2 BUSINESS REQUIREMENTS

Now that we understand the context and purpose of UAT the next stage is to understand the business requirements. This chapter explains what requirements are, how requirements are written and how the business requirements relate to UAT. It will also become clear why requirements may be out of sync with the real business requirements by the time UAT takes place and how this limitation can be overcome.

Topics covered in this chapter

- Business requirements
- Business intent and user expectations
- Acceptance criteria
- The requirement types
- Prioritising business requirements
- The relationship between business requirements and UAT
- The relationship between development and UAT
- Scope of UAT
- Building a UA test basis

BUSINESS REQUIREMENTS

In Chapter 1 the fit between an IS and its business users was referred to as being like the fit of a garment that is 'made to measure'. Using the same analogy a requirement identifies something that the customer wants to specify about the item of clothing about to be made for them. Similarly, a business requirement is something a business wants to specify about the IS it is about to have built or is about to acquire. In other words a requirement is simply something that we need and in the context of an IS, a requirement is something we need from the system. The better the desired outcomes and the detail of what the system ought to do is understood, the better the chance of delivering what is needed. Requirements describe what a system ought to do, what will be built and what will be tested during UAT.

Having captured the needs of the organisation the project team will document them in a requirements document using clear and simple language and using the terminology of the user. These requirements, if too general to be of use to developers, can then be translated into a more technical version using a structure aimed at helping the development team to understand and build the system. The resulting document is known as a requirements specification (RS) and the detailed needs of the business contained in it are the requirements.

UNDERSTANDING THE SYNTAX OF REQUIREMENTS

What do requirements look like?

Requirements can be expressed in many different ways and they do not all look the same. Requirements may be recorded in a document, in a spreadsheet or in software designed specifically for recording requirements.

Here are some simple examples of requirements but bear in mind that any requirements document you are given for UAT may contain much more than this very basic information.

Example 2.1

Table 2.1 Business requirements

CR1	An administrator is able to add a new user
CR2	An administrator is able to adjust existing user details
DB1	The system supports that 100 previous versions of a contract are available in a version history

As long as the language and terminology used is familiar to the people who requested it and to those who will develop it, then these are good examples of what requirements ought to be; namely a clear description of a requirement of the system.

However you may find that the requirements document you have been given contains much more than just this very basic information.

Example 2.2

These requirements, related to the security of a website, were entered into a system as opposed to written in a document. As well as a description of the requirements this document also contains some other valuable information.

Table 2.2 The requirements document

Project	Reference	Category	Description	Priority	Current status	Actions
UNT v0.2	S_01	Security	Users will be able to change their own password	1	None	Edit/ Delete
UNT v0.2	S_01.1	Security	There will be an option for users to change their password on the main menu	3	None	Edit/ Delete
UNT v0.2	S_01.2	Security	There will be an option from the login page for users to request a password reset if they have forgotten their password	3	None	Edit/ Delete

The name of the project
This field may not be needed where a requirements document relates to one project only; however, it may be relevant to software that can record requirements related to multiple projects.

A unique identifier of the requirement
Every requirement should have a unique identifier, which ensures that any further project documentation that is created can refer back to which requirement an activity or line item relates to. Because the purpose of the project is to meet the business requirements, the unique identifier is one of the cornerstones of all the project documentation that follows the RS. Note that there is a difference between the unique identifiers in the two examples above.

In the second example you can see that there are two different IDs that split the requirements into one larger overall requirement (S_01), and the detailed requirements that belong to it (S_01.1 and S_01.2), whereas there is just one level of ID in the first example (CR1).

The category
It probably makes sense for most RS documents to group together requirements in categories that cover similar ground. These categories cover any related parts of the functionality that the project is delivering and can be logically grouped together under headings such as: 'Creating a contract', 'Invoicing', 'Proposal maintenance', 'Workflow' or

'User interface design'. Categories may include functional aspects of the system – what it ought to do – as well as non-functional aspects – how it does what it ought to do. The non-functional requirements may be written by the project team if they were not requested by the business and cover topics such as the speed and usability of the system.

The priority
It is useful in most, if not all, cases to understand the priority of a requirement as it will help to schedule the development work and prioritise testing. In theory it is possible to use any scale; however, it is quite common to use 'critical' and 'non-critical', or to use a scale of 1–3 or 1–5, 1 being the highest priority and 3 or 5 being less important. The assumption made when writing the RS is that all the requirements are needed – hence the name 'requirement' – but that some are more critical than others.

Current status
It may or may not be important to track in the requirements document whether the requirement has simply been requested or whether it has been built or tested. If it is important, a status field or column can be created for the purpose in the RS.

Edit/Delete
In the example for project UNT v.02, the system that the requirements are logged on keeps track of the version history of requirements and the Edit/Delete buttons allow users of the document to make changes to the requirements. This may not be relevant in your case; however, it is useful to remember that requirements can, and often do, change. The RS should be revisited and reviewed during its life in order to make sure that the documented requirements still reflect what is happening on the project and what the current business needs are.

Exercise 2.1

The case study we will use throughout the text is of an accounting system named Excelsior, which has been created to deal with the accounting-related processes such as purchase orders, contracts and payments. It also has a number of modules that allow staff to carry out HR-related tasks. HR details can be updated and

Table 2.3 Excelsior requirement

Key	Name and description	Rank	Priority	Assigned to	Status
Request saving and background updating					
BW4	The system will allow saved requests to be automatically updated Once a request has been saved, users will receive request updates when and if they become available	34.0	High	Nobody	In progress

requests can be made for other changes such as absence, training or expenses. What information can you glean from the following business requirement?

The answer can be found in Appendix B.

Please note that if you are asked to review a requirements document or specification and you do not understand the requirements in that document, you may need to ask some questions of those that do.

BUSINESS INTENT AND USER EXPECTATIONS

We established that requirements are gathered and documented in a business requirements document, a set of statements that describes all of what a user community wants from the IS. The requirements as a whole should reflect the overall objectives and expectations of the business. These overall objectives and expectations, although they are often forgotten, should also be documented.

Every IS is created for a purpose: to achieve some business benefit such as improved productivity for staff or reduced costs of operation. The purpose of the system is usually called its 'business intent'. Business intent is important because it identifies what the business is expecting to be able to achieve when it has the system installed and operating.

Arising from the business intent the sponsor, managers and end-users will have a perception about what the system should do and how it should behave. Users of systems already in place will also have expectations based on their perception of the way they currently work and the way they are expecting to work with the system. Expectations may not be exactly aligned with business intent but they are an important element of perceived success, so they should be captured as an input to UAT.

A key driver of success in IS implementation is the transparency of business intent and user expectations so that all stakeholders, including the developers, have a common understanding of what is required and what is expected in an IS solution. The absence of effective transparency is a key risk factor because the parties may be operating from different assumptions, and UAT is the final opportunity to identify and resolve any misunderstandings that may have arisen during development. If the business requirements match the business intent and the system meets the business requirements, the business intent will have been achieved.

Example 2.3 – Business intent

- The system will enable the business to manage 250 customer accounts simultaneously.
- The system will enhance throughput of widgets by 10 per cent.
- The system will enable the business to offer five new services.
- The system will enable the business to deliver customer orders within 48 hours of order.

The business intent should be measurable and immediate. Once the system is installed it will be possible to determine if the business intent has been achieved, so it is also feasible to test for achievement of the business intent at UAT. A more strategic long-term objective, such as reducing staff numbers for example, cannot be tested at UAT. For this reason it is preferable to base the achievement of longer-term objectives on improvements that can be seen and verified immediately.

EXERCISE 2.2 – BUSINESS INTENT

For the project you are currently involved in, or for a previous project, list the overall goals that make up the business intent from memory or, if no business intent was written, from your knowledge of the project:

- ______________________________
- ______________________________
- ______________________________
- ______________________________

Were you able to list the overall goals? Were they specific enough to be measurable?

Unless the stakeholders, managers, developers and testers are aware of the overall requirements the IS must meet and understand how they should be measured, it will be harder to meet those goals. Whether or not the goals were met is an essential ingredient of UAT. The ability to decide together, based on an agreed measure, whether the project has been a success or not and whether the system should be accepted is what UAT should be able to achieve.

ACCEPTANCE CRITERIA

Before we look in more detail at the requirements and the different requirement types, we should examine the acceptance criteria.

Acceptance criteria

The exit criteria that a component or system must satisfy in order to be accepted by a user, customer or other authorized entity. (ISTQB Glossary)

The business requirements tell us what the system must do for the business so we will base our tests on them. Acceptance criteria, on the other hand, tell us how we will know that the system is fit for release to its users. In other words acceptance criteria are our way of determining when we have done enough testing.

Example: at least 90 per cent of all business requirements must be tested.

This acceptance criterion prevents a scenario in which only a small sample of requirements end up being tested. Acceptance criteria can be quality-related as well as system-related. For example we would not want a system released if all requirements had been tested and all the tests had failed. We can guard against this eventuality by imposing quality-related criteria.

Example: there must be no critical defects outstanding at release.

Acceptance criteria are the conditions that must be satisfied before a system can be accepted by the users or stakeholders. Accurately defining the acceptance criteria is one of the most important elements of a successful development project and clear acceptance criteria are critical to the success of testing. Inaccurate or incomplete acceptance criteria can lead to low customer satisfaction levels, missed deadlines and increased development costs.

The fact that all documented requirements should be met would be an implicit acceptance criterion, although it could also be stated explicitly. Other criteria would extend this to define the status of the system when it is offered for release. Examples might be:

- All requirements shall be tested before release.
- There shall be no critical defects and not more than two high-priority defects outstanding on completion of acceptance.
- There shall be no unallocated incident reports outstanding.

These three criteria should ensure that what is released is complete and in a stable state, which is clearly important for the user population.

Acceptance criteria have another useful purpose. If, at the scheduled release date, the acceptance criteria have not all been met, they can be used to provide a measure of outstanding work and the possible delay if the system is held until all criteria are met.

For example if a system has one critical defect and four high-priority defects outstanding, we can discover relatively easily how long it will take to clear the defects and retest to ensure the fixes are correct. This provides a measure of possible delay. We could also consider whether the effects of any of the outstanding defects could be mitigated by some kind of ‘workaround’. Acceptance criteria set up a dialogue between UAT, developers, users and sponsor through which a release date acceptable to all can be negotiated.

THE REQUIREMENT TYPES

So far we have identified that the RS is a document, usually written by a business analyst or a developer, which defines the business requirements in a form that developers can understand and respond to. However, the requirements are not homogenous and we can identify different types of requirements: functional requirements, informational requirements, behavioural requirements and environmental requirements. The RS should ideally contain all the different types of business requirements as well as expressing business intent, capturing user expectations and defining acceptance criteria.

Business requirement types

All business requirements contain information about what the business wants the new system to do, but there are different types of business requirements that we can distinguish between. In many organisations only two requirement types are used: functional and non-functional requirements, but there are at least four distinguishable types of business requirements:

- functional requirements;
- informational requirements;
- behavioural requirements;
- environmental requirements.

Functional requirements

Functional requirements are the type we are already familiar with, in that they define what the system must do. Functional requirements cover business processes that could be independent from technology, for instance 'purchase order sign-off' or 'shipment', and as a result are often easiest to understand and define. The system will apply logic to an input and produce the required output. On the basis of functional requirements the development team will build the parts of the IS that the end-user interacts with.

Example: the system shall validate that all passwords have a mixture of upper- and lower-case characters and at least one number.

Informational requirements

Informational requirements define the information provided to the end-users to help them do their work. They also cover to what extent information sharing and access control are required and set out the requirements for the information that is generated as input data, output data and stored data for the processes defined in the functional requirements. Some additional data and metadata that are technology-related rather than process-related may also be defined as part of the informational requirements.

Example: customer account files shall contain the fields: CustID, CustName, OrdDate, OrdQty and OrdPrice.

Behavioural requirements

Behavioural requirements describe how the solution has to act. They are often called non-functional requirements and they can be defined quantitatively or qualitatively.

Example: a qualitative behavioural requirement might be that the user interface must be user-friendly. This can be made verifiable in a number of ways: by relating it to existing standards, for example the Microsoft UX Guide, which is a guide to user experience interactions; or by defining parameters of the user interface, such as the number of colours used and the allowable uses for each colour.

Quantitative measures easily translate into numbers but qualitative behavioural requirements also need to be translated into a quantitative measure. The user interface requirement that covers the responsiveness of the system, for example, will have to be quantified in a way that can be measured and tested.

Example: response time for recalling customer address details will not exceed 1.5 seconds.

Environmental requirements

Environmental requirements, sometimes referred to as ‘constraints’, describe the context within which the IS will operate. This may include where each component of the IS is to be physically or geographically located, any industry standards to which it must adhere and any constraints on the use of technology; for example the system must be accessible from platforms running MS Windows, Unix or Linux. They should also cover any regulatory restrictions.

Many of the requirements in the last three categories, the non-functional requirements, are less likely to be requested or written by stakeholders and more likely to be added by the project team. Although the requirements do describe the needs and wants of the business, a lot of other information represented in the RS is often of a much more technical and abstract nature.

In practice a RS may not contain all of the information we need and the translation from business-speak to developer-speak that we mentioned earlier may have ‘blurred’ some aspects of the business requirements. Worst of all, the RS may have to be written and approved before development can begin so that it represents a snapshot of what the business requirements were believed to be at the beginning of development.

We will, in all probability, need to supplement the RS before we can plan, design and run effective UAT and we will see how we do that in Chapter 6. For now we will work on the basis that whatever documentation we have is an adequate expression of business requirements.

PRIORITISING BUSINESS REQUIREMENTS

Neither time nor resources are unlimited for UAT so tests probably have to be prioritised in some way that balances the expectations of each stakeholder group. Prioritisation also ensures that if time runs out, everyone can feel reassured that the most important (highest priority) aspects of the system have been tested.

Sponsors will naturally be most concerned about the business intent so a way to express the intent is needed, and business criticality is one way to determine which aspects should be tested first. Similarly users and managers will have concerns about how the system operates so their concerns will be encapsulated in the idea of usability.

Business criticality

Business criticality is a measure of how important any individual requirement is to the success of the business or to the achievement of the business intent. We could define any number of levels from absolutely critical to not at all critical, but what is usually done is to define just two – critical and non-critical. Within these two broad categories priorities are then attached to each individual requirement (or possibly to groups of related requirements) to enable the organisation to prioritise development and plan accordingly. For UAT we can use the same measures of criticality and priority to manage the UAT priorities or, if priorities have not already been assigned to requirements, we can consult the sponsor to determine the criticality for testing.

Usability

Usability has a technical meaning that gives rise to a specific type of testing known as usability testing. For UAT we have neither the expertise nor the time or resources to do usability testing in a rigorous way, but we do have a set of expectations that we can use to determine some simple parameters for testing. For example if the sponsor has an expectation that the system will be able to handle 30 specific transactions per hour then managers will have an expectation that the throughput of a single terminal multiplied by the number of terminals will at least equal the required throughput of transactions, while the end-users will want to satisfy themselves that the user interface at any single terminal will enable them to handle the expected number of transactions overall.

For each requirement or group of requirements the business criticality can be used to generate a priority for testing and the usability criteria for those requirements acquire the same level of prioritisation.

Business process

The definition of UAT included testing the system according to ‘user needs’ and ‘business processes’. We will need at least some of our tests to exercise business processes so this will be a third focus for our prioritisation of requirements for testing; we will group together aspects of the system related to specific business processes and test them together so that we can determine whether the system correctly implements the business process.

THE RELATIONSHIP BETWEEN BUSINESS REQUIREMENTS AND UAT

The RS is the document we usually see as the point of contact between users, developers and testers. It is intended to express to developers what the users want so they can build the system, and simultaneously to express to the UA testers what the users want so they can test the system independently of the developers. The word ‘independent’ is very important here; it is the key characteristic of UAT.

UAT is an expression of the RS (as interpreted by UA testers); the system itself is an expression of the RS (as interpreted by the developers). Comparing these two expressions by running the UA tests is the only way we can determine whether the outcome of development matches what the users said they wanted. So it is absolutely vital that UA testers work independently of the developers in designing their tests and it is equally vital that they work from an expression of the requirements that has not already been interpreted by the developers; otherwise UAT would simply reflect the developers' interpretation.

We have shown previously that, although it should be the embodiment of what the business wants and needs, the RS may be out of date, incomplete or inappropriate because:

- The needs of the business were not documented in detail (or at all).
- The needs of the business changed during the development project and the RS has not been kept up to date.
- The documentation was deliberately written only in outline at the outset and was then developed iteratively and incrementally.

These situations are all likely to lead to a state of affairs at the end of development in which the delivered system is not aligned with (current) user expectations. In addition to the fact that the RS may not be completely accurate, further mistakes may have occurred down the line. The problem that sponsors, managers, end-users and developers have in describing what an IS must do is that the sponsors understand the business background but (usually) not the technology; users and managers understand what the system is meant to do but (usually) not how it will work; while the developers understand the technology but (usually) not the business background. A middle ground is hard to find because the four stakeholder groups use different ways to communicate and have different views of the world. What is usually done is to try to define the business need in terms the users understand and then 'translate' it into 'technology speak'.

This process can cause problems for two very important reasons:

- Translations are never simple or completely accurate, so the translation process introduces errors.
- Both groups have a set of information that, as far as they are concerned, 'everybody knows'. These assumptions are seldom stated in requirements because the author's mindset does not recognise any gap in the information. Meanwhile readers take what is written literally and try to deliver it according to their own assumptions. This is a major source of serious requirements gaps.

So an RS is likely to be a flawed expression of business requirements, especially if it is drafted and approved at the start of the project. In addition:

- Requirements defined at the beginning of a project may contain errors and/or omissions that need to be corrected.
- Users may gain new insights during development that lead to requirements changes.
- The business environment may change in a way that forces requirements to change (for example in a system related to taxation there may be revisions to taxation law).

We will therefore need to assess the RS and, if necessary, update the requirements as expressed in the RS at the end of the project and document these as a basis for testing.

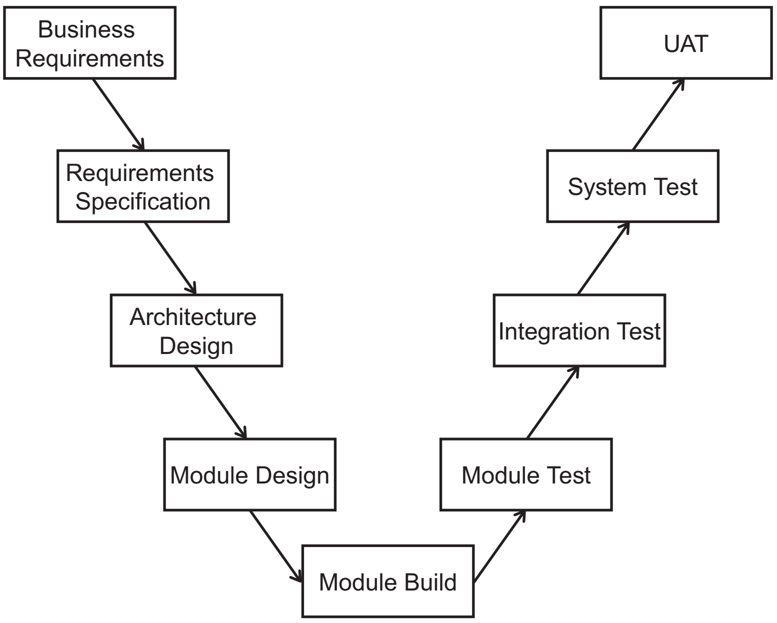
THE RELATIONSHIP BETWEEN DEVELOPMENT AND UAT

There are many ways to develop software but most fall into three broad categories: sequential development, iterative development and component-based development.

Sequential development

Sequential development uses a sequence of development stages that typically follow a V shape as shown in Figure 2.1. Discovering and documenting initial requirements is the first stage in this process and UAT is the final stage. The approach breaks a system down into manageable chunks and functionality is not normally delivered until the end of the project.

Figure 2.1 A sequential development life cycle

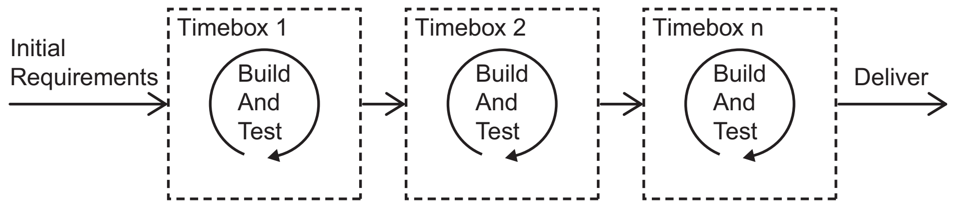


Notice that the test levels relate to different stages of design in which the requirements are analysed and then broken down into smaller and smaller building blocks. The test levels then test the outcomes of each of these design stages. UAT is the final test level that tests the completed system against the requirements.

Iterative development

In an iterative approach (such as agile development) the design and testing takes place during short development 'sprints' so that system functionality becomes available incrementally at the end of each sprint (Figure 2.2). In this case requirements are implemented incrementally according to a plan, but not all requirements are documented initially. UAT will be required before roll-out to confirm that the system is acceptable, but there will be no requirements documentation to form a test basis.

Figure 2.2 An iterative development life cycle



Component-based development

Other common ways to develop systems build on one or other of these fundamental models. For example a system can be constructed from commercially available components integrated into an architecture that meets the business requirements. In a component-based system architecture the component 'building blocks' are likely to be well defined but the integration will involve greater risk. UAT should encounter relatively few problems with the way the system functions but problems can occur, for example where integration of components does not deliver exactly what was required. A related approach, using COTS software, acquires most of the functionality in a collection of existing modules; development involves configuring the modules to meet the business requirements. In this case much of the focus of UAT will need to be on whether the solution is a good fit with business intent.

SCOPE OF UAT

The scope of UAT is defined in the UAT plan and is of vital importance to its success. It defines the extent of UAT: what will and what will not be tested. Clearly the requirements will be tested but the scope explains the extent of the deliverables and the boundaries, including those things that are out of scope. We saw earlier that within this parameter we have to ensure that the most critical requirements are tested first according to the perspectives of the most important interested parties. But are there any additional things we should test or definitely should not test?

What we must test

Firstly one thing that is not negotiable is the testing of contractual requirements. If the system is being delivered in fulfilment of a contract, then we must carry out contract acceptance testing to ensure that the delivered system meets all the requirements stated in the contract. Even in this case, though, there are likely to have been variations, some documented and some not, that will need to be taken into account.

Secondly we have to aim to test the business intent and user expectations by prioritising requirements and usability according to the business criticality of each requirement. This must include anything explicitly mentioned as a deliverable in a contract and anything explicitly defined in the RS.

Thirdly we need to ensure that we test every business process from end to end. Business processes have not been tested prior to, and must form the framework for, UAT. Any functional requirements should be tested as part of the known existing or future processes as requirements that do not relate to business processes may not be relevant in the new system.

Fourthly we must test any defect fixes that are outstanding and any fixes for defects that we find.

Finally we must carry out a regression test for every change that occurs. Regression testing checks whether the defect has been fixed and confirms that the fix has not caused any other problems.

What we will not test

With the limitations of time and resources we have already mentioned, it would clearly be impossible to test everything about the IS, but then many aspects will already have been tested. Unit testing (testing the code), integration testing (testing whether parts of the code or system work together) and system testing (testing whether each part of the system functions correctly) have already been carried out, and repeating any of these tests is outside the scope of UAT.

So UAT will not test anything that has already been tested in one of the development testing phases (provided we have evidence of the tests) and, in particular, it will not set out to test all of the functions of the software.

For reasons stated earlier we will not attempt to test usability in detail; nor will we attempt to test performance, except that we will embed some usability and basic performance measurements in business process testing.

ARIANE 5 – THE IMPORTANCE OF SCOPE

On 4 June 1996 the unmanned rocket Ariane 5, built to launch satellites for communications, earth observation and scientific research, was launched. It exploded just 40 seconds after lift-off on its first voyage after a decade of development costing £4.5 billion. The Ariane 5's cargo was valued at an additional £324 million. The cause of the failure was a software error in the inertial reference system (SRI – derived from the French equivalent of the name), the navigation system that calculates the position, orientation and velocity of the rocket during take-off. The SRI had been designed with a backup so that when the on-board computer (OBC) detected a critical issue the backup SRI would take over. Unfortunately the main SRI and the backup SRI contained the same error (an internal SRI software exception was caused during data conversion where the number that was converted had a value greater than what could be represented in the converted value format). This resulted in an operand error causing both SRIs to fail. The complete loss of guidance meant that the launcher veered off its flight path, broke up and exploded.

The inquiry board in the official report on the causes of the disaster stated that:

> Testing at equipment level was in the case of the SRI conducted rigorously with regard to all environmental factors and in fact beyond what was expected for Ariane 5. However, no test was performed to verify that the SRI would behave correctly when being subjected to the countdown and flight time sequence and the trajectory of Ariane 5. It should be noted that for reasons of physical law, it is not feasible to test the SRI as a 'black box' in the flight environment, unless one makes a completely realistic flight test, but it is possible to do ground testing by injecting simulated accelerometric signals in accordance with predicted flight parameters, while also using a turntable to simulate launcher angular movements. Had such a test been performed by the supplier or as part of the acceptance test, the failure mechanism would have been exposed.

The scope, or what is included in the requirements and thereby in UAT, is key to the success of an implementation. It is easy to make assumptions about what ought to be included based on what has worked well previously or how a requirement ought to be tested. The scope must be viewed in light of the current project, making as few assumptions as possible, and a culture should be encouraged where project team and stakeholders feel collectively responsible for the content of the IS and the extent of the testing, and empowered to speak up if anything seems to have been omitted.

BUILDING A TEST BASIS FOR UAT

Test basis

All documents from which the requirements of a component or system can be inferred. The documentation on which the test cases are based. If a document can be amended only by way of a formal amendment procedure, then the test basis is called a frozen test basis.

The life cycle of an IS begins with identifying and documenting business requirements, which then become the basis for development. In theory, then, we can use the requirements documentation as a basis for our UAT. In reality, however, this is not always the case, so we need a foolproof way of defining a test basis for UAT that does not rely on a complete set of initial requirements. The key sentence in the definition of a test basis (see box) is the second one: 'The documentation on which the test cases are based.' We need a test basis for UAT as a set of documents and what we have at present is an RS. In some situations we will have to test strictly against the original requirements; for example if a contract for development was based on them. In most cases we need to extend the information in the RS.

Evaluating/enhancing the RS

The requirements as a whole and each individual requirement need to be checked to ensure that they are of the required quality. If stakeholders are unfamiliar with how business requirements are formulated they may make their requirements too:

- high level;
- ambiguous;
- focused on the solution instead of the business need;
- general and not focused enough on specific user groups;
- all-encompassing, lumping together multiple requirements.

Requirements are checked and updated by reviewing them as a team of interested parties and stakeholders. Members of the team may also individually review the document with a checklist (containing the characteristics of good requirements for example) to note any comments and questions before a walk-through takes place. Review techniques will be explained in more detail in Chapter 3.

CHARACTERISTICS OF GOOD REQUIREMENTS

Gathering requirements is an imperfect process and mistakes will occur. Very few people have been trained how to elicit, analyse and document quality requirements and the RS may contain many badly written requirements. So what should we look out for when reviewing a requirements document? Here are some key things to consider.

Ambiguity

A good requirement is unambiguous. The reader should only be able to interpret the requirement in one way and draw one conclusion. The requirement should contain no jargon or ambiguous language such as ‘fast’, ‘user-friendly’, ‘several’ or ‘efficient’.

Mistakes

A good requirement should be correct. Only the user representatives can identify whether the requirement reflects what they intended. The end-users or other appointed stakeholders should sign off or otherwise agree that the document accurately reflects their needs.

Feasibility

A good requirement is achievable. Good requirements must be implementable within the project’s technological, budget and resource restrictions. This is not directly UAT’s concern, but any infeasible requirements will raise questions about what was actually implemented, and these need to be explored.

Necessity

A good requirement is needed. A requirement should meet a genuine need of the business and should have been requested or agreed to by someone with the appropriate level of authority.

Priority

A good requirement is prioritised. Especially for the purposes of UAT, a key part of risk management is to test the highest-priority requirements first. Without priorities UAT becomes much more difficult to manage.

Verifiable

A good requirement can be tested. A requirement should contain a concrete quantitative measure that can be tested even when covering requirements related to quality.

GOOD AND BAD REQUIREMENTS

Bad examples:

- ‘The website must be user-friendly and fast’ (too high level and contains two requirements). What exactly is meant by user-friendly and how would we test this? What would happen if we found it to be user-friendly, but not fast? Would it fail our test or pass?
- ‘All documents must be branded’ (ambiguous). What kind of branding? What are the documents involved?
- ‘We need visibility of what contracts are about to expire’ (not thinking in terms of user groups). Does this affect every user or only a specific user group?
- ‘An email must be sent automatically on enquiry’ (focusing on the solution instead of the business need). Is email the right answer? Perhaps the real requirement is that the customer be notified? The correct answer may be by email, fax, letter or SMS based on their preference. What if the enquiry did not come in via email?

Good examples:

- The design should be simple enough so that users will be proficient to use the IS after a day’s training.
- All non-managerial sales staff need to be able to see which of their contracts will be expiring within the next three months.
- The IS should be accessible from Internet Explorer and Google Chrome.
- Contracts should open in less than 0.5 seconds.

If a UAT team works from the requirements as originally documented, the UAT exercise is likely to hit all sorts of problems. Our review of the RS may discover some of the problems, but it cannot solve them and we will need to be sure that we test thoroughly in areas where we believe the requirements were flawed. If we find requirements gaps we can still go back to the users to supplement the RS by revisiting business intent, user expectations and business processes. With these three sources brought up to date we should be able to form a test basis.

Our next step is to look at these additional sources to find a way of enhancing what we have in the RS.

Capturing the business intent

If the business intent is not already documented, has not been documented well or has changed, the first step will be to discuss and document the business intent with the stakeholders. The business intent represents the overall goal of introducing the IS and is a great first step in creating the test basis because it gives UAT a very clear focus on what is needed. The business intent, like the requirements, must be clear, unambiguous and measurable.

Capturing user expectations

We can capture user expectations by talking to users. Our conversations can be helped and supported, though, by a simple technique known as 'user stories'. We can ask users to write down, or help us to write down, user stories that encapsulate their expectations.

User stories

A user story is a very brief description of what the system does for its users to help them perform their work more easily. User stories are used in agile development 'to facilitate the discussion and support planning' (Adzic, 2009: 160). They are a valuable tool for us because they avoid any details of specific requirements or of how they might be implemented. They simply provide a very brief and simple expression of user expectations.

A typical user story is made up of one or two sentences in the everyday language of the end-user that covers what a user needs to do as part of their job function. One common format is to use the formula 'as a .... I want .... so that ....'. This gives a clear link to expected benefits without asking users to articulate any details of specific requirements and is ideal for our purposes.

Example 2.4

As a team member I want to be able to complete a draft contract so that it can be processed.

As a manager, I want to be able to approve a contract created by my team.

The 'three Cs of user stories' (http://www.xprogramming.com/xpmag/expCardConversationConfirmation.htm) provide a good practical basis for capturing and exploring user expectations.

Card
User stories can be written on small index cards; small enough to capture the key idea and not large enough for a complete requirement. The cards can then be used as tokens for discussion.

Conversation
The conversations related to user stories can explore the relationships between stories and, if necessary, add detail for clarification.

Confirmation
Each user story must be verifiable so that we have a way of knowing whether it has been correctly implemented or not.

An exercise in collecting, documenting and discussing user stories can add not only a clear idea of what user expectations are, but also a means of identifying the results we would expect to get from a test based on each user story. This is exactly what we need.

Capturing business processes

We will need to review the business processes and check that they interface correctly with the requirements, for instance that the requirements contain all the functions and accept inputs as defined by the business processes. If necessary we may have to supplement them. We will describe the review technique in greater detail in Chapter 3.

Where business processes interface with the computer we can use the concept of use cases to capture what the software should do for each user activity and then compare that with the RS.

Use cases

Use case

A sequence of transactions in a dialogue between an actor and a component or system with a tangible result, where an actor can be a user or anything that can exchange information with the system.

The definition (see box) describes precisely what we are looking for. It involves a dialogue between an actor and the system that makes up a sequence of transactions with a tangible result. An actor is a representative of some role (such as an end-user who is making a purchase or an end-user who is completing an order); the transactions are the actions accomplished by the system for the actor. So here we have a description of how a user role would interact with the system to achieve some goal (such as placing an order).

As well as actors, use cases are normally associated with scenarios, where a scenario is a sequence of steps representing one specific action. In our case we can build a scenario for the 'normal' process, meaning the scenario in which everything works correctly and the process completes successfully. We can then add supplementary scenarios to explore the 'error' cases, for example where the process diverges from the 'normal' case in some way.

Use cases can be expressed in simple diagrammatic form or as a sequence of numbered steps. For our purposes the latter is probably the better option so that users can work with us as we build out the use cases that represent the business processes and their interactions with the system.

Table 2.4 Use case

Book cruise

1. The agent suggests a travel option for the client
2. The system searches for date options according to the criteria and the suitable options are listed for the agent
3. The agent chooses Select Holiday
4. The system verifies there are cabins available and reserves a cabin
5. The agent supplies payment details to finalise the booking
6. The system books the cabin and issues the ticket

Alternatives

4a. Cabin is not available in the desired class

4a1. The system informs the agent that no cabins are available according to the price criteria

4a2. The agent specifies another price range

4b. Cabin is not available at all

4b1. The system informs the agent that the cruise is fully booked

4b2. The agent specifies another cruise

Example 2.5

In this use case the actor role is the travel agent (an end-user of the system who uses it to sell holidays). The alternative scenario explores what happens when no suitable cabins are available.

This is a simple use case but it is easy to identify business process steps not involving system interaction (step 1) and system interactions (steps 2–6). The interactions can be described in more or less detail, as can the parts of the business process not involving the system. Use cases are potentially much more detailed than user stories and are normally aimed at providing developers with complete requirements.

If use cases have been used in capturing the initial requirements we gain the benefits for very little additional effort. If not, use cases provide one of the simplest and quickest ways to capture key information.

We will see how we can deploy user stories and use cases in setting up a UA test basis in Chapter 6.

CHAPTER SUMMARY

In this chapter we have looked at what business requirements are and how we capture them for the developers and for UAT. We also learned that, in addition to the detailed requirements, the business intent and user expectations ought to be documented so that these three aspects together meet both the goals and expectations of the business.

We learned that there are different types of requirements and the RS contains more than just the requirements. We recognised a number of pitfalls that can cause problems and a number of reasons why requirements, however good initially, rarely make it to the end of a project without becoming out of date. The test basis is created to overcome these issues, producing a valid and up-to-date document from which to generate tests.

Techniques such as user stories and use cases enable us to enhance the RS to provide a current and reasonably complete test basis for UAT.

What have you learned?

Test your knowledge of Chapter 2 by answering the following questions. The correct answers can be found in Appendix B.

1. How could you best define the business intent?

   A. The main reason for and benefit of the new system
   B. The reason for carrying out UAT
   C. The expectations users have of how the system will work
   D. The purpose of the business

2. What are business requirements?

   A. A strategy for UAT
   B. The test basis
   C. A set of statements that describe what the system ought to do
   D. Both (b) and (c) are true

3. What is a limitation of requirements?

   A. No requirements may be recorded
   B. Things have changed since the requirements were written
   C. The requirements may not be recorded in enough detail
   D. All of the above are true

4. Which of the following is a characteristic of an unsuitable requirement?

   A. Too much detail
   B. Too high level
   C. Not technical enough
   D. Focused on the business need not the solution

Some questions to consider (our responses are in Appendix B)

1. You are a stakeholder new to the project and are asked to read the RS to get an idea of what the project is about, but you do not understand some, most or all of the requirements. What would you do?
2. What is the difference between an RS and a UA test basis?