Systems Development Essentials
Contents

How to use this workbook.................................................................3
Introduction .........................................................................................5
Module 1 – System Development Roles and Responsibilities ............. 11
Module 2 – Architecture .................................................................... 22
Module 3 – Systems Development Lifecycles and Practices ............... 32
Module 4 – Systems Investigation .................................................... 53
Module 5 – Methods - Structure and Content.................................... 81
Module 6 – Methods - Models .......................................................... 96
Module 7 – Quality Assurance .......................................................... 123
Module 8 – System Design, Deployment and Maintenance ............... 147
Module 9 – CASE, CAST, Application Lifecycle Management Tools . 163
Examination Hints and Tips ............................................................. 171
How to use this workbook

Activity
Alongside this icon you will find details of the group/individual activity or a point for everyone to discuss.

Definition
Where a word with a very specific definition (or one that could be described as jargon) is introduced this will highlight that a definition is provided.

Exam tips
This icon indicates an examinable technique or will highlight information that you may find useful in the exam.

Expansion materials
This manual contains examinable materials. The QA++ symbol contains further information. Skip over these during class. They are not needed for the examination.

Glossary
Definition of a term.

Helpful hint
This icon guides you to tips or hints that will help you avoid the standard pitfalls that await the unwary practitioner or to show you how you might increase your effectiveness or efficiency in practising what you have learnt.
Important idea or concept

Generally this icon is used to draw your attention to ideas that you need to understand by this point in the course. Let your trainer know if you do not understand or see the relevance of this idea or concept.

Key point

This icon is used to indicate something that practitioners in this field should know. It is likely to be one of the major things to remember from the course, so check you do understand these key points.

Reference material

When we have only touched briefly on a topic this icon highlights where to look for additional information on the subject. It may also be used to draw your attention to International or National Standards or Web addresses that have interesting collections of information.

Reinforcement

From time to time, there will be places within the course where it is useful for you to reinforce your understanding. This might be in the form of a question to ponder or a short end-of-module test.

Useful tool

This icon indicates a technique that will help you put what you have learnt into action.

Warning

This icon is used to point out important information that may affect you and your use of the product or service in question.
Introduction

Welcome to QA’s Systems Development Essentials course. During the next few days, you will learn the skills, techniques and knowledge required to attempt the BCS examination in this subject. Full details of the syllabus are at:

www.bcs.org.uk

Course Administration

Before we begin the course, your instructor needs to take you through a number of administrative points as shown below.

- Safety
- Timings
- Breaks/Meals
- Rooms
- Security
BCS Course Objectives

BCS specify that holders of the Certificate in Systems Development Essentials should be able to:

- Identify the tasks and disciplines required for systems development and the implementation
- Describe the relationship between systems development and the wider term solution development
- Interpret business requirements and produce systems requirements
- Describe the commonly used development lifecycles defined in the syllabus
- Select a particular development lifecycle based on specific characteristics
- Describe in detail one method that embraces one (or more) of these generic lifecycles
- Describe the structure, activities and deliverables of this method
- Identify the key roles and responsibilities within the chosen method and describe how these can be used to form teams
- Describe, interpret and quality assure the key models that the selected method uses for defining process, static and event perspectives of the system
- Explain the difference between logical and physical models
- Make effective use of different methods of interpersonal communications
- Quality assure the systems requirements documentation and models
- Identify different architectures for systems development solutions
- Conduct a quality review
- Explain how CASE, CAST and Applications Lifecycle Management tools might be used to support the chosen method
BCS Systems Development Essentials Exam

- 1-1.5 pages
- Includes details of an organisation and a project

Case study

- Course manual
- Notes
- Exercises
- Solutions

Open book

- No writing
- No highlighting
- No applying or moving sticky tabs

15 minutes’ reading time

- 50 marks available
- A minute per mark
- 50% to pass

60 minutes’ writing time

- Read these first
- Answers specific to the case study
- Be concise

4 or 5 questions

- All delegates must produce photo ID before the exam (BCS’ rules)

Photo ID
## BCS International Diploma in Solution Development

<table>
<thead>
<tr>
<th>Core</th>
<th>Knowledge-based Specialism</th>
<th>Practitioner Specialism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems Development Essentials</td>
<td>Foundation Certificate in Systems Development</td>
<td>Business Analysis Practice</td>
</tr>
<tr>
<td>Systems Modelling Techniques</td>
<td>Intermediate Certificate in Enterprise and Solution Architecture</td>
<td>Practitioner Certificate in Enterprise and Solution Architecture</td>
</tr>
<tr>
<td></td>
<td>ISTQB-BCS Certified Tester Foundation Level</td>
<td>Systems Design Techniques</td>
</tr>
<tr>
<td></td>
<td>Foundation Certificate in IT Service Management</td>
<td>Integrating Off-the-Shelf Software Solutions</td>
</tr>
</tbody>
</table>

*Both of the above plus 1 of the above and 1 of the above*
## BCS International Diploma in Business Analysis

<table>
<thead>
<tr>
<th>Core</th>
<th>Knowledge-based Specialism</th>
<th>Practitioner Specialism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Analysis Practice</td>
<td>Commercial Awareness</td>
<td>Modelling Business Processes</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>Foundation Certificate in IS Project Management</td>
<td>Systems Modelling Techniques</td>
</tr>
<tr>
<td></td>
<td>Foundation Certificate in Business Analysis</td>
<td>Systems Development Essentials</td>
</tr>
<tr>
<td></td>
<td>Foundation Certificate in Business Change</td>
<td>Benefits Management and Business Acceptance</td>
</tr>
</tbody>
</table>

*Both of the above plus*  
1 of the above and  
1 of the above
BCS Diploma Oral Examination

** see the candidate guidelines and syllabus on the BCS website

Recommended Reading References

- **Developing Information Systems**
  - Author: James Cadle (Editor)
  - Publisher: BCS
  - Publication Date: 2014
  - ISBN: 978780172453

- **UML and the Unified Process**
  - Author: Jim Arlow and Ila Neustadt
  - Publisher: Addison Wesley
  - Publication Date: 2005
  - ISBN: 978-0321321275

- **Object-Oriented Systems Analysis and Design Using UML**
  - Author: Simon Bennett, Steve McRobb and Ray Farmer
  - Publisher: McGraw Hill
  - Publication Date: 2005
  - ISBN: 0077092444

Also see the BCS syllabus for further and recommended reading.
Module 1 – System Development Roles and Responsibilities

This section sets the scene by looking at the roles within system development and implementation, together with their responsibilities and characteristics, and how these roles may be put together in teams.

Topics

In this section of the course, we will cover:

- The actors/roles and responsibilities within system development and implementation
- Characteristics of these roles
- Team structure
What do you think?

Who are the actors in systems development?
Clearly the actors/roles in system development will vary in different organisations, depending on the business and IT department standards and methodologies, and the project needs. Common roles in use today include business analyst, systems analyst/designer, developer, tester and technical architect.

We will consider the characteristics of each of these roles, and then consider some of the Agile extensions.

**Business Analyst Responsibilities**

Although different organisations define the role differently, there does seem to be some common ground about the tasks that most business analysts perform:

- Work with senior business and IT staff to conduct high level feasibility studies, develop business requirements and, where appropriate, business cases
- Investigate business problems and requirements to identify and recommend appropriate solutions
- Produce written reports and presentations for management, colleagues and the business on various aspects of analysis undertaken
- Role in solution development, which may involve changes to the People, Organisation, Processes and the Technology

There is a clear distinction between the business analyst role and the systems analyst role:

- Business analysts are involved earlier in the Business Change Lifecycle and work at a higher level than systems analysts. Their role is in **solution** development which may involve changes to people, organisation, processes as well as technology
- The systems analyst’s role is in **system** development, which relates specifically to changes to the technology needed to support the business change, especially applications development
Module 1: System Development Roles and Responsibilities

Systems Analyst/Designer

• What are the primary responsibilities of the Systems Analyst/Designer
• What skills/qualities should they ideally possess?

Primary responsibilities of a systems analyst, their skills and qualities:
**Systems Analyst’s Role**

Systems analysis is about analysing requirements and designing specifications for IT systems so that organisations can evaluate software packages or develop bespoke IT systems to support their business processes. Systems Analysis involves the use of techniques such as UML data modelling, process and function modelling.

A systems analyst has responsibilities relating to every stage in the system development lifecycle (SDLC). These responsibilities include:

**Analysis:**

- Understand the role of the software in the processes
- Investigate current IT systems
- Document ‘as is’ system using standard models
- Define new IT system requirements
- Identify changes to technology in support of changes to the business

**Design:**

- Design software systems to meet requirements
- Write specifications for developers
- Participate in reviews and testing

**Implementation:**

- Documentation
- Training
- Testing
- Conversion
- Handover
Module 1: System Development Roles and Responsibilities

Review:

The systems analyst will also participate in post-project and possibly post implementation reviews:

- Review of live system
- Review of development method
- What went well
- What didn’t
- Lessons learnt
- What next

Systems Analyst’s Skills and Qualities

A systems analyst should be:

- An effective written and verbal communicator
- Able to use an appropriate range of analysis and modelling techniques
- Able to use appropriate tools
- Methodical

AND take personal responsibility for quality

Technical Architect Responsibilities

Technical architects are responsible for designing the solution architecture from a technology perspective (we will look at architecture later in the course). They define the overall structure of the technical solution (operating system / network / database) so that non-functional requirements will be met, including performance, usability, availability. As well as the Operating system, the network and any database requirements.

Developer Responsibilities

Developers need to understand the specification that has been written by the systems analyst. They read the system specification, highlight anything incomplete / ambiguous and talk through it with the analyst. Developers produce a detailed component or code design.
based on the systems analyst’s specification. They design the component or code algorithm, plan testing, write the code, set up and perform the unit tests and evaluate results.

Once the code module testing is complete then integration, system and acceptance testing can follow.

**Tester Responsibilities**

Many organisations have independent test teams to aid with system and user acceptance testing. Test analysts are involved in planning and specifying the test scenarios. Testers are responsible for executing the tests, as well as recording and checking the test results.

**Other Actors**

There may be many other actors in the systems development process. This list may vary depending on the organisation and the type and style of development being undertaken, and could include:

- End users
- Project managers
- Database analysts
- 3rd-party suppliers

**How These Roles May Be Structured Into Teams**

A traditional team structure might be as follows:

- A project manager runs the management environment for the system development
- Business analysts work for the project manager and liaise with the systems analyst to specify a working fit-for-purpose system that meets the business needs
- The systems analysts also work for the project manager, and liaises with the BA to produce a working fit-for-purpose system
• Developers writing the software will be supervised by the Systems analyst/designer
• During system development support may be provided by testers, a data architect or database administrator (DBA), and a technical architect

Agile Team Roles

When an Agile approach to systems development is taken, typically the teams include a team lead/Scrum Master/’Coach’ who facilitates the team members’ efforts, and co-ordinates/obtains resources.

Self-organising (“Kanban”) team members carry out the work, including coding and testing, while the product owner represents the product’s business users and other stakeholders. The product owner may be a business analyst in fact, representing the product’s sponsor.

Support may be provided by technical experts, domain experts and independent testers.
Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The actors/roles within system development and implementation include analysts, designers, developers, testers and technical architects.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>These roles have differing responsibilities within systems development and implementation.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The focus of the systems analyst role is system development, whereas the focus of the business analyst role is solution development.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Systems analysts ideally possess effective communication skills, are methodical and are familiar with analysis and design techniques.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Team structure depends on the development approach e.g. traditional or Agile.</td>
<td></td>
</tr>
</tbody>
</table>
Module 1: System Development Roles and Responsibilities

Post Test 1

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 21 at the back of the book.

1. List three common responsibilities of a Systems Analyst.
2. List three other roles (other than the Systems Analyst) which might be involved in the system development.
3. What is the difference in focus between a Business Analyst and a Systems Analyst?
4. List three characteristics (skills/qualities) of a Systems Analyst.
5. What roles would you expect to find in a team using a traditional approach to systems development?
6. What roles would you expect to find in an Agile team?

Further Reading


Answers to Post Test 1

1. Investigation of the current system, documenting ‘as is’ system using standard models, defining system requirements.


3. Business Analyst focus is on solution development includes changes to People, Organisation, Process and Technology. Systems Analyst focus is on the system development for the Technology part of the change.

4. Any three of: effective written and verbal communicator, methodical, able to use a range of analysis and design techniques, and supporting tools, takes personal responsibility for quality.

5. Project Manager, Business Analyst, Systems Analyst, Developer.

6. Team lead/Scrum Master/Coach, Team member, Product Owner.
Module 2 – Architecture

The aim of this section is to explain the different architectures that are relevant to the analyst/designer.

Topics:

In this section of the course, we will cover:

- Different levels of architecture
- Inputs at enterprise level
- Inputs at solution and infrastructure level
- Impacts of design decisions

Architecture Granularity

There are two different levels of architecture: Enterprise and Solution. Enterprise architecture is high-level, more abstract and coarse-grained. It covers a longer period and addresses all IS/IT systems.

Solution architecture is slightly lower level, slightly less abstract, slightly less coarse-grained. It is bounded by the business requirements, although it may require integration across several systems. Solution Architecture is a subject of debate. Some organisations use Solution Architects in place of Enterprise Architects and in some organisations Enterprise Architects do the role of Solutions Architects. Some organisations don't have Solution Architects but Business, Data, Application and Technology Architects who report to an Enterprise Architect who is responsible for solution development (see TOGAF phases E and F - later).

Both levels cover the four main architecture domains.

Architecture Domains

The four different domains (or layers, or types – the term used in TOGAF) of architecture are:
Module 2: Architecture

- Business
  - Processes, roles, functions, services…
- Data
  - Data entities, data schemas and data models
- Applications
  - Groups of business functionality, although often thought of as application software
- Technology
  - Supporting systems to the applications, such as databases, file management, directory management, security, networking

**What is an Architecture?**

*“The fundamental concepts or properties of a system in its environment embodied in its elements, relationships and in the principles of its design and evolution.”*

ISO/IEC 42010:2011

Architecture describes the structural components of a system and how they work together to achieve the goals of the system. The style
guiding the evolution and construction of the components is included too.

Any significant changes to systems should started by an Architecture phase. This concentrates on understanding what the business need, the general background (or context), and will include a high-level view of the appropriate architecture along with a plan to transition systems from their baseline to the target state.

There is no such thing as an inherently good or bad architecture; architectures are either more or less fit for some stated purpose. The fitness or otherwise of the architecture should be weeded out by Architecture Board governance during the architecture development.

**Inputs at Enterprise Level**

The primary purpose of creating an enterprise architecture is to ensure that business strategy and IT investments are aligned. As such, enterprise architecture allows traceability from the business strategy down to the underlying technology. Many large companies have applied Enterprise Architecture to improve their business architectures as well as in order to improve business performance and productivity.

Enterprise/Business Drivers are identified at this level and passed down to all other levels such as Solutions, Business, Data, and Technology Architecture. At this level you also want some idea of what the issue/concern/problem is, regarding why they need to undertake Architecture to change something in the organisation. Initial Organisation Structure may be gathered at this level also then investigated in more depth at the lower levels.

The factors shown below will influence the enterprise architecture:

- Enterprise mission and vision
- Corporate Strategy
- Corporate Goals including financial target
- Major systems used within the enterprise
- Organisational structure and distribution
Inputs at Solution Level

The factors shown below will influence the solution architecture:

- Essential drivers
- Organisation
- Locations
- Business processes
- Business data
- Products and marketplaces
- Financial targets
- Baseline Architecture
TOGAF

The Open Group Architecture Framework (TOGAF) includes:

- The Architecture Development Method (ADM) – a set of processes for developing, planning and governing the development of systems. When talking about “systems” in relation to the ADM we are not thinking of IT systems, but collections of interdependent components that make up a solution to a problem/issue/concern
- Guidelines and techniques for use in the ADM
- The Content Framework that details how to effectively describe architecture and provides an indication of what content to produce at various phases of the ADM
- The Enterprise Continuum – basically a library system for storing architecture information; it is a way of Classifying/Categorising content developed during architecture that can then be stored in a Repository
- Reference Models that provide high level ideas on how to structure an IT system and how it should behave
- Architecture Capability, with strong advice on how to govern architectures. The Architecture Capability Framework tells you what to do when setting up an Architecture Practice/Capability. Part of this phase deals with setting up an Architecture Governance Board

There are in fact many other frameworks that are less well-known.
TOGAF’s ADM

TOGAF’s ADM, affectionately known as the ‘crop circle’, is shown below. TOGAF suggests via this model a process for developing and sustaining an enterprise architecture.
**Application Architecture**

Best practice in application development demands these days an object-orientated (OO) approach to software construction. OO has many advantages as an approach since for example it promotes re-use and ease of maintenance.

MVC (Model-View-Controller) is a well-known architectural pattern for OO applications. In this pattern objects are grouped into distinct layers in accordance with their responsibilities within the software.

---

**Impacts of Design Decisions**

Design decisions made by enterprise architects can affect solution development considerably. For example, decisions related to distributed data/processing, policies on interoperability, use of commercial off-the-shelf (COTS) packages and the designation of a single solution for a single business problem (standardisation).

Architectural decisions can affect an enterprise for many years.

Good architectural design is flexible and enables an agile response to change, whereas poor design results in inflexible systems which are costly and difficult to make changes to.

An approach to integrated design using an Enterprise Architecture (EA) framework, and a sound System Development Lifecycle (SDLC), should promote good design outcomes.
Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Architecture defines the form and characteristics of a system, and may take into account social and economic factors alongside the more obvious technical ones.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Generically architectures cover enterprise architecture, and within that, business architecture, solution architecture and infrastructure architecture.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>There are two levels of architecture: Enterprise and Solution</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>There are four layers or domains of architecture: Business, Data, Applications and Technology</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TOGAF is a well-known architectural framework TOGAF includes a model of change and development called the Architecture Development Method (ADM).</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Architectural design decisions will impact the design of solutions.</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 2

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 31 at the back of the manual.

1. What does the word “architecture” mean?
2. What are the two levels of architecture?
3. What are the four layers in architecture?
4. Name two inputs at Enterprise level.
5. Name two inputs at Solution level.
6. What are the results of good architectural design?

Further Reading


Answers to Post Test 2

1. “The fundamental concepts or properties of a system in its environment embodied in its elements, relationships and in the principles of its design and evolution.”

2. Enterprise and Solution.


4. Any two of: Enterprise mission and vision, Corporate Strategy, Corporate Goals including financial target, Major systems used within the enterprise, Organisational structure and distribution


6. A good architectural design gives flexibility and enables an agile response to change.
Module 3 – Systems Development Lifecycles and Practices

The purpose of this section is to consider different System Development Lifecycles (SDLC), the advantages and disadvantages of each approach, and how to select an appropriate approach. We shall also look at different development practices and their relationship to the SDLC.

Topics:

In this section of the course, we will cover:

System Development Lifecycles (SDLC)

- Waterfall
- V model
- Incremental or phased delivery
- Spiral or iterative
- Advantages and disadvantages of each approach
- Selection of a lifecycle

Practices

- Bespoke development vs. software package solutions
- Evolutionary/Agile
- Prototyping
- Component-based development
Waterfall Lifecycle

This is the traditional approach to systems development. The classic approach is a rigid sequence of stages, with each stage focused on a specific aspect of the software product to be produced.

- **Analysis**: Investigate, document and assess current system. Gather and document new requirements. Requirements documentation produced and signed off.
- **Design**: Design new/enhanced business system, design user interfaces, and produce business system specifications.
- **Build and Test**: Perform detailed technical design. Deconstruct business system specifications further into units or components. Specify these components. Code and test components.
Components are brought together to form higher level systems and tested.

- **Implementation:** Train staff, user acceptance testing, and handover.
- **Review:** Post implementation review of project and system.

The Waterfall lifecycle can be adapted for a commercial off-the-shelf (COTS) package solution. After the analysis phase, select, acquire and configure a software package would replace the design and build phases. The Structured Systems Analysis and Design Methodology (SSADM) is often cited as an example of a traditional waterfall methodology; however it was really more a case of the way SSADM was used than anything intrinsically ‘waterfall’ in the method.

**V-model**

The V model is not really an SDLC in itself, but a model of the testing workflow, using a waterfall style.

On the left of the V Model is ‘static testing’ which is the inspection of documentation about the software, such as requirements. The benefit of doing this is to catch any defects early in the lifecycle when
fixing things is relatively quick and cheap to do. Also on the left, test plans are laid out for various aspects of the software.

On the right of the V model we see the execution of the various tests that were planned, which includes reporting on test findings and taking appropriate action.

Although variants of the V-model exist, a common type of V-model uses five test levels, corresponding to five development levels. The five levels are:

- Component (unit) testing
- Component integration testing
- System testing
- System integration testing
- Acceptance testing

**Waterfall Approaches**

The waterfall lifecycle seems the natural way to run an information systems project. The project proceeds along clearly defined phases and a preceding phase must be completed and ‘signed off’ before the next starts. Phase completion is judged by the outcome of the phase matching the requirements defined by the previous phase.

But this lifecycle approach has some shortcomings: except for well understood cases, it may be difficult to completely define requirements at the beginning. Hence, prototyping is often used to aid requirements elicitation, which also improves user involvement. Additionally, with this type of lifecycle it is almost impossible to accommodate changes to requirements at a late stage. The requirements were agreed and ‘frozen’ at the analysis phase. Finally only the implementation phase produces a non-documentation deliverable, so the business user has no early vision of the system.

Therefore a waterfall lifecycle is suitable when:

- Low degree of business change likely
- Requirements clearly understood
- Working with well-understood tools, architectures and infrastructures

It is not suitable for a volatile business environment.
Advantages

- Safe and ‘signed off’
- Thorough
- Quality built in
- Easy to manage
- Design based on all requirements and interfaces
- Good documentation

Disadvantages

- Can’t go back
- Lack of user involvement
- No early vision of system
- Difficult to add missing requirements
- Business change has high impact
- Fosters ‘silo’ mentality

Incremental or Phased Approach

The incremental approach is a method of software development where the system is designed, built, tested and implemented incrementally (a little more is added each time) until the product is finished. The product is defined as finished when it satisfies all of its requirements.

The product is decomposed into a number of components, each of which are designed and built separately (termed as builds). Each component is delivered to the client when it is complete. This allows partial utilisation of the product and avoids a long development time. It provides the user with earlier use of some of the system functionality and avoids a large initial capital outlay with the subsequent long wait for the IT system. This model of development also helps ease the traumatic effect of introducing a completely new system all at once. There are, overall, few problems with this model.

Developing a system incrementally allows the developer to take advantage of what was learned during the development of earlier, incremental, deliverable versions of the system.
Advantages

- Early vision of system
- Earlier deliverables
- Reduces complexity and risk can be tackled earlier
- Easier to test and debug smaller increments

Disadvantages

- Difficult to manage
- Users can change their mind!
- May get partial solutions
- Additional costs of parallel implementation and regression testing
Iterative (Spiral) Systems Development

Spiral or iterative systems development (such as the Spiral model shown below, created by Barry Boehm) may be used when a complete set of requirements cannot be gathered at the start of the project. This approach allows requirements to be gathered iteratively. Prototypes are demonstrated to the business user, feedback captured and further requirements built into the next prototype. The system is delivered once, after a number of iterations.
Spiral or Iterative Approaches

*Iteration* means a repeating cycle of steps, which commonly includes visiting several aspects of the software product. Development is divided into a number of time boxed iterations. Some people have likened each iteration to a mini-waterfall!

The idea behind this approach is to start with a simple implementation of a subset of the software requirements and incrementally enhance the product until the full system is implemented. At each iteration, design modifications may be made and new functional capabilities may be added. Developing a system in this way allows the development team, which includes users, to take advantage of what was learnt during the development of earlier deliverable versions of the system. Learning comes from both the development and the use of the system, wherever possible.

Advantages

- Requirements evolve
- Flexible, incorporates change
- Plenty of user involvement
- Early vision of system
- User buy-in is higher
- Cost can be controlled

Disadvantages

- Difficult to manage multiple iterations and multiple iteration teams
- Shortcuts can be taken
- Hard to scope
- Scope creep, ‘gold-plating’
- Complexity of integration testing can lead to higher costs

The Dynamic System Development Method (DSDM) and the Unified Process (UP) are frameworks which support the iterative development of software.
DSDM was first released in 1994 by a consortium of organisations including Norwich Union, British Airways, American Express, and Oracle, to provide discipline to Rapid Application Methods (RAD).

DSDM fixes cost, quality and time at the outset and uses the MoSCoW prioritisation of scope into musts, shoulds, coulds and won’t haves to adjust the project deliverable to meet the stated time constraint.

Today, the DSDM consortium – a not-for-profit organisation – is the guardian of DSDM Atern, the latest (2007) version, which incorporates Agile practices. DSDM is in fact one form of Agile in the modern context.
The Unified Process (Rational)

The Unified Software Development Process (USDP), from the authors of the UML, is also an iterative approach; in fact USDP is the ancestor of Agile.

USDP is commonly referred to as the Unified Process or UP. Projects are broken into small subprojects (the iterations) that deliver system functionality in chunks, or increments, leading to a fully functional system.

Each iteration contains all of the elements of a normal software development project: planning, analysis and design, construction, integration and test. Each iteration generates a baselined increment that comprises a partially complete version of the final system and associated project documentation. The baselines build on each other over successive iterations until the product is considered releasable and able to contribute business value.

Selection of an Appropriate Approach

Here we can see some of the considerations when selecting an approach.

Note in practice it is more helpful to think of these options as choices along a spectrum of approaches, from a formal structured approach at one end to a flexible, informal approach at the other.
Hence, for each project we should ask ourselves “should the development be run more towards the *waterfall* end of the spectrum or more towards the *iterative* end?”

<table>
<thead>
<tr>
<th></th>
<th>Waterfall</th>
<th>Incremental</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirements</strong></td>
<td>Requirements are clear at start</td>
<td>Requirements are clear at start</td>
<td>Requirements are unclear or volatile</td>
</tr>
<tr>
<td><strong>Delivery</strong></td>
<td>Single delivery essential</td>
<td>Phased delivery essential / desirable</td>
<td>Staged delivery possible/desirable</td>
</tr>
<tr>
<td><strong>Business Environment</strong></td>
<td>Low degree of business change</td>
<td>Some business change likely</td>
<td>High degree of business change</td>
</tr>
<tr>
<td><strong>Enterprise considerations</strong></td>
<td>Many stakeholders across the enterprise; large scale enterprise development</td>
<td>Enterprise wide or large scale development</td>
<td>Few stakeholders; small scale local developments; proof of concept</td>
</tr>
<tr>
<td><strong>Complexity of problem</strong></td>
<td>Simple problems</td>
<td>Problems that can be broken down</td>
<td>Complex: tackle complexity and risk early</td>
</tr>
<tr>
<td><strong>High regulatory requirements</strong></td>
<td>Good documentation and assurance</td>
<td>Good documentation and assurance</td>
<td>Chose an agile framework e.g. DSDM</td>
</tr>
</tbody>
</table>
Development Practices

Whichever lifecycle style is adopted it may also be necessary to select certain development practices. The BCS syllabus lists the following as practices:

- Bespoke development vs. Software Package solutions (COTS Commercial off-the-shelf)
- Evolutionary / Agile
- Prototyping
- Component-based development

A decision is needed at some stage as to whether the system will be acquired using a bespoke development or buying a software package.

If a bespoke approach is chosen, will an Agile practice be indicated? Will prototyping be useful as a way to confirm user requirements? Can existing components be re-used and re-configured?

Acquisition Options – Build or Buy

Software can be acquired by commissioning a bespoke development (build), where software is written to the customer’s exact specification, or a Commercial off-the-shelf (COTS) package may be purchased (buy) when the requirement is for standard software already available from vendors.

- Bespoke should be considered for software that supports unique enterprise processes, or processes crucial to the enterprise value creation process
- Packages should be considered to solve routine business problems, e.g. Payroll
- Enterprise Resource Planning (ERP) packages cover most of the routine problems for an entire business, e.g. SAP
Bespoke

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed and built to meet a set of agreed requirements</td>
<td>Expensive to produce – either in-house or outsourced</td>
</tr>
<tr>
<td>Not paying for unrequired features and functionality</td>
<td>Have to invest in maintenance and support for the product</td>
</tr>
<tr>
<td>Could provide competitive advantage</td>
<td>Documentation tends to be poor</td>
</tr>
</tbody>
</table>

ERP

An important variant on the package route is to use Enterprise Resource Planning (ERP) packages. An ERP is designed to cover most of the routine functions for an entire business, e.g. SAP provides software to cover functions like customer relationship management, human capital management, financial management, procurement and much more.

The advantages of ERP are that you automatically get interoperability between modules and common data designs. Disadvantages include a huge up front cost and considerable risk as the business tries to adapt to the constraints imposed by the ERP.

Cloud Option

Yet another option for software acquisition that has become increasingly important is the use of Cloud Services. The basic premise behind using Cloud Services is to ‘rent’ the software and pay only for what is used, rather like water or electricity. The advantages include avoiding the costs of ownership of IT, like infrastructure costs, easy to scale and 24x7 round the world availability. Disadvantages include security concerns and, at least at the moment, broadband coverage is not guaranteed everywhere.
Agile/Evolutionary Development Practice

Bespoke development could be conducted as an Agile project. Agile is one form of the iterative evolutionary approaches discussed earlier, and has become very popular. The focus is on working software produced quickly and in direct collaboration with the software users.

The Agile Manifesto for Software Development states:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.”

Agile as a practice is summed up as:

“A group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The Agile approach is:

- Iterative
- Adaptable
- Rapid
- Cooperative
- Quality-Driven” (Source: QA - Agile and Scrum Awareness Course)

Agile’s main principles include:

1. **Satisfy the customer** through early and continuous delivery of valuable software.
2. **Welcome changing requirements**, even late in development.
3. **Deliver frequently**
4. Business and IT must **work together** daily
5. Build projects around **motivated individuals**.

6. Face-to-face **conversation**.

7. **Working software**, work done is the primary measure of progress.

8. Agile processes promote **sustainable development**.

9. Continuous attention to **technical excellence** and good design enhances agility.

10. **Simplicity** – the art of maximizing the amount of work not done

11. Architectures, requirements, and designs **emerge** from **self-organising teams**.

12. The team **reflects** on how to become more effective, & **adjust accordingly**

(Source: QA - Agile and Scrum Awareness Course)

**Evolutionary Delivery**

Going hand in hand with iterative development is evolutionary delivery. A large waterfall style project might take a year or more to deliver useful software, but in evolutionary delivery the business gets a new version to test, use and critique as often as every couple of weeks. In addition, at the beginning of every cycle the business users have the opportunity to set priorities for the next version. This lets the business start using the high-priority features right away, and makes sure that the software meets their needs. As an added bonus, the business users are never left wondering, "What are those IT people doing?"

Examples of iterative, incremental, evolutionary approaches are the Dynamic Systems Development Method (DSDM) and the Unified Process (UP), including Agile/Extreme Programming (XP).

**Prototyping**

All iterative approaches depend on using prototyping to some extent. A prototype is a mock-up of the required product so that the customer can appreciate and comment on what is being built.

Prototypes can be:
• Paper based drawings
• Screen dumps
• Demo software
• Working software

Prototypes in most environments are thrown away, but for methods like Agile and UP, the production software is really just a prototype for the next version – the idea of perpetual evolution!

The advantages of prototyping are that customers find it easier to comment on something visual, and prototyping supports incremental development, which provides the customer with early vision of the system and its functionality.

The disadvantages are that the requirements and design work can become confused, as customers want to design the solution while the prototype is being used to gather requirements. Also, because the prototype may look like a ‘real’ system, users may believe that the production process is more advanced than it really is.

**Package**

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost and Time</td>
<td>Not 100% fit to requirements</td>
</tr>
<tr>
<td>Quality</td>
<td>Ownership issues</td>
</tr>
<tr>
<td>Documentation and Training</td>
<td>Financial stability of the vendor</td>
</tr>
<tr>
<td>Maintenance and Enhancement</td>
<td>Lack of competitive advantage</td>
</tr>
<tr>
<td>Try before you buy</td>
<td>Limited legal redress</td>
</tr>
<tr>
<td></td>
<td>Changing nature of requirements</td>
</tr>
</tbody>
</table>
Component-based Development

A software component is a system element offering a predefined service, and able to communicate with other components; a reusable program building block that can be combined with other components to form an application. Component based development is at the heart of the OO approach.

A component should meet the following five criteria to fulfil the definition:

- Multiple use
- Non context-specific
- Composable with other components
- Encapsulated i.e., non-investigable through its interfaces
- A unit of independent deployment and versioning

A simpler definition can be: A component is an object written to a specification. It does not matter what the specification is: COM, Java Beans, etc., as long as the object adheres to the specification. It is only by adhering to the specification that the object becomes a component and gains features like reusability and so forth.

Component-based development states that software should be developed by gluing prefabricated components together much like in the field of electronics or mechanics. It claims that software components, like the idea of a hardware component used e.g., in telecommunication, can be ultimately made interchangeable and reliable.

Components may come from different sources: Internal or external, bespoke or off-the-shelf, or from Web/Cloud sources. End-user computing may also be based around components; e.g. Microsoft’s VBA

Component Granularity

Components exist at different levels of granularity:

- Small, low level components may be developed in-house and used across many projects, or bought in and reused in many applications
• High level components tend to be bought in. For example, whole COTS products could be components in broad, multi-application solution
## Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A waterfall lifecycle is the traditional lifecycle for systems development. It consists of rigidly sequential phases.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>The V model is not really a system development lifecycle, it describes how testing activities can occur in parallel with other tasks in the project lifecycle.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>With incremental and phased approaches, the system is designed, built, tested and implemented incrementally (a little more is added each time) until the product is finished</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>With iterative approaches, a partial system is built by developing an evolutionary prototype in build cycles, which are time boxed. These approaches allow frequent reassessment of scope, requirements and solutions. Examples are DSDM and UP</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>As well as choosing a lifecycle, a decision must be made regarding using one or more development practices, i.e. bespoke vs. COTS (plus ERP or Cloud), Agile, component based and the use of prototyping.</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 3

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 52 at the back of the manual.

1. What are the three main approaches to systems development?
2. What is the main difference between the two approaches?
3. Which phase of the traditional lifecycle is split in the V-model and why?
4. What is component based development?
5. What development practices should be considered within any given System Development Lifecycle (SDLC) approach?
6. What are the advantages and disadvantages of a package solution?

Further Reading

Answers to Post Test 3

1. Waterfall, spiral/iterative, incremental/phased approaches.

2. The waterfall approach is a sequential production process with sign off after each phase. With incremental/phased approaches software is designed, built, tested and implemented in increments. With spiral/iterative and incremental approaches the software product evolves through a series of iterations, until it is releasable, generally within a set time box.

3. The testing phase is split, so that testing activities can occur in parallel to lifecycle activities.

4. Component-based development involves assembling a software product from reusable program building blocks.

5. Development practices including Agile practices, Commercial off-the-shelf (COTS) packages and prototyping should be considered.

6. The advantages of a package solution are, that the software is:
   - A proven solution
   - Relatively inexpensive over the product lifecycle
   - Based on best practice
   - Delivered with good documentation and support
   - Kept up to date with upgrades

   The disadvantages for the customer are:
   - They have to put up with the facilities offered
   - They will be locked into a supplier
   - The package might constrain the business
Module 4 – Systems Investigation

This section covers the identification of stakeholders in the project, the fact finding approaches that might be used to identify their requirements for the system, and the classification of requirements as functional or non-functional.

In the Unified Modelling Language (UML), requirements are documented as Use Cases, so we will look at the UML Use Case diagram.

Finally we will consider some of the social aspects relating to systems investigation and the introduction of change.

Topics:

In this section of the course, we will cover:

- Fact finding approaches
  - Workshops
  - Prototyping
  - Interviewing
  - Questionnaires
  - Scenario analysis
- Functional requirements definition
- Non-Functional requirements definition
- Documenting system requirements
- Human aspects of system investigation and the introduction of change
Stakeholders

Stakeholders are those people who are affected either directly or indirectly by a specific change, often characterised as the people ‘who care’.

The business sponsor is a very important stakeholder. He/she:

- Holds the budget and ideally has executive authority
- Will often have a good idea of who the stakeholders may be

Systems analysts will need to consult the system end users, business managers, subject matter experts (SMEs), members of the project team, technical architects and testers.

Besides the business sponsor, business analysts will need to consider those affected by the system change, who are internal to the organisation and those who operate externally to the organisation.

Other stakeholders may include those in the following groups: customers, partners, suppliers, competitors, regulators, owners, employees and managers.

Many IT solutions fail because stakeholders have not been correctly identified, or the fact finding methods chosen have failed to elicit a complete and correct set of system requirements.

What Information to Gather

“I keep 6 honest serving-men (they taught me all I knew);
Their names are What and Why and When
And How and Where and Who”
Rudyard Kipling

Software sits in a business context. In order to create software that is fit for purpose it is essential the analyst captures enough information about the current and future business systems to be able to specify
an appropriate product. This is classic Business Analysis work, but the Systems Analyst needs to have an appreciation of all this too.

Kipling’s famous poem was about his children, who were always asking questions! Still, this is quite a good way of focusing the analysis task. When the analyst knows the answers to all these questions within the scope of the business problem, then a large part of the analysis work is done.

Inevitably most of this information will have to come from the stakeholders. There are a large number of investigative techniques available to enable the analyst to elicit this information. Those covered by the BCS syllabus here are: workshops, prototyping, interviewing, questionnaires and scenario analysis.

**Workshops**

A facilitated workshop is a team-based information gathering and decision making technique designed to accelerate business planning and development. It is an interactive communication technique involving experienced and empowered personnel working in one or more sessions run by an independent facilitator. A workshop is a process to be implemented when there is a requirement to make decisions, explore ideas and exchange knowledge to solve a business problem.

The strength of the facilitated workshop technique is that it enables the exchange of information between key individuals and enables them to reach decisions that are mutually acceptable. A workshop provides a forum for exchanging views and achieving consensus decisions in a structured framework across and within areas of the business. Clear deliverables are produced during the workshops enabling all attendees to review decisions taken by the group.

Workshops offer the following benefits:

- **Speed:** By gathering all the relevant stakeholders in a workshop, it is possible to achieve an agreed set of user requirements, for example, in a day or two as opposed to weeks or months using traditional methods
- **Ownership:** Stakeholders who are part of the group decision-making process are more likely to be committed to its decisions
and to feel a greater sense of ownership for subsequent development work

- **Productivity**: The workshop often benefits from the participants building on each other’s ideas and gaining a better understanding of each other’s viewpoint. Often misunderstandings in the business areas can be swiftly clarified during workshops

- **Consensus**: The workshop provides an opportunity for the participants to discuss relevant subject matters, including the major issues and problems, with a view to reaching a consensus on all the important decisions

- **Quality of decision-making**: Because stakeholders are active players in the decision-making process, the levels of confidence and understanding of the workshop output are likely to be high

- **Overall perspective**: Workshop participants can appreciate the importance of business areas contained within the project, but in which they are not directly involved

It is not all good news, however. Workshops are hard to schedule as we need to fit in with all of the participants’ availability. Group dynamics need careful managing as some people may turn up with their own hidden agendas.

**Using Workshops**

- Enable requirements to be captured from several perspectives at once
- Must have the right size group
  - Too many and people get lost
  - Too few and there is no synergy
- Useful at different project stages
  - Identifying problems
  - Eliciting / negotiating requirements
  - Investigating solutions
- Group must be empowered to make decisions
Workshop Roles

Sponsor

The sponsor ‘owns’ the workshop and its objectives and is the ultimate decision-maker. The sponsor (or delegated person) works with the facilitator to create the terms of reference for the workshop. They will agree the rules for the workshop and the sponsor empowers the facilitator to enforce them. The sponsor also empowers the participants of the workshop to make the necessary decisions.

Facilitator

The facilitator is the key player in the workshop. He or she ensures the group meets the agreed objectives in the agreed timeframe. It is important that the facilitator manages the workshop process and NOT the content of workshop.

The facilitator maintains the focus of the workshop, manages the group dynamics and encourages individual contributions. Naturally the facilitator also upholds the workshop rules.

The facilitator needs a range of attributes, including:

- Detachment, neutrality and objectivity
- Professionalism and integrity
- Positive, patient, cheerful, organised demeanour
- Active listening ability
- Ability to question and to give feedback
- Ability to maintain group participation and consensus
- Ability to handle group dynamics and conflict

Participants

The participants are required to contribute fully and positively to the workshop by providing the ideas and expertise. They must work towards group success and stay within the workshop rules. It is vital that the participants are empowered to speak and make decisions on behalf of their areas.
The Scribe
The scribe works closely (and often interchangeably) with the facilitator. He or she is responsible for ensuring workshop output is captured accurately and published as formal documentation. The scribe takes no other part in the process.

Prototyping
A common complaint from analysts is that users are much better at telling you what they do and don’t like or want when something is presented to them, than they are at telling you what they want off the top of their heads.

By creating a ‘demonstration system’, prototyping shows the users how a system might work. This can help clarify their thinking and firm up requirements before we are too far down the development path. The ‘system’ can be anything from a set of screen mock-ups, or a paper prototype, to a fully-fledged application. Prototyping is an integral part of iterative / incremental systems development, and has a strong link with scenario analysis (covered later).

Advantages
- Helps the user to really determine the requirements
  - Especially user interface, performance, navigation paths
- Validates requirements
- Reduces risk of “getting it wrong”

Disadvantages
- Can run out of control
- Can raise unrealistic expectations
- Can lead user to overestimate progress

Interviewing
An interview is a structured discussion used to elicit facts and information. Interviewing is an excellent elicitation technique and a key analysis skill used for gathering information for requirements and to quickly build up rapport with the stakeholders.
Among the advantages of using an interview is that it enables us to build a working relationship with key stakeholders as we are in a confidential, one-on-one setting with them. If we were to rely exclusively on interviews, however, it would be very time consuming, and each interview only gives us one stakeholder’s point of view.

Whichever fact finding approach is used, there are a number of steps to go through. For an interview these steps include:

- **Planning**: What to ask, who to interview, how to conduct interview, what topics to cover, and where the interview will take place.
- **Conducting**: listen, watch non-verbal signals and verbal signals, be aware of attitude and blocks, use open questions i.e. questions starting what, where, when, how, why or who. Decide how to record the interview results.
- **Following up**: write up interview notes as soon as possible after the interview, and then prepare a formal record of the interview to be checked and confirmed by interviewee.

### Questionnaires

Questionnaires are a very useful way of eliciting limited amounts of information from a large group of people, especially if they are geographically spread. The skill comes in formulating the questions so that you are able to get the needed information in a way that is easy to analyse and turn into requirements.

A questionnaire may be used to elicit information about the usability of an existing system. Questionnaires may be sent to prospective suppliers of a COTS solution, to gather information to determine whether the package will meet the user requirements.

There are software tools available to assist with questionnaire creation, administration and results analysis.

**Advantages**

- Can reach a large audience, even if geographically separated
- Can uncover common problems
- Can determine attitudes
Disadvantages

- People don’t like them
- They must be kept small and focused
- It takes skill to formulate them

Scenario Analysis

Scenario analysis is essentially telling the story of a task or transaction ‘holistically’, including any use of IT. This technique is sometimes referred to as mapping the ‘Customer Journey’.

Its value is that it enables users who may be uncertain about what is needed to visualise a task from its initial trigger through the steps needed to achieve a successful outcome. The analyst needs to ask the ‘what if’ questions to ensure that all paths have been covered. For complex scenarios, it may be easier to analyse different paths as separate scenarios.

The advantages of scenarios are that if we take the user through every step it is less likely that any elements get taken for granted. By helping the user visualise all possibilities (‘what if…?’), beginning with the ‘sunny day’ or normal course of events, we reduce the possibility of missing detailed ‘exception’ requirements. This technique is also useful to build test scenarios.

However working through scenarios with a number of users is very time consuming and large user tasks may produce complex scenarios.
Requirements

What is a Requirement?

The stakeholders are the principle source for the requirements of the IT system we wish to specify.

BCS ‘Business Analysis’ Glossary Definition

“A feature that the business users need the new system to provide”

This syllabus is concerned with IT system requirements. A requirement is a specification of a want or need that an eventual application solution should satisfy. Hence it is important that requirements themselves are expressed in a solution-free way.

Requirements can be classified initially as Functional Requirements (FR) and Non-functional Requirements (NFR):

- FR specify what the product must do, what actions it must take, in order to satisfy the product’s goals, as defined by the stakeholders
- In a software product FR amount to some form of data/information manipulation (Create/Read/Update/Delete - CRUD)
- NFR specify how the product must be, what characteristics it must have
- NFR are about specifying the way the functionality is to be delivered (form)
- NFR are often regarded as constraints on the way the functionality operates
Why are Requirements Often Poorly Defined?

Users often find it difficult to visualise what they need and so are unable to describe a system requirement. Tacit knowledge may be a problem, i.e. users may take for granted that the analyst knows what the system needs to do, or users may present us with solutions rather than requirements.

All this leads to requirements that are ambiguous, incomplete, inappropriate or incorrect.

Poorly defined requirements will result in far larger and more costly problems during system development.

Separate Functional and Non-functional Requirements, and Solutions

One challenge we often face is that stakeholders give the business analysts a mixed bag of requirements, complete with solutions. We need to ensure that we can liaise with the stakeholders and engineer a usable set of requirements so that we have a sound basis for developing the required product.

Requirements in this context are expressed as statements of what an IT system must do in order to meet its business goals (functional requirements) and statements of the constraints that must be imposed upon the system, that condition the delivery of the
functionality (non-functional requirements). All requirements must be implementable and testable and/or be the subject of SLAs.

**Functional Requirements**

A functional requirement is a statement of what an IT system must do in order to support business processes. It is a description of the functionality, relating to data capture, manipulation or reporting, which must be provided by the IT system. Examples of Functional Requirements:

**The system will:**

- Capture a Customer’s details
- Update a Customer’s details
- Delete a Customer
- Produce a report of all current Customers

The BCS give us the following categories for functional requirements:

<table>
<thead>
<tr>
<th>Data entry</th>
<th>Data maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Gathering and recording data</td>
<td>• Changes to data, including data deletion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedural</th>
<th>Retrieval requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Implementation of business rules</td>
<td>• Reporting, responding to enquiries</td>
</tr>
</tbody>
</table>
Non-functional Requirements

In systems engineering and requirements engineering, non-functional requirements (NFRs) are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviours. This should be contrasted with functional requirements that specify specific behaviour or functions.

Some NFRs will be common across the IT application, whilst in other cases FRs have specific NFRs attached to them. Other terms for non-functional requirements are "quality attributes", "constraints" and "quality of service requirements".

NFRs can contribute enormously to the success of the IT product. NFRs can dramatically affect the choice of feasible technical options in terms of automation, hardware requirements, etc. They are therefore a very important but separate dimension from what the system will do. Many NFRs are required operational characteristics.

Examples of NFRs

_Only the Accounts Supervisor may change a customer’s credit limit_

- This is an access constraint – the product must _have_ appropriate access control

_Reprint the report on demand_

- This is an availability constraint – when this functionality must _be_ available

_Manage an enquiry on available customer credit within 5 seconds_

- This is a performance constraint – the system must _have_ a response time of < 5 seconds when delivering this functionality_
The BCS give us the following examples of typical categories of NFRs:

<table>
<thead>
<tr>
<th>Performance</th>
<th>Security</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Speed of processing transactions</td>
<td>• Security levels for protection of data</td>
<td>• Permissions, who has access to which functionality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backup &amp; recovery</th>
<th>Archiving &amp; retention</th>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Protection against loss of data</td>
<td>• Duration, methods, eventual deletion</td>
<td>• Reliability, data integrity, user error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Availability</th>
<th>Usability</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Timeframe for availability of functionality</td>
<td>• Ease of learning, ease of use</td>
<td>• Data volumes, transaction volumes, user volumes</td>
</tr>
</tbody>
</table>

**Requirements Hierarchies**

Requirements are often hierarchical and have dependencies. It is very important that the analyst organises the requirements recognising these dependencies.

A functional requirement may have non-functional requirements specific to it, e.g. Reprint the report on demand.

A non-functional requirement could lead to the need for functionality, e.g. Access control leads to functionality for:

- Identification of the user
- Authentication of the user
Solution Language

The analyst must try to avoid including solution language in a requirement statement. Business people often think in terms of solutions, but it’s the abstract need that interest us.

Requirements are a step that connects solution and problem. Solutions are volatile, whereas requirements are generally quite stable. Some solutions may be mandatory (e.g. SAP), but we still need to know what the requirement is, in order to make sense of the solution!
Documentation Styles

There are many different documentation styles to choose from - Requirements Catalogue, Business Requirements Document (BRD), Prioritised Requirements List, Functional Specification, Software Requirements Specification (SRS), User Story, Use Case, IEEE 830, IEEE 1233, and so on. In the Unified Process (UP) it is customary to document the required system functionality as Use Cases.

Use Cases

A use case is a ‘structured requirement’ that describes a piece of system functionality. During analysis, use cases are descriptions of the external sequence of events between an actor and the system needed to meet a functional requirement.

Later, in design, the use case description will include UML models that describe the internal co-operation of the components which are needed to realise each use case. These models are covered in other BCS Diploma courses (cf. SMTU/SDT).

Use Cases are found by examining the business process(es) and procedure documents, discussing requirements with the business and understanding what functionality the IT system should provide.

Focus of Use Cases

A Use Case is expected to deliver a result of value to an actor who is using the IT system, in order to further the aims of a business process that he or she is a ‘worker’ in.

The scope of a Use Case is a single business transaction that a user (actor) undertakes with the IT system in order to achieve a relevant business goal. For example: Change Supplier Address might be a Use Case.

Use Cases describe only interactions with the IT system and not any events outside of it, and should not be confused with business processes. The use case is a piece of IT functionality; the business process is an end-to-end set of business tasks, some of which may be supported by IT.
Actors and the System Boundary

Actors represent named user roles that interface with the system, and could be human, i.e. the users, or non-human, e.g. another system or even time.

A system actor is given the stereotype <<system>> and time is labelled <<time>> on a use case diagram. Actors may provide input to the system and/or receive output from it.

They are external to the IT system under study and therefore define the system boundary.

Sales clerk  <<system>>  Finance  <<time>>  Weekly
Use Case Diagram Notation

The three main symbols are Actor, Use Case and the Association line that says: “This Actor is involved with this Use Case.”

On-line Customer

Place Order

Use case diagrams are very simple. If no association exists between an actor and a use case then, when the system is built, it should NOT be possible for a person logged on in the role of that actor to gain access to that use case. Note: in UML 2 the association should only have an arrow on it when an actor is the recipient of output but hasn't been involved directly with the use case.

Use Case Diagram

Use Case Diagrams are often created to capture, at a high level, the scope of the required functionality of an IT system. They are particularly helpful during workshop situations.

All use cases are, by definition, within the scope of the new system – indeed the sum of the use cases is “The System” to be built.

The naming of use cases is important; use cases should be named using a verb-noun phrase such as ‘Create Purchase Order’, ‘Accept Delivery’, etc. because they represent Functional Requirements.
How to create a use case diagram:

- Examine the business processes under review and discuss with the stakeholders what functionality IT might usefully provide for them in their jobs. The stakeholders that will use the system are the Actors; by convention such actors are called Primary Actors. (NB: actors can be ‘time’ or another IT system in this context). What they want to do with the system are the Use Cases.
- Each use of the system by an Actor to accomplish a single business goal is a Use Case – so we must think of a verb-noun phrase to describe it.
- Draw the boundary, name the scope of the IT system, draw on the Use Cases and Actors, and connect them via association lines.
Working through the Requirements Specification

It is crucial to ‘organise’ the requirements as a coherent set.

1\textsuperscript{st}: when faced with specifying an IT system, decide on the required functionality that addresses the business problem \textit{directly}. This will lead to an initial set of Use Cases.

2\textsuperscript{nd}: decide what Non-functional constraints are applicable to this functionality, either across the board (like Access Control) or specific to a given function. UML doesn’t have a way of documenting NFRs. General NFRs are documented as \textit{supplementary specifications}. FR-specific NFRs are documented with the Use Case specification.

3\textsuperscript{rd}: work out if any \textit{additional} functionality derives from the NFRs. This will usually be functionality specified \textit{within} a Use Case.

Each Use Case is documented separately. Initially a \textit{summary} Use Case Description is produced, followed by a more \textit{complete template} later in the development. (Note: this is covered in the SMTU syllabus.)

\textit{Summary Use Case Description Example for a library system}

\begin{tabular}{|l|}
\hline
\textbf{Name}  \\
Borrow Item  \\
\hline
\textbf{Goal}  \\
To create a loan record for each item borrowed.  \\
\hline
\textbf{Brief Description}  \\
A member brings the item, or items, to be borrowed to the checkout counter. The member selects the borrow items option and provides membership identification. The member then identifies which item(s) are being borrowed. The library system makes a loan record for each item lent and the member takes the item(s).  \\
\hline
\end{tabular}
Use Case Template

Eventually more detail is added, based on a set format. NB: UML does not itself have a standard for use case documentation, but there is a de facto set of fields generally used, some of which are shown here.

<table>
<thead>
<tr>
<th>Member</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Selects ‘Borrow Item’</td>
<td>Prompts for member’s reference.</td>
</tr>
<tr>
<td>2. Enters the member’s reference into the system.</td>
<td>Retrieves matching member’s details. Requests confirmation. A1</td>
</tr>
<tr>
<td>3. Confirms member.</td>
<td>A2</td>
</tr>
<tr>
<td>4. For each item being borrowed:</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Prompts for item reference.</td>
</tr>
<tr>
<td>7. Confirms item to be loaned.</td>
<td>Creates loan record. Marks item as ‘out on loan’. Prompts to continue. A4</td>
</tr>
<tr>
<td>8. Selects continue or completed.</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Use case ends.</td>
</tr>
</tbody>
</table>

Post-condition: Loan record created for each item lent

Alternatives:

- A1. Member not found,
- A2. Wrong member,
- A3. Item not found,
- A4. Wrong item

Agile Approach - User Stories

In Agile, Use Cases are not used. User Stories are the preferred way to identify what the system has to do.

A User Story is a more granular form of Use Case, and is written by the Product Owner. The idea though is very similar to a Use Case.

The narrative of a User Story has a set formula: As [role] I want to [use a feature] so that I can [achieve a goal]. Here is an example:
As a user I want to be able to compare vehicles side by side so I can see different features and prices

Priority: Should

On the back of the card are written a list of acceptance criteria (see later session on Quality Assurance).

- Vehicles added to the shortlist must be of the same type (eg. passenger vs. SUV)
- Up to 3 vehicles can be added to the shortlist
- A minimum of 2 vehicles must be in the shortlist to enable compare
Module 4: Systems Investigation

User Story Characteristics

<table>
<thead>
<tr>
<th>Brief, informal expression of a requirement</th>
<th>Written by a business stakeholder</th>
<th>The basis of a feature to be developed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close contact between customer and developer</td>
<td>Little documentation</td>
<td>Used in XP and other agile methodologies</td>
</tr>
</tbody>
</table>

Some projects will benefit from an Agile approach, e.g.:

- Clarifying requirements
- Proof of concept
- Small scale, local, departmental software
- Construction Phase of UP

Agile is often criticised for not keeping enough information about the software, and just focussing on the working code. Such an approach may give rise to problems when the software needs to be modified at a later date.

Human Aspects of Systems Investigation

Analysts must take into account the human aspects of systems investigation.

Analysts require effective communication skills to carry out systems investigation. Communication is the giving, receiving or exchange of information, opinions or ideas so that the material communicated is completely understood by all concerned.

Ineffective communication costs organisations, and individuals, money, time and relationships. It is therefore important – both professionally and individually – to communicate as well as we can.

There are different types of communication:

- Formal vs informal
- Spoken or verbal communication: face-to-face, telephone, teleconference and other media.
- Non-verbal forms of communication: body language, gestures, how we dress or act.
- Written communication: letters, e-mails, books, magazines, the Internet or via other media.
- Visualisations: graphs, charts, models, maps and logos.

Analysts must make an appropriate choice of the method of communication, taking into account the purpose of the communication, the target audience, their level of understanding and any barriers to communication.

**Barriers to Communication**

Whichever investigation method is chosen, analysts need to be aware of these barriers:
- Language that is not appropriate
- Too much information
- Poorly defined objectives
- Distortion
- Wrong medium
- Anxiety preventing full understanding
  - Redundancy
  - Loss of earnings

**Human Aspects of Introducing Change**

Systems analysts need to be aware that the systems they are developing are part of a wider solution, and that IT systems must fit with the changes to people, organisation and technology to deliver improvements in the way an organisation carries out its business.

People’s reaction to change will be affected by whether the change is planned, reactive or as a result of a crisis, and there is likely to be a negative impact on their performance.
Stage 1: Change is launched. There may be a small increase in performance if the launch commitment is strong. However, some people will adopt a ‘wait-and-see’ attitude, others will be indifferent, others concerned.

Stage 2: Performance problems arise as people struggle to cope with the new system.

Stage 3: Turning point as most problems are ironed out.

Stage 4: Productivity starts to increase as the system becomes familiar.

Stage 5: New patterns established at a higher performance level than before.

Change specialists have also identified the SARAH curve to model personal (and group) reaction when confronted with change.

SARAH stands for: Shock, Anger, Rejection, Acceptance and Hope.
Dealing with Resistance

People’s resistance to change may come from a number of sources:

- Anxiety: ‘will things be better… or worse in the long run?’
- Shame: ‘is it because we weren’t doing our jobs well enough?’
- Fear of performance: ‘will I be able to cope with the new system?’
- Uncertainty: ‘I cannot think what will happen’

Identify which approach to use:

**Education and Communication:** use where there is a lack of (or misunderstood) information. If persuaded, people will help with implementation; the disadvantage is that this is time consuming.

**Participation and Involvement:** use where initiators do not have all the information to design the system, or there are strong pockets of resistance. People who participate will be committed and their information integrated. But will they come up with something acceptable?

**Facilitation and Support:** use where people are normally resisting change: i.e. adjustment problems, as no other approach will work with adjustment problems. This can be time consuming, expensive, and still fail.
## Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stakeholders are individuals or groups (internal or external) who have an interest in (or may be affected by) the issue at hand.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>There are eight key “types” of stakeholders – partners, suppliers, regulators, employees, managers, owners, competitors and customers.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>There are a range of system investigation techniques: workshops, interviewing, prototyping, questionnaires and scenario analysis. You need to select a technique that is appropriate to the stakeholder.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A requirement is a feature that the business users need the new system to provide.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Requirements engineering activities are used to produce requirements that may be signed off as clear, unambiguous and correct.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A functional requirement is a statement of WHAT a system must do in order to meet business goals.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Non-functional requirements determine HOW WELL the functional requirements will be provided.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A use case is a ‘structured requirement’ that describes a piece of system functionality.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>An actor is an external entity performing a named role with respect to the system.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Use cases are always described from the actor’s point of view and yield a result of value to that actor.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Use cases should not be confused with business processes.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>We must always consider the human aspects of systems investigation and introducing change.</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 4

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 80 at the back of the manual.

1. What is a stakeholder?
2. Name four system investigation techniques.
3. What is the difference between a Functional and a Non-functional requirement?
4. Name four types of non-functional requirements.
5. What are the key components of a Use Case diagram?
6. List four barriers to communication.
7. What does SARAH stand for and what does it represent?

Further Reading


Module 4: Systems Investigation

Answers to Post Test 4

1. Those who have an interest in, or may be affected by, the issue under consideration. They may be internal to an organisation or operate externally to the organisation.

2. Any of Interviews, Workshops, Questionnaires, Scenario Analysis and Prototyping.

3. A Functional requirement is a statement of WHAT some ‘system’ must do in order to meet its (business) goals and Non-functional requirements define HOW the functional requirements will be provided.

4. Four from: performance, security, access, backup & recovery, archiving & retention, robustness, availability, usability, capacity.

5. System boundary, use cases, actors, and associations.

6. Language not appropriate, poorly defined objectives, too much information, distortion, wrong medium, anxiety.

7. Shock Anger Rejection Acceptance Hope, represents an individual or group response to change.
Module 5 – Methods – Structure and Content

This section introduces the structure and content of the chosen method to represent system requirements: the Unified Process and the Unified Modelling Language. We will look at three representative models from the method.

**Topics:**

In this section of the course, we will cover:

- Structure and content of a chosen representative method
  - Unified Process/Unified Modelling Language
- Roles and team structures within the method
- Products within the chosen method

**RUP/UP History**

*The Unified Process (UP) was specified by Ivar Jacobsen, Grady Booch and James Rumbaugh in their book The Unified Software Development Process. UP was derived from their work in developing a process for Rational (RUP) to accompany their work on the Unified Modelling Language (UML). RUP is a commercial version of UP, which was owned by Rational Corporation (now IBM). Our course concentrates on the standard UP version, but adopting the RUP workflows.*

**RUP/UP Origins**

*The Swedish company Ericsson did work as far back as 1967 on modelling systems as a set of interconnected blocks (subsystems/components). They also derived these from “traffic cases” – a forerunner of today’s use cases.*
The Objectory process was created in Sweden in 1987 by Ivar Jacobson as the result of his experience with Ericsson. This process became a product at his company, Objectory AB, a subsidiary of Ericsson. Centred on the concept of use case and an object-oriented design method, it rapidly gained recognition in the software industry and has been adopted and integrated by many companies worldwide. A simplified version of the Objectory process was published as a text book in 1992.

Objectory AB was acquired by Rational Software Corp. in 1995. They set about merging Objectory with various practices being developed at Rational as well as Rumbaugh's Object Modelling Technique (OMT). The result was Rational Objectory Process which became known as the Rational Unified Process (RUP) in 1998. This baselined version forms the basis of the 'generic' Unified Process as described in “The Unified Software Development Process” by Jacobsen, Booch & Rumbaugh: 1998.
Rational have since enhanced RUP as a product which provides a slightly different set of workflows and models as well as enhancements such as Tool Mentors and Templates.

UP and OO/UML

The Unified Process (UP) is a framework for software development, using Object Oriented (OO) analysis and design techniques. It is not essential for the technical environment to be OO, but it helps! The Unified Process uses the Unified Modelling Language (UML) as its standard modelling notation. UML is a modelling notation, not a methodology, and does assume an OO approach.

Unified Process

The Unified Process is a …

- Use Case driven,
  - Use cases express the functional requirements of the required system
  - The services that the product must provide, seen from the perspective of all the users of the product
  - The development approach is focused around use cases; analysis, design, implementation, testing, …

- Risk focused,
  - UP tries to de-risk the project as early as possible, by early analysis of critical use cases, early exploration of feasible architecture and focussing on executable software

- Architecture-centric,
  - Early and continuing concentration on sound architecture
  - Architecture is represented by several views in UP, each view deals with a sub-set of the whole architecture
  - Use cases are realised across the architecture, but architecture must support the required functions now and in the future!
• Iterative,
  o Repeated cycle of steps, successively adding more and more
detail or functionality

• and Incremental
  o Each iteration results in an increment, a step in the evolution
  of the final product. Each increment is of the \textit{required} quality
  and is a baseline for internal or external release

\textbf{... Approach to software development}

\textbf{Use Cases}

Use Cases express the Functional requirements of the required
system; they are the services that the product has got to provide,
seen from the perspective of all the users of the product.

The Unified Process development approach is focused around Use
Cases; analysis, design, implementation, testing ...

\textbf{Risk Focused}

Risks are events that could occur in the future that would jeopardise
the success of the Project. They should be identified and actions
prescribed to deal with them

Unified Process tries to de-risk the project as early as possible
through:

• Early analysis of critical Use Cases
• Early exploration of feasible architecture
• Focus on executable software
Architecture

Architecture describes the most significant elements in the software product, and how they relate to each other. The 'Product' is not just the final code; it should include a number of other things too.

Architecture is represented by several Views in the Unified Process: each View deals with a sub-set of the whole architecture. Architecture should reflect best practice and often uses recognised patterns.

Sound architecture will allow the product to evolve in a low cost, low risk way.
Iterative and Incremental

Iteration is about repetition of cycles of similar steps. After each cycle there should be an improvement in quality or an increase in functionality or scope.

Incrementation is about achieving the delivery of the whole product by starting with something less than the whole and adding to it over a series of deliveries. Each increment is of the required quality, is built through iteration and is a baseline for internal or external release.

Unified Process Lifecycle

UP: 4 Phases, 4 Milestones

Most people familiar with the unified process will be familiar with the four phases of a cycle: inception, elaboration, construction, and transition. There is a major milestone achievement at the end of each phase:

- Inception ★ Lifecycle Objectives
- Elaboration ★ Product Architecture
- Construction ★ Initial Operational Capability
- Transition ★ Product Release

This high level view helps reinforce the scope of each cycle.
Where the system supports a stable part of the business environment, there may be periods of system stability between cycles. In this situation new versions/cycles are usually necessary to react to substantial change, such as a change in the business or the need to re-engineer technically.

Where the system is supporting a more dynamic environment, such as a web-site or intranet development, there may be a need for almost continuous development. Some might prefer to treat this as an ongoing project that is never completed; in terms of control, it is probably better to set up a series of projects with specific objectives, etc.

**UP: Workflows**

UP has six workflows, spread across the four phases:

- Business Modelling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

Each workflow is a sub-process consisting of a set of related activities that various project workers perform to produce UP artefacts. Together the Workflows create the Software Product.

Note that in RUP, workflows are called disciplines and workers are called roles.
The Unified Process

UP: Iteration in the Phases

All phases may be composed of any number of iterations, especially in elaboration & construction. Each iteration is a mini-project with defined objectives across all workflows and each iteration delivers an increment. How many iterations in each phase depends on what is required. 1 in inception, 2 in elaboration, 3 in construction and 2 in transition is typical.
Composition of the Software ‘Product’

In UP, the product is not just the code, it should also include everything we’ve used to produce the code e.g. requirements, use cases, non-functional requirements, test cases. This is so that we can maintain the product into future cycles and releases.

UP focuses on building 6 Models to accomplish the comprehensive needs of the product: Use case, Analysis, Design, Deployment, Implementation and Test (UADDIT). Executing the workflows over the phases completes the 6 models (views) required.

**UP: 6 models – Logical and Physical**

The 6 models together represent the product’s architecture; both logical and physical. The models are described by relevant artefacts. The models are all 'empty' at the start of the project. The various workflows and activities seek to improve each model through the lifecycle; the project is complete when the models are reasonably complete.
Each model is a self-contained view of an aspect of the software product, seen from a particular perspective. By ‘self-contained model’, we mean that we do not need to refer to the other models to interpret it.

A model is not just a diagram, although a diagram is often included. A model satisfies a ‘view’, for example the ‘use case’ view uses a use case diagram + the template specs + activity diagram, etc.

The 6 models have dependencies on each other, in particular the physical realises the logical. The logical view helps users to focus on ‘WHAT’ is required and step away from the solution that will be implemented physically (the HOW and WHO), otherwise we are in danger of reproducing the current physical system.
UP has conceptual, logical and physical models:

- Conceptual: use cases and analysis
- Logical: design
- Physical: implementation (coding) and deployment

Testing should occur at all 3 levels (remember the V model!)

**Roles and Team Structures**

UP can be operated under any of the roles and team structures referred to earlier in the course.

- Traditional roles and structure
  - Using UP with more formality and pre-defined sequences
  - Each iteration is a mini-waterfall!
- Agile/XP Roles and structures
  - Agile is derived from the UP approach; Agile reflects ‘the spirit of the UP’
  - Roles and structures are adapted to an iterative, minimum formality approach

**The Unified Modelling Language (UML)**

UML is a software modelling language, which is visually orientated and is designed to be implemented by UML modelling tools. UML does NOT provide a methodology; UP is the preferred method for UML. UML models the world as systems of interacting objects. An object is a cohesive cluster of data and function.

UML provides 13 diagrams, including:

- Activity diagram to model process
- Class diagram to model data
- State machine diagram to model events
Full UML Diagram Set

There are 13 different types of UML diagrams.

UML models have at least two dimensions: graphical (to visualise the model using diagrams and icons), and textual (the specification of the modelling elements).

In the UML there are static models, which capture the things and the structural relationships between things e.g. class diagram and dynamic models, which in turn capture how things interact to generate the required behaviour of the software system e.g. state machine diagrams.

There is no specific order in which UML diagrams are created, though usually a use case diagram is drawn first to define the system scope. Diagrams are drawn in parallel, and are cross checked as
information is uncovered to ensure that the system specification is complete and consistent.

## Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Unified Process (UP) is a framework for software development, using object oriented analysis and design techniques.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>UP uses the Unified Modelling Language (UML) as its standard modelling notation.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The UP is use case driven, risk focussed, architecture-centric iterative and incremental.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>The Unified Process considers that the life of a software system can be represented as a series of development cycles, each cycle has four phases: inception, elaboration, construction and transition. There are iterations within each phase.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>There are 6 workflows spread across the four phases: business modelling, requirements, analysis &amp; design, implementation, test and deployment.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>In UP, 6 models together represent the product’s architecture, both logical and physical.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>The Unified Modelling Language (UML) is a visually oriented software modelling language, with both static and dynamic views of the system.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>The class model is a static model. Use case diagrams, activity diagrams and state machine diagrams are all dynamic models.</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 5

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 95 at the back of the manual.

1. How are UP and UML linked?
2. List the four phases of a UP development cycle.
3. Name the six UP workflows.
4. The three perspectives used to model systems are process, data and event; name a UML model for each perspective.

Further Reading

Answers to Post Test 5

1. The Unified Process is a framework for software development which uses the Unified Modelling Language as its standard modelling notation.

2. Inception, Elaboration, Construction and Transition.


Module 6 – Methods - Models

This section shows a selection of models from the chosen method that are helpful in analysing and designing the required system. Three UML models are introduced to represent process, data and event perspectives on the required software system.

Topics:

In this section of the course, we will cover three representative models from UP/UML:

- Process perspective: activity diagrams
- Data perspective: domain class diagrams
- Event perspective: state machine diagrams

Activity Diagrams

Activity Diagrams show *processing logic*. They can show triggering events, actors, tasks, decisions and process flows. Activity diagrams are mainly used for modelling business processes, and to model the flow of use cases. The same diagram type is often used by business analysts and technical architects, so communication is greatly facilitated.

Activity diagrams are widely supported in CASE tools, Visio, etc. Hand drawing/sketching is still possible though.
Notation

- Initial Node (indicates where flow starts)
- Final Node (many permitted for layout)
- Action Node (atomic)
- Activity Node – calls the activity diagram corresponding to its name
- Activity Edge or Control Flow
- Decision Node or Merge Node
- Guarded Edge (condition must be true)

Activities and Actions

An activity is a network of nodes and edges. An action is atomic behaviour i.e. it cannot be decomposed further.

Diagramming conventions use activities with nodes and edges.

Nodes are Initial, Action or Final and edges are the path of control, shown by arrows.
Example

We can use a simple activity diagram to illustrate its usage. Let us imagine our student is a young male living in a bedsit. The activities undertaken during his day are shown in the activity diagram below. After finishing his start of day activity, the student must decide whether or not he needs to attend lectures. If the decision is no, he drinks all day and falls asleep to end the process. If the decision is yes, he undertakes the sub-process of attending lectures, which will have its own activity diagram. When ‘attend lectures’ is completed, the student drinks and sleeps.

Example – Start Day Activity Diagram

Here is the breakdown of the ‘Start day’ activity: It starts with the ‘get up’ action. Naturally this might be a protracted affair involving, scratching, yawning, groaning but we can use the ‘description’ box in the action’s properties to describe the action in as much detail as we want. If the ‘get up’ action is too complex to describe easily, then perhaps we should break it down into; ‘wake up’, ‘perform getting up rituals’, and ‘actually get out of bed’ actions. If it’s even more
complex than that, then we might make ‘get up’ an activity that will be described by its own activity diagram.

After performing his ablutions (which we will not break down further here), the student undertakes two actions at once, in that he reads the paper whilst eating his beans. The fork and join symbols allow us to show parallel paths. Note that they do not have to happen at exactly the same time, but the process cannot continue until both of them are complete.
Partitions

Bands that partition an activity diagram (aka swimlanes) can represent departments, locations, roles or systems. We can assign activities and actions to specific partitions to indicate where responsibilities lie.

Further Notation

- **Send Signal Node**: asynchronous – doesn’t wait for the response
- **Receive Signal Node**: wake up the action (an event and alternative way of initiating an action)
- **Flow Final Node**: terminates its own path but not the whole activity
- **Time Event**

Send Signals are signals or messages sent to entities external to the activity. Signals are sent asynchronously, meaning the activity does not wait for the response, but moves on to the next action after the
signal is sent. The recipient will probably do something in response, but that isn't modelled on the activity diagram.

A Receive Signal captures an event generated outside the activity. If the receive is active, the activity is expecting the signal to arrive.

A receive signal can trigger an activity, and replace the start node.

**Example - Use of Sending and Receiving Signals**

It is likely that the customer is outside the scope of the process and therefore not within the control of the process. There is little point in modelling aspects that are out of scope. Therefore the interaction with the customer would be better modelled using send and receive nodes.
Flow Final Nodes

In this diagram we have 2 independent flows – which one will finish first? If we use the activity final node, then whichever flow finishes first will terminate the whole activity, even if the other flow hasn’t finished. The answer is to use a flow final node, which terminates its flow, but not the whole activity.

Time Events

*Time events* are a special form of receive signal. Sometimes time is a factor in an activity. For example, it may be necessary to model a wait period, such as waiting 3 days after shipping an order before sending the invoice, or perhaps kick off an activity at a regular interval, such as the monthly reading of a utility meter. Time events are drawn with an hour glass symbol.

The text next to the hour glass – *wait 3 days* – shows the amount of time to wait. The incoming edge to the time event means that once the time event is activated, the invoice is sent after a three day wait.
A time event with no incoming edge is a recurring time event, meaning it is activated with the frequency of the text next to the hour glass. There is no initial node necessary here because a time event is an alternate way to start an activity.

The time event symbol can be used together with a synchronisation bar. The diagram shows that although payments are raised, they will not be sent until after 3pm.
Activity Diagrams in support of Use Case Descriptions

Instead of a textual flow (as below) an activity diagram could be used. Alternatives would be represented with a decision node.

<table>
<thead>
<tr>
<th>Member</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Selects ‘Borrow Item’</td>
<td>Prompts for member’s reference.</td>
</tr>
<tr>
<td>2. Enters the member’s reference into the system.</td>
<td>Retrieves matching member’s details. Requests confirmation.</td>
</tr>
<tr>
<td>3. Confirms member.</td>
<td></td>
</tr>
<tr>
<td>4. For each item being borrowed:</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Prompts for item reference.</td>
</tr>
<tr>
<td>7. Confirms item to be loaned.</td>
<td>Creates loan record. Marks item as ‘out on loan’. Prompts to continue.</td>
</tr>
<tr>
<td>8. Selects continue or completed.</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Use case ends.</td>
</tr>
</tbody>
</table>
Class Diagrams

A clear understanding of the meaning and structure of the required data is at the heart of effective requirements analysis and systems design. This is true of projects both large and small, and in all technical environments.

There are two common standards in use for modelling data: the ‘Entity Relation Diagram’ from structured systems analysis, and the ‘Class Diagram’ from the unified modelling language.

A class model defines what data is required, how it is structured, what it means and the integrity rules about that data. It is a model of the business domain’s data, defining:

- The classes – types of things of interest to the business
- The associations between classes
• The attributes of the classes

The class model is a logical model of the business data. That is, it models the data requirements independent of:

• The technology that will be used to store the data
• The ways in which individual processes require to view the data
• Any performance constraints

A class diagram documents the data requirements of the business domain, providing a basis for the design of flexible, robust database structures which enables the “sharing” of data and controls redundancy so data is collected, recorded, and stored once and once only.
A Class is...

...anything about which information needs to be recorded.

So in a garage business some classes would be:

- People and organisations like: supplier, customer, and employee
- Things like: vehicle model, vehicle, and part
- Events like: sale, service
- Places like: garage (assuming they have several)

Class

A class is shown as a simple, rectangular container with its name in the uppermost of its two compartments. The name should be a singular noun phrase. If the name has more than one word, by convention, the first letter of each word should be capitalised and spaces should be omitted.

Example names are

- Customer
- QuoteType
- BankAccount
Objects and Classes

Objects can be considered as real world things e.g. a car, a computer keyboard etc. A class is a template (blueprint) for objects; objects are instances of classes.

At any moment in time, the object’s attributes have values that may determine the state of the object.

Attributes are...

....the things we need to know about the class.

Attributes are shown in the second compartment of the class, as you can see below.
Attribute names are normally shown with a lower case first letter and all subsequent words in the name having uppercase first letters. Attributes may show visibility. This refers to whether the attribute can be accessed by an operation from another class or not. Attributes are usually private (‘-‘ sign).

If attributes are private, their values can only be accessed by operations of the same class. So if you wish to know, for example, a customer’s address, you must use the operation of the customer class that is designed to access that piece of data. You can use the operation because it is public (‘+‘ sign). The benefit of this arrangement has to do with the OO principle of encapsulation, which in turn reduces dependencies and promotes maintainability. Operations are shown in the third compartment.

Note that it is acceptable to show classes with one, two or three compartments.

**An Association is...**

...a logical meaningful relationship between two classes.

Here is one possible association between the employee and the vehicle classes. For example, it will support the requirement to know, for a given salesperson, which vehicles they sold; and for a given vehicle, which salesperson sold it. Associations need to be named...
precisely to reflect the business rules they are supporting. We only include associations that are direct and of interest to the business.

**Association**

An association is shown as a simple line between two classes. To enhance meaning, it is often labelled and given an arrow to help understand the direction of the description so, for example, we know that the customer owns the vehicle, rather than the vehicle owning the customer!
Multiplicity

Numbers at each end of an association indicate how many instances of one class may be associated with each instance of the other class. This is shown as a range (..) from minimum to maximum. Note that the asterisk (*) means unspecified ‘many’ in UML.

Examples of multiplicities include:
- 0..1 (zero to one)
- 0..10
- 0..* or * - both mean the same
- 1..*
- 1..1 or 1 – both mean “one and only one”

Here is an example of the use of multiplicities:

This means that each Customer owns zero to (possibly) many Vehicles and that each Vehicle is owned by zero or one Customer.

You may well ask how a vehicle can be “owned” by zero customers, but consider those vehicles that have not yet been purchased. A Class model must be ‘true’ at all moments in time – i.e. it is static and timeless.

Showing Requirements

The question now is, “How do we take our requirements and construct a data model that supports them?” Consider the following requirements:
- The system is to record, for a given employee, which vehicles they sold as new
• The system is to record details of all services booked for each vehicle
• The system is to record which vehicles are currently owned by which customers

You can see that there are four key “things” here:
• Customer
• Vehicle
• Service
• Employee

This will give us four classes. You must consider then for each class, which other classes are associated with it, and think about the appropriate multiplicity. See the suggested solution below:
Cross-checking

Recall that use cases are built from **functional** requirements. Do all the use cases actually *do* something? Is all the data used? Does the system process data items that are never created? Can data be deleted? We often use the CRUD Matrix to ensure that we know which use cases create, read, update and/or delete each class.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Customer</td>
</tr>
<tr>
<td>Take Payment</td>
<td></td>
</tr>
<tr>
<td>Record Rental</td>
<td></td>
</tr>
<tr>
<td>Record Sale</td>
<td></td>
</tr>
<tr>
<td>Manage Customer</td>
<td></td>
</tr>
<tr>
<td>Add Vehicle</td>
<td></td>
</tr>
<tr>
<td>Record Service</td>
<td>R</td>
</tr>
</tbody>
</table>

**Housekeeping Use Cases**

Often we may find that we are missing a use case whose job is, for example, to delete obsolete service records (in our garage scenario). This might simply be because it never came up during interviews and the analyst didn’t think of it at the time. It may be that a completely different system is responsible for this but **at least we will have checked**. *The CRUD Matrix helps us do this.*
Use cases like this are often called “housekeeping” use cases, and often do not appear in the project until some form of validation is performed.


State Machine Diagrams

This diagram models all of the states that a typical object from a domain class may assume during its ‘lifetime’. For each state, the model shows all the events that can legally occur for an object in that state. Modelling the object’s lifecycle allows us to specify more robust systems, and ensures that all required use cases have been identified by cross-referencing the identified events to the use case diagram. These diagrams are also known as ‘state diagrams’ or ‘state charts’.

UML State Notation

- **Start State**
- **Transition – must be labelled**
- **State**
- **End State**

**Employed**
Transitions and Events

A transition moves an object from one state to another.

The event causing the transition is written on it. Transitions indicate the potential of moving from the first to the second state.

Events are generated by use cases or by time passing, ‘after’ and ‘when’ are keywords for time events, e.g. after (50 years).

Example – States of Interest

The DVLA maintain records of all vehicles in the UK. The events of interest for the class ‘Car’ might be:

- First registration
- Pay VED
- Declare off road (SORNed)
- Write off

Example - Events

The resulting ‘state’ of a Car’s record after undergoing each event:

<table>
<thead>
<tr>
<th>Event</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>First registration</td>
<td>Car becomes</td>
</tr>
<tr>
<td>Pay VED</td>
<td>Car becomes</td>
</tr>
<tr>
<td>Declare off road</td>
<td>Car becomes</td>
</tr>
<tr>
<td>Write off</td>
<td>Car becomes</td>
</tr>
</tbody>
</table>

Registered  VED paid  Taxed
The state machine diagram captures business rules. It determines the constraints on the modification of the object and the actions that must be carried out as a result of attempts to modify the object. These rules must eventually be implemented.

Not all classes will need a state machine diagram, only those that have significant changes of state. For others we just check we know the event cycle for create, update & delete.
More Notation

Some events may leave the object in the same state, or return an object to a previous state. The keywords *after* and *when* are available to indicate time events.
The U-curve

The modelling of physical and logical is known as the U-curve process pattern. Although it originated in early systems development methods, it is still applicable today.

**Logical and Physical Aspects of a System**

Each of the models we have looked at can be used to document the current system, representing WHAT the system does as well as HOW the system works, to produce a physical model.

Models that show WHAT the system does are logical models.

Physical and Logical models can be produced to show the current (as-is) and the proposed (to-be) system.

The typical order is:

- Current physical
- Current logical
- Proposed logical
- Proposed physical

If there is no current system then you start with the proposed logical system.
Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Activity diagrams show processing logic. They are mainly used for modelling business processes and to model the flow of use cases.</td>
<td>✔</td>
</tr>
<tr>
<td>2</td>
<td>Activity diagrams can show triggering events, actors, tasks, decisions and process flows.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The class diagram is a model of the business domain’s data, which defines the classes, their attributes and the associations between them.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A class is shown as a simple, rectangular container with its name in the uppermost of its compartments.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Attributes are the things we need to know about the class.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>An association is a meaningful relationship between classes. It has a name and direction to enhance meaning.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Multiplicities are the number of possible instances at each end of a relationship.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>The CRUD Matrix affords the analyst a useful way to check that the functionality of the use case diagram is complete.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>State machine diagrams model all of the states that a typical object from a class may assume, and for each state, all the events that can legally occur for an object in that state.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>State machine diagrams document business rules.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>The U-curve is used to explain how models can be used to show the physical and logical aspects of a system.</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 6

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 122 at the back of the manual.

1. What does the activity diagram model do?
2. What are the conventions to be applied to class names?
3. What do we call the ‘numbers’ at each end of an association between two classes?
4. What is the purpose of the CRUD Matrix?
5. What do we depict on a state machine diagram?

Further Reading


Answers to Post Test 6

1. An activity diagram shows processing logic.

2. The name should be a noun phrase and start with a capital letter. Names consisting of multiple words should not be separated by spaces and they too, must have initial capitals.


4. The CRUD Matrix affords the analyst a useful way to check:
   a. The effect of each UC on each Domain Class is understood.
   b. The set of Use Cases is complete

5. State machine diagrams model all of the states that a typical object from a Domain Class may assume during its lifecycle, and for each state, all the events that can legally occur for an object in that state.
Module 7 – Quality Assurance

This section covers techniques that are used in the systems development lifecycle to ensure the quality of the IT system.

**Topics:**

In this section of the course, we will cover:

- Definitions of quality
- Requirements driven testing
- Types of walkthrough and inspection
- Post project reviews
- Service Level Agreements

**What is Quality?**
What is Quality?

“Quality is the totality of features and characteristics of a product or service which bear on its ability to satisfy a given need.”

BS4778

Quality Assurance Objectives

To see that all the right things are being done:

- In the right order
- By the right people
- To professional standards
- In a controlled manner

Getting reasonable assurance about the quality of the product is one of the reasons it is important to have a defined approach for the system development lifecycle, like UP.

In UP, testing, including document reviews, is contained in its own workflow.
Module 7: Quality Assurance

Quality Assurance Techniques

The techniques that we shall look at are:

- Requirements driven testing
- Walkthroughs and inspections
- Post project reviews
- Service Level Agreements

Requirements Driven Testing: What is Testing?

"The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are *fit for purpose* and to *detect defects*"

ISTQB®, 2010
Why is Testing Necessary?

- Software systems are everywhere
  - Business applications (e.g. banking, e-commerce)
  - Consumer products (e.g. cars, washing machines)
- Failed software may result in:
  - Financial loss (e.g. business unable to sell product, or customers leave)
  - Wasted time (e.g. time to fix and recover from failure)
  - Loss of reputation (e.g. bad press or negative market reaction)
  - Injury or death (e.g. safety control system failure)
- … Or may have an insignificant effect
- Testing reduces the risk of software failure
  - And ensures software meets user needs

Causes of Software Defects

Error

- A human action that produces an incorrect result
  - e.g. A typo by a developer or an ambiguous requirement by a business analyst
- May introduce a…

Defect

- A flaw in a system that can cause it to fail to perform its required function (also known as a fault or bug)
  - e.g. An incorrect statement in program code or document
- May result in a…

Failure

- Deviation of a system from its expected delivery, service or result
  - e.g. The system crashes, gives incorrect result or performs too slowly
- May be caused by data, hardware or environmental conditions (such as radiation, magnetic fields or pollution) as well as defects
Fundamental Test Process (ISTQB)

The ISTQB identifies a set of activities it calls the Fundamental Test Process. This consists of:

- Test planning and control
  - Includes knowing the Test Basis
- Test analysis and design
  - Designing the test cases
- Test implementation and execution
  - Setting up and running the test
- Evaluating exit criteria and reporting
  - Assessing if enough testing has been done, reporting results
- Test closure activities
  - Closing the testing activity

Testing effort must be prioritised – it is impossible to test everything, so a risk-based approach is required. What/where are the risks in using this software?

**Testing must be Risk-based**

Because it is virtually impossible to test everything, testing must normally be targeted, based on risk. We need to identify what the main risk areas are in using this software, and use testing to mitigate these risks.

Testing may find defects but, even when these are fixed, we cannot guarantee there are no further defects in the product. Also software could be released with ‘known defects’ if they are low risk and the business is made aware of them.

Professional testers will do a better job of testing than developers, because they have a different mentality, are trained in techniques, and offer an independent pair of eyes.

Testing requires as much planning and support as all project activities, especially as testing is often ‘squeezed’ as budget and time constraints get challenging.
Requirements Driven Testing

This is an approach to testing in which we base our testing on each requirement in the test basis. Using UML, functional requirements are expressed as Use Cases, so the test basis is the set of Use Cases expressed in the Use Case Diagram and elaborated using suitable templates. We need to test the non-functional requirements as well.

This type of testing is called Black-box Testing since it only tests the external use of the product not the interior design. White-box Testing may also be necessary, for example testing the predicted execution of the code. White-box Testing is outside of the scope of this course.

Each Use Case must be converted to a set of Test Cases. This is fairly easy to do since Use Cases are already specified as a Main Flow plus a number of alternatives and exceptions.

Each Test Case covers a number of Test Conditions, in line with the logic of the Use Case. Test Conditions are statements of what should be tested. To make a Test Case we examine each suitable test condition, based on the Use Case description, to determine the input data, the data expected to be on the system and the predicted result if the test is successful.
The Unified Process (Reminder)

Remember that the Test workflow cuts across all four phases.

Test Conditions and Test Cases

A test condition is defined as “... an item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element” (ISTQB Glossary)

A test case is defined as “... a set of input values, execution pre-conditions, expected results and execution post-conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.” (ISTQB Glossary)

Type of Test Data

INPUT DATA
- Input required to test the condition
- E.g. Customer code, Product code

BASE DATA
- Stored data
• E.g. Customer and Product details

**Use Case to Test Case Example**

**Use Case: Register New User**

*Pre-condition: Primary Actor has Administrator privileges*

<table>
<thead>
<tr>
<th></th>
<th>Administrator</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Selects ‘Register New User’.</td>
<td>Shows ‘Register User’ screen.</td>
</tr>
<tr>
<td>2</td>
<td>Enters User Details: (new) User ID, (existing) User Profile.</td>
<td>Validates entries.</td>
</tr>
<tr>
<td>3</td>
<td>Confirms registration.</td>
<td>Creates User with privileges in accordance with User Profile. Shows transaction confirmation screen.</td>
</tr>
<tr>
<td>4</td>
<td>OK.</td>
<td>End Use Case.</td>
</tr>
</tbody>
</table>

*Post-condition: User created with correct privileges*

*Alternatives:*
A1. User details are invalid
A2. Administrator does not confirm registration and quits

Such Use Case descriptions may be supported by Activity Diagram which would assist the tester in deriving Test Cases. See the following activity diagram below.
Finding Test Cases

Test Cases readily appear from the main flow (happy day) and the alternatives, e.g.:

- TC1: Main Flow
- TC2: User Id and User profile valid + Administrator quits
- TC3: User Id invalid + Administrator quits
- TC4: User Id valid and User profile invalid + Administrator quits

Consider which input and base data to use and remember to make it specific.
Designing the Test Case (TC1) for the Main Flow

We can design a test case for the Main Flow using a table like the one below. Note that this covers a number of test conditions (things that should be tested), which we think are significant for the acceptability of the function.

<table>
<thead>
<tr>
<th>UC Step</th>
<th>Test Condition “test the ...”</th>
<th>Input Data</th>
<th>Base Data</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Selection of the ‘Register New User’ option</td>
<td>Option selection</td>
<td>n/a</td>
<td>‘Register User’ screen appears.</td>
</tr>
<tr>
<td>2.</td>
<td>Input of a (new) valid User ID</td>
<td>User ID = ‘GB1234’</td>
<td>User ID ‘GB1234’ is not there</td>
<td>Input accepted by the system.</td>
</tr>
<tr>
<td>2.</td>
<td>Input of a (existing) valid User Profile</td>
<td>User Profile ID = ‘GuruGroup’</td>
<td>User Profile ‘GuruGroup’ exists</td>
<td>Input accepted by the system.</td>
</tr>
<tr>
<td>3.</td>
<td>Confirmation of data entered</td>
<td>Confirmation signalled</td>
<td>n/a</td>
<td>New user registered with privileges in accordance with their User Profile. Transaction confirmation screen displayed.</td>
</tr>
<tr>
<td>4.</td>
<td>Selection of OK</td>
<td>OK selected</td>
<td>n/a</td>
<td>Return to previous function.</td>
</tr>
</tbody>
</table>
Designing the Test Case (TC3) for the alternate flow (A1)

Then we can design test cases for the Alternate Flows, to test the system copes with invalid data.

<table>
<thead>
<tr>
<th>UC Step</th>
<th>Test Condition “test the ...”</th>
<th>Input Data</th>
<th>Base Data</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Selection of the ‘Register New User’ option</td>
<td>Option selection</td>
<td>n/a</td>
<td>‘Register User’ screen appears.</td>
</tr>
<tr>
<td>2.</td>
<td>Input of a invalid User ID</td>
<td>User ID = ‘GBooch’</td>
<td></td>
<td>Display error message ‘User ID invalid format’</td>
</tr>
</tbody>
</table>

Test Procedures Support the Testing

Test procedures are created to support each Test Case (or group of Test Cases). The procedures should include instructions for:

- any grouping of test cases
- initiating the test run/session
- executing each test case
- recording the test results
- terminating the test run/session
- inspecting the results

Testing in Agile

Testing in Agile takes the form of a number of ‘acceptance criteria’ annotated at the back of the User Story.

These notes are there for the guidance of the developer so that they know what the function should and shouldn’t do. The tests are there to be able to determine when the User Story is done.

In fact this approach is often called ‘Test-driven requirements engineering’, as the functionality is being driven out by understanding the behaviour of the function through the named tests.
Testing NFRs

It is of course vital to test NFRs as well as FRs (Use Cases). Test Conditions could be set up at any stage of the Test Case to test NFRs. Considering the level of risk is especially important for NFRs.

However a number of common NFRs are difficult if not impossible to test prior to release, for example Availability, Performance and Reliability.

In order to get the quality assurances we need therefore, such NFRs should be the subject of SLAs signed with the operations organisation.

Types of Walkthrough and Inspections

There are several different types of review that may be carried out on the work products of the systems development. The reviews vary in the level of formality.
Types of Review

Informal reviews

- No formal process, documentation optional
  - Useful and cheap, but not as effective as formal review

Walkthroughs

- Scenarios, dry runs, peer group reviews led by author

Technical Reviews

- Documented, defined fault-detection process
- Includes peers and technical experts
- Moderator ideal but not mandatory

Inspections

- Formal process based on rules, checklists and entry/exit criteria
- Led by trained moderator
- Defined roles

Inspection is the most formal review type. The less formal reviews will also reduce the cost of subsequent testing and are essential to do before passing to quality assurance for formal review. However, informal reviews are insufficient if they are the only checking activity, as there is a much lower probability of errors being found in an informal process. Sometimes there may be a combination of reviews carried out, e.g. informal followed by technical.

Informal Review Characteristics

With this type of review there is no formal process. It is the least expensive method of review and even an informal review is better than none! Making a record of findings is optional.

Examples are:

- ‘Buddy reviews’
- Desk check
• Technical lead reviewing designs and code

When people see the benefits of the informal reviews, such as reduced number of failures, they may be willing to invest time in more formal types and start recording the lessons learnt.

**Walkthrough Characteristics**

Walkthroughs contain formal elements but, unlike inspection, the meeting is led by the author. During the meeting, scenarios and dry runs of the product take place. These are less formal, more ‘open-ended’ sessions. Optionally there will be pre-meeting preparation by reviewers, a review report, a list of findings and use of a scribe. However the scribe should not be the author. The main purposes of walkthroughs are: learning, gaining understanding and defect finding.

**Technical Review Characteristics**

This type of review is aimed at getting the best out of a design, so it may include roles like technical architect. A technical review has some formal elements (guidelines, checklists etc.) and some informal elements, as it may be done as a peer review rather than a specially called meeting.

A technical review is a documented, defined defect-detection process that includes peers and technical experts. It may be performed as a peer review without management participation. Ideally these reviews are led by a trained moderator (not the author). There will be pre-meeting preparation by the reviewers, along with the option of the use of checklists, a review report or a list of findings.

Technical reviews vary in practice from quite informal to very formal. The main purposes are to discuss the product, make decisions, evaluate alternatives, find defects, solve technical problems and check conformance to specifications and standards.
Inspection Activities

An inspection is a formal process based on rules, checklists and entry/exit criteria, led by a trained moderator and with defined roles for those taking part in the inspection. The stages of inspection are:

- **Planning**: select the personnel, allocate roles; define the entry and exit criteria and select which parts of documents to look at
- **Kick-off**: distribute the documents, explaining the objectives and checking entry criteria
- **Individual preparation**: before the review meeting individuals note potential defects, questions and comments
- **Review meeting**: discussion or logging, with documented results or minutes
- **Rework**: fixing defects found – this is typically done by the author
- **Follow-up**: checking that defects have been addressed, gathering metrics and checking on exit criteria

Roles and Responsibilities

The **Moderator/Leader** is responsible for managing all the review team activities. Duties include: planning, motivating, facilitating meetings, training checkers and monitoring the whole review process. The moderator should be trained in the review process. They set the agenda; control the pace during the inspection and record summary results for quality management and follow-up action.

The **Author** prepares the material to be reviewed and actively participates in the review process, describing the product being reviewed during the inspection meeting. The author provides summaries of the document(s) under inspection, rather than reading it verbatim. After the inspection the author incorporates agreed changes. The author should not serve as moderator.

**Reviewers** find defects in the product under review. They should include stakeholders in the artefact. Reviewers may be assigned different areas of focus e.g. conformance to standards.
The **Manager** may not participate directly in the review, but ensures that time is planned for all those involved, or preparation will not be thorough enough. The manager decides whether review objectives have been met.

The **Scribe** documents all the issues, problems and action points that were identified during the meeting.

Note that these are roles, not job titles: one person could play a number of roles within the process, particularly the leader who may serve as scribe and reviewer too. All participants act as reviewers.

**Review Rules**

Whichever type of review is adopted, the rule must apply that the product is assumed to be guilty and the author always innocent!

**Post Project Reviews**

The objectives of the post project review are:

- To review the development project
- To document lessons learnt during the project
- To document problem areas
- To prepare a Post Project Report

Ideally, this review is carried out by the project team; all those who have been involved in the development (Analysts, PM and Developers, for example).

In addition an independent / impartial person (moderator) may chair the review.

**When?**

The post project review should take place as soon as possible after the project has gone live, before the team has dispersed.
It is essential to **plan** for the review early in the lifecycle, during implementation planning, and to **organise** the people involved to **gather** the input during the project.

Information gathered should include details on the…

- Approach – systems lifecycle and practice
- Techniques – process models, data models etc.
- Communications – internal and external
- Technology – use of software tools
- Project Management - planning, monitoring and controls
- Estimating – quality of estimates and other metrics
- Risks and issues - identified and resolved
- User involvement - sufficient and effective

…used during the project.

The review will **evaluate** the information gathered, compare the project plan against the actual project, and identify **problems** and **opportunities**.
Post Project Review Report

A report is produced to document the findings and lessons learnt in the review, and should include a plan to action the lessons.

It is important to get stakeholder agreement on the report and then to action the report, following change control procedures. Action must be taken to:

- Realise the benefits of the new system
- Avoid ‘reinventing the wheel’
- Improve estimates and plans
- Re-use what worked
- Avoid what didn’t work
- Correct any mistakes

Therefore the Post Project Review report should be circulated to other teams. Finally the post implementation review is planned.

Service Level Agreements

A service level agreement (SLA) is a document that describes the relationship between two parties: the service provider and the service user. This is clearly an extremely important item of documentation for both parties. If used properly it should:

- Identify and define the user’s needs
- Provide a framework for understanding
- Simplify complex issues
- Reduce areas of conflict
- Encourage dialogue in the event of disputes
- Eliminate unrealistic expectations

Specifically it should address a number of factors; amongst these are usually the following:

- Services to be delivered
- Performance, tracking and reporting
- Problem management
- Legal compliance and resolution of disputes
• Customer duties and responsibilities
• Security
• Intellectual Property Rights (IPR) and confidential Information
• Termination

Service elements are the most critical section of the SLA as it describes the services and the manner in which they are to be delivered. The information on the services must be accurate and contain detailed specifications of exactly what is being delivered. The services provided come from the agreed requirements.

Service Level Agreements Example

Service
The Vehicle Registration System (VRS) will provide the DVLA with accurate, up-to-date information on all vehicles currently registered in the UK.

Service Level
1. VRS will be available 08.00-1800 Monday to Saturday excluding statutory holidays
2. Time to restore service less than 4 hours for each reported fault
3. Provide an average 2 second response time for vehicle detail enquiries during peak periods 09.00-10.00 and 17.00-18.00, 1 second at all other times
4. Provide an average 5 second response time for vehicle detail updates during peak periods 09.00-10.00 and 17.00-18.00, 2.5 seconds at all other times

Measurement
Performance of agreed service levels will be calculated over any 6 consecutive months.
Use of recognised SDLC frameworks

The use of a recognised SDLC framework like UP is also a quality assurance technique. The SDLC encapsulates best practices and is based on use and experience in many situations. SDLC often include in their framework the use of many techniques and practices which will tend to improve the quality of the end products, including the methods and models you have seen on this course.
## Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Quality is the totality of features and characteristics of a product or service that bear on its ability to satisfy a given need.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Testing, walkthroughs, inspections and post project reviews are quality assurance techniques.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Testing ensures that the software conforms to the specification.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Inspections are a formal process based on rules, checklists and entry/exit criteria, led by a trained moderator, with defined roles for the participants.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Walkthroughs are a less formal process, led by the author, e.g. dry runs using scenarios.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Technical reviews are a documented, defined fault-detection process.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Informal reviews are useful and cheap, but not as effective as a formal review.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A post project review is carried out as soon possible after the project goes live to review the project, document lessons learnt during the project, any problems areas and to produce a report, which must be actioned.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A Service Level Agreement (SLA) is a formal document that identifies what is to be delivered and the agreed quality level in objective and measurable terms.</td>
<td></td>
</tr>
</tbody>
</table>
Module 7: Quality Assurance

Post Test 7

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 146 at the back of the manual.

1. What is meant by software quality?
2. Name three quality assurance techniques.
3. What is requirements driven testing?
4. What are the roles involved in an inspection?
5. What is the least effective form for review?
6. When is the post project review carried out?
7. What should a Service Level Agreement include?

Further Reading


Answers to Post Test 7

1. Quality is the totality of features and characteristics of a product or service which bear on its ability to satisfy a given need.

2. Testing, Inspections, Walkthroughs, Post-project review.

3. An approach to testing. Acceptance criteria are generated for each requirement, test conditions are created for each criterion, and test cases are produced for each test condition.


5. An informal review.

6. As soon as possible after the project goes live, before the team disperses.

7. The services elements and management elements.
Module 8 – System Design, Deployment and Maintenance

This section briefly considers the other major activities that occur within the System Development Lifecycle.

- Security is always a major concern in software development these days, so design must take this into account.

- Moving beyond analysis, which uses for example the models we have been reviewing, the development continues with design activity, including the User Interface design (UI). Design is covered in more depth in the BCS SD Diploma module Systems Design Techniques.

- Once complete and tested, the software must be packaged and moved into the live environment. This is the critical stage, and close support of the user environment is generally needed.

- Once deployed and stabilised, the software must be supported and maintained, and may be the subject of further system development lifecycles.

Topics:

In this section of the course, we will cover:

- Controls and security
- Verification and validation
- Interface design
- Design principles and constraints
- Different types of implementation
- Sign off and deployment
- Post implementation reviews
- Different types of maintenance and support
Controls and Security

Businessdictionary.com defines a control as a “device or mechanism installed or instituted to guide or regulate the activities or operation of an apparatus, machine, person, or system.” The analyst works with audit, user management and user staff to determine and document the need for security and control measures identified below. Part of this will be a risk assessment, as the cost of security and control measures must be in proportion to the risks associated with the system under consideration.

- Security requirements including physical and logical security, and identification/authentication methods
- Contingency planning, Backup and recovery requirements and methods. For databases roll-back and roll-forward recovery are options to cater for
- Audit trails to record who has accessed the system and what operations he/she has performed during a given period of time (forensics). Audit trails are useful for identifying security breaches and for recovering lost transactions
- Legislative and ethical constraints including Intellectual Property Rights (IPR), government regulators, Data Protection and Freedom of Information Acts, ‘Big Brother’ issues such as monitoring internet usage

Security

Security is about “the protection of assets.” The data that is held and processed on our systems is a vital business asset and so systems need to be protected from threat events, such as:

- Unauthorised access, physical and logical
- Inappropriate alteration of data
- System failure, such as being unavailable

The types of protective measures that we can put in place are:

- Prevention - measures to prevent assets being lost or damaged
- Detection - measures to inform about damage to an asset, the nature of the damage and what caused it
• Reaction - measures by which assets can be recovered or repaired

ISO 27000 is the gold standard for IT security.

ISO 27000 series

ISO/IEC 27000 series defines an Information Security Management System (ISMS)

• ISO 27001: spec for an ISMS, i.e. what is meant by ISMS
• ISO 27002: code of practice, include best practice controls
• ISO 27003: implementation guidance
• ISO 27004: measurement and metrics
• ISO 27005: Information Security Risk Management
• ISO 27006: accreditation guidelines for ISMS certifiers
• ISO 2700x: ... there will be others in the series

Essential aim is to ensure CIA - Confidentiality, Integrity and Availability of Information

Physical Security

Prevention is better than cure! Examples of physical security include:

• Secure physical access, e.g. locks, barrier, safes
• Environmental controls
• Uninterrupted power
• Contingency planning
• Backup and recovery
• Security personnel

Contingency Planning

Actions must be planned to react to disasters. Contingency planning is generally referred to as Business Disaster Recovery and
Continuity plans. Businesses will have Hot, Warm and Cold Sites. The actions in the plans must be rehearsed to ensure they work.

**Backup and Recovery**

“Backup” - A security copy of a file or database for use in the event of loss or damage to the original. The term is most commonly used to refer to a copy of critical files on a computer's discs, which is made periodically and kept on removable medium, stored securely.

“Recover” or “Restore” - To bring back the backup copy so that it replaces a file or database which has been lost or damaged.

It is important to classify the data that we hold to ensure we are not wasting expensive storage space with irrelevant data. “Critical” data is essential to the business; a key business function(s) would stop without this data. “Needed” data will be used over the long term but is not immediately required to keep the business running. “Not important” data is the stuff that the business could function without.

**Logical Security**

Examples of logical security measures include:

- Anti-virus, spyware and intrusion software
- Penetration testing
- Encryption
- Digital Signature
- Forensics
- https, Firewalls and Demilitarised Zone (DMZ)
- Application Access Security

**Application Access Security**

- *Identification* - announce who you are (e.g. User Name)
- *Authentication* - prove that you are who you claim to be
  - Up to 3 factors: something you ‘know’ (password), something you ‘possess’ (ID Card), something you ‘are’ (fingerprint)
- *Authorisation* – what ‘rights’ do you have
- *Access* – grant access only in accordance with the authorised rights
Audit Trails/Forensics

An audit trail is a record showing who has accessed a computer system and what operations they performed during a given period of time. It is useful for:

- Identifying unauthorised or fraudulent activity
- Checking integrity of system data by tracing transactions
- Recovering lost transactions
- Providing a record for compliance

Most accounting systems and database management systems include an audit trail component.

In addition, there are separate audit trail software products that enable network administrators to monitor use of network resources.

Legislative and Ethical Controls

Legislation and regulation are common in most industries today. Software controls must often be put in place to reduce the risk of breaking the law. Fines for breaching the regulations may be heavy, with loss of reputation a risk here too.

It is important for companies to make personnel aware of the extent to which their actions are being monitored by the system. Companies need certain controls and audits, but they should be transparent and open.

Verification and Validation of Data in Design

Software Design has to be conscious of the need to \textit{validate} and \textit{verify} data input. An information system depends upon the quality of its basic transaction data capture, so a lot of attention should be paid to this.
Verification of Input Data

Verification is a process whereby we can try to assess whether the data is ‘true’ (correct, accurate).

Examples of verification techniques are:

- Double entry: e.g. re-typing an email address
- Build in verification steps: e.g. send an email to the entered address, such that the process can only proceed if that occurs successfully
- Cross-reference to other data to see if the data entered seems feasible and get user verification: e.g. if the user is aged 11, does she really earn £100K per annum?
- Confirmation: get users to confirm ‘drastic’ actions like deletes etc.
- Self-checking codes: check digits for critical codes e.g. account numbers

Validation of Data

Validation is a process whereby we can assess whether the data input is correct according to standards that have been set. There are many forms of validation, some examples are:

- Existence checks: a matching record exists in a reference file
- Format Checks: e.g. an NI number must conform to a certain format
- Length Check: e.g. minimum and maximum digits allowed
- Lookup Table: e.g. a code must coincide with a list in a table
- Range Check: the data’s value has a minimum and maximum, e.g. date of birth
- Cross-field validation: between related data items

Unfortunately data passing validation doesn’t guarantee that the data is ‘true’. For example “The Cat resembled false confidence” is a valid English sentence, but it doesn’t make much sense!
Design for Validation and Verification

Input

- Any data input by the user should be validated as much as possible
- The amount of data required from user freeform input should be minimised
  - Use codes, lists, radio buttons, check boxes etc.
- Long data entry processes should be broken up and checked in stages
- Error messaging should be clear, relevant and HELPFUL

Output

- Data output should undergo a series of sanity checks
  - e.g. Avoid sending a final demand for a bill of -£0.01!
- Thorough testing of boundary conditions
- Scrutiny and audit by (real) people

Interface Design

Human Computer Interaction (HCI)

Many, if not most, applications require human interaction, supplying Input and interpreting Output. Unfortunately (!?) people and IT have very different information processing characteristics.

The Human-Computer Interface (HCI) deals with the most effective way to bridge this difference. For most users the HCI is the application, and the range of technical possibilities is increasing all the time, with technologists developing interface solutions in all directions.

Poor design may lead to inefficient and ineffective use of the application even if it is technically sound.

HCI is a major field of study in its own right, with psychology and ergonomics having a major role here.
Interface Technology Options in Design

- Screen, form and document design
- Input capture solutions
- Output technologies

Design Principles

Software design is as much art as it is science – here’s some useful design principles proposed by David Hooker, who compiled this list from a number of famous developers’ thoughts:

- The 1st Principle: The Reason It All Exists; *is … to provide value to the stakeholders*
- The 2nd Principle: KISS; *Keep It Simple, Stupid!*
- The 3rd Principle: Maintain the Vision; *all good software is based on a clear vision*
- The 4th Principle: What You Produce, Others Will Consume; *someone else will have to understand what you have done*
- The 5th Principle: Be Open to the Future; *never design yourself into a corner – keep asking "what if?"
- The 6th Principle: Plan Ahead for Reuse; *reuse saves time, effort and risk and contributes to agility*
• The 7th Principle: Think!; *no amount of methods, techniques and methodology is a substitute for thinking*

Adherence to these principle, and basing design decisions upon them, will contribute significantly to producing a successful and durable product in the context of enterprise software applications.

**Design Constraints**

Of course, we do not usually have a totally free rein when designing an IT system. Any design we produce must accommodate any constraints we have been given, for example:

- Legal e.g. Data protection, Financial Conduct Authority, Employment law
- Ethical e.g. personal data usage, tracking
- Financial e.g. affordability, budget

**Sign-off and Deployment**

Preparation for going live is essential to ensure the success of the system. The steps in the process include considerations for:

- Sign-off from the project team into the live environment
- Configuration, Change and Release Management (CCRM), according to corporate standards
- User Training may be required
- Data conversion may be required
- Business readiness activity
- The software may be part of a wider change
- Range of handover methods
Different Types of Implementation Approach

A decision should be made early in the lifecycle regarding the type of implementation suitable for deploying the IT system into the live environment. The four main choices, which may be combined, are:

- **Pilot**: the whole system is delivered to a small group of users e.g. at a branch, while the old system is running elsewhere. Generally followed by a gradual roll out to the rest of the users.
- **Parallel running**: the old and new systems run in parallel until an agreed date. Generally the old system remains ‘live’.
- **Phased**: phased delivery of functionality i.e. some functionality given to all users. This based on there being distinct ‘modules’ defined. The modules are rolled out in a prescribed order. The term ‘phased’ may also be used to indicate an approach where software is rolled out incrementally on a geographic basis.
- **Big Bang**: turn off the old system and turn on the new

**Handover Considerations**

The appropriate choice will depend upon a balance of cost, risk, time and resources. We are aiming for high confidence, minimum disruption for the business, and a good fall-back position.
Deciding the Approach

We need to weigh up the advantages and disadvantages of each approach and decide which will be the most appropriate for our particular project.

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Consider when ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Big Bang</strong></td>
<td>Obtain benefits quickly. Usually the cheapest approach.</td>
<td>High risk, especially in greenfield sites.</td>
<td>No other way. Inescapable deadlines. Small changes.</td>
</tr>
<tr>
<td><strong>Pilot</strong></td>
<td>Low risk approach, allows problems to be resolved before full deployment. Easy to roll back.</td>
<td>Delays benefits. Not always possible. Are pilot sites really representative?</td>
<td>Greenfield software, or combined with process change.</td>
</tr>
<tr>
<td><strong>Phased</strong></td>
<td>Concentrates resources. Minimises disruption.</td>
<td>Not always possible. Might require special interfaces.</td>
<td>Software that is clearly modular with little coupling elsewhere.</td>
</tr>
</tbody>
</table>

Post Implementation Review

A post implementation review of the system is carried out sometime after the system has gone live (and has had time to ‘settle’) in order to validate that we have delivered what the user needs, and answered the question: ‘Did we meet those objectives and constraints as set in the business case?’

The objectives are:

- To review the newly implemented system
- To evaluate it against its original objectives including costs and benefits
- To document problem areas
- To prepare a Post Implementation Report

Ideally this review is initiated by the project sponsor and will be carried out by the users, representatives of the project team, and
audit staff. In addition, an independent / impartial person (moderator) may chair the review.

**When**

There is no ‘rule’ when the review should take place, but typically the system should be reviewed 6 months after it has gone live, and:

- Give time for the system to settle
- Start planning early

<table>
<thead>
<tr>
<th>J</th>
<th>F</th>
<th>M</th>
<th>A</th>
<th>M</th>
<th>J</th>
<th>J</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

*Go live*  *PIR*

**Approach to the post implementation review**

It is essential to **plan** for the review as soon as the IT system has gone live. The business case is an important input to the review. Information **gathered** should include:

- Incident reports
- Error logs
- Performance figures
- Support requirements

The review will **evaluate** this information and determine whether the objectives defined in the business case were met and whether there are any changes to the requirements. A comparison of actual costs and benefits against predicted costs and benefits is carried out. A report is produced to document the findings, **identify** any problems or opportunities, including whether anything needs to be done to realise the benefits. The **report** should include an action plan.
Types of Maintenance and Support

It has been estimated that 80% of the development effort that goes into a system over its lifetime is maintenance, usually performed outside of the System Development Lifecycle environment. This maintenance could fall under any of the following categories:

- Corrective - fixing something that is broken
- Perfective - making improvements, e.g. for improved efficiency
- Preventative - making changes to prevent a future problem
- Adaptive - adapting software for a different use

Different levels of support are put in place to respond to issues that occur once the system has gone ‘live’. If the first level of support is unable to resolve the issue, it will be escalated to higher, more technical or specific support staff. It is important that any changes made to resolve the issue are controlled appropriately.
Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the design stage we consider controls and security requirements.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Verification is a process to assess whether data input is ‘true’ (correct, accurate).</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Validation is a process to assess whether data input is correct according to standards that have been set.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Design of forms, documents and interfaces must meet the users’ needs.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Keep design simple and use a consistent, standard format, following the natural sequence of steps carried out by the business. Ease of use will reduce errors.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>System design must incorporate legal, ethical and financial design constraints.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>The different approaches to implementation are: pilot, phased, parallel running and big bang.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Sign-off and deployment should not be a continuation of testing.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Handover considerations must balance risk, cost, time and resources against high confidence in the handover, minimum disruption to ‘business as usual’ and a secure fall back system.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A post implementation review of the system is carried out sometime after the system has gone live and has had time to 'settle', in order to validate that we have delivered what the user needs.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>A system may be subject to corrective, perfective, adaptive and preventative maintenance during its lifetime.</td>
<td></td>
</tr>
</tbody>
</table>
Support will be put in place for the ‘live’ system to respond to issues that occur once the system has gone ‘live’.

Post Test 8

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 162 at the back of the manual.

1. Give four examples of controls and security that should be considered in the design stage.
2. Give examples of three input devices and three output devices.
3. What is the difference between a pilot and a phased approach to implementation?
4. When is the post implementation review carried out?
5. Name four different types of maintenance that may be carried out on a system.
Answers to Post Test 8

1. Risk assessment, backup and recovery, audit trails, contingency planning, legislative and ethical controls.

2. Input devices: barcode scanner, mouse, keyboard and three output devices: printer, screen, and microfilm/fiche.

3. A pilot approach gives all the system functionality to a small group of users whereas phased approach gives some of the functionality to all of the users.

4. Sometime after implementation when the system has had time to 'settle in'.

5. Perfective, preventative, adaptive, corrective.
Module 9 – CASE, CAST, Application Lifecycle Management Tools

This section introduces the range of software tools available to assist the analyst, including Computer Aided Software Engineering (CASE) and Computer Aided Software Testing (CAST), and identifies the stages of the system development lifecycle where the tools may be used.

Topics:

We will cover:
- Software support for systems development
- Features of CASE and CAST tools
- Features of Application Lifecycle Management
- Lifecycle coverage
- Configuration and version control

Software Support for Systems Development

Claims made by suppliers are often exaggerated, but most will provide a mix of the following features:
- Diagramming tools
- Diagram validation
- Automatic generation of first-cut low-level diagrams
- Report production
- Code generation

But CASE tools do not carry out analysis and design for the systems analyst; instead they provide support by helping with diagram creation and amendment, and by providing an element of diagram consistency checking.
On small projects CASE tools may have a negative impact on productivity, because of the effort involved in inputting required information.

**Support for Team Development**

CASE tools enable the parallel development of systems models by allowing decomposition of models into controlled units. Integration with version control systems ensures that team members consistently work on the latest version of a model or document. Integration with requirements tracking systems enables forwards traceability from requirement to model, and backwards traceability, from model or document to originating requirement.

Publishing the models produced by the CASE tool on the web provides increased developer communication. The use of standard designs aids communication, maintenance and facilitates re-use.

As models are held within a central repository, branching and merging of development streams is also supported.

**Features of CASE and CAST tools**

CASE is the use of a computer-assisted method to organise and control the development of software, especially on large, complex projects involving many software components and people. Using CASE allows designers, code writers, testers, planners, and managers to share a common view of where a project stands at each stage of development. CASE helps ensure a disciplined, check-pointed process. A CASE tool may portray progress (or lack of it) graphically. It may also serve as a repository for, or be linked to, document and program libraries containing the project's business plans, design requirements, design specifications, detailed code specifications; the code units, test cases and results, and marketing and service plans.

**Common CASE Tool Features**

The shared repository (database) for all models, provided by the CASE tool, enables user and group management of access to the models. The tool also provides support for:
Module 9: CASE, CAST, Application Lifecycle Management Tools

- Traceability and change impact analysis
- Configuration management and version control
- Reporting (mostly web-based)
- Code generation

Traceability enables analysts and developers to set up and track relationships between requirements, to verify that high-level requirements are represented in the detailed software requirement specifications and models and to determine the impact of any changes to the system.

Using CASE tools has the following:

Advantages
- Shared repository (database) for all models
- Easier and quicker to create and change models
- More professional and readable results
- Support for traceability and change impact analysis
- Support for configuration management and version control

Disadvantages
- Cost of software
- Time to train analysts in use of CASE

CAST Features

CAST is a software product that supports one or more test activities, such as planning and control, specification, building initial files and data, test execution and test analysis. (ISTQB Glossary). The software supports testing throughout the lifecycle and is likely to include features for:

- Test management
  - Enables test managers to plan, review, evaluate and report on testing programme
- Test design
  - To design test scenarios and test data from requirements
- Automated testing
  - Enables automatic execution of repeatable tests
- Test analysis
To provide feedback on defects to developers, and identify improvements to test designs.

Using CAST tools has the following:

Advantages
- Testing process more efficient and effective
- Reduces the cost of testing
- More predictable and consistent results
- Objective results
- Provides test management information

Disadvantages
- Initial investment and ongoing costs
- Detailed analysis required to implement tool
- Effort to maintain tool

Lifecycle Coverage

All aspects of the software development lifecycle can be supported by software tools, and so the use of tools from across the spectrum can, arguably, be described as CASE: from project management software through tools for business and functional analysis, system design, code storage, compilers, translation tools, test software and so on.

Examples:
- Project management tools e.g. Microsoft Project
- Computer Aided Requirements Engineering (CARE) tools e.g. IBM’s Requisite Pro
- Computer Aided Software Engineering (CASE) tools e.g. Visual Paradigm for UML
- Integrated Development Environments (IDE) also known as workbenches e.g. Microsoft Visual Studio
- Computer Aided Software Testing (CAST) tools e.g. Quality Centre
Configuration and Version Control

Configuration management is concerned with controlling any changes made to project deliverables, e.g. documents, models, in order to ensure that the changes are made in a disciplined manner and traceability is sustained.

There are two key areas in configuration management:

- **Configuration identification**: identify the deliverables to be brought under configuration management and decide the identifier and version numbering scheme to be used
- **Configuration control**: agree a process for controlling changes to the deliverables

Configuration management is essential throughout the system development lifecycle, from requirements gathering through to developed code. Software tools (CARE, CASE and CAST) provide features to:

- Identify configurations
- Apply version and change control
- Enable traceability and impact analysis

Application Lifecycle Management Tools

Application Lifecycle Management (ALM) is a continuous process of managing the life of an application through governance, development and maintenance.

ALM exploits tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management.
Summary

The key points of this chapter are as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>Subject</th>
<th>Prepared?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software tools are available to provide support throughout the system development lifecycle</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Computer Aided Software Engineering (CASE) tools support the development of software.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Computer Aided Software Testing (CAST) tools support one or more testing activities.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>These tools enable traceability; backwards from the code and forwards from the requirements.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Application Lifecycle Management tools support the continuous process of managing the life of an application through governance, development and maintenance.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Software tools exist to support the whole system development lifecycle</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Features of CARE, CASE and CAST support configuration and version control, which are essential throughout the system development lifecycle</td>
<td></td>
</tr>
</tbody>
</table>
Post Test 9

To reinforce the materials we have just covered, try out the following questions in your own time. You’ll find the answers on page 170 at the back of the manual.

1. Give examples of four types of tools that will support the system development lifecycle.
2. What are the advantages of using CASE tools?
3. What is meant by traceability?
4. What is Application Lifecycle Management?

Further Reading

Answers to Post Test 9

1. CASE, CAST, CARE, IDE.

2. Shared repository (database) for all models, easier and quicker to create and change models, more professional and readable results, support for traceability and change impact analysis, support for configuration management and version control.

3. To set up and track relationships between requirements, to verify that high-level requirements are represented in the detailed software requirement specifications and models.

4. Application Lifecycle Management is a continuous process of managing the life of an application through governance, development and maintenance.
Examination Hints and Tips

Note that these notes are for guidance in approaching the written exam only.

- The exam is about applying the SDE syllabus techniques to a business scenario
- The exam lasts for 75 minutes in total: 15 minutes reading time to look at the scenario, questions and reference materials; 60 minutes to answer the questions
- The key to answering the questions is to apply the techniques and come up with specific answers based on the scenario
- No credit is earned for generic answers and answers copied from the course notes
- The paper is worth 50 marks and the pass mark is 25 (50%)

Reading Period (15 mins)

- Valid reading materials include the examination paper scenario and questions, and your course notes (including any notes you make during the course). Use this time wisely to understand the questions and the scenario
- Read the questions first and understand what you are being asked. Be aware of the number of marks available for each question
- Read the scenario to pick up ideas for answers on your first pass through
- Make use of your reference materials but note that no writing, marking, highlighting or annotating is allowed during this reading period

Writing Period (60 Mins)

- Manage your time effectively. Try to earn a “mark a minute”, i.e. spend 10 minutes answering a 10 mark question; this will allow 10 minutes at the end to check your answers
Examination Hints and Tips

• Answer only what you have been asked for, nothing more. If you are asked for a specific number of examples, give that number (no more). If you are asked to explain your answer, give explanations; otherwise don’t waste time giving something that is not required.
• Do not rewrite the question!
• For each of the techniques, look at the number of marks at stake and how many components requested to give you an idea of how much time to spend coming up with the answer.
• Play the percentages – the pass mark is 50%. 100% correct answers for each question are almost impossible.
• Write as neatly as you can – it makes marking easier, and the marker cannot give you marks if your answer is illegible.
• It is “Open Book” – open your books and look at the examples, but don’t copy them.

Case Study

The Case Study Scenario will be between one and one-and-a-half pages long and will include a general description of an organisation, its operations, a business need and an outline project to meet that need, as well as some requirements for change. These requirements, for an IT application, may be presented as several paragraphs of prose or as numbered lists.

When you refer to a requirement in your answers it may be enough to give the requirement’s number (if it has one).

However, if the requirement needs to be broken down, you will need to indicate which part of the requirement you are referring to; the best way of doing this is by quoting the relevant part of the requirement verbatim.

Important Note

Our markers regularly see papers in which candidates have lost marks (sometimes many marks, sometimes making the difference between pass and fail) through either failing to read the questions or failing to follow the instructions. READ THE QUESTIONS CAREFULLY AND DO WHAT THEY ASK OF YOU.
Examinable Topics

While all areas of the syllabus are in theory examinable, the reality is that you will not be asked to define anything, as this is included in your notes. Below is a list of the examinable techniques followed by details on how to approach a question on that technique. Note that you are unlikely to be asked more than one question in each topic area, so if you have a class diagram question you won’t also have a state machine diagram question.

- Systems Development Lifecycles and Development Practices
  - Lifecycles
  - Development practices
- System Investigation
  - Stakeholders
  - Investigation techniques
  - Functional requirements definition
  - Non-functional requirements definition
- Methods
  - Activity diagrams
  - Class diagrams
  - State machine diagrams
- Quality Assurance
  - Test acceptance criteria, conditions, cases
  - Service Level Agreements
- System Design, Deployment and Maintenance
  - Handover approaches

Other topic areas will not be included in the written exam, but must be revised in preparation for the (closed book) diploma oral exam.

The scenario will give details of an organisation, and a project they are undertaking. All of the questions relate to the scenario and all of the answers must too.

Lifecycles and Approaches

Recommend a lifecycle and/or development practice for the project described and give reasons for your recommendation. You may be asked to say why you have not recommended the alternatives.
Stakeholders and Investigation Techniques

Identify, from the scenario, a number of business stakeholders for the project who you would need to gather information from and say which investigation technique you would recommend. You may also be asked why you recommend this technique and/or what information you would hope to gather. Only give the number asked for.

Requirements

Identify a number of functional and non-functional system requirements from the scenario. Typically you will be asked to categorise the non-functional requirements. Only give the number asked for.

Diagrams

Review an activity diagram, class diagram or state machine diagram to look for notational errors or mismatches with the given requirements or scenario. Only give the number of errors asked for.

Testing

From a requirement expressed as a Use Case, construct a Test Case identifying a number of test conditions with input, base data and expected results.

Service Level Agreements

Identify a number of services and associated SLAs. Only give the number asked for. Ensure that the SLAs relate to the service and that they are reasonable in context of the scenario.

Handover Approaches

Recommend a handover approach for the project described and give reasons in support of your recommendation. You may be asked to say why you have not recommended the alternative handover approaches.
Index

Activity Diagrams, 93, 96
Actors, 11, 17
Agile, 18, 19, 20, 32, 40, 42, 45, 47, 50, 52, 91
Application Lifecycle
  Management, 163, 167, 168, 169, 170
Architecture, 23, 27, 29, 30, 83, 86, 89
Association, 109, 110
Attributes, 108, 109, 120
BCS SDE Syllabus, 5
Bespoke, 32, 45
Business Activity Model
  Benefits of, 102
Business Analyst, 13, 17, 19, 20, 21
CASE, 6, 96, 163, 164, 166, 168, 169, 170
CAST, 6, 163, 165, 166, 168, 170
Class Diagram, 105
Class Diagrams
  CRUD, 113
Communication barriers, 75
Context Diagram, 67
COTS, 34, 43, 50, 52, 59
Deployment, 87, 89, 91, 93, 95, 155
Developer, 16, 21
Development Practices, 43
DSDM, 40, 46, 50
Evolutionary, 46
Functional, 53, 62, 63, 67, 79, 80, 83
Handover, 15, 160
Implementation, 15, 34, 87, 89, 91, 93, 95, 141, 156, 157, 158
Incremental, 32, 36
Inspection, 136, 137, 138
Interviewing, 53, 58
iteration, 39, 41, 84, 88, 91, 42, 50, 52, 84
Logical and physical aspects, 119
Maintenance, 159
MoSCoW, 40
Multiplicity, 111
Non-functional, 62, 64, 78, 79, 80, 89
Objectives, 86, 124
Objects, 108
Performance Change Curve, 76
Phased, 36, 156
Post Test 1, 20, 21
Post Test 2, 30, 31
Post Test 3, 52
Post Test 4, 79, 80
Post Test 5, 94, 95
Post Test 6, 121, 122
Post Test 7, 145, 146
Post Test 8, 161, 162
Post Test 9, 169, 170
project, 13, 16, 17, 34, 35, 40, 41, 42, 45, 46, 50, 56, 83, 87, 88, 89, 101, 114, 123, 125, 139, 140, 144, 146, 157, 164, 166, 167
Project Manager, 17, 21, 139
Prototyping, 32, 43, 46, 53, 58, 80
Index

Quality, 6, 56, 124, 125, 136, 144, 145, 146, 166
Questionnaires, 53, 59, 80
Requirements
  Definition of, 61
Requirements Engineering, 78, 166
Review, 16, 34, 134, 136, 137, 138, 139, 141, 145, 157, 158
SARAH, 76
Scenario Analysis, 60, 80
Security, 148
Service Level Agreements, 123, 141, 142
SSADM, 34
Stakeholder, 54
Stakeholders, 54, 55, 78
State Machine Diagram, 115
Support, 18, 77, 104, 159, 161, 163, 164, 165
syllabus, 6
Systems Analyst, 14, 15, 16, 17, 19, 20, 21
team, 17, 18, 20, 55, 138, 139, 146, 157, 164
Technical Architect, 16, 18, 21
Tester, 17, 21
Testing, 15, 91, 125, 128, 144, 145, 146, 166, 168, 169
TOGAF, 26, 27, 29, 31
U-curve, 119
UML, 41, 51, 81, 83, 91, 92, 93, 94, 96, 111, 115, 121, 166
Unified Process, 41, 51, 81, 82, 83, 86, 88, 93, 95, 121
Use Case, 53, 67, 69, 79, 83, 90, 93, 104, 113
Validation, 151, 160
Verification, 147, 151, 160
V-model, 34, 35
Waterfall, 32, 33, 34, 35, 42, 50, 52
workshop, 55, 56, 57, 69
Zachman, 29