13 DELIVERING THE REQUIREMENTS

Debra Paul and James Cadle

INTRODUCTION

Once the requirements have been defined, attention shifts to considering how they will be delivered. The business analysis work could provide the basis for a large scale, broad scope business change programme or could be much more focused on a particular area. As a result, delivering the requirements could include some or all of:

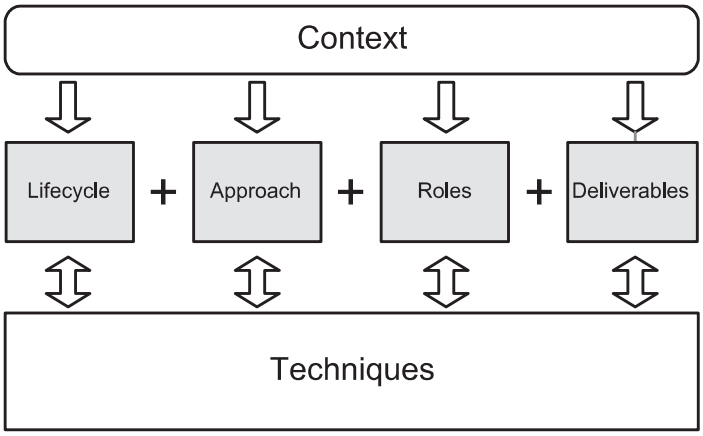
- **Business process changes**: the business processes may be altered to be simpler, faster, more accurate or more effective.
- **People changes**: the jobs of the people involved may be redefined and new tasks and skills added. In turn, this gives rise to further requirements for training, coaching or mentoring, and to changes in the way people are recruited and appraised.
- **Changes to organisational structure**: often, posts disappear, new ones are created and sections or departments are merged or split. Sometimes, introducing a new structure (projects, for example) is seen as key to the business improvements sought.
- **Changes to the IT systems**: the software that supports the business processes may need to be enhanced or replaced in order to provide the features and information used by the organisation.

These elements must not be considered in isolation and it is important that the business analyst keeps their interdependence in mind. A holistic approach to the delivery of the requirements is essential for the successful delivery of the changes to the business.

DELIVERING THE SOLUTION

There are several lifecycles, methods and standards that may be used when developing solutions to meet the requirements. In deciding how to deliver the requirements it is important to consider a number of factors and these are summarised in Figure 13.1.

Figure 13.1 Factors in deciding the delivery approach



The elements are concerned with the following areas:

- **Context**: the nature of the organisation and project that will provide the basis for deciding how the solution will be delivered.
- **Lifecycle**: the process adopted for developing and implementing the solution.
- **Approach**: the methods and standards that are used during the lifecycle.
- **Roles**: the key roles to be filled during the project.
- **Deliverables**: the products to be delivered by the project team.
- **Techniques**: the management and development techniques used to plan, analyse and document the project work.

These areas are described in further detail in the following sections.

CONTEXT

The context for the organisation and the project will provide the basis for deciding how any business changes are to be delivered. Organisations are subject to external pressures from customers, competitors and other business demands and these usually determine the need for change and the constraints within which those changes need to be met. For example, if an organisation works in a safety-critical or highly regulated environment it may be essential for the entire set of change requirements to be defined in great detail and implemented in one group. Alternatively, if an organisation operates in a highly competitive business environment then it is likely that a rapid response to change will be required.

The key factors to take into account are as follows:

- **The culture and underlying philosophy of the organisation**: here, we can consider questions such as what type of organisation is this, what is the nature

of the business domain within which it operates, what are the values and beliefs of the senior managers?

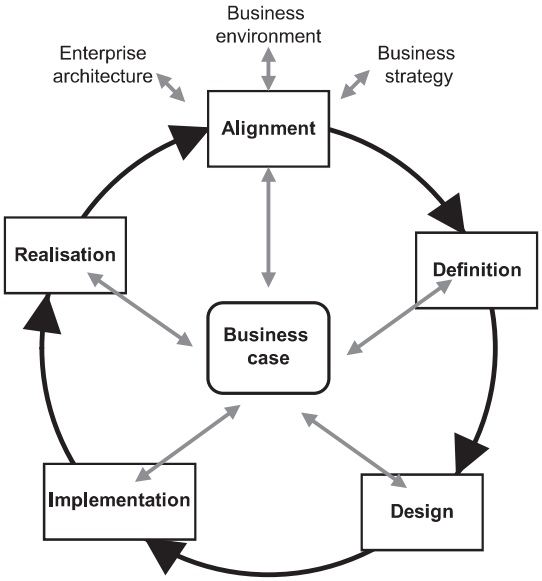
- **The business context for the proposed changes**: for example, what is the organisation hoping to achieve in terms of business benefit as a result of this project?
- **Any constraints that impinge upon the project**: for example, what is the timescale for delivering the solution, what is the budget, what resources are to be made available, what standards does the organisation use?
- **The prioritised needs of the business**: for example, improved public image may be more important than cost savings or alternatively, reducing costs may be the top priority.
- **The drivers for the project**: for example, is this project based upon a need to comply with new legislation or is it concerned with offering additional or enhanced services to customers?

The context will vary from project to project or across change programmes. Whichever is the case, it is important that this is understood in order to determine that we adopt the most appropriate approach to deliver the requirements.

DELIVERY LIFECYCLE

The business change lifecycle defined in Chapter 1 and reproduced in Figure 13.2 shows in overview the sequence of stages to be carried out when analysing, developing and delivering business changes.

Figure 13.2 The business change lifecycle



However, while this lifecycle shows in overview the areas of business change activity needed, it does not indicate *how* the solution is developed and delivered. There has been a great deal of effort devoted to defining lifecycles for IT systems development and it is important to understand the nature of these lifecycles as they provide a clear basis for the project work. Although they focus on software development, it is important to align this work with the broader business changes, such as the definition of new processes and revised job roles. The systems development lifecycles are described below.

The concept of a systems development lifecycle

The development approaches to IT systems are known as systems development lifecycles (SDLCs). These approaches have evolved over many years and are based upon underpinning philosophies that determine the route taken from analysis through to deployment of the system solutions.

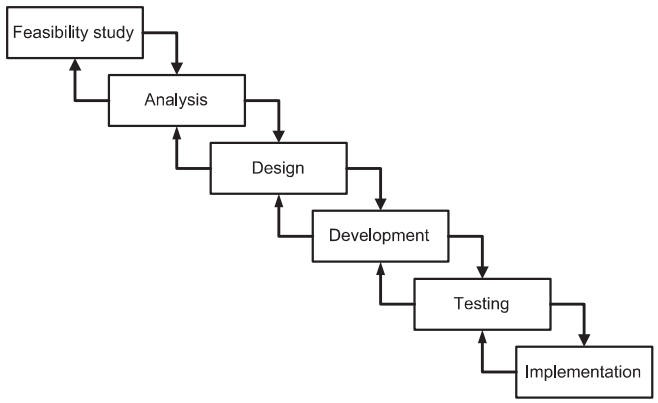
A systems development lifecycle sets out the stages, and their sequence, that are undertaken in the process to develop and implement an IT system. The lifecycle covers the entire life of a system from feasibility study to operation. There are two main philosophies that underpin systems development lifecycles: the linear approach, whereby steps are carried out in sequence and are completed before moving on; and the evolutionary approach, whereby the solution to be delivered emerges during iterative periods of software development based upon the use of prototyping and multi-disciplinary teams.

The earliest established model of a linear SDLC is the waterfall lifecycle but, over the years, variants and alternatives have been devised and each has advantages and disadvantages.

The waterfall lifecycle

The waterfall lifecycle, illustrated in Figure 13.3, shows the development proceeding through a series of sequential stages. In theory, each is reviewed and signed off before the next starts. Thus, we should only begin analysis once a feasibility study

Figure 13.3 The waterfall lifecycle



has been approved, and design should be based on an agreed set of requirements from the analysis. The backwards-facing arrows in the lifecycle indicate the need to check back at each stage to ensure that the project has not expanded its defined scope, that the present stage builds logically on its predecessor and that modifications to the deliverables from the previous stage may be made where required.

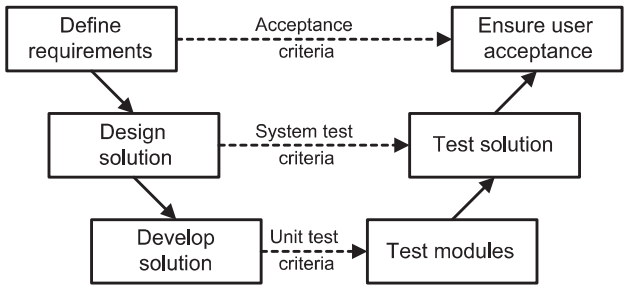
BAs would typically be involved in the feasibility study and analysis stages of the project. They would also probably assist in the user acceptance that forms part of the testing stage and help the business users during implementation. During design and development, BAs would be available to answer queries about the business and solution requirements raised by the development team.

The principal benefit of the waterfall lifecycle is that it provides good control from a project management perspective. In addition, the requirement to sign off each stage before the next one starts should lead to a high-quality system. However, the highly structured nature of this lifecycle is also its weakness as it can lead to a long-drawn-out development that can leave the business in limbo. In addition, it does not handle change particularly well since a whole sequence of deliverables may require adjustments to adapt to the change; for example, a change that occurs during development would require adjustments to the analysis and design documentation and possibly to the feasibility study as well.

The 'V' model

The 'V' model, shown in Figure 13.4, is a variant of the waterfall model and consists of similar stages and sequence of the work.

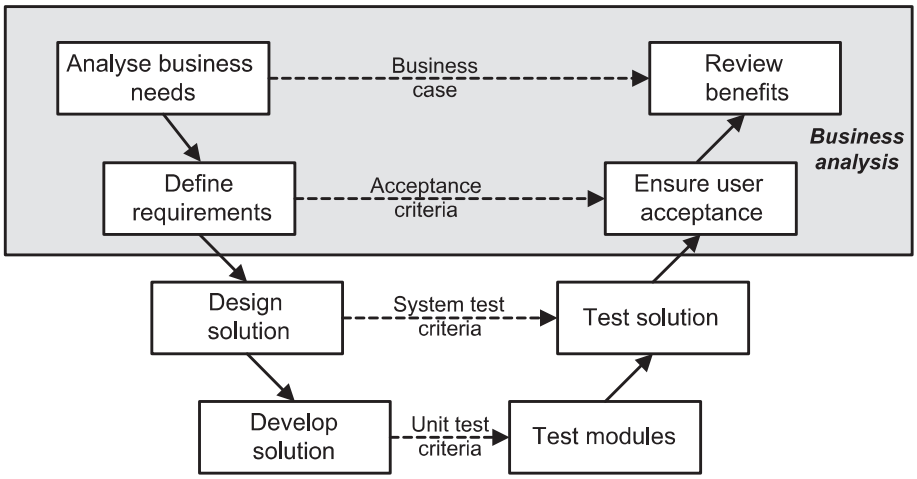
Figure 13.4 The 'V' model



While the 'V' model is based upon the same principles as the waterfall lifecycle, in effect the waterfall has been bent back on itself following the development activity. This adds another dimension, showing explicitly the connection between the earlier – developmental stages of the project and the later – testing – stages. In particular, the derivation and usage of the test criteria is made explicit at each stage, for example, showing how the acceptance criteria should be defined as part of the requirements documentation and used during user acceptance testing.

An extended 'V' model has been developed to show the context and nature of business analysis work. This model is shown in Figure 13.5.

Figure 13.5 Extended 'V' model



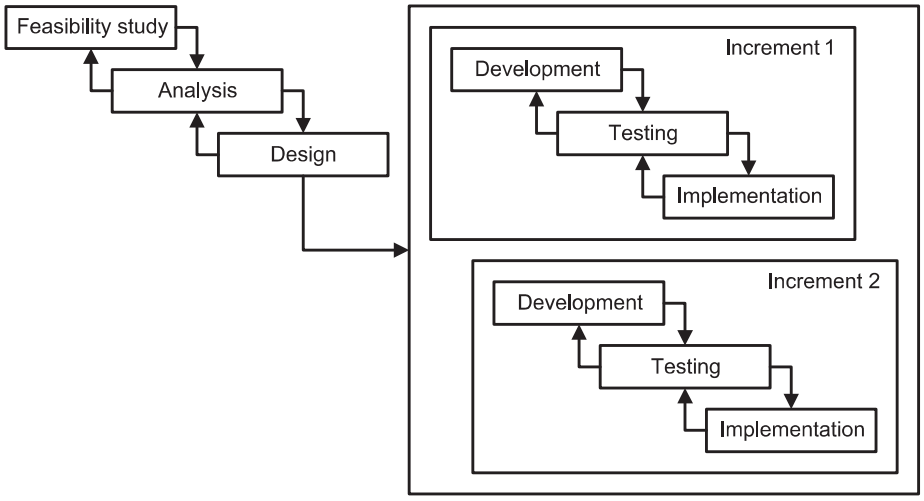
In this model, the initial work concerns an analysis of the business needs and the development of the business case that justifies the recommended solution. This solution sets out the scope of the business change work to be explored further in the requirements definition stage. Again the corresponding leg of the 'V' model shows how the deliverables from the initial stages are used in testing the solution. The business case is used during the benefits review to test the success of the recommended solution by reviewing the predicted business benefits against those delivered by the new business system. The aim is to confirm that the benefits have been achieved, or 'realised', and to identify further actions if this has not yet happened. This model is also used to show the range of business analysis work in the light of an IT solution lifecycle. As can be seen in Figure 13.5, the business analyst is involved at the outset in assessing the business needs and defining the requirements. The later activities of ensuring user acceptance and reviewing the benefits also form part of the business analyst role. Business analysis is not shown within the other stages of the lifecycle as they focus on the design, development and testing of the IT element of the solution. The design, development and testing stages are primarily the remit of technical architects, developers and software testers. However, the business analyst should still be involved during these stages for the following reasons:

- to ensure that the business needs continue to be met;
- to develop the process and job role definitions that will be implemented alongside the IT solution;
- to assess the impact of proposed changes.

Incremental delivery

When analysing the requirements for the business change solution, some requirements will be more important, or more urgent, than others. For example, a regulatory deadline or an expected move by a competitor may make it imperative to implement some requirements quickly whereas others may be less urgent and can be delivered at a later point. One way of achieving this is to develop and deliver the solution in a series of increments. The incremental delivery lifecycle is illustrated in Figure 13.6 and shows the analysis and design for the solution being completed initially. This is then followed by two incremental delivery phases, each consisting of development, testing and implementation. The most pressing requirements are in increment 1 and the rest follow in increment 2.

Figure 13.6 Incremental lifecycle



Something to be remembered if using the incremental lifecycle is that the total cost of delivering the solution is likely to be higher than delivering the solution in one release. This is because of the need to ensure that each increment will not compromise the system when it is implemented. Also, in the second and subsequent increments, it will be necessary to carry out a specific form of testing known as regression testing, to make sure that the additional features do not cause something already implemented to stop working. What is more, the high-level analysis and design for the solution needs to be defined in order to future-proof the design for the later increments. If this is not done, aspects of the solution may be discovered during later increments that will necessitate changes to parts of the system already implemented. This lifecycle shows both incremental development and delivery while utilising the structure of sequential stages defined in the waterfall lifecycle. The need for a complete set of requirements and an overall design is one of the fundamental

differences between the incremental lifecycle, described here, and the iterative approach which is described next.

Iterative systems development

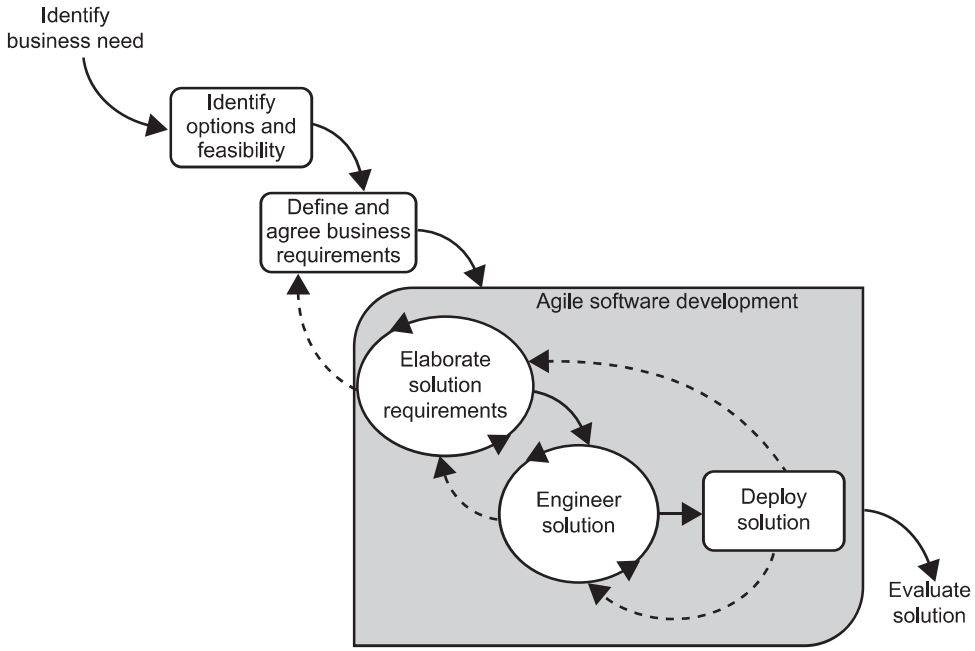
A feature common to the waterfall, 'V' and incremental models is that a complete set of requirements is gathered at the start of the project and these form the basis for all subsequent work. An alternative approach is sometimes represented by a spiral model which was originally devised by Barry Boehm in his article: 'A spiral model of software development and enhancement' (Boehm 1986). Boehm's original perception of the spiral model incorporated the concept of iterations (or 'rounds' as Boehm called them) and used prototyping to reduce risk when developing software for complex situations. This developed into the basis for early Agile approaches such as Rapid Application Development. With the advent of Agile methods such as DSDM and Scrum, iterative development has increasingly used prototyping and related techniques in order to evolve the detailed requirements and the corresponding software features.

The iterative development approach is founded upon some fundamental generic principles:

- **Evolutionary** – Detailed requirements are evolved through a series of iterative prototyping development stages.
- **Empowerment and collaboration** – It is a fundamental requirement that the team, including business staff and IT developers, are empowered to make decisions during the software development and work as a collaborative team.
- **Fitness for purpose** – The deliverables are required to be fit for purpose rather than aligned slavishly with a set of defined requirements.
- **Testing all the time** – Testing is an integral part of the iterative development approach so the software should be tested continuously. Automated testing tools are very useful in doing this.
- **Re-factoring** – Iterative development adopts an exploratory approach so any work may be reversed if it does not add value. This is not perceived as being a mistake or waste of time but instead as an integral part of the iterative process.
- **Incremental delivery** – The system is likely to be deployed in increments, with each subsequent increment providing additional functionality or improved performance.
- **Prioritisation** – An approach such as MoSCoW (see Chapter 11) is used to identify the different levels of priority amongst the features to be delivered.
- **Timeboxing** – The concept of a 'timebox' whereby time limit is set, at the outset, for the development of part of the system. Prioritisation is used to provide some contingency in the timebox, for example by including features of a lower priority that may be deferred or dropped if time does not allow.

A generic Agile model which encompasses the key principles of the iterative software development is shown in Figure 13.7.

Figure 13.7 Generic Agile model



The model begins with the identification of a business need; typically it will be the work of the business analyst to uncover the needs to be met. This initiates the generic Agile process depicted in Figure 13.7. The stages of this model are described next.

Identify options and feasibility

A feasibility study is conducted to determine the options that the business might pursue in order to address the business need. This will set out the options available, in particular the costs, benefits, impacts and risks of each alternative option, and possibly a preferred option.

Define and agree business requirements

Once a preferred option has been selected, the high-level business requirements need to be defined, from which more detailed solution requirements can be derived. This stage identifies the scope for the systems development project by defining a set of high-level features to be delivered by the solution. These high-level requirements are also prioritised during this stage.

Elaborate solution requirements

The high-level business requirements are explored and elaborated into more detailed requirements for the solution, using techniques such as storyboards, wire frames and prototyping. An iterative approach

	is adopted in order to develop a final, operational prototype. This confirms the functional requirements to be built and deployed in the current release. As Agile development projects typically deliver functionality incrementally, this stage and the following two stages (engineer solution and deploy solution) will be repeated for each incremental solution delivery.
Engineer solution	Once the functionality for the current delivery has been agreed, a production-ready solution is built that can be deployed into the live environment. This stage will addresses the non-functional requirements and technical infrastructure required to turn the operational prototype into a fully fledged, working solution.
Deploy solution	The tasks necessary to deploy the working solution into the live operational environment are conducted, including data take-on, data conversion, preparation of documentation, end-user training, installation of the live environment and transition to the service delivery team.
Evaluate solution	Once the solution is 'live', typically some period of time after it has been deployed, an evaluation of the solution is undertaken to determine whether it has met the business objectives and is working satisfactorily. This review may lead to perfective maintenance to improve aspects such as performance and usability. Once the final version is deployed, a benefits review will be conducted to examine whether or not the benefits have been realised and, if not, identify any further actions required to enable their realisation.

The arrows in Figure 13.7 show the overall sequence. The dotted arrows show optional revisiting of previous stages.

Advantages and disadvantages of the lifecycles

The waterfall, 'V' model and incremental lifecycles incorporate a high level of control which offers many advantages. However, in practice there are some real difficulties in the way:

- There may not be time to define all the requirements at the outset and only the most important or urgent ones can be defined in the time available.
- It is unlikely that the business actors will know exactly what they want, particularly if they are moving into new and uncharted areas of business. Even if they are reasonably knowledgeable about the requirements, it is likely that

there will be areas where they are less certain. These areas could concern the exact details for some requirements or non-functional aspects.

- It is important to establish some of the business changes quickly in order to demonstrate capability and achievement, and keep the support of the key stakeholders.
- The pace of business change is so rapid these days that a completely defined set of requirements can become out-of-date very quickly.

The most flexible delivery approach is achieved by utilising iterative development with incremental delivery together, in a co-ordinated and controlled way. This is the approach adopted during Agile software development, for example, using methods such as Scrum or DSDM which are described later in the chapter.

Developing the business solution

While the lifecycles described above are concerned with software development, the basic principles also apply to other business changes. For example, where the project is concerned with business process improvement, the requirements for the new processes may be defined prior to the development of the processes or an explorative, evolutionary approach may be adopted. Given that most requirements are delivered through a mix of IT and process changes, the lifecycle chosen for the software elements may impose a lifecycle and constraints on the other aspects of the business solution. For example, where an incremental software release is to be delivered, we will need to define the process, task and job changes that will make that increment workable in practice, and these may need to change again with the release of the next increment. Where a complete solution is to be implemented, the entire set of process, task and job definitions will be required at the same point.

DEVELOPMENT AND DELIVERY APPROACH

Whichever lifecycle is adopted it will also be necessary to select an approach to carrying out the project work. The approach will determine the methods and standards to be adopted on the project. There are several published approaches to business process improvement and software development that offer techniques and standards. The Unified Modeling Language (UML) is a popular approach, providing a range of modelling techniques that may be used to represent different views of a system. Organisations will need to take account of the context and the lifecycles described above when deciding which approach to adopt. In deciding this, there are two key areas that should be considered: the approaches to the development and the delivery of the solution.

- **Development**: there are two key questions to answer:
  - are we able to work collaboratively with the business users during the development of the solution or is this not possible?
  - are the requirements unclear or well understood?

When deciding on the development approach, we need to consider if the detailed requirements will need to be defined by working collaboratively with the business users

during the solution development. Where this is the case, it is vital that the business users are available to work closely with the project team so that the detail of the requirements may be uncovered as the solution evolves. Techniques such as scenario analysis and prototyping will be invaluable in this situation. However, if the requirements are sufficiently well understood to be defined prior to the solution development and testing, and there is likely to be limited contact with the business users, formal documentation techniques, such as detailed process and data models and a well-formed requirements catalogue, will be needed.

- **Delivery**: do we require one delivery of the entire solution or a phased delivery using incremental releases? The context described above will provide the basis for deciding whether the organisation requires the delivery of all of the requirements in one release, or if a phased approach that delivers incremental change, is necessary.

Software development approaches

There are several published, defined approaches to developing software solutions, each of which provides a framework and standards. Two key approaches are described below.

The **Unified Process** (UP) from the Object Management Group (OMG) is a generic software development process which can be configured to meet the requirements of an organisation. Its structure acknowledges that no single development process fits all organisations, development environments and cultures. It is designed to fit small releases such as enhancements to existing systems as well as large systems development projects. The UP provides a guide on how to use the Unified Modeling Language (UML) effectively; techniques from the UML are discussed in Chapter 12. The UP is both an iterative and incremental approach. It is based upon the principle of using UML modelling techniques to explore and elaborate requirements through a series of iterations. Increments are developed that may be combined for one release of the entire solution or may be implemented as phased releases.

Agile software development approaches include DSDM and Scrum. In recent years, these have become increasingly popular in organisations for a number of reasons:

- The need for organisations to respond quickly to fast-changing business situations.
- The difficulty – indeed, sometimes the impossibility – of knowing what is wanted at an early stage of the project.
- The importance of flexibility when deciding how a requirement will be delivered. For example, a requirement may be defined to protect certain areas of the data but there may be several possibilities as to how this is achieved. The solution to deliver this requirement may be decided through an exploration of various mechanisms.

The Agile methods provide a means of developing IT systems using an iterative approach while still ensuring the quality of the software solution. This approach assumes that it is impractical to expect that systems can be built during one development phase for the entire system. It is based upon the principle that 80 per cent of a solution

can be delivered using 20 per cent of the development effort needed to produce the total solution. When using an Agile approach, there is the danger that the emerging prototype systems are not documented sufficiently or are not coherent, which leads to considerable difficulties later when they are in live operation. The methods described here address this danger quite explicitly and stress the need for documentation and testing throughout the development activity.

Scrum is a widely adopted manifestation of the Agile approach. The name refers to the method of re-starting a game of rugby after an infringement of the rules and, in the method, is the name given to the daily progress meeting between members of the development team. The ScrumMaster is a facilitator for these meetings but is not a project manager in the conventional planning and directive sense. The development team is self-governing and is guided in its work by the product owner who acts as the 'voice of the customer'.

In a Scrum project, the development work proceeds in a series of 'sprints' which are timeboxed periods of effort, typically from one week to one month in duration. At the start of each sprint, there is a sprint planning meeting where the product owner nominates the work to be done by selecting tasks from the 'product backlog'. The daily Scrum meetings focus on what has been accomplished yesterday, what is planned for today and what obstacles have been encountered.

As with other Agile approaches, Scrum depends on close co-operation between the user community (represented by the product owner) and the developers, and on the ability to prioritise the work to be done. Prioritisation is explored in more detail in the next section.

The importance of prioritisation

Prioritisation is extremely important during solution development as there are always many requirements and limitations on time and budget. Clearly, all requirements are not deemed of equal importance, and this helps determine what should go into each work package or iteration. Whichever approach is adopted, it will be important to identify where the priorities lie in order to ensure that the available effort and funding is targeted at the requirements designated the highest priority. Careful and accurate prioritisation is vital if we are using an incremental delivery lifecycle, possibly combined with iterative development. It is here that a prioritisation scheme such as DSDM's MoSCoW structure, which was introduced in Chapter 11, is particularly relevant. The MoSCoW prioritisation categories are related to the development and delivery of the solution as follows:

- 'Must have' requirements will be delivered in the first deployed increment; these form the 'minimum usable subset' of requirements. If an iterative development approach is to be adopted, these requirements will form the first elements of the prototypes.
- 'Should have' requirements provide one of the mechanisms for introducing contingency and flexibility. They could be implemented as manual 'work-arounds' in the short term, which is extremely helpful when deadlines are tight. Any of these requirements that have not been delivered in the first release will be allocated a 'must have' priority in the second increment.

- 'Could have' requirements may be included in the set of requirements under development, particularly if they are relatively easy and inexpensive to incorporate with the higher priority requirements. Where timeboxes are used, these requirements provide some contingency, as they can be left out should the development team run out of time.
- Finally, the 'Want to have but not this time around' requirements are recognised as those that will be set aside and considered during one of the later increments. This is an essential element for incremental delivery approaches as it provides a means of identifying requirements for later phases of the solution and specifically annotating them as such. These requirements will be allocated a different priority once the point arrives for their delivery to be considered. For example, there may be a specific date when some of these requirements become mandatory and, when planning for that release, the 'W' prioritisation may be changed to an 'M'.

The MoSCoW approach is also extremely useful for prioritising other types of business changes. For example, when developing process documentation we can identify the elements that must be included at the outset, those that can wait for the next version and those that could be dropped if we run out of time; or when developing team capability, we can prioritise the different skills into the MoSCoW categories in order to highlight those that are most important and those that can be deferred, even if only for a short time.

Use case diagrams, described in Chapter 12, are also useful in prioritisation as they can provide a highly visual way for business actors and developers to understand the level of priority assigned to each of the use cases. While modelling the potential scope of the entire solution, a use case diagram can be used to partition the solution into practical implementation packages for the short or longer term and provide a means of considering the different options available to deliver the business requirements.

Software package approach

So far in this chapter, we have considered the lifecycles and approaches that may be used when developing a software solution, using either an in-house development team or an outsourced software development supplier. However, in many situations, organisations prefer to adopt commercial off-the-shelf (COTS) solutions where possible, using these to implement best practice and engender related business process changes. The reasons are not hard to find:

- A COTS solution is almost certainly going to be cheaper than the organisation having to fund the entire development process since, by definition, the development costs are shared by all the users of the package.
- Implementation should be faster, as the solution exists and only has to be set up in the way the customer wants.
- Support and maintenance packages are available from the software vendors.
- COTS vendors keep their software up to date, for example in line with changes in legislation.

As against these advantages, however, there are some drawbacks, including:

- No COTS package is likely to be a perfect fit with the requirements, so either the customer must adapt their processes to what the package can do or they must pay for expensive tailoring and customisation, thereby partly negating one of the benefits of the COTS approach.
- If competitive advantage is required, it is unlikely to come from a software package, since all organisations working in a particular business domain can buy the same software and adopt the same work practices. However, the deployment and use of the software also needs to be done effectively, which may offer an opportunity for differentiation from competitors.

If a COTS solution is desired, care must be taken that the requirements focus on what the system is required to do, rather than how it is expected to work (unless the 'how' really is vital to the business, for example to improve competitiveness). And the most effective use of a software package requires a willingness to change the business processes and procedures where necessary to align with how the package works.

Whichever approach is taken – bespoke development or COTS solution – and whichever lifecycle and approach are chosen if the former is selected, the business actors must make a decision on how far they want to go with automating the processes and jobs. For example, there may be situations, such as those requiring complex decision-making, where it is preferable for the work to be undertaken manually by staff who can make judgements on a case-by-case basis.

ROLES

The roles that are required to deliver the solution will depend upon these three factors:

- the context of the organisation and project;
- the nature of the lifecycle selected;
- the approach adopted.

Typically, there will be a project team that will include the following roles:

- project manager;
- business analyst;
- developer;
- tester.

However, the point at which these roles are required will differ depending upon the nature of the solution, the lifecycle to be used and the approach chosen. If the solution is not concerned with changes to the IT systems but to the business processes that use them, the role of the business analyst will be to analyse and define the business process improvements but there will not be a need for the IT developer or software testers. Where there is a software solution, and the 'V' model or the incremental lifecycle

are selected, the business analyst may complete much of the early work without the need for the involvement of the developer or tester. However, if a software solution is required and an Agile approach is adopted, a collaborative project team that includes development and testing skills will need to be set up at an early point. Whichever approach is adopted, there may also be a need for other roles. For example, a technical architect may be needed where the solution to be developed is complex and needs to be integrated with other software applications. Or, a data analyst may be needed where the software is underpinned by a complex data structure that will need to be designed carefully to ensure that all of the requirements and business rules are understood.

Within the business community there are also key roles such as that of the project sponsor and the business user. These roles play an essential part in the development of the solutions. The project sponsor is the business representative who is the point of contact for the project manager and is responsible for providing the resources to the project team. The business users provide the details of the requirements. However, the nature of the work carried out by these roles, in particular that of the business user, will vary depending upon the approach adopted. Where a waterfall, 'V' or incremental lifecycle is adopted, the business user will be involved in the definition of the requirements and the acceptance of the solution. Where an Agile approach is adopted, the business users have to work collaboratively with the development team during the software development.

DELIVERABLES

The products that are to be delivered will also vary depending upon the nature of the solution, the standards of the organisation and the lifecycle and approach adopted. Sometimes we can deliver the requirements through a training exercise to improve the skills of the business users. In this case, the deliverables may be course descriptions and training manuals. Sometimes, the business rules applied by the business users need to be defined and documented and a follow-on training exercise conducted. In this case, the deliverables will also include the procedure descriptions. Where business process changes are aligned with or supported by IT systems changes, the deliverables will be more extensive and will be linked to the software development lifecycle and approach. Early methods for systems analysis and design, such as the structured systems analysis and design method (SSADM), placed a strong emphasis on formal documentation throughout all stages of the systems development lifecycle. While this method is rarely used these days, some of the principles concerning documentation are still applied. For example, many organisations develop extensive requirements documentation prior to deciding upon the delivery approach, and maintain that documentation throughout the development lifecycle. Where an iterative development approach is to be adopted most organisations document the requirements in overview before exploring the selected requirements set in greater detail through the use of prototyping techniques. In this situation, the deliverables include the prototypes developed during each iteration. However, many organisations also enhance the requirements documentation as part of the iterative development work and ultimately may produce similar deliverables to those resulting from a waterfall-type lifecycle.

One of the important aspects about the deliverables is that they should always be suitable for the purpose. Where documentation standards exist – whether organisational

standards or part of a particular approach – it is useful to adhere to the standards as long as the deliverables are valuable in the particular project context. There is little benefit to be gained (and potentially a lot of time to be wasted) in developing deliverables that do not contribute to the aims of the project.

TECHNIQUES

The techniques to be adopted during the development of the solution can vary considerably from project to project. They will depend upon the scope of the solution, the standards of the organisation and the lifecycle and approach adopted. Some typical situations are described below:

- The use of a waterfall-based lifecycle will require the use of formal documentation that has been reviewed and 'signed off'. This necessitates the use of formal and rigorous techniques for documenting and modelling requirements. For example, a technique such as entity-relationship modelling or class modelling is likely to be required.
- Where an Agile approach has been adopted for a project, this will determine the techniques to be used. As stated earlier, prototyping is a vital element of the Agile approaches but equally techniques such as wireframes, scenarios and user stories are extremely useful.
- Organisations may have defined templates for requirements documentation that include standards for modelling business processes and IT requirements. They may also have templates for organisational documents such as job role definitions and job descriptions.
- Many organisations have adopted support tools that impose both a development process and the modelling standards to be used.

SUMMARY

All of the aspects described above need to work together to ensure that a coherent approach to delivering the requirements is adopted. It is important to recognise that the context for the organisation and the project should determine this. An inappropriate lifecycle and approach will not reap the business benefits and is likely to result in wasted time and budget, possibly leading to the failure of the change initiative. The standard lifecycles and approaches are used in the main to develop software (or some stages are used when procuring an off-the-shelf package) but we also need to consider them when delivering the other required aspects – the business processes and procedures, and the new jobs and people skills. In many situations we cannot deliver all of the requirements as we would ideally, or even ultimately, desire. The nature of the business constraints may mean that we have to explore alternative means of delivering some requirements in the short, or even longer, term. The role for the business analyst is to identify where this is required and seek out the options for delivery that will ensure the business needs are met.