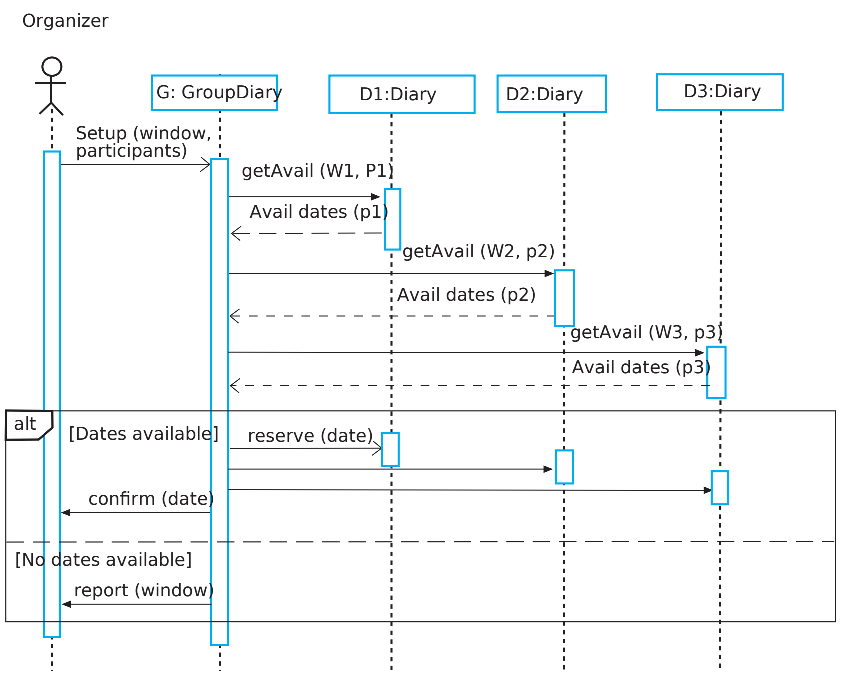


7.3 Using the UML graphical notation for object classes, design the following object classes, identifying attributes and operations. Use your own experience to decide on the attributes and operations that should be associated with these objects.

- 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.