Chapter 1. Introduction

Table of Contents

Objectives 1
Introduction 1
Information systems and software 2
    The importance of software engineering 2
    Systems 3
System boundaries 3
Categories of information system and software programs 4
    Information systems 4
Legacy systems 7
The benefits of software systems 7
    Tactical benefits 7
    Strategic benefits 8
The reasons for change 8
Software myths 8
    Management myths 9
    Customer / end-user myths 9
    Programmer myths 9
Review 9
    Questions 9
    Answers 12

Objectives

At the end of this chapter you will have acquired an introductory understanding of what software and software engineering are, as well as an understanding of some of the common myths surrounding the practice of software engineering.

Introduction

It could be argued that information systems are vital components of any civilisation. The human desire to record information goes back thousands of years to when humans first started painting on stones. However, it is the practice of recording and displaying information in a systematic manner that warrants the use of the term “information system”. Such practices can be easily found in great civilisations, such as those of ancient Egypt, Greece and Rome.

The development and use of information technology (IT) has led to the birth of new generations of information systems. Computer-based information systems (software systems) have dramatically influenced our behaviour and the way in which we conduct every day activity. It is not surprising that the standing of any society in the world is now strongly linked to the level of penetration that software systems have in that society.

In this module we shall use term *software systems* to refer to information systems that contain or might contain software components. Although there are many information systems that do not involve computers, such as a card filing system of a small library or a manager's list of contacts, almost all modern information systems either use computers, or could use computers, to perform some of their functions (such as cash registers for point-of-sales processing, and stock control systems for small businesses).

Further, while we will occasionally mention information systems, this module is ultimately interested in the *software* components of an information system, and how to *engineer* software that can be

reliably used by other people. Software is integral to computerised information systems. Without the underlying software, the system will not be able to do what its users intend, and if the software functions incorrectly, so will the information system.

Information systems and software

Software systems are made up of the following components:

- **Users** — the people who add information to the system, request information from the system, and perform some of the information processing functions.
- **Procedures** – the tasks performed by the human components of the information system.
- **Information** – meaningful data that the system stores and processes.
- **Documents** – manuals on how to use the system, sometimes even files of data which should not or could not be stored electronically.
- **Hardware** – not only the computers in the system but also any networks linking the computers, the input devices and output devices.
- **Software** – computer applications performing some of the system functions to record, process, and regulate access to some of the information worked with by the information system

Importantly, we need to consider what software is. Software is typically defined to be *instructions* that provide desired features, functions, and performance. They contain *data structures* which allow the software program to manipulate the information contained in an information system. Importantly, software also includes documentation describing how the software performs the actions that it does, and how the software may be used. Notice that some of the documentation is for the software's users, while other portions of the documentation are for its developers and maintainers.

There are some important properties of software that you should consider when thinking of the discipline of software engineering.

- First, software is *engineered* rather than manufactured. Once the software has been developed, there remains no significant “manufacturing” process that could possibly lower the software's quality (i.e., introduce software errors, cause the software to deviate from what the customer requested, and so on). The cost of developing software lies almost completely in the engineering of the software, and not in the “manufacturing” of a “product” that customers can be hold in their hands.
- Software does not wear out with use, as hardware might. However, this does not mean that software does not *degrade* over time. A software program is continuously changed over its lifetime. If these changes occur too frequently, the bugs introduced by each change may slowly degrade the performance and utility of the software as a whole. Also, when software degrades in quality, there are no “spare parts” which can be used as replacements.
- Unlike hardware, most software remains custom built, rather than built using “off the shelf” components.

The importance of software engineering

Over the last few decades, software systems and the software that run them have become an important component to many aspects of our society, from commerce to medicine, engineering, the sciences and entertainment.

Importantly, the infrastructure of all developed countries rely heavily on software systems. Because of this, it is important that the software we use and rely on are of a high quality and fulfil our requirements of them. Gaining this high quality does not happen randomly or by accident — rather we need to engineer that quality into the software that we use.

When software fails, people may be bankrupted and even killed (consider safety critical systems which run planes, medical equipment, and so on). Because so much depends on software, software has become important to business and the economy. This means that the software engineer is always part of a larger environment, consisting of customers, other software engineers, managers, and so on. It is important that the engineer be able to interact appropriately with all of these individuals, and this, too, is part of software engineering.

Systems

Because information systems are what we build using software, it is important to consider exactly what it is that we are building: what is a system?

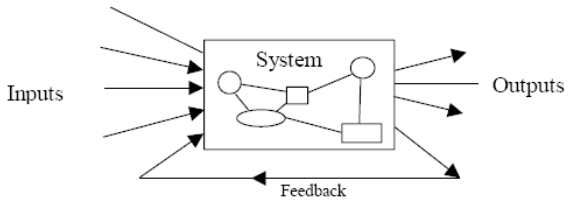
The word system is used regularly to refer to a coherent group of elements (or components) that together aim to achieve a certain objective, or to have a specific purpose. A more rigorous definition would describe a system as a group or combination of interrelated, interdependent or interacting elements forming a collective entity (a whole) with a control mechanism that helps the system achieve its goal.

Systems live in environments that are relevant to their existence. They receive inputs from the environment, and produce outputs for the environment. Systems tend to have a boundary that is defined in relation to the external environments in which they reside. The survival of a system relies on a control mechanism that regulates the processes of receiving inputs and producing outputs using feedback.

Figure 1.1, “A diagrammatic representation of a system” illustrates many of the important features of a system:

- There may be many different inputs
- There may be many different outputs
- The system is composed of interconnected components — the components may be of different kinds.
- Some of the outputs of the system are fed back into the system, so that the system can control itself and make corrective changes when unexpected or undesirable outputs occur

Figure 1.1. A diagrammatic representation of a system



These components play a role in the software that we create: the inputs to the system will become inputs to the software. The outputs of the software will eventually be the system's outputs.

System boundaries

The smaller the system, the sharper its boundary. Large systems may have multiple boundaries as they interface with multiple systems. The boundary also depends on the point from which it is viewed in relation to other systems with which it interfaces.

An example of the complexity and difficulties of defining system boundaries is the ATM (automated teller machines). Consider the following candidates for system boundary:

- The physical machine itself
- The customer and the machine
- The machine and the bank's local database of customer accounts
- The machine and the bank's national network and central database of transactions and account balances
- The machine and the staff that loan and run regular software checks of the machine

Which of the above might be the “right” boundary for the analysis of the system in which the ATM is central? Different analysts may choose different boundaries, and these boundaries may change during talks with the customers wanting the system built, and with the ultimate end-users of the system. An important skill which systems analysts and software engineers must develop is to determine which boundaries are important to consider.

The software which we as software engineers develop are only a component of larger information systems in an organisation or group. This module concerns itself with the development and engineering of software, but to effectively engineer such software we must be able to understand the information system as a whole, and relate the software to it.

For effective and successful software engineering, software engineers have the difficult job of learning how the information system users perform their daily jobs, as well the tasks they attempt to achieve. If the software does not support the users in their tasks, then the software will not be used, and the software development has failed.

American Federal Aviation Authority example

During the development of the American Federal Aviation Authority (FAA) automated flight control system software engineers tried to produce electronic counterparts for the flight strips — little bits of paper giving details of a particular flight which would be moved between operators and used for quick reference. This task proved to be impossible and the final system not only still has these bits of paper but the consoles have special slots to put them in.

This example illustrates an important point; that is any artificial system has a boundary which defines what is part of it and what is not. Sometimes this boundary is clear cut, say when the computer screen and keyboard are the boundary for a system, other times it is not, say when people and bits of paper form an integral part of the system. In the case of the FAA system some of the functions that were considered part of the system are solely performed by humans, but still within the boundary of the information system and the software being developed.

Categories of information system and software programs

Information systems

There are many kinds of software systems. Clearly, software used by each kind of system will differ. It is useful (and common practice) to break them down into categories such as the following:

- Data-processing systems
- Real-time systems

- Decision support systems
- Knowledge-based systems

Different people and authors may break down the categories further, or provide other categories. The list above is informal but useful, since (as we shall describe below) each category of system can be distinguished from other kinds of information system.

Data-processing systems

Such a system generally has some large database of information and the purpose of the system is to provide quick, easy access and processing of data.

Depending on the degree to which data is processed and analysed, systems may be classified as either transaction processing systems or management information systems. A system which basically manages the data necessary to perform the daily business is a transaction support system. A system which summarises data in a form useful for the management of a business is a management information system.

Real-time systems

The environment outside the boundaries of the system is not under the system's control. Therefore a system will need to be able to respond to data whenever it arrives — real time systems must respond quickly to changes in the inputs from the environment. Typical response times would be of the order of a few milliseconds or even microseconds. To achieve such a fast response the system needs to prioritise its tasks, often dividing them into several processes that may interrupt each other. However, as soon as there are several tasks, they must be able to communicate properly with one another and not interfere with each other. Because of environmental interactions, real-time systems have to be robust to accidents, errors and failures in the external parts of the system. That is, they must respond in a safe and controlled manner in (almost) all conceivable circumstances.

A typical example is an auto-pilot that must adjust the engines and ailerons of an aeroplane to keep it on course. An automated manufacturing system — such as is used in car factories — has to detect when specific parts have arrived in designated zones of the factory, and then make sure that they are correctly assembled.

Decision support systems

Although a data-processing system may help to identify a problem in the business, it does not suggest any solution to the problem. In a decision support system, given a problem, the system attempts to fit the data on the problem into some model and thereby suggest a solution to the problem. A decision support system may have different models, say operational research or statistical models. The manager chooses the model appropriate to the problem.

Ultimately, of course, any decision is made by the manager and not the system. The manager may know or guess something which cannot be represented in the system's models, and this may affect their decision. The system is simply there to clarify the problem and suggest solutions as far as it is able.

Knowledge-based systems

In some situations it is not a large amount of data that needs to be handled, but a large amount of *knowledge*. *Knowledge* is a combination of rules, laws, constraints and previous experience. A knowledge-based system encapsulates a knowledge-base, like a database but filled with knowledge, and enables the user, possibly unskilled in the problem area, to use the knowledge-base to solve problems.

In business systems, the knowledge-based system often takes the form of what is called an “expert system”. Expert systems embody the knowledge of a particular class of experts, such as medical doctors, and the system (ideally) provides the same answers as an expert of that class would. In the

case of a medical expert system, this could be a diagnosis of an illness, and perhaps a recommendation for a specific choice of treatment or for further tests.

Software

Software programs can be categorised in the following manner:

- System software
- Application software
- Engineering / scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software

System software

System software is software written to service and support other programs. Compilers, editors, debuggers, operating systems, hardware drivers, are examples of such software. Most system software deals heavily with computer hardware, multiple users, concurrent operations and process scheduling, resource sharing and virtualisation, complex data structures and multiple external interfaces.

Application software

Application software are programs designed to solve a specific business need. Most software operating with business and technical data are application software, as are most real-time systems.

Engineering / scientific software

This software supports the use and production of scientific and engineering data sets. They are used in almost all engineering and scientific disciplines, from simulating water flow, lighting conditions, and aerodynamics, to examining the large scale structure of the universe. Engineering software is also used for design purposes (such software is called CAD software, for *Computer Aided Design*) and for automating the manufacturing of goods (CAM: *Computer Aided Manufacturing*).

Embedded software

This is software that resides directly within some hardware component in order to provide control over the component or other vital features required for it to function. Embedded software is widespread, and can be found in everything from phones and microwave ovens to cars, aeroplanes and medical equipment.

Web applications

A *web application* is an application accessed via a web browser over a network. Web applications offer a variety of functions, and some application software are now implemented as web applications, such as Google Docs.

Artificial intelligence software

Artificial intelligence software (AI) has been defined as "the science and engineering of making intelligent machines" (John McCarthy). Application domains that make use of AI software include robotics, expert systems, pattern recognition, theorem proving and game playing.

Legacy systems

While a lot of software that is in use is fairly current, state of the art software, companies often rely on software that has been in use for years, or even decades. This older software is called *legacy software*.

Legacy software is often, by today's standards, of poor quality: inextensible, convoluted, badly documented, and written in languages which are generally no longer used.

Worse, legacy software have often been continuously changed over decades to meet ever changing business and system requirements, contributing to the software's degradation.

Unfortunately, such legacy software is long lived precisely because it supports critical business systems. This makes it important that legacy systems be reengineered to remain usable in the future. One can think of this as a slow evolution of the legacy software. This evolution, these *changes*, are often done for one of the following reasons:

- The business is making use of a new computing environment or technology, and the software must be updated to support this.
- The software must meet new business requirements.
- As newer information systems, databases, and so on, become available, the software must be updated to be interoperable with them.
- Software must be modified to operate in a networked environment.

Changes to software are inevitable and are not unique to legacy systems. Software engineering must take this into account. This process of change is often referred to as *software maintenance*.

The benefits of software systems

The benefits of introducing new software are not always easy to identify. The person (or people) who are considering the introduction or extension of a software system (we shall refer to such a person as the “customer”) may be very enthusiastic about the possible benefits of the new software, such as providing “better service” and “more control”. However, given it is unlikely that the existing (manual or partly software-based) system is ruining a business, the advantages of the new software may be very difficult to quantify. In fact, unless the software is well-designed and properly thought out, it may bring no overall benefits at all.

The potential for significant benefits and the differences information systems have made in other organisations are an important motivation for many organisations to investigate the development of new or extended software systems in their businesses.

Organisations, their information systems, and the software they employ, are complex, and therefore the costs and benefits of information system and software development are hard to estimate and measure. Such systems are hard to analyse, design and implement. For this reason there has been much study into the relationship of information systems and software within organisations, as well as into information system and software development.

Broadly speaking, possible benefits can be divided into two categories:

- tactical benefits
- strategic benefits

Tactical benefits

Tactical benefits are ones which improve the day-to-day running of an organisation or group in measurable terms.

Customers almost always anticipate or require that new software and systems deliver cost benefits. In some cases software replaces an existing automated system, but the new software has cheaper hardware and lower maintenance overheads. For instance, where a single, centralised computer system is replaced by a network of smaller PCs. Or the software system replaces a manual system and so savings may be made on paper, paper storage space like filing cabinets, and office space. There may be an improvement in communications resulting in fewer telephone calls and faxes. In both cases there may well be a saving on the number of staff required to support the old system.

Another frequently cited benefits of software systems are their speed and accuracy. Information can be retrieved more quickly and with greater confidence in its accuracy. This can improve the productivity of employees. It may also improve the movement of goods and the supply of goods to customers. This could result in an organisation being able to handle more transactions and expand its business.

Strategic benefits

Strategic benefits are about improving the nature or abilities of a group or organisation. With a new software system, a business may be able to offer their customers new services. Or by examining the information held, could new customers or products could be identified. More speculatively, enhanced functionality might allow managers to quickly identify trends in sales or spending. This could lead to a competitive advantage in the market place.

With the introduction of expert systems, knowledge previously confined to a handful of individuals can be distributed and made available throughout a company. This could improve the functioning and performance of a company in many ways.

However, in many ways strategic benefits are even more difficult to quantify than tactical benefits, and without careful planning and design they will simply not appear.

The reasons for change

People do not usually embrace change merely for the sake of it. There is always risk involved in changing existing practices — to paraphrase an English saying “If it is not broken, do not fix it!”.

The simplest reason might be that people like the look or sound of new technology and a company wants to be seen at the “cutting edge” of technology. Installing a new hi-tech software system shows the company off to good advantage. Companies can use this opportunity to reap some of the other benefits that a new system might bring.

Or a company may realise that new technology brings new processing facilities and methods within their reach.

More normally, companies develop new software systems as a result of external forces. Costs may be rising and new software could reduce overheads. There may be competitive pressures and unless changes are made to the business processes, a company will lose its competitive edge.

More mundane reasons for change includes new legislation requiring an update to business processes. A shortage of appropriate staff can also drive the need for new software.

Whatever the reason, once a company decides to install new software, it makes sense to maximise the return on the investment. Not only should the system address the problem at hand, but it should also bring as many other benefits as possible. It is this escalation of requirements which can transform a hi-tech business solution into a software catastrophe. We discuss this next.

Software myths

All people who come into contact with software may suffer from various myths associated with developing and using software. Here are a few common ones.

Management myths

Our company has books full of standards, procedures, protocol, and so on, related to programming software. This provides everything that our programmers and managers need to know. While company standards may exist, one must ask if the standards are complete, reflect modern software practice, and are — importantly — actually used.

If we fall behind schedule in developing software, we can just put more people on it. If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

Customer / end-user myths

A vague collection of software objectives is all that is required to begin programming. Further details can be added later. If the goals / objectives for a piece of software are vague enough to become ambiguous, then the software will almost certainly not do what the customer requires.

Changing requirements can be easily taken care of because software is so flexible. This is not true: the longer that development on the software has proceeded for, the more work is required to incorporate any changes to the software requirements.

Programmer myths

Once the software is written, and works, our job is done. A large portion of software engineering occurs after the customer has the software, since bugs will be discovered, missing requirements uncovered, and so on.

The only deliverable for a project is the working program. At the very least there should also be documentation, which provides support to both the software maintainers, and to the end-users.

Software engineering will make us create a lot of unnecessary documentation, and will slow us down. Software engineering is not about producing documents. Software engineering increases the quality of the software. Better quality reduces work load and speeds up software delivery times.

Review

Questions

Review Question 1

Read the following scenario about an online reservation system, then carry out the tasks below.

Global-Travel, an on-line reservation system

Global-Travel is an airline that sells all its tickets through an on-line system on the Internet. Prospective travellers register their details by filling in the form available on the company's web site. Once a registration has been confirmed as valid, the travellers can proceed by selecting the destination to which they wish to travel, the date of travel, and the type of ticket they wish to purchase. Only those of destinations to which Global-Travel flies appear on the list of destinations for customers to choose from. Once a destination has been selected, the date box displays only those days of the week on which Global-Travel flies to the selected destination.

Once all ticket details are entered, the customer submits the details to the systems by clicking on the appropriate button. The system responds by either accepting the reservation or rejecting it. If a

reservation is rejected, the system displays the reason (e.g., no available places on the specified date or no places available on the specified date for the selected type of ticket). The customer may then choose to change the date of travel, upgrade his/her ticket or abandon the reservation.

If a reservation is accepted, the system prompts the traveller to proceed to the payment section to purchase the ticket or reserve it for 24 hours. Global-Travel web site has a secure server and accepts payments by most credit and debit cards. Once the traveller completes the payment section, the system displays the details of the purchased ticket and requests the traveller to check the details and confirm or abandon purchase by clicking on the appropriate button.

The marketing department at Global-Travel has another information system that is linked to the online site and monitors sales of tickets on all flights. It uses this information to frequently display various promotions and special offers on the site. The accounting department system is also linked to the online site and receives all payment information.

Review question tasks - Answer the following questions based on this scenario:

- Describe the context of Global-Travel online sales system.
- Describe the collection of information that is relevant to that context.
- Specify who has access to this information.

A discussion of this question can be found at the end of the chapter.

Review Question 2

Do the following, based on the Global-Travel scenario:

- Describe the inputs and outputs of the system.
- From the point of view of a prospective traveller, describe the feedback loops in the system and how this affects the input and the output of the system.

A discussion of this question can be found at the end of the chapter.

Review Question 3

Read the following scenario about an automatic train system, then carry out the tasks below.

ARTC, an on-line reservation system

A railway authority have been asked to fit all its trains with an automatic signalling control system within five years. The main objectives of the project was to increase the safety of rail travel by:

- reducing the number of trains that passes through red signals or possibly preventing trains from passing any red signal.
- reducing the number of train accidents that result from head on collision or moving trains colliding with stationary ones.

The “Automatic Railway-Train Control System” (ARTC) system should alert the train driver when the train is approaching a red signal. The alert of a red signal must take the form of audible sound and visible red light in the driver's cabin. The alert must start at a specified distance from the red signal (called the *red signal alert point*), and at the very least the light must continue to be on until the signal switches back to green. The alert sound must also stay on until the driver switches it off. The switching off of the light is used as an acknowledgement that the driver has heard the alert. The level noise from the alert sound increases until it gets switched off.

The ARTC system checks and records the train speed each time it passes a red signal alert point (RSAP). If the train speed was higher than it should be at this point, the system should alert the driver to slow the train down.

The action taken by the driver after his/her train reaches a RSAP depends on the system that would be installed. Currently there are three different systems available.

The first, called *Fully Automatic Railway-Train Control System* (FARTC), takes over the breaking and the driving systems of the train after it passes a RSAP and gradually brings the train to a halt before the red signal.

The second, called *Semi-Automatic Railway-Train Control System* (SARTC), does the same as FARTC but has to be triggered by the driver and can be over-ridden also by the driver.

The third, called *Automatic Alert Railway-Train Control System* (AARTC), simply alerts the driver of the red signal and expects him/her to stop the train before the signal.

The choice of system will depend on the budget and the time-scale allowed.

Tasks

Answer the following questions based on this scenario:

- Describe the possible boundary of each of the proposed system.
- Discuss how they differ and why.

A discussion of this question can be found at the end of the chapter.

Review Question 4

Do the following, based on the ARTC railway scenario:

Describe some of the control (feedback) mechanisms that were mentioned in the case study and their function. State the inputs and output of these mechanisms and how the feedback process could affect them.

A discussion of this question can be found at the end of the chapter.

Review Question 5

Describe what is meant by an information system?

A discussion of this question can be found at the end of the chapter.

Review Question 6

Describe one example from your own experience of each of the following types of systems:

- Real-time systems
- Data-processing systems
- Decision-support systems
- Expert systems

A discussion of this question can be found at the end of the chapter.

Review Question 7

The United Kingdom has been debating whether to join the single European Currency for a while now. What might be some of the consequences to organisations whose business is the export of hand-made furniture to EU countries and the USA? How would these consequences influence decisions about software and information system development?

Legacy systems are old software which continues to be used today, even though continual, ad-hoc updating of the software has introduced many bugs and inconsistencies. What are the potential pitfalls to keep in mind when updating software in order to avoid producing (in the long-term) inefficient software that we would consider to be legacy software?

A discussion of this question can be found at the end of the chapter.

Review Question 8

A seminal work on software engineering is Fred P. Brooks's paper, “No Silver Bullet — Essence and Accident in Software Engineering”.

An important concept that he tells us is that:

> Fashioning complex conceptual constructs is the *essence*; *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.
>
> —Fred Brooks

In the above quote, Brooks is arguing that our ability to manage the direct tasks of the programming and production of the software itself is no longer a major problem: tools and techniques exist to help us handle this. *The* problem of software engineering is in managing the conceptual aspects of it: the software's specification and design, and the testing of this specification and design.

We cannot expect tools to automatically manage or drastically lesson the conceptual problems related to software engineering: there is no “silver bullet”, no tool that will solve it all; there is only sound engineering practice.

This paper is available online in an abridged form. Find a copy and read it.

There is no discussion section for this review question.

Answers

Discussion of Review Question 1

Describe the context of Global-Travel online sales system.

The system is in the context of providing a service to potential passengers, providing a communications medium for messages from the marketing department and providing booking, reservation and payment data to the sales function of the company.

Describe the collection of information that is relevant to that context.

Information is collected from the customer, the marketing department, the seat availability database, and the credit reference system.

Specify who has access to this information.

The customer has access to information fitting their request. Sales has access to booking, reservation and payment information. Marketing has access to the sales information and the seat availability, on which to base its promotion decisions.

Discussion of Review Question 2

Describe the inputs and outputs of the system.

Inputs from the customer include:

- Flight requirements

- Reservation/booking decision
- Payment requirements

Inputs from the marketing department include:

- Offer availability messages to display

Inputs from the sales department include:

- Seat availability
- Payment acceptance decisions

Output to the customer include:

- Details of available flights meeting requirements
- Request for booking/reservation decision
- Request for payment
- Payment acceptance decision
- Display of offers (as suggested by marketing)

Outputs to the marketing department include:

- Details of sale information

Outputs to the sales department include:

- Customer's flight requirements
- Customers booing/reservation details
- Customers payment details

From the point of view of a prospective traveller, describe the feedback loops in the system and how this affects the input and the output of the system.

The system displays only information about flights meeting the traveller's requirements. If the requirements change, the system responds with new suggestions.

If there are seats meeting traveller's requirements the system requests a booking/reservation decision.

The system responds to it information on the validity of the payment offered by the user.

Discussion of Review Question 3

Describe the possible boundary of each of the proposed system.

The boundary of FARTC is simply the system's interaction with the RSAP, the braking system itself and some method for the system to know how hard the brakes need to be automatically applied.

The boundary of SARTC needs to include the driver, since the trigger by the driver will make the SARTC system start to break the train. The system must also include the braking system to be applied and the RSAP signal.

The boundary of AARTC should include the RSAP and the driver. Since this system does not make the brakes come on, there is no need to include the braking system as part of this information system.

Discuss how they differ and why

The first system does not need to include the human driver — all that is required is a signal requesting that the noise stop playing.

The second system must include both the driver and the braking system, since the driver needs to engage the SARTC system, and can also provide the input of overriding the system.

The final system does not include the braking system at all, since it is simply an alarm system to be switched off by the driver.

Discussion of Review Question 4

The FARTC system has three main feedback loops:

- The RSAP signal input triggers the brake
- When the train is stationary, we might assume the brake mechanism releases
- The driver switched off the light

The SARTC system has five main feedback loops:

- The RSAP signal input triggers the warning
- The driver triggers the system to start braking
- The driver can trigger the system to stop braking
- When the train is stationary, we might assume the brake mechanism releases
- The driver switched off the light

The AAARTC system has two main feedback loops:

- The RSAP signal input triggers warning

The driver switched off the light

Discussion of Review Question 5

We might define an information system as a system of interrelated elements working together to achieve some goal, composing of:

- A context,
- A collection of information that is relevant to that context, and
- System functions to record, process, and regulate access to the information.

Obviously in this module we are most interested in information systems that are comprised chiefly of software components.

Discussion of Review Question 6

Real-time systems: Examples should be systems that respond to changes in very short times. Examples might include:

- automated braking systems on cars (that detect jamming and unlock the breaks for a fraction of a second)
- automatic pilot systems (that detect undesirable plane behaviour and alert the human pilot)

- computer games (detecting joystick movements and changing the display in fractions of a second to give the effect of movement)
- An autonomous robot fish (that swims in a river filtering poisons, avoid objects and surfaces to recharge its solar cells

Data-processing systems: Examples should refer to systems that process large amounts of data, and possibly provide communication with some larger network of computer-based systems. Examples might include

- Point of sales terminals that process (and validate) credit card transactions
- Electronic mail systems to allow employees to communicate with each other in different rooms or buildings
- Booking systems, such as an air-plane reservation system whereby a travel agent places a reservation for a seat into the records for a particular air-plane company, to prevent other agents booking the same seat

Decision-support systems: Examples should be about systems that attempt to analyse data in terms of certain models, and perhaps predictions of future outputs based on extrapolations of the data. Examples might include:

- stock re-ordering systems, that monitor stock changes and attempt to predict optimal ordering to reduce stock keeping costs but ensure orders can be met without delay
- marketing systems that model likely demand and sales for new products based on timing and advertising budget decisions

Expert systems: Examples should be about systems that model human expertise and decision making. Examples might include:

- expert systems that process inputs against a knowledge-base of past situations and decision rules to make (and justify) suggestions for decisions (such as diagnosing illnesses or categorising applicants for life insurance or loan applications)
- a car-diagnosis system to guide a telephone centre operator to ask questions about a car to locate the problem so the right parts can be taken along by a rescue vehicle

Discussion of Review Question 7

There are a number of consequences, including:

- currency changes (EU countries would no longer need any change in currency, while the currency to be exchanged with the USA customers will be different)
- legal changes — there may be new or changed import/export laws between the UK and EU countries, possible less or different taxation

Implications for software system are that any existing systems will need to be changed. It might be that the company has a sister-company elsewhere in the EU, so perhaps a decision about which information system (or which bits of each) should be retained so that both companies will move towards using the same system. The jobs done by staff for processing EU sales will probably be simpler (and different) from the processing of USA sales, and the human and computer information system needs to be adapted to make such different order processing straightforward.

An important thing to consider when updating software is that the changes are properly engineered: while all changes to software can introduce inconsistent behaviour, inefficient behaviour, and bugs, when bad software engineering practices are followed this is much more likely to occur.