

UR.A An Example of Structure for a Requirements Document

UR.A.1 Introduction

UR.A.1.1 Purpose of the requirements document

UR.A.1.2 Scope of the product

UR.A.1.3 Definitions, acronyms and abbreviations

UR.A.1.4 References

UR.A.1.5 Overview of the remainder of the document

UR.A.2 General Description

UR.A.2.1 Product perspective

UR.A.2.2 Product functions

UR.A.2.3 User characteristics

UR.A.2.4 General constraints

UR.A.2.5 Assumptions and dependencies

UR.A.3 Specific requirements covering functional, non-functional and interface requirements. These should document external interfaces, functionality, performance requirements, logical database requirements, design constraints, system attributes and quality characteristics.

Appendices

Index

This is a generic document structure and organizations should adapt it to their own specific needs. Irrespective of its organization, the standard should define the information which you would expect to find in a requirements document. This may include the following.

UR.A.3.1 An overview of the system and the benefits of developing the system.

UR.A.3.2 A glossary explaining the technical terms used.

- UR.A.3.3 A definition of the services or functional requirements of the system.
- UR.A.3.4 A definition of system properties or non-functional requirements such as reliability, safety, etc.
- UR.A.3.5 Constraints on the operation of the system and the system development process.
- UR.A.3.6 A definition of the system's operating environment and likely changes to that environment.
- UR.A.3.7 Detailed system specifications expressed as system models showing the relationships between system components. These may include data processing models, data structure models, etc. You should only include them if you need to produce a very detailed requirements document.

A requirements document standard must allow for differences between systems. You should be able to omit parts of the document and to add new sections. Part of your standard should therefore be an introductory page which explains allowed variances from the defined standard.

To allow for variants, you may find it helpful to define the standard as a list of stable and variant parts.

To establish this document structure standard, the SOM or delegate should:

- a) Examine the structure of several existing RDs to find common characteristics;
- b) Evaluate the pros and cons of existing RDs with document users to arrive at the best structure for the organization.

UR.B A Checklist to Follow in Gathering and Defining Requirements

Gather Requirements

- UR.B.1 Exercise sensitivity to the organizational and political factors influencing requirement sources;
- UR.B.2 Identify and consult with potential stakeholders i.e. people who benefit from the system including end-users, managers, customers, software developers etc., to uncover whether there are additional specific requirements or constraints;
- UR.B.3 Keep a record of the source(s) where the requirements were derived;
- UR.B.4 Identify and describe any business goals and concerns from which the requirements are based;
- UR.B.5 For a specific requirement, ensure that the view points of existing or potential stakeholders are gathered.

Define requirements

- UR.B.6 Defining the system's operating environment including other interacting hardware and software systems;
- UR.B.7 Studying and defining policies, regulations and limitations that proposed requirements have to consider when applied e.g. company security policies, safety regulations, racing rules etc.
- UR.B.8 Prototyping poorly-understood requirements to demonstrate the facilities the simulated system can provide;
- UR.B.9 Using scenarios (interaction sessions between end-user and system) to explain/define requirements;
- UR.B.10 Defining the business's operating environment and processes the system supports;
- UR.B.11 Reusing requirements wherever possible from other systems within the same application area.

UR.C Guidelines on How the Requirements Document can be Designed to be Easier to Read, Modify & Produce

- UR.C.1 Use wide margins so that text lines are not too long. Psychological research has shown that long lines are more difficult to read than short

lines. Unless you arrange your document in columns, you should not right justify the lines in the document as, again, research has found that documents with a ragged right margin are easier to read (yes, we know the text in this book is right justified; publishers think it looks neater). Wide margins have the additional advantage that readers can make their own document annotations.

- UR.C.2 Use section and sub-section headings with a consistent style for sections and sub-sections. Leave white space around the section and sub-section headings.
- UR.C.3 Use emphasis sparingly and consistently. You can emphasize information by using a bold or italic font or by underlining it. Once you have decided what should be emphasized, always use the same emphasis technique for the same type of information.
- UR.C.4 Use tables, bulleted or numbered lists to present sets of related information items rather than list them within a sentence.
- UR.C.5 Where a number of items of information have to be presented with some stable and some variant parts, use a table to show commonalities and differences.
- UR.C.6 Separate equations from the text using white space and present them using a different font.
- UR.C.7 If you are describing a sequence of events or a sequential process, use diagrams to show the steps in the process. However, you should also provide explanations of the diagrams as people from different backgrounds sometimes interpret diagrams in different ways.
- UR.C.8 Do not use complex diagrams. Complex diagrams become very confusing and you should normally have less than 12 elements in a diagram. It is much better to use several simpler diagrams than a single complex diagram.

Your organizational guides for requirements document layout should be presented as part of your document standards.

- UR.C.9 Modifications become a problem when changes to one part of a document affect other parts of the document. Therefore, when developing requirements documents, you should try to ensure that the different sections of the document are loosely rather than tightly integrated. You should try to minimize explicit relationships between the different parts of the document and design the document in such a way that parts of it may be selectively replaced.

There are some general techniques which may be used to improve the modifiability of a requirements document.

- UR.C.9.1 Produce documents in loose-leaf binders rather than in bound versions. Users may then selectively replace parts of the document. This may not always be possible where documents are externally distributed, but for internal use loose-leaf versions should be maintained.
- UR.C.9.2 Many word processors allow the automatic insertion of change bars to indicate that text has been changed. Where appropriate you should use this facility. However, the problem is that automatic change bars do not distinguish between significant changes and minor changes which do not affect the meaning of a document (e.g. to fix a spelling mistake or remove superfluous white space).
- UR.C.9.3 When writing documents, avoid references to other page numbers in the document. These page numbers will change as the document is modified.
- UR.C.9.4 Ensure that all figures and tables are labelled and always refer to them by that label rather than, for example, 'the table below'.
- UR.C.9.5 Keep chapters short so that entire chapters may be replaced by document users.
- UR.C.9.6 Start new chapters on separate pages. This means that the chapter may be replaced without disrupting the remainder of the document.
- UR.C.9.7 Always number pages relative to chapters, i.e. the first page in chapter 2 is 2.1, the eighth page of chapter 12 is 12.8, etc. Avoid numbers of the form Page X of Y where Y is the total number of document pages. The total page count is liable to change with time.
- UR.C.9.8 If available, use the facilities in your word processor which allow for relative references to figures, tables, etc.

UR.D Guidelines on the Design of Templates for Describing Requirements

To ensure that natural language requirements are readable, you should define some structure for the description of each requirement. This structure should include fields to note the information that you wish to associate with the requirement. For example, if the requirement is a functional requirement, your standard may specify that the inputs, the processing and the outputs must be specified in the requirements definition.

A standard form or template should be defined to record requirements information. These forms should have named fields which remind the analyst which information is to be collected and recorded. These replace unstructured sentences which describe the requirement. This template should be based on the type of system which you are specifying and the specific information needs of your organization. You will need to define several templates to cater for different types of requirement.

Within your requirements template you may wish to include fields to support other guidelines such as the requirements reference number and the rationale for the requirement.

You should provide templates for these forms to all those who are likely to have to collect requirements. You can then be sure that the forms can be supported on all computers in your organization.

However, you should not force analysts to fill in all fields on a form and you should not use special-purpose software which requires the fields in a form to be completed in a particular order. Requirements discovery is a difficult process and different people collect information in different ways. Imposing a single approach is likely to be resisted. Always allow for a 'get out' where analysts may simply note the requirement in any way they wish in some comments field of the form. Pre-defined forms cannot cover everything and there needs to be some flexibility in your system.

Some readers of requirements simply want a high-level description and do not want to read pages and pages of forms. You should allow for this by including fields on the form which may be extracted (automatically if possible) to create a shorter (perhaps incomplete) description of the requirement. Make sure, however, that you maintain links from this shorter description to the complete requirements specification in the standard template. This will then allow you to generate a new short description when changes to the requirement are agreed.

UR.E Guidelines for Defining System Boundaries

To implement this guideline, you must examine each requirement and classify it as either a system requirement, a process requirement or a requirement which should be rejected. Individual judgements will vary and different analysts will partition the requirements in different ways. Customers will also have their own ideas about where the system boundaries lie. Therefore, several people should classify the requirements then negotiate which requirements are outside the system boundary.

This process should be carried out as part of the overall analysis of the requirements. The questions in the analysis checklist may help you make decisions about the system boundary. Other questions, which might be helpful in making decisions about the system boundary, are as follows.

UR.E.1 *Does a requirement imply the need for some decision-making based on incomplete or unreliable information?* If so, consider making this a process requirement, as people are much better than computers at making this kind of decision.

UR.E.2 *Will the implementation of a requirement need information, which is outside the defined database for the system?* If so, you should consider making this a process requirement where the operators of the system are responsible for providing the information. If you do not do this, you need to review the data, which should be managed by the system.

UR.E.3 *Is a requirement concerned with the core functionality of the system?* Most systems are intended to support some critical processes such as command and control for military organizations, controlling aircraft systems for pilots, providing financial information for managers, etc. If the requirement is not concerned with these critical processes, you should consider whether or not it is really needed.

UR.E.4 *Is a requirement concerned with the functionality or performance of equipment, which is external to the system?* In this case, it may be impossible to implement the requirement, as you do not have control over that equipment. The requirement is therefore outside the system boundary.

For the requirements which have been associated with the operational process and which have been classified as outside the boundaries of the system, you must prepare technical and/or economic arguments why these have been excluded. These arguments should be based on the defined business objectives of the organization or on the results of the system feasibility study.

UR.F Guidelines on How Checklists can be Developed and Used

A checklist is a list of questions, which the analyst uses to assess each requirement. Analysts should check items on this list as they read through the requirements document. When potential problems are discovered, these should be noted either in the margins of the document (if your organization permits document annotation) or on a separate analysis list.

This analysis list can be implemented as a spreadsheet where the rows are labelled with the requirements identifiers and the columns are the checklist items. You then fill in the appropriate cell with comments about potential problems.

Checklists are an organizational resource, which must evolve with experience of the requirements analysis process. The questions on the checklist should usually be general rather than restrictive. If the questions are too specific, they will be irrelevant for most systems.

You can create an initial checklist based on the questions shown in Figure UR.F.

Checklist item	Description
Premature design	Does the requirement include premature design or implementation information?
Combined requirements	Does the description of a requirement describe a single requirement or could it be broken down into several different requirements?
Unnecessary requirements	Is the requirement 'gold plating'? That is, is the requirement a cosmetic addition to the system, which is not really necessary?
Use of non-standard hardware	Does the requirement mean that non-standard hardware or software must be used? To make this decision, you need to know the computer platform requirements.
Conformance with business goals	Is the requirement consistent with the business goals defined in the introduction to the requirements document?
Requirements ambiguity	Is the requirement ambiguous i.e. could different people read it in different ways? What are the possible interpretations of the requirement? Ambiguity is not necessarily a bad thing as it allows system designers some freedom. However, it has to be removed at some stage in the development process.
Requirements realism	Is the requirement realistic given the technology, which will be used to implement the system?
Requirements testability	Is the requirement testable, that is, is it stated in such a way that test engineers can derive a test which can show if the system meets that requirement?

Figure UR.F Analysis checklist items

The goal of analysis is to discover requirements problems; it is not a quality management activity (this comes later in the process). Consequently, the checklist should not include general quality questions concerned with conformance to organizational standards, etc.

You should limit the size of checklists to about 10 questions. People cannot hold too many items in their head, so long checklists are useless when reading through a document. Furthermore, long checklists inevitably mean that most questions are irrelevant to most requirements and they will be applied in a perfunctory way. If checklists grow too long, you can consider splitting them to create several checklists. Different analysts can work with different checklists, thus giving complete checklist coverage.

Checklists for analysis can be introduced by organizations at any level of requirements engineering process maturity. Organizations at the initial level should develop short and simple generic checklists with questions like those that we suggest here. More mature organizations can use more detailed checklists for requirements checking.

UR.G Technologies that can be Used for Electronic Negotiations

There are four basic technologies, which can be used for the electronic negotiation of requirements.

- UR.G.1 Electronic mail (e-mail) systems where stakeholders exchange messages about the requirements.
- UR.G.2 Bulletin board systems where problems are posted on a common 'bulletin board' which are accessible to all stakeholders. Discussions of these problems may be organized separately using the bulletin board facilities.
- UR.G.3 Shared database systems such as Lotus Notes where a repository of requirements, problems and comments may be accessed by all involved in the process. These systems are based on a central database plus copies of part of this database held on personal machines. The system automatically supports the synchronization of the central and the local databases.
- UR.G.4 Intranet technologies where Internet software such as World-Wide-Web browsers are used to access an organization's internal data.

Electronic mail systems are now widely used and are the simplest and cheapest approach to electronic requirements negotiation. Stakeholders are identified as a group and requirements problems and comments mailed to all group members. To use them effectively, you need to make one member of the group a problem manager. The problem manager is responsible for keeping track of the problems raised and circulated by e-mail, the responses to the problems and the agreed solutions. Where necessary, he or she must also prompt people for responses to mail messages.

Bulletin boards usually have a fixed structure consisting of a posting plus an associated discussion thread. Problems are 'posted' to the board and a discussion is based on the problems. They may be linked with the electronic mail systems so that posted problems are also circulated by e-mail. These systems allow for a more focused discussion as it is possible to link individual problems with responses to these from different stakeholders. Problems and responses are linked and are visible to all group members. There is less need for problem management, although some moderation to curtail discussions is usually necessary.

Shared database systems or computer conferencing systems can be used in a similar way to bulletin boards. However, they provide more sophisticated data management facilities and can also be used for requirements management. It is therefore possible to integrate problems

and responses with the requirements themselves. Database systems allow for more flexible structures but, of course, these need to be defined in advance.

Intranet systems are a form of shared database system where an organization uses World-Wide-Web browsers to access a shared database, which is structured as a set of hypertext pages. Browsers can be tailored to specific needs by associating scripts with pages, which are automatically executed when that page is displayed. Many companies are implementing these systems for organizational data. This approach is relatively low-cost and easy to use and certainly has a lot of potential. However, experience with Intranets is still limited and we cannot really comment on their cost-effectiveness at this stage.

You need to be connected to bulletin boards to access the information on them. Shared database systems, however, allow information to be transferred to and from a private database copy held on the user's machine. This simplifies independent working. It is generally easier to integrate work from different people.

UR.H Guidelines on How to Prioritize and Classify Requirements

UR.H.1 Prioritization

Ideally, requirements priorities should be assigned during the requirements elicitation process. When expressing requirements, stakeholders should decide how important they are. However, people find it difficult to assign priorities at this stage because they do not have a complete picture of the system requirements. They usually do not completely understand the implications of the requirements, which they propose. Priority assignment is therefore only realistic after a reasonably complete set of requirements has been collected and some preliminary analysis has been carried out. Priorities should be noted in a field on the form used for collecting the requirements.

Priorities should be assigned in discussions between requirements analysts and stakeholders. In many cases, different stakeholders will assign a different priority to the same requirement. This may reflect real needs or may simply reflect different stakeholder perceptions. Try to resolve the differences informally. If this is impossible, the priority conflicts should be discussed and resolved at negotiation meetings.

You should decide on a relatively small number of priority classifications such as 'essential', meaning that it must be included in the system, 'useful' meaning that the system will be less effective without it, and 'desirable' meaning that the facility is not a core system facility but makes the system more attractive to users in some ways (e.g. by providing a graphical interface).

UR.H.2 Classification

The simplest way to classify requirements is to use what is called a faceted approach. You identify a number of dimensions or facets and identify keywords, which describe each of these.

Possible keywords, which you might use to classify requirements, are as follows.

UR.H.2.1 *System.* This should be applied to requirements, which affect the entire system such as performance or reliability requirements.

UR.H.2.2 *User interface.* This should be applied to requirements, which are concerned with user interaction. More detailed

classifications may also be used such as ‘user input’, ‘data presentation’, ‘error indication’, etc.

- UR.H.2.3 *Database.* This should be applied to requirements, which are concerned with the data managed by the system.
- UR.H.2.4 *Communications.* This should be requirements, which are concerned with the external communication facilities in the system. Again, it may be broken down into more detailed classifications reflecting, perhaps, the data communications equipment.
- UR.H.2.5 *Security.* This should be applied to requirements, which are likely to have security implications.

These are possible examples and you must devise the most relevant classifications for the requirement, which you are classifying. We recommend that you should have no more than five or six classifications. If you have more than this, classes tend to be too narrow. You will find that you only have a few requirements in each class.

After deciding on an initial set of classification terms, you should then go through the requirements, associating one or more of these keywords with each of them. You can classify the requirements using as many of the keywords as is necessary. Therefore, a requirement, which states that ‘A graphical user interface to the account data should be provided’, would be classified as ‘User interface’, ‘Database’ and ‘Security’.

Once requirements have been classified, you extract groups of requirements with the same classification terms for comparison and analysis. To do this, you need to use some software, which will allow you to tag each requirement with its classification terms and then construct a request to the system to find all requirements with a given tag. This is possible if you use a word processor system to manage your requirements, but it is simpler if requirements are managed in a database.

UR.I Guidelines on the Use of Interaction Matrices

The simplest way to construct an interaction matrix is to use a spreadsheet program and to label the rows and the rows and the columns of the spreadsheet with the requirement identifiers. Each requirement is then considered and compared with other requirements. You can then fill in values in the spreadsheet cells as follows.

- For requirements which conflict, fill in a 1.
- For requirements which overlap, fill in a 1000.
- For requirements which are independent, fill in a 0.

If you cannot decide whether requirements conflict, you should assume that a conflict exists. If you assume a conflict where none exists then it is usually fairly cheap to fix this problem; it can be much more expensive to resolve undetected conflicts.

The advantage of using numeric values is that you can then sum each row and column to find the number of conflicts (i.e. the remainder when the total is divided by 1000) and the number of overlaps (the total divided by 1000). Requirements, which have high values for one or both of these figures, should be carefully examined as part of the analysis process. A large number of conflicts or overlaps means that any changes to that requirement will probably have a major impact on the rest of the system.

This technique only works when you have a relatively small number of requirements, as it requires each requirement to be compared with every other requirement in the system. The upper size limit is probably about 200 requirements. It can be applied to small systems or to related groups of requirements in a viewpoint or in the same class. We therefore recommend that this guideline should be implemented at the same time or after you have introduced some requirement classification scheme.

A similar approach is used in the QFD (quality-function-deployment) model when considering quality attributes of a system. The application of QFD in software development is described in an article ('Quality Function Deployment: Usage in Software Development'), published in the January 1996 issue of the Communications of the ACM.

UR.J Guidelines on How to Perform Risk Analysis for Requirements

Risk analysis is a difficult process and there is no general method, which is applicable to all types of requirement. You need to involve experienced people to be involved in the analysis process. They use their judgement to decide on the requirement risk.

Risk analysis is particularly important for new systems where there are unknown factors involved. In these cases, requirements risk analysis helps avoid downstream development problems. Requirements which have high risks associated with them are the most likely to cause development difficulties. Identifying these risks at an early stage means that you can look for more information to reduce the risks before the requirements are finalized.

The types of risk, which you should consider, are as follows.

- UR.J.1 *Performance risks.* Implementing the requirement may adversely affect the overall performance of the system.
- UR.J.2 *Safety and security risks.* Implementing the requirement may cause problems in meeting overall system requirements for safety and security.
- UR.J.3 *Process risks.* Implementing a requirement may require changes to the normal development process, such as the introduction of mathematical specifications and proof (for a safety requirement) or the use of unfamiliar prototyping systems (for a user interface requirement).
- UR.J.4 *Implementation technology risks.* Implementing a requirement may require the use of unfamiliar implementation technology, such as AI techniques, the use of N-version programming for fault tolerance, etc.
- UR.J.5 *Database risks.* Implementing a requirement may involve non-standard data, which is not available in an existing system database.
- UR.J.6 *Schedule risks.* Implementing a requirement may be technically difficult and may threaten the planned development schedule for the system.
- UR.J.7 *External risks.* Implementing a requirement involves external contractors.
- UR.J.8 *Stability risks.* The requirement may be volatile and subject to evolution during the development process.

You need to decide on the appropriate risks for each system being specified and assess each requirement against these risks. Risk assessment is imprecise so you should not use a numeric assessment scheme for risks. Rather, you should assess risks using 'fuzzy' categories such as 'high', 'medium' or 'low'.

UR.K Guidelines on Building Complementary System Models

Complementary system models show different aspects of the system specification. System models are usually developed to present either a behavioural view or a structural view of the system specification. A behavioural view models the behaviour of the system in response to stimuli from its environment. These stimuli may be either events (such as the arrival of a message) or input data (such as the availability of a report). A structural model shows how the system or its data are decomposed. You should normally develop at least one behavioural and one structural model.

Examples of the different types of system model, which might be produced as part of the analysis process, are as follows.

- UR.K.1 *A data-processing model.* Data-flow diagrams may be used to show how data is processed at different stages in the system.
- UR.K.2 *A composition model.* Entity-relation diagrams may be used to show how some entities in the system are composed of other entities.
- UR.K.3 *A classification model.* Object class/inheritance diagrams may be used to show how entities have common characteristics.
- UR.K.4 *A stimulus-response model.* State transition diagrams may be used to show how the system reacts to internal and external events.
- UR.K.5 *A process model.* Process models may be used to show the principal activities and deliverables involved in carrying out some process.

Other types of model such as timing models may also be required for some types of system such as telecommunication systems.

There is no ideal set of system models. The set of models, which is best for your application, depends on the following things.

- *The type of information that you need to specify.* The value of system models is that they can add more detailed information to a natural language specification. Models should be developed when detailed specification cannot be left until the system is designed. Therefore, if parts of the system exchange data, you should develop a detailed structural model of that data; if a particular sequence of processes must be followed in response to an input, you should define a data processing model to show that process sequence.
- *Likely readers of the system models.* Models must be expressed in a notation, which is understandable, by the people who need to read them. Although a Petri net model may be a good way to show concurrent processing, there is no point in

developing this if only a few people in your organization understand this notation.

- *Skills of the model developers.* A bad model is probably worse than no model. You should not attempt to develop system models, which are outside the experience of your requirements engineers.
- *The availability of CASE tools.* CASE tools reduce the time and effort needed to produce system models. If you do not have tool support for some types of system modelling notation then you will probably find that it is impractical to develop models in these notations.

System models are usually represented graphically. They show system or problem entities and the relationships between them. Graphical representations are used as these are often easier to understand than textual descriptions and are good for showing relationships between the system entities.

You can use an informal approach to system modelling where you define your own notation (often simple boxes and lines) to describe the system specification. This is particularly appropriate for fairly high-level models in stakeholder specifications. These present a system overview and leave details of the system unspecified. Alternatively, you may use the techniques and tools defined as part of a structured method of analysis for model description.

Graphical models often appear to be easy to understand but they do suffer from some problems

- Graphical models are imprecise and may be interpreted in different ways by different people.
- Most graphical notations, which are used for system modelling, do not include facilities for specifying how exceptions should be handled. Models are a good way of presenting an overall picture of 'normal' operation but graphical models become very cluttered when exception information is included.
- Graphical notations are not standardized. Readers of one notation may misinterpret diagrams in another notations because they make assumptions about the meanings of graphical symbols.
- We do not have useful notations for modelling non-functional requirements or relating these non-functional requirements to models, which specify the system. Many non-functional requirements apply to the system as a whole rather than to identifiable system components. This is almost impossible to illustrate on a graphical model.

Because of these problems, some people advocate developing system models using formal mathematical notations. These notations are less ambiguous than informal

graphical notations so the system models are a precise system specification. This is particularly important for systems, which have rigorous reliability, safety or security requirements.

UR.L Guidelines on Building the Environmental System Model

The environmental model is a model of the context in which the system is used. It should include the following things.

- UR.L.1 Other systems which are directly interfaced to the system being specified. This includes systems which already exist and planned systems which are being developed at the same time.
- UR.L.2 Other systems (where known) which may co-exist with the system and the possible interactions between them. For example, it may be assumed that an electronic mail system is always available to end-users and that information may be cut and pasted from mail messages into the system.
- UR.L.3 The business processes in which the system is used. This may simply be the name of the process or it could be a detailed process description. This depends on the processes and on the familiarity of the specification readers with these processes.

Normally, environmental models are defined using informal block diagrams. The principal components of the environment are shown in boxes and the main relationship between the system and these components are illustrated as lines linking these boxes. Of course, the boxes and lines need to be annotated with additional information to add detail to the model. If you use a data dictionary, the names used in the environmental model should be added to it.

When you define business processes as part of the environmental model, you should define the activities involved, their inputs and outputs, the people responsible for these processes and the software, which they use for process support.

Environmental models should be one of the first system models, which you develop. In some cases, these models may be developed as part of the requirements elicitation process. The environmental model can be used during requirements elicitation and analysis to help decide whether or not a requirement is within the scope of the system.

The process of producing an environmental model involves the following.

- *Finding out about the environment by consulting system stakeholders.* This can be a surprisingly difficult and time-consuming task. The people involved in maintaining the system's environment may not be interested in the system being developed; they may not have time to provide the environmental information, which you need. Some people may not wish to discuss their work because they are fearful of change and consider the new system to be a threat to them. If the system is a new system, the environment itself may not be well defined.

- *Deciding what to include and what to leave out of environmental models.* The environment of a system is usually very complex. There are often subtle, implicit relationships between different parts of that environment. It can be difficult to decide whether or not parts of the environment are relevant for the system being developed. Decisions about the system's environment may not be finalized and the environmental information may be unstable.
- *Producing an initial environmental model.* The initial model should reflect your understanding of the system's environment. You will almost certainly get this wrong but you need some kind of model to discuss with system stakeholders.
- *Analyzing and refining the model.* Once you have an initial model, you need to go back to the system stakeholders and refine this model. Several iterations may be necessary before you have a complete model of the environment.

The process is iterative and you will probably have to repeat these steps a number of times before the final environmental model is established.

UR.M Guidelines on Building the Architectural System Model

Architectural models are usually presented as simple block diagrams showing the main sub-systems and the relationships between them. Main sub-systems are those parts of the system, which can exist as independent entities. They take information from and provide information to other sub-systems. For example, a car might have a braking sub-system, a steering sub-system, an engine management sub-system, etc.

Sub-systems are represented as boxes in a block diagram and relationships between the sub-systems as lines joining these boxes. Different types of relationship may exist, e.g. a sub-system may pass information to another sub-system, they may share the same processor, etc. You need to be careful to distinguish between these in your architectural model, perhaps by using different styles of line to link the sub-system boxes. For large systems, each sub-system is itself a substantial system in its own right and you may need to develop separate architectural models of each of these.

Designing the architecture of a system is a creative activity. Requirements analysts and engineers must use their general knowledge of similar systems and architectural patterns to design an appropriate system architecture. There are a number of 'standard' architectural patterns, which are commonly used. You may find it helpful to develop architectural models using these patterns. Examples of architectural patterns, which have been identified in software systems, include:

- UR.M.1 client-server systems where shared system facilities are provided via general-purpose servers, and user facilities are provided on client systems which call on servers when information is required
- UR.M.2 layered systems where system facilities are provided by calling on facilities offered by a lower-level layer
- UR.M.3 repository-based systems where sub-systems communicate through a shared repository
- UR.M.4 pipeline systems where each element in the system carries out some computations and passes on information to another element for further processing.

The characteristic of all of these patterns is some kind of separation between parts of the system. This makes it easier to incorporate system changes as these changes may be confined to one part of the architecture.

You might think that the system architecture is a design issue and that it is premature to model the system architecture as part of the requirements engineering process. We disagree with this.

- Many system engineering problems have resulted because the requirements as specified could not be mapped onto a coherent system architecture, which was resilient to requirements change.
- For some systems, the system architecture may itself be a requirement. This may be due to organizational policies on hardware utilization, the need to achieve a given level of system reliability or the constraints imposed by other systems. For example, systems which have high reliability requirements may have to be developed using *N*-version programming and this immediately imposes a system architecture.

UR.N Guidelines on How to Apply Systematic Approaches and the Use of the Data Dictionary in System Modelling

UR.N.1 Systematic Approaches

Systematic Approaches mostly fall into two categories.

UR.N.1.1 Procedural methods where the behaviour of the system is represented by functional models of some kind. Data structures are defined using entity-relation models.

UR.N.1.2 Object-oriented methods where object models which combine structural and behavioural information are produced

The vast majority of systematic approaches that are in use are procedural methods. These were developed in the 1970 and came into widespread use during the 1980s. There are many analysts trained in their use and mature toolsets to support these methods. Object-oriented methods are more recent but are becoming increasingly widely accepted, as systems are developed using object-oriented programming languages such as C++. Object-oriented methods are particularly appropriate for modelling interactive systems.

The main components of a systematic approach are:

- A set of recommended system models which should be developed and a defined notation which should be used to develop these models
- A set of model rules which should be applied both to individual models and the overall model formed by integrating the different individual models; these may range from simple rules regarding the use of names to complex properties which complete models should satisfy
- A set of good practice guidelines for producing high-quality system models
- A description of the process which should be followed to create a system model using the method
- Reports on the system specification which should be produced during the application of the method

Methods are not universally applicable and different methods are appropriate for different types of system and application domains. For

example, methods such as SSADM and Structured Analysis are best for modelling commercial data processing systems, Statecharts for real-time systems, and SDL for telecommunication system modelling. Object-oriented methods such as OMT and Objectory are sometimes claimed to be more universal. Frankly we are sceptical about this. These methods are still relatively new and we do not have enough experience of their use in large systems to know their limitations.

Many organizations invested heavily in systematic approach and associated CASE tools in the 1980s. They anticipated that this would result in significant increases in analyst and designer productivity and system quality. Although some companies reported very significant improvements, the majority of companies found that the benefits of systematic approaches had been oversold. They now use structured methods where appropriate and do not insist on their use for all projects.

UR.N.2 Data Dictionary

Data dictionary software is an integral part of many CASE systems. If you use these CASE systems for developing and documenting the system model, the names are automatically entered in the data dictionary and you can check the dictionary when new names have to be invented. However, these systems may make assumptions about what information should be maintained in a data dictionary. They may not allow you to put the information which you need into the dictionary. If you do not use a CASE toolset, data dictionaries are simple to implement using commercial database systems on workstations or PCs.

Each name used should be entered as a record in the data dictionary. At least the following information should be maintained.

- UR.N.2.1 The defined name of the entity in the model. This should be the key to the data dictionary.
- UR.N.2.2 Any aliases or alternative versions of the name which are used. It is best to avoid aliases if possible but sometimes you need to use them. For example, if you use versions of Microsoft Windows before Windows 95, your file names are restricted to eight characters. You might have a logical file name used in the system model and an alias for it, which conforms to the file system standard.
- UR.N.2.3 The type of the named entity, e.g. process, object, attribute, etc.
- UR.N.2.4 A description of the named entity and why it has been introduced in the system model.

- UR.N.2.5 Any constraints on the named entity which are known. These can relate to the use of the entity, its structural representation, its lifetime, etc.
- UR.N.2.6 Links to related entities. This improves the traceability of the system. When the meaning of a name has to be changed, the related entities should be checked to see if this change affects them.

The data dictionary is a shared resource so it must be accessible to everyone who is developing or using the system model. This means that one of the following two things must happen:

- It must be maintained on a server computer, which is easily accessible to everyone developing the system model without having to leave the model development system. This is the approach, which we recommend.
- It must be copied daily from a central system to and from the personal computers of the model developers. This approach may have to be used if your model development tools do not support access to a shared server system or where development is taking place at remote sites connected only by slow communications links. To implement this approach, you need software which will integrate changes to the data dictionary, detect name clashes and report these to model developers. Lotus Notes may be used for this purpose, as it has been designed to support remote working on a shared database.

General-purpose database software such as Filemaker Pro or Microsoft Access may be used to implement your data dictionary. These are adequate for small or medium-sized system models. There are also special-purpose data dictionary systems available for mainframe computers. These are often provided with database systems such as ORACLE. They may be appropriate if you use the host database for requirements management. Powerful systems of this type must be used if you have a very large number of names to maintain.

UR.O Guidelines for Checking that the Requirements Meet Organizational Standards

An analyst or an engineer who is familiar with the requirements standards but who has not been involved in the system requirements specification should be responsible for this initial standards check. It is not necessary for the document checker to understand the requirements in detail.

The checker should compare the structure of the requirement document to the defined standard and should highlight missing or incomplete sections. The search and outline facilities, which were provided with most word-processing systems, may be used to find parts of the document and display the document structure. If you have a standard for individual requirements, the checker should also check each requirement for compliance with that standard.

This initial check should also check that all pages in the document are numbered, that all diagrams and figures are labelled, that there are no requirements, which are unfinished or labelled 'To be completed' and that all required appendices in the document have been completed.

After this process has been completed, there are two possible options if deviations from the standard are found.

- Return the document to the requirements engineering team to correct deviations from the standards. This option should be chosen if there is enough time to allow for a re-issue of the document.
- Note the deviations from the standards and distribute this to document reviewers. This saves the time and cost of creating a new version of the requirements document. However, the deviation from standards may increase the difficulty of the requirements review process. This option should be chosen when there is a tight deadline for the requirements review or when the deviations from the standard are minor, easily correctable and do not affect the understandability of the document.

UR.P Guidelines for Organizing Formal Requirements Inspection Meetings

The requirements inspection is a formal meeting. It should be chaired by someone who has not been involved in producing the requirements, which are being validated. During the meeting a requirements engineer presents each requirement in turn for comment by the group and identified problems are recorded for later discussion. One member of the group should be assigned the role of scribe to note the identified requirements problems.

A formal requirements inspection should involve a team of inspectors from different backgrounds, who read the requirements document and record problems with the system requirements. The document reviewers may use a checklist (see Appendix 6.17) to help focus their attention on particular aspects of the requirements and the requirements document.

Unlike program inspections where errors are simply reported to the program author for correction, requirements inspections involve the group making some decisions on actions to be taken to correct the identified problems. Actions that might be decided for each problem are as follows.

- UR.P.1 *Requirements clarification.* The requirement may be badly expressed or may have accidentally omitted information, which has been collected during requirement elicitation. The author should improve the requirement by rewriting it.
- UR.P.2 *Missing information.* Some information is missing from the requirements document. It is the responsibility of the requirements engineers who are revising the document to discover this information from system stakeholders or other requirements sources.
- UR.P.3 *Requirements conflict.* There is a significant conflict between requirements and the stakeholders involved must negotiate to resolve the conflict.
- UR.P.4 *Unrealistic requirement.* The requirement does not appear to be implementable with the technology available or given other constraints on the system. Stakeholders must be consulted to decide whether the requirement should be deleted or modified to make it more realistic.

A good practical book on formal inspection is *Software Inspection*, by T Gilb and D. Graham (Addison Wesley, 1993). Although this is mostly concerned with program inspections, most of the advice is equally applicable to requirements validation.

UR.Q Guidelines for Defining and Using Checklists/Questionnaires

The use of checklists for requirements analysis has already been discussed in Appendix 6.6, *Use Checklists for Requirements Analysis*, and similar checklists may also be used in the validation process. These checklists are oriented towards individual requirements; validation checklists should also be concerned with the quality properties of the requirements document as a whole and with the relationships between individual requirements. This cannot be checked during requirements analysis, as the requirements document is unfinished at that stage.

Questions which might be included in such a checklist should be based on the following general issues.

- UR.Q.1 Are requirements complete, that is, does the checker know of any missing requirements or is there any information missing from individual requirement descriptions?
- UR.Q.2 Are the requirements consistent, that is, do the descriptions of different requirements include contradictions?
- UR.Q.3 Are the requirements comprehensible, that is, can readers of the document understand what the requirements mean?
- UR.Q.4 Are the requirements ambiguous, that is, are there different possible interpretations of the requirements?
- UR.Q.5 Is the requirements document structured, that is, are the descriptions of requirements organized so that related requirements are grouped? Would an alternative structure be easier to understand?
- UR.Q.6 Are the requirements traceable, that is, are requirements unambiguously identified, do they include links to related requirements and to the reasons why these requirements have been included? See 6.23 for guidelines on traceability.
- UR.Q.7 Does the requirements document as a whole and do individual requirements conform to defined standards?

Checklists should be expressed in a fairly general way and should be readily understood by people such as end-users who are not system experts. As a general rule, checklists should not be too long. Checklists should not normally have more than ten items. If you have more than this, checkers cannot remember all items and must continually re-consult the checklist.

The danger, of course, is that checklist becomes too vague and it is impossible to answer the checklist questions in any useful way. You have to find the right balance

between generality and detail. Unlike program inspections, low-level checklists concerned with very specific faults are not so good for requirements inspections because of the differences between the requirements for different types of system.

Checklists can be distributed and simply used as a reminder of what people should look for when reading the requirements document. Alternatively, they can be used more systematically where, for each requirement, an indication is given that the checklist item has been considered. This may be done on paper or you can manage this type of checklist completion by using a simple database or a spreadsheet. The checklist items are shown along the horizontal axis and the requirements along the vertical axis. The checker should fill in each cell with an appropriate comment. You should not force checkers to use an automated approach, as people may read the document in places and at times when they do not have computer access.

UR.R Guidelines for Prototyping System

Prototypes for validation must be more complete than elicitation (ie gathering and defining) prototypes. Elicitation prototypes can simply include those requirements which are particularly difficult to describe. They may leave out well-understood requirements. While a validation prototype need not include all system facilities, there must be a sufficient number of facilities implemented in a reasonably efficient and robust way that end-users can make practical use of the system. Otherwise, the validation activity will be unnatural. You also need to ensure that the validation prototype includes installation instructions, instructions on how to recover when things go wrong and some end-user documentation describing how to use the system.

During elicitation, the requirements engineer and the end-user may work together, whereas during validation, end-users are more likely to work alone, experimenting with the prototype. If possible, however, requirements engineer should spend some time observing how end-users make use of the system. This can reveal particular problem areas. You can also see if the end-user has developed coping strategies for dealing with system features which they find useful but which are awkward to use in their current form.

You should design the prototype trials so that they do not take too long. End-users will normally have other things to do apart from experiment with the prototype system. Therefore, the system must be reliable. It should not crash and, if there are facilities missing, you must tell users about these in advance.

Unstructured end-users experimentation with the system is unlikely to provide complete coverage of the system. If possible, you should select end-users who do different jobs so that different areas of system functionality will be covered. You should provide end-users with a problem report form (paper or electronic) that they use to record their problems and what they were doing when these problems were discovered. Ideally, you should use scenarios for validation where users work through a defined scenario and discover if it is a correct representation of how they work.

UR.S Develop Requirements Test Cases

For each requirement in the requirements document, you should analyze that requirement and define a test, which can objectively check if the system satisfies the requirement. The objective of proposing test cases for requirements is to validate the requirement rather than the system. You need not be concerned with practicalities such as testing costs, avoiding redundant tests, detailed test data definition, etc. Therefore, you do not have to propose real tests which will be applied to the final system. You can make any assumptions you wish about the way in which the system satisfies other requirements and the ways in which the test may actually be carried out.

To define the test cases, you can ask the following questions about a requirement.

UR.S.1 What usage scenario might be used to check the requirement? This should define the context in which the test should be applied.

UR.S.2 Does the requirement, on its own, include enough information to allow a test to be defined? If not, what other requirements must be examined to find this information? If you need to look at other requirements, you should record these as there may be dependencies between requirements, which are important for traceability (see Chapter 9).

UR.S.3 Is it possible to check the requirement using a single test or are multiple test cases required? If you need several tests, this may mean that there is more than one requirement embedded in a single requirement description.

UR.S.4 Could the requirement be re-stated so that the required test cases are fairly obvious?

You should design a test recording form, which you fill in for each requirement checked. This should include at least the following information:

UR.S.4.1 the requirement identifier

UR.S.4.2 related requirements

UR.S.4.3 a brief description of the test which could be applied and why this is an objective requirements test

UR.S.4.4 a description of requirements problems which made test definition difficult or impossible

UR.S.4.5 recommendations for addressing requirements problems which have been discovered.

Test case forms may be implemented as electronic documents and managed in a database with links to the associated requirement. Ideally, forms should be electronic but you must also allow for hand-written forms and, perhaps, budget for secretarial support to type these forms. Some people find it more convenient to review and check documents away from a computer (e.g. while travelling) and you should accommodate this style of working.

UR.T Guidelines for Paraphrasing System Models

Converting a system model to natural language text should be done in a systematic way. The actual technique used should depend on the type of system model, but we recommend using some kind of form or table where different components in the model are described in different fields or columns. For example, in a data-flow diagram, you might use a template with the following fields to describe each transformation.

- UR.T.1 *Transformation name.*
- UR.T.2 *Transformation inputs and input sources.* Gives the name of each input to the transformation and lists where that input comes from.
- UR.T.3 *Transformation function.* Explain what the transformation is supposed to do to convert inputs to outputs.
- UR.T.4 *Transformation outputs.* Gives the name of each output and lists where the output goes to.
- UR.T.5 *Control.* Any exception or control information which is included in the model.

In some cases, it may be possible to partially automate this process. Some CASE tools will generate reports on the system models, which can be a starting point paraphrasing. However, these are normally fairly stilted and you have to modify them to make them more readable and add information, which cannot be automatically derived. You may find that it is actually less effort to ignore the generation facilities and simply work directly from the system model.

It is important to avoid reading information into the model, which is not there. Therefore, the paraphraser should not try to interpret the model or provide rationale for model elements. Ideally, the paraphraser should be familiar with the type of system being specified but should not have been involved in the development of the specification.

If you wish to introduce formal specification techniques, we strongly recommend that you should implement this guideline. Formal models of systems are usually only accessible to specialists. They must be translated into natural language if you wish end-users and domain experts to be involved in the model validation.

UR.U Guidelines on How to Define Policies for Requirements Management

Requirements management policies are a basis for the quality management of the system requirements. Policies are not the same as standards but are clearly related to them. Policies, loosely, set out what should be done; standards describe how the policy should be implemented in a particular situation. Some policies may be reflected in organizational standards but individual project managers are sometimes the best judges of how policies should be implemented.

Organizations should define a general set of requirements management policies. For each project, you must look at this general set of policies and select those which are relevant to that project. They may be amended in some ways to suit the specific needs of the project.

General requirements management policies should include:

- UR.U.1 a set of objectives for the requirements management process and rationale associated with each of these objectives
- UR.U.2 the reports which should be produced to make the requirements engineering process visible and the activities which are expected to produce these reports as deliverables
- UR.U.3 the standards for requirements documents and requirements descriptions which should be used
- UR.U.4 change management and control policies for requirements
- UR.U.5 requirements review and validation policies
- UR.U.6 relationships between requirements management and other system engineering and project planning activities
- UR.U.7 traceability policies which define what information on dependencies between requirements should be maintained and how this information should be used and managed
- UR.U.8 criteria when these policies can be ignored; in these situations, managers use their own judgement on how to implement a requirements change.

Of course, it is unrealistic to expect to define and introduce all of these policies at the same time. They should be developed incrementally and updated after you have practical experience of their application in requirements management.

You must build some flexibility into your policies otherwise they will be unworkable. There is sometimes such an urgent need for system change, that it is impossible to follow through a systematic requirements management policy before implementing that change. You must have some mechanism for managing these exceptions. It

should suggest how you maintain the information, which can eventually be used to update the requirements that are being managed.

UR.V The Use of a Database for Storing Requirements Information

The most appropriate way to implement this guideline depends on the type and the number of requirements, which must be managed and the particular ways of working in your organization. Issues which influence design choices are as follows.

- UR.V.1 How are requirements expressed? Do you use natural language, graphical models, mathematical expressions, etc.? Do you need to store additional information such as photographs as part of the requirements rationale? If you need to store more than just simple text, you may need to use a database with multi-media capabilities.
- UR.V.2 How many requirements do you typically need to manage? Is it tens, hundreds, thousands or tens of thousands? The requirements for a small to medium sized system can be managed using commercial PC databases. Larger systems usually need a server-based system and a database, such as ORACLE, which is designed to manage a very large volume of data.
- UR.V.3 Are the requirements always developed and managed by teams which work together at the same site and which use the same types of computers or do you need to access the requirements database from different sites? If you need multi-site access from different types of equipment, you should be using an Intranet-based solution that provides access to the requirements database through a WWW browsing system.
- UR.V.4 Do you already use a database for software engineering support? Do you have a company policy on database use? What types of computer do you use? These factors obviously constrain the type of database that you can use.
- UR.V.5 What in-house database expertise do you have available? Will requirements engineers be responsible for database administration or will this be a separate responsibility? Database management costs can be high and a solution, which requires a specialist database administrator, may not be cost-effective.

If you maintain your requirements in a database, you can design the requirements database to include traceability information. With each requirement in the database, you should include at least two fields for traceability information. These should be filled in with references to other requirements, which the requirement depends on and with references to requirements, which are dependent on that requirement. Obviously, if you wish to maintain other types of traceability information such as requirements-architecture links, you must include database fields to record information about each different type of relationship.

Using the database system has the advantage that the database itself usually includes facilities for browsing and report generation. You can browse the requirements in the database then move immediately to related requirements. You may also write simple scripts that scan the database and generate the specific traceability information which you require.

Relational databases are now the most commonly used type of database. Relational databases were designed for storing and managing large numbers of records, which have the same structure and minimal links between them. A requirements database, however, may have relatively few records (hundreds rather than hundreds of thousands) each of which includes many links such as links to documents, text files and other requirements. Maintaining these links is possible with a relational database. However, it is inefficient, as it requires operations on several different tables. For very large numbers of requirements, you may find that this type of database is too slow.

To use a relational database, you may have to keep the text of the requirements in a separate file or files and access these files through the database. The database tables (which use the requirements identifiers as keys) are used to index requirements and to store links between related requirements. You find linked requirements using the relational join operation.

Object-oriented databases have been developed relatively recently and are structurally more suited to requirements management. They allow different types of information to be maintained in different objects and managing links between objects is fairly straightforward. However, this type of database is still immature and low-cost object-oriented databases are not widely available. They are also probably more expensive to manage than relational databases.

If you have a very large number of requirements, you will need to use a powerful database management system such as ORACLE. This might be organized as a database server on a workstation with access through other client workstations or PCs. This is an expensive approach but the power of these systems allows for fast database manipulation. They also usually have excellent report generation tools, which can be used to create skeletal requirements documents from the requirements database. These databases can support many simultaneous users and provide good facilities for backup and recovery in the event of system failure.

Lower-cost approaches are possible using simpler PC database systems. Modern PC databases include facilities for managing multimedia information such as diagrams and photographs. Some of them can use object sharing facilities (such as OLE) to link directly to word processor files where the requirements text is stored. Their support for simultaneous access may be limited to locking the whole database when it is in use and they have limited facilities for error recovery. The requirements database must provide shared access and access control to the requirements.

Whatever database you use, some data administration will be required. The data administrator must set up and manage the database scheme. He or she should also be

responsible for executing backup and recovery procedures and providing support to database users. Database administration can take up quite a lot of time; a large requirements database may need several hours per week of a professional database administrator's time.

If you require access to your requirements database from remote sites and using a variety of different computers, you might consider using WWW-based information system as a front-end for a requirements database server. At the time of writing, various off-the-shelf solutions of this type have just become available. We do not know of any experience reports of using this type of information system for requirements management but it is an interesting development, which has significant future potential.

A requirements database may be used in conjunction with special-purpose CASE tools for requirements management. There are now several of these tools on the market, ranging from simple PC systems to large and complex workstation-based tools intended to manage large requirements databases. These tools provide built-in traceability support and some of them support the automatic creation of requirements databases from a natural-language requirements document. We do not endorse any specific tool but include details of tool suppliers in the book's WWW pages.

UR.W Guidelines on How to Write Traceability Policies

Traceability policies are written policies, which should define the following.

- The traceability information which should be maintained. This is described in more detail below.
- The techniques which may be used for maintaining traceability. This is described in more detail below.
- A description of when the traceability information should be collected during the requirements engineering and system development processes. You should also define the roles of the people, such as the traceability manager, who is responsible for maintaining the traceability information.
- A description of how to handle and document policy exceptions, that is, when time constraints make it impossible to implement the normal traceability policy. Realistically, there will always be occasions where you have to make changes to the requirements or the system without first assessing all change impacts and maintaining traceability information. The policy exceptions should define how these changes should be sanctioned. The traceability policies should also define the process used to ensure that traceability information is updated after the change has been made.

Traceability policies should be written so that they are independent of any particular system. As part of your quality planning process, you should select the most relevant traceability policies and tailor them to the specific needs of the system, which is being specified. These should then be defined in the system traceability manual.

Maintaining traceability information is expensive because it involves managing large volumes of information. Requirements changes may involve making changes to this traceability information in several places to record new or changed dependencies. You must be realistic in defining your traceability policies and you should not make them too bureaucratic. It is better to have light-weight policies, which are followed rather than more comprehensive traceability policies that are ignored by project managers.

This guideline may be implemented by organizations at any level of requirements engineering process maturity. Simple traceability of requirements to their sources and other requirements may be implemented by organizations at the basic level in the requirements engineering process maturity model. As organizational maturity increases, more complex traceability policies may be introduced.

UR.W.1 Traceability information

There are different types of traceability information that you might want to maintain. These are shown in Figure 6.23.1.

Traceability type	Description
Requirements-sources traceability	Links the requirement and the people or documents which specified the requirements
Requirements-rationale traceability	Links the requirement with a description of why that requirement has been specified.
Requirements-requirements traceability	Links requirements with other requirements which are, in some way, dependent on them. You should always try to maintain this type of information.
Requirements-architecture traceability	Links requirements with the sub-systems where these requirements are implemented. This is particularly important where sub-systems are being developed by different sub-contractors.
Requirements-design traceability	Links requirements with specific components in the system, which are used to implement the requirement. These may be hardware or software components. It is particularly important to maintain this type of information for critical systems.
Requirements-interface traceability	Links requirements with the interfaces of external systems, which are used in the provision of the requirements. Should be maintained where there is a high dependency on other systems.

Figure UR.W.1 Types of traceability information

There are three basic techniques, which may be used to maintain traceability information. These are as follows.

- UR.W.1.1 *Traceability tables.* A cross-reference matrix is produced where the entries in the table indicate some kind of traceability link between the items in the rows and the items in the columns.
- UR.W.1.2 *Traceability lists.* Each requirement has a list of associated traceability information.
- UR.W.1.3 *Automated traceability links.* The requirements are maintained in a database and traceability links are included as fields in the database record.

UR.W.2 Traceability tables

Traceability tables show the relationships between requirements or between requirements and design components. The requirements are listed along the horizontal and vertical axes and relationships between requirements are marked in the table cells. They can be implemented using word processor or spreadsheet tables; a requirements database is not necessary.

Traceability tables for showing requirements dependencies should be defined with requirement numbers used to label the rows and columns of the table. In the simplest form of traceability table you simply put a mark such as an * in the table cell where there is some kind of dependency relationship between the requirements in the cell row and column. That is, if the requirement in row X (say) depends on the requirement in columns P, Q, and R, you should mark table cells (X, P), (X, Q), and (X, R). By reading down a column, you see all requirements, which depend on a requirement; by reading across a row, you see all requirements, which the requirement in that row depends on.

Depends-on

	R1	R2	R3	R4	R5	R6
R1			*	*		
R2					*	*
R3				*	*	
R4		*				
R5						*
R6						

Figure UR.W.2 A simple traceability table

The figure above shows a very simple system with six requirements.

Each row in the table shows dependencies so that R1 is dependent on R3 and R4, R2 is dependent on R5 and R6, etc. Therefore, if a change to R4 is proposed, we can see by reading down the R4 column that requirements R1 and R3 are dependent requirements. You should therefore assess the impact on R1 and R3 of the proposed change to R4.

You can extend the simple model of traceability tables by distinguishing between the types of relationship between requirements and by indicating each of these, using a different symbol, in each table cell. Possible relations between requirements, which might exist, are as follows.

- UR.W.2.1 specifies/is-specified-by. This relation indicates that some requirement B adds detail to another requirement A. For example, if A is a general security requirement which states that data should be encrypted, B might specify the characteristics of the encryption algorithm which should be used.
- UR.W.2.2 Requires/is-required-by. This relation indicates that some requirement B requires the result provided by some other requirement A. For example, A might specify that the system should maintain a record of the current time and date in some specific format; B might specify that each transaction processed by the system should be date stamped.
- UR.W.2.3 Constrains/is-constrained by. This relation indicates that some requirement B is constrained by some other requirement A. For example, B might specify that some real value should be displayed and A might state that all real numbers should be implemented as 12-digit fixed-point numbers.

If you have a relatively small number of requirements (up to 250, say), you can implement and manage traceability tables using a spreadsheet. Traceability tables become more of a problem when you have hundreds or thousands of requirements as you end up with very large and thinly populated tables. Sometimes, dependencies between requirements are confined to requirements groups and you can create separate traceability tables for these groups. Dependencies across groups can be specified separately. If this is not the case, we recommend that other traceability techniques such as traceability lists should be used.

A common error, which arises with the use of traceability tables, is confusion between the rows and columns. Therefore, rather than filling in Row R1 which shows that R1 depends on R3 and R3, column R1 is filled in showing that R3 and R4 are dependent on R1. Obviously, these do not mean the same thing and can cause problems to users of the traceability tables.

UR.W.3 Traceability lists

Traceability lists are a simplified form of traceability tables where, along with each requirement description, you keep one or more lists of the identifiers of related requirements. Traceability lists are more compact than traceability tables and do not become as unmanageable with large

numbers of requirements. They are probably less prone to error than traceability tables.

Requirement	Depends-on
R1	R3, R4
R2	R5, R6
R3	R4, R5
R4	R2
R5	R6

Figure UR.W.3 A traceability list

Traceability lists are simple lists of relationships, which can be implemented as text or as simple tables. Figure 6.23.3 shows a traceability list for the dependencies shown in Figure 6.23.2.

You may use several lists, one for each type of relationship such as *requires*, *is-required-by*, *specifies*, etc., or may simply keep a single list of related requirements. The disadvantage of these lists compared to traceability tables is that there is no easy way to assess the inverse of a relationship. We can easily see that R1 is dependent on R3 and R4 but, given R4, we must look through the whole table to see which requirements depend on it. If you wish to maintain this 'backward-to' information, you need to construct another table showing these relationships

Traceability lists can either be maintained manually as word processor or spreadsheet tables or created from the requirements database. To create the list automatically, the database record for each requirement must include a link to a list of related requirements.

UR.X Guidelines on How to Maintain a Traceability Manual

The traceability manual is a central record of the traceability policies for a specific project and all of the relevant traceability information. Your general traceability policies should be specialized to take into account the characteristics of the project. This may involve leaving out some traceability information, deciding on exactly how traceability information should be represented, deciding on the responsibilities for traceability information collection, etc.

The specific traceability policies, which should be used for a project, depend on a number of factors. These factors include the following.

- UR.X.1 *Number of requirements.* The greater the number of requirements, the more the need for formal traceability policies. However, if there is a very large number of requirements, you have to be realistic about what traceability policies can be implemented in practice. Complete requirements-design traceability is impractical for most large systems.
- UR.X.2 *Estimated system lifetime.* More comprehensive traceability policies should be defined for systems, which have a long lifetime.
- UR.X.3 *Level of organizational maturity.* Detailed traceability policies are most likely to be cost-effective in organizations, which have a higher level of process maturity. Organizations at the basic maturity level should focus on simple requirements – requirements traceability.
- UR.X.4 *Project team size and composition.* With a small team, informal discussions between team members may be all that is required to assess the impact of proposed changes. With larger teams, however, you need more formal and detailed traceability policies. This is particularly true if all members of the team are not located together.
- UR.X.5 *Type of system.* Critical systems such as hard real-time control systems or safety-critical systems need more comprehensive traceability policies than non-critical systems.

The traceability manual should normally be developed incrementally as the system is specified, designed and implemented. The first chapter should always include the project traceability policies. Requirements dependencies can then be documented as soon as the requirements document is agreed but design traceability, documentation traceability, etc., must be added at later stages of the development process.

Traceability information must be regularly updated if it is to remain useful. If the document is maintained on paper, there will always be a lag between changes made to the paper document and the document, which is used by the engineers maintaining the

requirements and/or the system. Furthermore, if a document exists on paper, there is always a temptation to use this, even if it is out-of-date. This often results in errors or misunderstandings.

We strongly recommend that the traceability manual should be implemented as a networked electronic document rather than as a paper document. When traceability information is required, the document is either consulted on-screen or the relevant sections of the document are printed. Some kind of hypertext system (e.g. a WWW-based system) may be used to implement the manual or, alternatively, it may simply be maintained as word processor text. There will be a different version of the traceability manual for each different version of the system. The traceability manual, therefore, should be managed using your normal configuration management procedures.

To ensure that the traceability manual is kept up-to-date, you should assign someone to be the traceability manual manager. He or she should work with system developers to ensure that changes to the requirements/design, etc., have been incorporated in the manual and should review and update traceability policies. The traceability manual manager should also be responsible for following up deviations from traceability policies and ensuring that the required information is subsequently added to the traceability manual.

UR.Y Guidelines on How to Identify and Document Global System Requirements

We recommend that you identify the global requirements during the requirements analysis and the requirements validation processes. During these activities, you are analyzing all the requirements, therefore very little additional effort is needed to discover global requirements. Alternatively, you can identify global requirements when you are creating your initial traceability information.

There is no easy way to identify global system requirements. You need to use your judgement and knowledge of the system. To help with this identification process, it is sometimes helpful to ask the following questions about each requirement.

- UR.Y.1 Is the requirement expressed in a very general way (for example, ‘the system must never corrupt data in the database’)?
- UR.Y.2 Does the requirement express some global, non-functional system characteristic (for example, ‘the system must process N transactions per second’)?
- UR.Y.3 Is the requirement a general user interface requirement?
- UR.Y.4 Does the requirement refer to specific system functionality or a specific system service, which must be provided?
- UR.Y.5 Can the requirement be mapped onto part of the system model?
- UR.Y.6 Does the requirement refer to specific data or components of the system?

If the answers to the questions UR.Y.1-3 are ‘yes’ or if the answers to questions UR.Y.4-6 are ‘no’ then the requirement may be a global requirement. If possible, you should try to identify global system requirements during analysis and negotiation activities. Where this is impractical, they may be identified at requirements reviews during the validation process.

You will not get the identification of global system requirements right first time. Some requirements, which appear to be sub-system requirements, will emerge as global system requirements. You must therefore expect new global requirements to be discovered throughout the requirements analysis, negotiation and validation process.

Global system requirements really require separate traceability management. The normal approaches to traceability where you link specific requirements or requirements and system components do not apply to these requirements. You cannot map them onto a set of system components. You should therefore maintain a list of global requirements as a separate section in the traceability manual.

UR.Z Guidelines on How to Maintain a List of Requirements that are Most Likely to Change

All system requirements may change but it is generally the case that some requirements change more frequently than other, more stable, requirements. To implement this guideline, experienced requirements engineers should examine all of the system requirements and identify those, which are likely to be particularly volatile. We recommend that you should identify these requirements during the requirements validation process or when your initial set of traceability information is created.

We have suggested that this guideline is an advanced guideline because you will not really get any benefits from it until you have requirements management and change management policies in place.

There are different types of volatile requirements.

- UR.Z.1 *Mutable requirements.* These are requirements that change because of changes to the environment in which the system is operating. For example, the requirements for a system, which computes tax deductions, evolve as the tax laws are changed.
- UR.Z.2 *Emergent requirements.* These are requirements which cannot be completely defined when the system is specified but which emerge as the system is designed and implemented. For example, it may not be possible to specify in advance, the details of how information should be presented. As stakeholders see examples of possible presentations, they may think of new presentations that would be useful to them.
- UR.Z.3 *Consequential requirements.* These are requirements, which are based on assumptions on how the system will be used. When the system is put into use, you will find that some of these assumptions will be wrong. You will also find that users will adapt to the system and find new ways to use its functionality. This will result in demands from users for system changes and modifications.
- UR.Z.4 *Compatibility requirements.* These are requirements, which depend on other equipment or processes. As these change, these requirements also evolve. For example, an instrument control system may have to be modified when a new display is added.

To identify volatile requirements, you must involve application domain specialists who will know which domain information is fairly stable and which is variable. If you can assign requirements to one of the above classes, that is also an indicator of volatility. It may also be helpful to discuss possible organizational change with senior management. However, they may not be completely candid here, as they do not want people to know of the organizational impact of planned changes.

You will not get change prediction 100% right and you will not be able to identify all volatile requirements. Unexpected changes will always occur. You should update your list of volatile requirements as you gain experience; you may find that some of the requirements, which you thought were volatile, are fairly stable.

To identify volatile requirements in a database, you should add a field to the requirements record whose value indicates the estimated volatility of the requirement. We suggest a three-point volatility scale where 1 means fairly stable, 2 means subject to medium term change and 3 means that the requirement is likely to change in the short-term.