

ESA PSS-05-02 Issue 1 Revision 1  
March 1995

# Guide to the user requirements definition phase

Prepared by:  
ESA Board for Software  
Standardisation and Control  
(BSSC)

**DOCUMENT STATUS SHEET**

DOCUMENT STATUS SHEET			
1. DOCUMENT TITLE: <b>ESA PSS-05-01 Guide to the user requirements definition phase</b>			
2. ISSUE	3. REVISION	4. DATE	5. REASON FOR CHANGE
1	0	1991	First issue
1	1	1995	Minor updates for public version

Issue 1 Revision 1 approved, May 1995  
Board for Software Standardisation and Control  
M. Jones and U. Mortensen, co-chairmen

Issue 1 approved, 1st February 1992  
Telematics Supervisory Board

Issue 1 approved by:  
The Inspector General, ESA

Published by ESA Publications Division,  
ESTEC, Noordwijk, The Netherlands.  
Printed in the Netherlands.  
ESA Price code: E0  
ISSN 0379-4059

Copyright © 1995 by European Space Agency

## TABLE OF CONTENTS

<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
1.1 PURPOSE .....	1
1.2 OVERVIEW.....	1
<b>CHAPTER 2 THE USER REQUIREMENTS DEFINITION PHASE.....</b>	<b>3</b>
2.1 INTRODUCTION.....	3
2.2 CAPTURE OF USER REQUIREMENTS .....	3
2.3 DETERMINATION OF OPERATIONAL ENVIRONMENT .....	4
2.4 SPECIFICATION OF USER REQUIREMENTS.....	5
2.4.1 Capability requirements.....	5
2.4.1.1 Capacity.....	6
2.4.1.2 Speed .....	6
2.4.1.3 Accuracy.....	7
2.4.2 Constraint requirements.....	7
2.4.2.1 Communications interfaces.....	7
2.4.2.2 Hardware interfaces .....	8
2.4.2.3 Software interfaces.....	8
2.4.2.4 Human-Computer Interaction .....	8
2.4.2.5 Adaptability.....	8
2.4.2.6 Availability .....	9
2.4.2.7 Portability .....	9
2.4.2.8 Security .....	10
2.4.2.9 Safety .....	10
2.4.2.10 Standards.....	10
2.4.2.11 Resources.....	11
2.4.2.12 Timescales.....	11
2.5 ACCEPTANCE TEST PLANNING.....	11
2.6 PLANNING THE SOFTWARE REQUIREMENTS DEFINITION PHASE .....	11
2.7 THE USER REQUIREMENTS REVIEW.....	12
<b>CHAPTER 3 METHODS FOR USER REQUIREMENTS DEFINITION.....</b>	<b>13</b>
3.1 INTRODUCTION.....	13
3.2 METHODS FOR USER REQUIREMENTS CAPTURE .....	13
3.2.1 Interviews and surveys .....	13
3.2.2 Studies of existing software.....	13
3.2.3 Study of system requirements .....	13
3.2.4 Feasibility studies .....	14
3.2.5 Prototyping .....	14
3.3 METHODS FOR REQUIREMENTS SPECIFICATION.....	14
3.3.1 Natural language .....	14

3.3.2 Mathematical formalism .....	14
3.3.3 Structured English .....	14
3.3.4 Tables .....	15
3.3.5 System block diagrams.....	15
3.3.6 Timelines .....	16
3.3.7 Context diagrams .....	16
<b>CHAPTER 4 TOOLS FOR USER REQUIREMENTS DEFINITION .....</b>	<b>17</b>
4.1 INTRODUCTION.....	17
4.2 TOOLS FOR USER REQUIREMENTS CAPTURE .....	17
4.3 TOOLS FOR USER REQUIREMENTS SPECIFICATION .....	17
4.3.1 User requirements management .....	17
4.3.2 Document production .....	18
<b>CHAPTER 5 THE USER REQUIREMENTS DOCUMENT.....</b>	<b>19</b>
5.1 INTRODUCTION.....	19
5.2 STYLE.....	20
5.2.1 Clarity .....	20
5.2.2 Consistency .....	20
5.2.3 Modifiability .....	21
5.3 EVOLUTION.....	21
5.4 RESPONSIBILITY.....	22
5.5 MEDIUM.....	23
5.6 CONTENT .....	23
<b>CHAPTER 6 LIFE CYCLE MANAGEMENT ACTIVITIES .....</b>	<b>31</b>
6.1 INTRODUCTION.....	31
6.2 PROJECT MANAGEMENT PLAN FOR THE SR PHASE .....	31
6.3 CONFIGURATION MANAGEMENT PLAN FOR THE SR PHASE .....	32
6.4 VERIFICATION AND VALIDATION PLAN FOR THE SR PHASE .....	32
6.5 QUALITY ASSURANCE PLAN FOR THE SR PHASE .....	32
6.6 ACCEPTANCE TEST PLANS .....	33
<b>APPENDIX A GLOSSARY .....</b>	<b>A-1</b>
<b>APPENDIX B REFERENCES.....</b>	<b>B-1</b>
<b>APPENDIX C UR PHASE MANDATORY PRACTICES .....</b>	<b>C-1</b>
<b>APPENDIX D INDEX .....</b>	<b>D-1</b>

## PREFACE

This document is one of a series of guides to software engineering produced by the Board for Software Standardisation and Control (BSSC), of the European Space Agency. The guides contain advisory material for software developers conforming to ESA's Software Engineering Standards, ESA PSS-05-0. They have been compiled from discussions with software engineers, research of the software engineering literature, and experience gained from the application of the Software Engineering Standards in projects.

Levels one and two of the document tree at the time of writing are shown in Figure 1. This guide, identified by the shaded box, provides guidance about implementing the mandatory requirements for the user requirements definition phase described in the top level document ESA PSS-05-0.

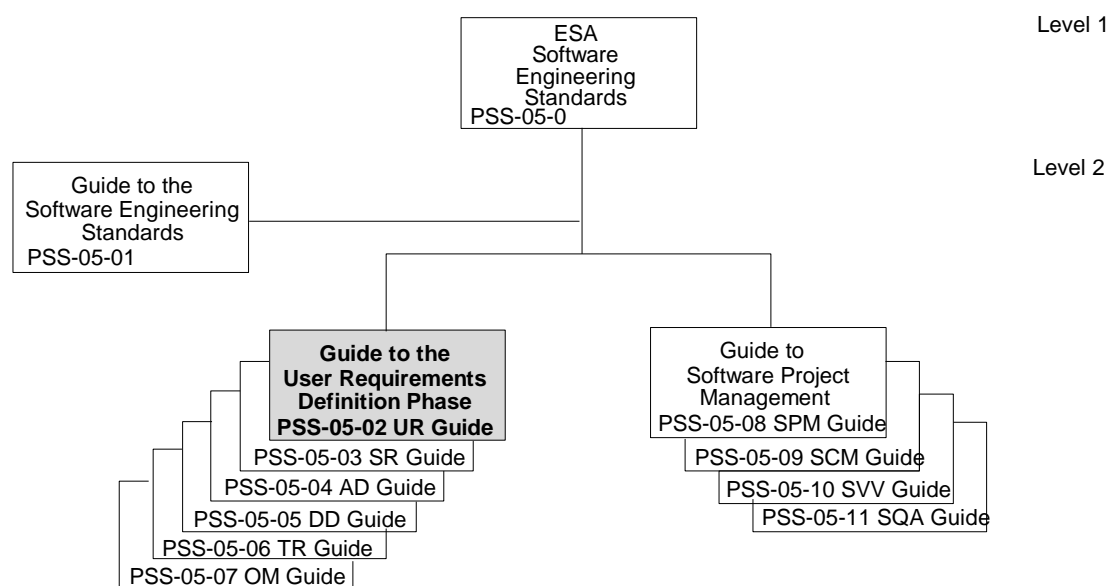


Figure 1: ESA PSS-05-0 document tree

The Guide to the Software Engineering Standards, ESA PSS-05-01, contains further information about the document tree. The interested reader should consult this guide for current information about the ESA PSS-05-0 standards and guides.

The following past and present BSSC members have contributed to the production of this guide: Carlo Mazza (chairman), Gianfranco Alvisi, Michael Jones, Bryan Melton, Daniel de Pablo and Adriaan Scheffer.

The BSSC wishes to thank Jon Fairclough for his assistance in the development of the Standards and Guides, and to all those software engineers in ESA and Industry who have made contributions.

Requests for clarifications, change proposals or any other comment concerning this guide should be addressed to:

BSSC/ESOC Secretariat  
Attention of Mr C Mazza  
ESOC  
Robert Bosch Strasse 5  
D-64293 Darmstadt  
Germany

BSSC/ESTEC Secretariat  
Attention of Mr B Melton  
ESTEC  
Postbus 299  
NL-2200 AG Noordwijk  
The Netherlands

## CHAPTER 1 INTRODUCTION

### 1.1 PURPOSE

ESA PSS-05-0 describes the software engineering standards to be applied for all deliverable software implemented for the European Space Agency (ESA), either in house or by industry [Ref 1].

ESA PSS-05-0 defines a preliminary phase to the software development life cycle called the 'User Requirements Definition Phase' (UR phase). Activities and products are examined in the 'UR review' (UR/R) at the end of the phase.

The UR phase can be called the 'problem definition phase' of the life cycle. The phase refines an idea about a task to be performed using computing equipment, into a definition of what is expected from the computer system.

This document provides a definition of what user requirements are, suggests how they can be captured and gives guidelines on how they should be stated in a URD. This guide should be read by all active participants in the user requirements phase, i.e. initiators, user representatives, software project managers and authors and reviewers of the URD.

### 1.2 OVERVIEW

Chapter 2 discusses the UR phase. Chapters 3 and 4 discuss methods and tools for user requirements definition. Chapter 5 describes how to write the URD, in particular how to fill out the document template. Chapter 6 summarises the life cycle management activities, which are discussed at greater length in other guides.

All the mandatory practices in ESA PSS-05-0 relevant to the UR phase are repeated in this document. The identifier of the practice is added in parentheses to mark a repetition. This document contains no new mandatory practices.

This page is intentionally left blank.



## CHAPTER 2

### THE USER REQUIREMENTS DEFINITION PHASE

#### 2.1 INTRODUCTION

The UR phase can be called the 'concept' or 'problem definition' phase of the ESA PSS-05-0 life cycle. User requirements often follow directly from a spontaneous idea or thought. Even so, wide agreement and understanding of the user requirements is more likely if these guidelines are applied. The definition of user requirements is an iterative process.

User requirements are documented in the User Requirements Document (URD). The URD gives the user's view of the problem, not the developer's. A URD may have to go through several revisions before it is acceptable to everyone.

The main outputs of the UR phase are the:

- User Requirements Document (URD);
- Software Project Management Plan for the SR phase (SPMP/SR);
- Software Configuration Management Plan for the SR phase (SCMP/SR);
- Software Verification and Validation Plan for the SR Phase (SVVP/SR);
- Software Quality Assurance Plan for the AD phase (SQAP/SR);
- Acceptance Test Plan (SVVP/AT).

#### 2.2 CAPTURE OF USER REQUIREMENTS

The capture of user requirements is the process of gathering information about user needs. ESA PSS-05-0 recommends that:

- user requirements should be clarified through criticism and experience of existing software and prototypes;
- wide agreement should be established through interviews and surveys;
- knowledge and experience of the potential development organisations should be used to help decide on implementation feasibility, and, perhaps to build prototypes.

Above all, user requirements should be realistic [Ref 5]. Realistic user requirements are:

- clear;
- verifiable;
- complete;
- accurate;
- feasible.

Clarity and verifiability help ensure that delivered systems will meet user requirements. Completeness and accuracy imply that the URD states the user's real needs. A URD is inaccurate if it requests something that users do not need, for example a superfluous capability or an unnecessary design constraint (see Section 2.4).

Realistic user requirements must be feasible. If the resources and timescales available for its implementation are insufficient, it may be unrealistic to put them in a URD.

When a system is to replace an existing one, the best way to make the user requirements realistic is to describe the current way of doing things and then define the user requirements in terms of the changes needed. The description of the current system should use the concrete, physical terms familiar to the user [Ref 7, 8, 9].

Methods for capturing user requirements are discussed in Chapter 3.

## **2.3 DETERMINATION OF OPERATIONAL ENVIRONMENT**

A clear description of the real world that the software will operate in should be built up, as the user requirements are captured. Chapter 3 describes several user requirements definition methods that can be used. This description of the operational environment must clearly establish the problem context.

In a system development, each subsystem will have interfaces to other, external, systems. The nature of these exchanges with external systems should be specified and controlled from the start of the project. The information may reside in an Interface Control Document (ICD), or in the design documentation of the external system.

The roles and responsibilities of the users and operators of software should be established by defining the:

- characteristics of each group (e.g. experience, qualifications);
- operations they perform (e.g. the user of the data may not operate the software).

## **2.4 SPECIFICATION OF USER REQUIREMENTS**

The specification of user requirements is the process of organising information about user needs and expressing them in a document.

A requirement is a 'condition or capability needed by a user to solve a problem or achieve an objective' [Ref 2]. This definition leads to two principal categories of requirements: 'capability requirements' and 'constraint requirements'. These categories and their subcategories are described in detail in this chapter. Unless otherwise indicated, all types of requirements can be stated in the User Requirements Document.

### **2.4.1 Capability requirements**

Capability requirements describe the process to be supported by software. Simply stated, they describe 'what' the users want to do.

A capability requirement should define an operation, or sequence of related operations, that the software will be able to perform. If the sequence contains more than approximately five related operations, the capability requirement should be split.

The operations should be organised to describe the overall process from start to finish. Where there are many operations to describe, it is recommended that they are grouped hierarchically to help manage the complexity.

Operations may be routine, (e.g. normal tasks) or non-routine (e.g. error handling, interruptions). Non-routine operations may be grouped separately from those related to the normal processing.

In the Software Requirements Definition Phase, capability requirements will be analysed to produce a set of functional requirements. If duplication of capability requirements occurs, the analyst may be able to replace them with a single functional requirement. A single function may support a process at many different times, therefore a function can map to many capability requirements.

Quantitative statements that specify performance and accuracy attributes should form part of the specification of capability. This means that a capability requirement should be qualified with values of:

- capacity;
- speed;
- accuracy.

The performance attribute is the combination of the capacity and speed attributes.

#### **2.4.1.1 Capacity**

The capacity attribute states 'how much' of a capability is needed at any moment in time. Each capability requirement should be attached with a quantitative measure of the capacity required. For example the:

- number of users to be supported;
- number of terminals to be supported;
- number of satellites that can be controlled simultaneously;
- amount of data to be stored.

#### **2.4.1.2 Speed**

The speed attribute states how fast the complete operation, or sequence of operations, is to be performed. Each capability requirement should be attached with a quantitative measure of the speed required. There are various ways to do this, for example the:

- number of operations done per unit time interval;
- time taken to perform an operation.

For example: '95% of the transactions shall be processed in less than 1 second', is acceptable whilst, '95% of the transactions will be done as soon as possible' is not.

Note that a system may react quickly to a command but take quite a long time to complete the operations requested. Such 'response' requirements should be stated as HCI requirements.

### 2.4.1.3 Accuracy

The accuracy of an operation is measured by the difference between what is intended and what happens when it is carried out.

Examples are:

- 'the accuracy of accounting reports shall be one accounting unit';
- 'the program shall predict the satellite's altitude to within 10 metres, seven days in advance'.

Accuracy attributes should take account of both systematic errors and random errors.

### 2.4.2 Constraint requirements

Constraint requirements place restrictions on how the user requirements are to be met. The user may place constraints on the software related to interfaces, quality, resources and timescales.

Users may constrain how communication is done with other systems, what hardware is to be used, what software it has to be compatible with, and how it must interact with human operators. These are all interface constraints.

An interface is a shared boundary between two systems; it may be defined in terms of what is exchanged across the boundary.

Interfaces are important kinds of constraints. The user may define external interfaces (i.e. state how interactions with other systems must be done) but should leave the developers to define the internal interfaces (i.e. to state how software components will interact with each other).

Users may constrain the quality required of the final product. Typical quality characteristics are: adaptability, availability, portability, security and safety.

#### 2.4.2.1 Communications interfaces

A communications interface requirement may specify the networks and network protocols to be used. Performance attributes of the interface may be specified (e.g. data rate).

The ISO reference model for Open Systems Interconnection, with its seven layers of abstraction, can be used for describing communications

interfaces. This means that a communications interface requirement should use terminology consistent with the model. Communications interface requirements should avoid mixing the layers of abstraction.

#### **2.4.2.2 Hardware interfaces**

A hardware interface requirement specifies all or part of the computer hardware the software is to execute on. This may be done by stating the make and model of the device, physical limitations (e.g. size, weight), performance (e.g. speed, memory), qualifications (e.g. project approved, space qualified) and also perhaps whether any hardware selected has to be derated (e.g. for operation at altitude). Environmental considerations that affect the selection of hardware may be stated (e.g. humidity, temperature and pressure).

#### **2.4.2.3 Software interfaces**

A software interface requirement specifies whether the software is to be compatible with other software (e.g. other applications, compilers, operating systems, programming languages and database management systems).

#### **2.4.2.4 Human-Computer Interaction**

A Human-Computer Interaction (HCI) requirement may specify any aspect of the user interface. This may include a statement about style (e.g. command language, menu system, icons), format (e.g. report content and layout), messages (e.g. brief, exhaustive) and responsiveness (e.g. time taken to respond to command). The hardware at the user interface (e.g. colour display and mouse) may be included either as an HCI requirement or as a hardware interface requirement.

#### **2.4.2.5 Adaptability**

Adaptability measures how easily a system copes with requirements changes. Adaptable (or flexible) systems are likely to live longer, although the extra design work needed may be extensive, especially for optimising modularity. An example of an adaptability requirement is: 'it shall be possible to add new commands without retesting existing commands'.

In the operations and maintenance phase the software may undergo continuous adaptation as the user requirements are modified by experience.

When considering the adaptability, note that any change involves some risk, and to change reliable parts of the system may not be acceptable.

#### **2.4.2.6 Availability**

Availability measures the ability of a system to be used during its intended periods of its operation. Availability requirements may specify:

- mean and minimum capacity available (e.g. all terminals);
- start and end times of availability (e.g. from 0900 to 1730 daily);
- time period for averaging availability (e.g. 1 year).

Examples of availability requirements are:

- 'the user shall be provided with 98% average availability over 1 year during working hours and never less than 50% of working hours in any one week';
- 'all essential capabilities shall be at least 98% available in any 48 hour period and at least 75% available in every 3 hour period'.

When a system is unavailable, some, or even all, of its capabilities cannot be used. A loss of capability is called a 'failure' and is caused by one or more 'faults'. The average time between the occurrence of faults internal to the software (i.e. 'bugs') measures the 'reliability' of the software. The average time taken to fix such faults measures its 'maintainability'. A system may also become unavailable due to external factors (e.g. loss of input service).

Users only need to state their availability requirements. The availability requirements are decomposed into specific reliability and maintainability requirements in the SR phase.

#### **2.4.2.7 Portability**

Software portability is measured by the ease that it can be moved from one environment to another. Portable software tends to be long lived, but more code may have to be written and performance requirements may be more difficult to meet. An example of a portability requirement is: 'the software shall be portable between environments X and Y'.

Portability can be measured in terms of the number of lines of code and/or the number of modules that do not have to be changed to port the

software from one computer to another. Either absolute or relative measurements can be used.

If migration to another hardware base or operating system is intended, then any requirements to run with different hardware and software interfaces should be stated as portability requirements. New interfaces should be described (e.g. name the new operating system or computer hardware).

#### **2.4.2.8 Security**

A system may need to be secured against threats to its confidentiality, integrity and availability. For example, a user may request that unauthorised users be unable to use the system, or that no single event such as a fire should cause the loss of more than 1 week's information. The user should describe threats that the system needs to be protected against, e.g. virus intrusions, hackers, fires, computer breakdowns.

The security of a system can be described in terms of the ownership of, and rights of access to, the capabilities of the system.

A secure system protects users from their own errors as well as the malicious interference, or illegal activities, of unauthorised users.

#### **2.4.2.9 Safety**

The consequences of software failure should be made clear to developers. Safety requirements define the needs of users to be protected against potential problems such as hardware or software faults. They may define scenarios that the system should handle safely (e.g. 'the system should ensure that no data is lost when a power failure occurs')

#### **2.4.2.10 Standards**

Standards requirements normally reference the applicable documents that define the standard.

Two kinds of standards can be specified: process standards and product standards. Examples of product standards are export file formats and legal report formats. Examples of the process standards are product assurance standards and accounting procedures to be followed. Adherence to process standards should be specified in the Software Project Management Plan.



A standards requirement may specify the methods that are to be employed by the developers in subsequent phases. Such methods must be compatible with the life cycle defined in ESA PSS-05-0.

#### **2.4.2.11 Resources**

The resources available for producing and operating the software are a constraint on the design. If this information is available then it should be stated in the Software Project Management Plan in terms of one or more of financial, manpower and material limits. As with any other product, the quality and sophistication of a software product are limited by the resources that are put into building it.

Resource requirements may include specifications of the computer resources available (e.g. main memory). They may define the minimum hardware that the system must run on (e.g. a 486 PC with 4 Mbytes of memory). Care should be taken to include only the necessary resource constraints.

#### **2.4.2.12 Timescales**

A constraint on the design of the software may be the acceptable timescales for its development and production. Requirements for the achievement of specific life cycle milestones may be stated in the Software Project Management Plan.

### **2.5 ACCEPTANCE TEST PLANNING**

Validation confirms whether the user requirements are satisfied when the software is delivered. This is done by performing acceptance tests in the Transfer Phase.

Acceptance Test Plans must be generated in the UR phase and documented in the Acceptance Test section of the Software Verification and Validation Plan (SVVP/AT). The Acceptance Test Plan should describe the scope, approach and resources required for the acceptance tests, and take account of the user requirements. See Chapter 6.

### **2.6 PLANNING THE SOFTWARE REQUIREMENTS DEFINITION PHASE**

Plans of SR phase activities must be drawn up in the UR phase by the developer. Planning of the SR phase is discussed in Chapter 6. Planning

should cover project management, configuration management, verification, validation and quality assurance. Outputs are the:

- Software Project Management Plan for the SR phase (SPMP/SR);
- Software Configuration Management Plan for the SR phase (SCMP/SR);
- Software Verification and Validation Plan for the SR phase (SVVP/SR);
- Software Quality Assurance Plan for SR phase (SQAP/SR).

## 2.7 THE USER REQUIREMENTS REVIEW

Producing the URD and the SVVP/AT is an iterative process. The initiator should organise internal reviews of a document before its formal review.

The outputs of the UR phase must be formally reviewed during the User Requirements Review (UR08). This should be a technical review. The recommended procedure is described in ESA PSS-05-10, and is derived from the IEEE standard for Technical Reviews [Ref 4].

Normally, only the URD and the Acceptance Test Plan undergo the full technical review procedure involving users, developers, management and quality assurance staff. The Software Project Management Plan (SPMP/SR), Software Configuration Management Plan (SCMP/SR), Software Verification and Validation Plan (SVVP/SR), and Software Quality Assurance Plan (SQAP/SR) are usually reviewed by management and quality assurance staff only.

The objective of the UR/R review is to verify that:

- the URD states user requirements clearly and completely and that a general description of the process the user expects to be supported is present;
- the SVVP/AT is an adequate plan for validating the software in the TR phase.

The UR/R should conclude with a statement about the project's readiness to proceed.

## **CHAPTER 3**

### **METHODS FOR USER REQUIREMENTS DEFINITION**

#### **3.1 INTRODUCTION**

This chapter discusses methods for user requirements capture and specification in current use. Methods can be combined to suit the needs of a particular project.

#### **3.2 METHODS FOR USER REQUIREMENTS CAPTURE**

While user requirements ultimately come from an original idea, one or more of the methods described below can be used to stimulate the creative process and record its output.

##### **3.2.1 Interviews and surveys**

Interviews should be structured to ensure that all issues are covered. When it is not practical to interview all the potential users, a representative sample should be selected and interviewed; this process is called a survey. Interviews and surveys can be useful for ensuring that:

- the user requirements are complete;
- there is wide agreement about the user requirements.

##### **3.2.2 Studies of existing software**

New software is often written to replace existing software. An investigation of the good and bad features of what exists can identify requirements for what is to be built. Examination of user manuals, requirements documentation and change proposals can be especially helpful.

##### **3.2.3 Study of system requirements**

If software is part of a larger system, many of the user requirements can be derived from the System Requirements Document.

### 3.2.4 Feasibility studies

A feasibility study is the analysis and design of the principal features of a system. The amount of detail in the design will not normally allow its implementation, but may show whether implementation is possible.

### 3.2.5 Prototyping

A prototype is a 'concrete executable model of selected aspects of a proposed system' [Ref 5]. If requirements are unclear or incomplete, it can be useful to develop a prototype based on tentative requirements to explore what the user requirements really are. This is called 'exploratory prototyping'. Hands-on experience can be an excellent way of deciding what is really wanted.

## 3.3 METHODS FOR REQUIREMENTS SPECIFICATION

### 3.3.1 Natural language

The obvious way to express a requirement is to use natural language (e.g. English). Natural language is rich and accessible but inconsistency and ambiguity are more likely. For example, the statement:

'The database will contain an address'

might be read as any of:

'There will be one and only one address'

'Some part of the database will be designated as an address'

'There will be at least one address in the database'.

### 3.3.2 Mathematical formalism

Mathematical formulae should be described or referenced in the URD where they clarify the statement of requirement. All symbols used in an expression should be defined or referenced.

### 3.3.3 Structured English

Structured English is a specification language that makes use of a limited vocabulary and a limited syntax [Ref 7]. The vocabulary of Structured English consists only of:

- imperative English language verbs;

- terms defined in a glossary;
- certain reserved words for logic formulation.

The syntax of a Structured English statement is limited to these possibilities:

- simple declarative sentence;
- closed end decision construct;
- closed end repetition construct.

Structured English is normally used to describe the basic processes of a system and is suitable for expressing capability requirements. Examples are:

Sequence:

```
GET RAW DATA  
REMOVE INSTRUMENT EFFECTS  
CALIBRATE CORRECTED DATA
```

Condition:

```
IF SAMPLE IS OF NOMINAL QUALITY THEN  
    CALIBRATE SAMPLE  
ELSE  
    STORE BAD SAMPLE
```

Repetition:

```
FOR EACH SAMPLE  
    GET POINTING DIRECTION AT TIME OF SAMPLE  
    STORE POINTING DIRECTION WITH SAMPLE
```

Formalising the English structure may allow automated processing of requirements (e.g. automated checking, analysis, transformation and display) and makes it easier to define acceptance tests.

### 3.3.4 Tables

Tables are an effective method for describing requirements completely and concisely. Used extensively in later phases, they can summarise relationships more effectively than a plain text description.

### 3.3.5 System block diagrams

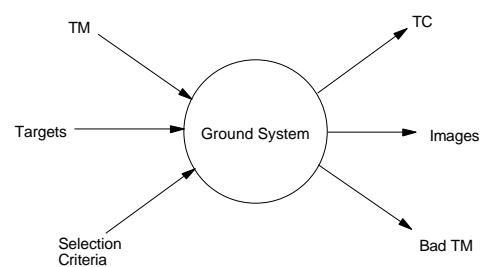
Block diagrams are the traditional way of depicting the processing required. They can also demonstrate the context the software operates in when it is part of a larger system.

### 3.3.6 Timelines

Timelines can describe sequences of operations that software must perform, especially if there is a real-time aspect or processing schedule. They convey a sense of interval more powerfully than a text description.

### 3.3.7 Context diagrams

A context diagram contains one bubble, representing the system, and dataflow arrows, showing the inputs and outputs. Context diagrams show external interfaces.



## **CHAPTER 4**

### **TOOLS FOR USER REQUIREMENTS DEFINITION**

#### **4.1 INTRODUCTION**

This chapter discusses tools for user requirements capture and specification. Tools can be combined to suit the needs of a particular project.

#### **4.2 TOOLS FOR USER REQUIREMENTS CAPTURE**

The questionnaire is the primary tool of a survey. To get useful data, careful consideration should be given to its contents and presentation.

A feasibility study may be performed using CASE tools for analysis and design. Similarly, prototyping of code may employ tools used for detailed design and production.

Capturing user requirements from studies of existing software may require building models that describe what the existing software does [Ref 7, 8, 9]. CASE tools are available for the construction of such models.

When the software is being developed as part of a system development, any tools used for system requirements analysis may also prove useful in identifying the user requirements.

#### **4.3 TOOLS FOR USER REQUIREMENTS SPECIFICATION**

##### **4.3.1 User requirements management**

Tools for managing user requirements information should support one or more of the following functions:

- insertion of new requirements;
- modification of existing requirements;
- deletion of requirements;
- storage of attributes (e.g. identifier) with the text;
- searching for requirements attributes and text strings;
- cross-referencing;

- change history recording;
- access control;
- display;
- printing, in a variety of formats.

Database Management Systems (DBMS), available on a variety of hardware platforms (e.g. PC, minicomputer, mainframe), provide many of these functions. For large systems, a requirements DBMS becomes essential.

The ability to export requirements data to the word processor used for URD production is essential for preserving consistency.

#### **4.3.2 Document production**

A word processor or text processor should be used for producing a document. Tools for the creation of paragraphs, sections, headers, footers, tables of contents and indexes all facilitate the production of a document. A spell checker is desirable. An outliner may be found useful for creation of sub-headings, for viewing the document at different levels of detail and for rearranging the document. The ability to handle diagrams is very important.

Documents invariably go through many drafts as they are created, reviewed and modified. Revised drafts should include change bars. Document comparison programs, which can mark changed text automatically, are invaluable for easing the review process.

Tools for communal preparation of documents are beginning to be available, allowing many authors to comment and add to a single document in a controlled manner.



## CHAPTER 5

### THE USER REQUIREMENTS DOCUMENT

#### 5.1 INTRODUCTION

The URD is a mandatory output of the UR phase (UR10) and must always be produced before the software project is started (UR11). The URD must:

- provide a general description of what the user wants to perform with the software system (UR12);
- contain all the known user requirements (UR13);
- describe the operations the user wants to perform with the software system (UR14);
- define all the constraints that the user wishes to impose on any solution (UR15);
- describe the external interfaces to the software system or reference them in ICDs that exist or are to be written (UR16).

The size and content of the URD should reflect the complexity of the problem and the degree of expertise and understanding shared by the initiator, users, URD author and software developer.

The URD needs to state the problem as completely and accurately as possible. The cost of changing the user requirements increases rapidly as the project proceeds through the life cycle.

When software is transferred to users after development, acceptance tests are held to determine whether it meets the user requirements. The URD should be detailed enough to allow the definition of acceptance tests.

The URD should be a balanced statement of the problem and should avoid over-constraining the solution. If the software described in the URD is a part of a larger system (i.e. it is a subsystem), then the URD may replace the descriptive information with references to higher level documents. The purpose of the software, however, should always be clear from the URD.

ESA PSS-05-0 defines the minimum required documents for a software project and the URD has a definite role to play in this documentation scheme. URD authors should not go beyond the bounds of that role.

The URD should not:

- contain an exhaustive analysis of the requirements on the software (this is done in the SR phase);
- define any design aspects (this is done in the AD and DD phases);
- cover project management aspects (which form part of the SPMP/SR);

If definition of design aspects is unavoidable, then such definitions should be categorised as constraint requirements.

The URD should define needs accurately and leave the maximum scope for the software engineer to choose the most efficient solution.

## 5.2 **STYLE**

The style of a URD should be plain and concise. The URD should be clear, consistent and modifiable. Wherever possible, requirements should be stated in quantitative terms to increase their verifiability.

### 5.2.1 **Clarity**

A URD is 'clear' if each requirement is unambiguous and understandable to project participants. A requirement is unambiguous if it has only one interpretation. To be understandable, the language used in a URD should be shared by all project participants and should be as simple as possible.

Each requirement should be stated in a single sentence. Justifications and explanations of a requirement should be clearly separated from the requirement itself.

Clarity is enhanced by grouping related requirements together. The capability requirements in a group should be structured to reflect any temporal or causal relationships between them. Groups containing more than about ten requirements should be broken down into sub-groups. Subgroups should be organised hierarchically. Structuring the user requirements is one of the most important ways of making them understandable.

### 5.2.2 **Consistency**

A URD is consistent if no requirements conflict. Using different terms for what is really the same thing, or specifying two incompatible qualities, are examples of lack of consistency.

Where a term used in a particular context could have multiple meanings, a single meaning should be defined in a glossary, and only that meaning should be used throughout.

### 5.2.3 Modifiability

A URD is modifiable if any necessary requirements changes can be documented easily, completely, and consistently.

A URD contains redundancy if there are duplicating or overlapping requirements. Redundancy itself is not an error, and redundancy can help to make a URD more readable, but a problem arises when the URD is updated. If a requirement is stated in two places, and a change is made in only one place, the URD will be inconsistent. When redundancy or overlapping is necessary, the URD should include cross-references to make it modifiable.

The removal of redundancy can lead to errors. Consider the situation of two similar requirements from separate users being combined. A change of mind on the part of one user may result in the removal of the combined requirement. The requirement of the other user has been lost, and this is an error. Source attributes should be retained when merging requirements to show who needs to be consulted before an update is made.

## 5.3 EVOLUTION

Changes to the URD are the user's responsibility. The URD should be put under change control by the initiator at soon as it is first issued. The document change control procedure described in ESA PSS-05-0 Issue 2, Part 2, Section 3.2.3.2.1 is recommended. This requires that a change history be kept.

New user requirements may be added and existing user requirements may be modified or deleted. If anyone wants to change the user requirements after the UR phase, the users should update the URD and resubmit it to the UR/R board for approval. Note that in the OM phase, the Software Review Board (SRB) replaces the UR/R board.

The initiator of the project should monitor the trend in the occurrence of new user requirements. An upward trend signals that the software is unlikely to be successful.

## 5.4 RESPONSIBILITY

The definition of the user requirements must be the responsibility of the user (UR01). This means that the URD must be written by the users, or someone appointed by them. The expertise of software engineers, hardware engineers and operations personnel should be used to help define and review the user requirements.

Typically the capability requirements are generated by the people who will use the system, while the constraint requirements may come from either hardware, software, communications or quality assurance experts. Human-computer interfaces are normally best defined by a joint effort of users and developers, ideally through prototypes.

In a system development, some of the user requirements for the software come from the System Requirements Document. The preferred approach is to refer to system requirements in the URD. Alternatively, relevant requirements can be extracted from the System Requirements Document, perhaps reformulated, and then inserted in the URD. This approach may pose problems from a change control point of view, but may also be the only possible alternative when the system requirements are not clearly identifiable or when the requirements applicable to the software components are embedded in other requirements.

It should never be assumed that all the user requirements can be derived from system requirements. Other techniques for capturing user requirements should always be considered (see Chapter 2). In other cases there could be multiple user groups, each having their own set of requirements. A single URD, with sections compiled by the different groups, or multiple URDs, one for each group, are both possible ways of documenting the user requirements.

In summary, there is no single scheme for producing a URD. Nevertheless:

- responsibilities should be clearly defined before URD production is started;
- the real users of the system are responsible for determining the capability requirements (UR01);
- the software engineers to be in charge of the development should take part in the URD creation process so that they can advise the users on the real practicalities of requirements, point out the potential of existing software and technology, and possibly develop prototypes.

The roles and responsibilities of the various people must be clarified and accepted by everybody involved before the process starts. Whatever the organisation, users should avoid dictating solutions while developers should avoid dictating capabilities.

## 5.5 MEDIUM

It is usually assumed that the URD is a paper document. There is no reason why the URD should not be distributed electronically to participants with the necessary equipment.

## 5.6 CONTENT

The URD should be compiled according to the table of contents provided in Appendix C of ESA PSS-05-0. This table of contents is derived from ANSI/IEEE Std 830-1984 'Software Requirements Specifications' [Ref 3].

Section 1 should briefly describe the purpose and scope of the software and provide an overview of the rest of the document. Section 2 should provide a general description of the world the software operates in. While rigour is not necessary, a clear physical picture should emerge. Section 3 should provide the formal requirements, upon which the acceptability of the software will be judged. Large URDs (forty pages or more) should contain an index.

References should be given where appropriate. A URD should not refer to documents that follow it in the ESA PSS-05-0 life cycle. A URD should contain no TBDs by the time of the User Requirements Review.

ESA PSS-05-0 recommends the following table of contents for a URD:

Service Information:

- a - Abstract
- b - Table of Contents
- c - Document Status Sheet
- d - Document Change records made since last issue

## 1 INTRODUCTION

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms and abbreviations
- 1.4 References
- 1.5 Overview

## 2 GENERAL DESCRIPTION

- 2.1 Product perspective
- 2.2 General capabilities<sup>1</sup>
- 2.3 General constraints
- 2.4 User characteristics
- 2.5 Operational environment
- 2.6 Assumptions and dependencies

## 3 SPECIFIC REQUIREMENTS

- 3.1 Capability requirements
- 3.2 Constraint requirements

Material unsuitable for the above contents list should be inserted in additional appendices. If there is no material for a section then the phrase 'Not Applicable' should be inserted and the section numbering preserved.

### 5.6.1 URD/1 INTRODUCTION

This section should provide an overview of the entire document and a description of the scope of the software.

#### 5.6.1.1 URD/1.1 Purpose (of the document)

This section should:

- (1) define the purpose of the particular URD;
- (2) specify the intended readership of the URD.

#### 5.6.1.2 URD/1.2 Scope (of the software)

This section should:

- (1) identify the software product(s) to be produced by name;

---

<sup>1</sup> This section has been inserted after ESA PSS-05-0 Issue 2 was published. Other General Description sections have been reordered.

- (2) explain what the proposed software will do (and will not do, if necessary);
- (3) describe relevant benefits, objectives, and goals as precisely as possible;
- (4) be consistent with similar statements in higher-level specifications, if they exist.

#### **5.6.1.3 URD/1.3 Definitions, acronyms and abbreviations**

This section should provide the definitions of all terms, acronyms, and abbreviations, or refer to other documents where the definitions can be found.

#### **5.6.1.4 URD/1.4 References**

This section should provide a complete list of all the applicable and reference documents, identified by title, author and date. Each document should be marked as applicable or reference. If appropriate, report number, journal name and publishing organisation should be included.

#### **5.6.1.5 URD/1.5 Overview**

This section should:

- (1) describe what the rest of the URD contains;
- (2) explain how the URD is organised.

### **5.6.2 URD/2 GENERAL DESCRIPTION**

This chapter should describe the general factors that affect the product and its requirements. This chapter does not state specific requirements but makes those requirements easier to understand.

#### **5.6.2.1 URD/2.1 Product perspective**

This section puts the product into perspective with other related systems. If the product is to replace an existing system, the system should be described and referenced. Ancestors of the product that are no longer in use might be mentioned. If the product is 'standalone', it should be stated here.

### **5.6.2.2 URD/2.2 General capabilities**

This section should describe the main capabilities and why they are needed. This section should describe the process to be supported by the software, indicating those parts of the process where it is used.

### **5.6.2.3 URD/2.3 General constraints**

This section should describe any items that will limit the developer's options for building the software.

This section should not be used to impose specific requirements or specific design constraints, but should state the reasons why certain requirements or constraints exist.

### **5.6.2.4 URD/2.4 User characteristics**

This section should describe those general characteristics of the users affecting the specific requirements.

Many people may interact with the software during the operations and maintenance phase. Some of these people are users, operators and maintenance personnel. Certain characteristics of these people, such as educational level, language, experience and technical expertise impose important constraints on the software.

Software may be frequently used, but individuals may use it only occasionally. Frequent users will become experts whereas infrequent users may remain relative novices. It is important to classify the users and estimate the likely numbers in each category. If absolute numbers cannot be stated, relative numbers can still be useful.

### **5.6.2.5 URD/2.5 Operational environment**

This section should describe the real world the software is to operate in. This narrative description may be supported by context diagrams, to summarise external interfaces, and system block diagrams, to show how the activity fits within the larger system. The nature of the exchanges with external systems should be specified.



If a URD defines a product that is a component of a parent system or project then this section should:

- outline the activities that will be supported by external systems;
- reference the Interface Control Documents that define the external interfaces with the other systems;
- describe the computer infrastructure to be used.

#### **5.6.2.4 URD/2.4 Assumptions and dependencies**

This section should list the assumptions that the specific requirements are based on. Risk analysis should be used to identify assumptions that may not prove to be valid.

A constraint requirement, for example, might specify an interface with a system that does not exist. If the production of the system does not occur when expected, the URD may have to change.

#### **5.6.3 URD/3 SPECIFIC REQUIREMENTS**

Specific requirements should be described in this section, which is the core of the URD. The acceptability of the software will be assessed with respect to the specific requirements.

Each requirement must be uniquely identified (UR02). Forward traceability to subsequent phases in the life cycle depends upon each requirement having a unique identifier.

Essential requirements have to be met for the software to be acceptable. If a requirement is essential, it must be clearly flagged (UR03). Non-essential requirements should be marked with a measure of desirability (e.g. scale of 1, 2, 3).

Some user requirements may be 'suspended' pending resources becoming available. Such non-applicable user requirements must be clearly flagged (UR09).

The priority of a requirement measures the order, or the timing, of the related software becoming available. If the transfer is to be phased, so that some parts of the software come into operation before others, then each requirement must be marked with a measure of priority (UR04).

Unstable requirements should be flagged. These requirements may be dependent on feedback from the UR, SR and AD phases. The usual method for flagging unstable requirements is to attach the marker 'TBC'.

The source of each user requirement must be stated (UR05). The source may be defined using the identifier of a system requirement, a document cross-reference or even the name of a person or group. Backwards traceability depends upon each requirement explicitly referencing its source.

Each user requirement must be verifiable (UR06). Clarity increases verifiability. Each statement of user requirement should contain one and only one requirement. A user requirement is verifiable if some method can be devised for objectively demonstrating that the software implements it. For example statements such as:

- 'the software will work well';
- 'the product shall be user friendly';
- 'the output of the program shall usually be given within 10 seconds';

are not verifiable because the terms 'well', 'user friendly' and 'usually' have no objective interpretation.

A statement such as: 'the output of the program shall be given within 20 s of event X, 60% of the time; and shall be given within 30 s of event X, 99% of the time', is verifiable because it uses concrete terms and measurable quantities. If a method cannot be devised to verify a requirement, the requirement is invalid.

The user must describe the consequences of losses of availability and breaches of security, so that the developers can fully appreciate the criticality of each function (UR07).

#### **5.6.3.1 URD/3.1 Capability requirements**

The organisation of the capability requirements should reflect the problem, and no single structure will be suitable for all cases.

The capability requirements can be structured around a processing sequence, for example:

- a) RECEPTION OF IMAGE
- b) PROCESSING OF IMAGE
- c) DISPLAY OF IMAGE

perhaps followed by deviations from the baseline operation:

d) HANDLING LOW QUALITY IMAGES

Each capability requirement should be checked to see whether the inclusion of capacity, speed and accuracy attributes is appropriate.

### 5.6.3.2 URD/3.2 Constraint requirements

Constraint requirements may cover any topic that does not directly relate to the specific capabilities the users require.

Constraint requirements that relate to interfaces should be grouped around the headings:

- communications interfaces;
- hardware interfaces;
- software interfaces;
- human-computer interactions (user interfaces).

If the software is part of a larger system then any documents (e.g. ICDs) that describe the interfaces should be identified.

Requirements that ensure the software will be fit for its purpose should be stated, for example:

- adaptability;
- availability;
- portability;
- security;
- safety;
- standards.

This page is intentionally left blank.

## **CHAPTER 6**

### **LIFE CYCLE MANAGEMENT ACTIVITIES**

#### **6.1 INTRODUCTION**

Plans of SR phase activities must be drawn up in the UR phase. These plans cover project management, configuration management, verification and validation, quality assurance and acceptance tests.

#### **6.2 PROJECT MANAGEMENT PLAN FOR THE SR PHASE**

By the end of the UR review, the SR phase section of the SPMP (SPMP/SR) must be produced (SPM02). The SPMP/SR describes, in detail, the project activities for the SR phase. As part of its introduction, the SPMP/SR must outline a plan for the whole project (SPM03).

A rough estimate of the total cost of the software project should be included in the SPMP/SR. Technical knowledge and experience gained on similar projects help make this estimate.

A precise estimate of the effort involved in the SR phase must be included in the SPMP/SR (SPM04). Specific factors affecting estimates for the work required in the SR phase are the:

- number of user requirements;
- level of user requirements;
- stability of user requirements;
- level of definition of external interfaces;
- quality of the URD.

An estimate based simply on the number of user requirements might be very misleading - a large number of detailed low-level user requirements might be more useful, and save more time in the SR phase, than a few high-level user requirements. A poor quality URD might imply that a lot of requirements analysis is required in the SR phase.

### **6.3 CONFIGURATION MANAGEMENT PLAN FOR THE SR PHASE**

By the end of the UR review, the SR phase section of the SCMP (SCMP/SR) must be produced (SCM42). The SCMP/SR must cover the configuration management procedures for all documentation, CASE tool outputs and prototype code, to be produced in the SR phase (SCM43).

### **6.4 VERIFICATION AND VALIDATION PLAN FOR THE SR PHASE**

By the end of the UR review, the SR phase section of the SVVP (SVVP/SR) must be produced (SVV09). The SVVP/SR must define how to trace user requirements to software requirements, so that each software requirement can be justified (SVV10). It should describe how the SRD is to be evaluated by defining the review procedures. It may include specifications of the tests to be performed with prototypes.

### **6.5 QUALITY ASSURANCE PLAN FOR THE SR PHASE**

By the end of the UR review, the SR phase section of the SQAP (SQAP/SR) must be produced (SQA03). The SQAP/SR must describe, in detail, the quality assurance activities to be carried out in the SR phase (SQA04). The SQAP/SR must outline the quality assurance plan for the rest of the project (SQA05).

SQA activities include monitoring the following activities:

- management;
- documentation;
- standards, practices, conventions, and metrics;
- reviews and audits;
- testing activities;
- problem reporting and corrective action;
- tools, techniques and methods;
- code and media control;
- supplier control;
- records collection maintenance and retention;
- training;
- risk management.

## 6.6 ACCEPTANCE TEST PLANS

The initiator(s) of the user requirements should lay down the acceptance test principles. The developer must construct an acceptance test plan in the UR phase and document it in the SVVP (SVV11). This plan should define the scope, approach, resources and schedule of acceptance testing activities.

Specific tests for each user requirement are not formulated until the DD phase. The Acceptance Test Plan should deal with the general issues, for example:

- where will the acceptance tests be done?
- who will attend?
- who will carry them out?
- are tests needed for all user requirements?
- what kinds of tests are sufficient for provisional acceptance?
- what kinds of tests are sufficient for final acceptance?
- must any special test software be used?
- if the acceptance tests are to be done by users, what special documentation is required?
- how long is the acceptance testing programme expected to last?

This page is intentionally left blank.



## APPENDIX A GLOSSARY

### A.1 LIST OF TERMS

Except for the definitions listed below, the definitions of all terms used in this document conform to the definitions provided in or referenced by ESA PSS-05-0.

#### **Capability requirement**

A capability requirement describes an operation, or sequence of related operations, that the software must be able to perform.

#### **Constraint requirement**

A constraint requirement restricts the way the software is implemented, without altering or describing the capabilities of the software.

#### **Initiator**

The person, or group of people, who originates a project and is responsible for accepting its products.

#### **Language**

A symbolic method of expressing information. Symbols convey meaning and are used according to convention.

#### **Outliner**

A word processing program that allows the viewing of the section headings in isolation from the rest of the text.

#### **Risk**

The amount of uncertainty in being able to satisfy a requirement.

**A.2 LIST OF ACRONYMS**

AD	Architectural Design
ANSI	American National Standards Institute
AT	Acceptance Tests
BSSC	Board for Software Standardisation and Control
DD	Detailed Design and production
ESA	European Space Agency
OM	Operations and Maintenance
PSS	Procedures, Specifications and Standards
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SPM	Software Project Management
SPMP	Software Project Management Plan
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SR	Software Requirements
SRB	Software Review Board
SVV	Software Verification and Validation
SVVP	Software Verification and Validation Plan
TBC	To Be Confirmed
TBD	To Be Defined
TR	Transfer
UR	User Requirements
UR/R	User Requirements Review

## **APPENDIX B REFERENCES**

1. ESA Software Engineering Standards, ESA PSS-05-0 Issue 2 February 1991
2. IEEE Standard Glossary for Software Engineering Terminology, ANSI/IEEE Std 610.12-1990.
3. IEEE Guide to Software Requirements Specifications, ANSI/IEEE Std 830-1984.
4. IEEE Standard for Software Reviews and Audits, IEEE Std 1028-1988.
5. Realistic User Requirements, G.Longworth, NCC Publications, 1987.
6. Software Evolution Through Rapid Prototyping, Luqi, in COMPUTER, May 1989.
7. Structured Analysis and System Specification, T.DeMarco, Yourdon Press, 1978.
8. SSADM Version 4, NCC Blackwell publications, 1991.
9. Structured Systems Analysis and Design Methodology, G.Cutts, Paradigm, 1987.

B-2

ESA PSS-05-02 Issue 1 Revision 1 (March 1995)  
REFERENCES

This page is intentionally left blank

## APPENDIX C

### UR PHASE MANDATORY PRACTICES

This appendix is repeated from ESA PSS-05-0, appendix D.2.

- UR01 The definition of the user requirements shall be the responsibility of the user.
- UR02 Each user requirement shall include an identifier.
- UR03 Essential user requirements shall be marked as such.
- UR04 For incremental delivery, each user requirement shall include a measure of priority so that the developer can decide the production schedule.
- UR05 The source of each user requirement shall be stated.
- UR06 Each user requirement shall be verifiable.
- UR07 The user shall describe the consequences of losses of availability, or breaches of security, so that developers can fully appreciate the criticality of each function.
- UR08 The outputs of the UR phase shall be formally reviewed during the User Requirements Review.
- UR09 Non-applicable user requirements shall be clearly flagged in the URD.
- UR10 An output of the UR phase shall be the User Requirements Document (URD).
- UR11 The URD shall always be produced before a software project is started.
- UR12 The URD shall provide a general description of what the user expects the software to do.
- UR13 All known user requirements shall be included in the URD.
- UR14 The URD shall describe the operations the user wants to perform with the software system.
- UR15 The URD shall define all the constraints that the user wishes to impose upon any solution.
- UR16 The URD shall describe the external interfaces to the software system or reference them in ICDs that exist or are to be written.

C-2

ESA PSS-05-02 Issue 1 Revision 1 (March 1995)  
UR PHASE MANDATORY PRACTICES

This page is intentionally left blank

## INDEX

## APPENDIX D INDEX

acceptance test plan, 33  
 Accuracy, 7  
 adaptability, 7, 8  
 availability, 7, 9, 10, 28  
 block diagrams, 15, 26  
 bug, 9  
 capability requirement, 15, 22  
 capability requirements, 5  
 Capacity, 6, 9, 28  
 CASE tool, 17, 32  
 Communications interfaces, 7  
 confidentiality, 10  
 constraint requirement, 22  
 Constraint requirements, 7, 28  
 Context diagrams, 16, 26  
 Essential requirements, 27  
 external interface, 7, 19  
 external interfaces, 26, 31  
 failure, 9  
 fault, 9  
 feasibility, 3  
 feasibility studies, 14  
 hardware interface requirement, 8  
 Hardware interfaces, 8  
 Human-Computer Interaction, 8  
 Human-computer interfaces, 22  
 ICD, 29  
 IEEE Std 830-1984, 23  
 integrity, 10  
 interface, 7  
 Interface Control Document, 4  
 internal interface, 7  
 interview, 13  
 ISO reference model for OSI, 7  
 maintainability, 9  
 method, 13  
 methods, 11  
 model, 17  
 Operational environment, 26  
 performance, 6  
 portability, 7, 9  
 prototype, 14, 32  
 Prototyping, 14  
 reliability, 9  
 Resources, 11, 33  
 response requirement, 6  
 safety, 7, 10  
 SCM42, 32  
 SCM43, 32  
 SCMP/SR, 3, 32  
 security, 7, 10, 28  
 Software interfaces, 8  
 Software Project Management Plan, 11  
 Speed, 6, 28  
 SPM02, 31  
 SPM03, 31  
 SPM04, 31  
 SPMP, 10, 11  
 SPMP/SR, 3, 20, 31  
 SQA03, 32  
 SQA04, 32  
 SQA05, 32  
 SQAP/SR, 3  
 SRB, 21  
 stability, 31  
 Standards, 10  
 Structured English, 14  
 Study, 13  
 survey, 13  
 SVV09, 32  
 SVV10, 32  
 SVV11, 33  
 SVVP/AT, 3  
 SVVP/SR, 3, 32  
 System Requirements Document, 22  
 TBC, 27  
 TBD, 23  
 Timescales, 11  
 Tool, 18  
 UR01, 22  
 UR02, 27  
 UR03, 27  
 UR04, 27  
 UR05, 27  
 UR06, 27  
 UR07, 28  
 UR08, 12  
 UR09, 27  
 UR10, 19  
 UR11, 19

D-2

ESA PSS-05-02 Issue 1 Revision 1 (March 1995  
INDE)

UR12, 19  
UR13, 19  
UR14, 19  
UR15, 19  
UR16, 19  
URD, 3, 19  
Validation, 11, 31  
verification, 31