

User's Manual as a Requirements Specification

**Daniel M. Berry, 1991, 1998, 2002, and 2003
with help from K. Daudjee, J. Dong,
I. Fainchtein, M.A. Nelson, T. Nelson, &
L. Ou**

Advice on writing UMs

You vs. User

There are two ways to describe the user in a UM:

- **“You” (second person) and imperative sentences with implied subject of “you”**
- **“the user” (third person)**

You vs. User, But Only One

In the last analysis, it does not matter which one you choose.

However, use *only* one; do *not* mix the two.

If you use both, the reader wonders if there are two kinds of users.

You vs. User Tradeoff

Second person is textually shorter than third person:

**“You enter ‘exit’.” is shorter than
“The user enters ‘exit’.”,**

and imperative:

“Enter ‘exit’.” is even shorter.

A Definite NO NO

If you use “the user”, remember that it is singular.

Therefore, the correct pronouns for it are:

“he”, “she”, and “he or she”

and NOT

“they”!

Grrrrrrrrr!!!

Glossary of Terms

From the very beginning of the writing of a UM, you should build and *maintain* a glossary (dictionary, lexicon) of technical terms for the manual.

This glossary is not only for the benefit of the reader, but also for your benefit as the author to avoid two terrible scourges of writing that tries to be interesting,

- **US = unnecessary synonymy and**
- **CP = confusing polysemy**

Unnecessary Synonymy

US is using more than one word for the same concept, e.g., ...

**“the program”, “the software”, “the system”,
“X” (where X is the name of the program),
“the X software”, etc.**

The reader is left wondering if there are even subtle differences between these different terms.

Necessary NonSynonymy

Occasionally you may need to distinguish between the software as an artifact supplied on a medium and an invocation of the software.

Necessary NonSynonymy, Cont'd

In that case, you make two entries into the Glossary:

“the *X* program = a copy of the *X* program on a CD” and

“*X* = an invocation of the *X* program running on your computer”,

and you carefully maintain the distinction in the writing.

Confusing Polysemy

CP is using one word for more than one concept, e.g., ...

Confusing Polysemy, Cont'd

One document that used “OS” to mean

operating system,

an open source program, i.e., an artifact,

**open source software as a kind of
software,**

**open source development, i.e., a process,
and**

**the open source phenomenon, i.e., a
metaprocess.**

Confusing Polysemy, Cont'd

It never talked about an open source operating system, which would be “OS OS”, but even so, it was a *very confusing* document.

Note that the authors *knew* which meaning of “OS” they meant each time they used it, but the readers had to *guess*.

Plural of Acronyms

Be careful of acronyms in which an interior noun or an irregular noun gets pluralized:

“request for proposals” = “RFP”

“requests for proposals” = “RFPs”

“brother-in-law” = “BIL”

“brothers-in-law” = “BILs”

“brethren-in-law” = “BILs”

Plural of Acronyms, Cont'd

The pluralizing “s” comes at the end of the acronym even when it comes in the middle or not at all in the expansion of the acronym.

***Never* let an acronym be its own plural, even if its pronunciation is word that is its own plural, e.g.,**

“FISH” and “FISHs”

UMs Should Lie

The manual should be written well enough that it deceives the reader into believing that the software really exists.

In fact, it's getting the picky details worked out well enough to write the deceptive UM that forces ironing out those synergistic problems that plague many requirements, and even design, documents.

Faking it [Parnas & Clements, Simon & Garfunkel], again!!!

Present Tense

Important rule for any specification of a CBS:

Use present tense to talk about what the CBS *does*.

Consistent with faking it, that the CBS is already implemented.

Why Rule is Needed -1

Rule is needed so that:

When it is necessary to talk about something that happens in the user's future, after the user's present in which he or she says something to the CBS, future tense can be used to distinguish the future event from the user's input.

A typical specifier writes a specification for a not-yet-implemented product in the future tense, talking about what the CBS *will do*.

Why Rule is Needed -2

After the CBS is implemented in the future, the user will enter some input, and the CBS will do something in response.

The specifier loses ability to distinguish between the user's present and the user's future.

Everything happens in the specifier's future.

“Shall” vs. “Will”

There is a convention that is observed in many design and engineering disciplines for writing specifications:

**When describing a requirement for the system to be built, say “The system shall ...” ...
(optative mood)**

**and leave “will” to describe future events.
(indicative mood)**

“Shall” vs. “Will”, Cont’d

In the vernacular, “shall” is often used as a future indicator with an additional compulsion component.

In specifications, “shall” is *reserved* for indicating requirements.

So be careful of how you use it!

Parentheses

A very common problem in technical writing is non-parenthetical parenthesized material.

Parenthesized material is any thing surrounded in a pair of parentheses.

Except for some very specific uses described later, parenthesized material is supposed to be parenthetical, i.e., not essential to the meaning of the containing sentence.

Parentheses, Cont'd

You're supposed to be able to remove the parenthesized material with no lost in meaning.

Probably because of our non-parenthetical use of parentheses in mathematics, we tend to use parentheses in technical writing for non-parenthetical purposes, e.g.,

'George saw Irving. He (George) said, "Hi!'"

Parentheses, Cont'd

**Here, the parenthesized material is essential.
Rewrite the example as**

'George saw Irving. He, George, said, "Hi!"'

**Often, each use of a pair of parentheses
expresses a different relationship between the
enclosed material and the containing material,
e.g.,**

**'The (security) (sub)system(s) need to be
subjected to hazard analysis.'**

Parentheses, Cont'd

Rewrite the offending text not to use parentheses, writing several sentences if necessary to get across the full meaning.

It *is* legitimate to use parentheses to introduce acronyms or to set off bibliographical citations. Such a use is actually parenthetical, even if only locally within the containing sentence.

Parentheses, Cont'd

In fact, except for these uses, it is really hard to think of any truly parenthetical use of parenthesized material in a specification. If it were truly not essential, you probably would not have said it in the first place.

Bottom line: Avoid parentheses, except in mathematical formulae, to introduce acronyms, and to set off bibliographical citations.

Quotation Marks

A pair of quotation marks is used legitimately to enclose a quotation or a phrase used as itself, e.g. as in

‘The word “word” has a “w” and three other letters.’

Quotation marks can be used to set off an ironic use of a phrase, e.g.,

‘John McCain is a “liberal”.’

Quotation Marks, Cont'd

A common use these days for quotation marks is to surround vague terms that cannot be defined precisely, e.g.,

'The system is "user friendly".'

to apologize for a poorly defined term.

Quotation Marks, Cont'd

Instead of apologizing, just define it as well as you can and then use it with no apology, e.g.,

**'Software is regarded as user friendly if
The system is user friendly.'**

Quotation Marks, Cont'd

Another typical other use of a pair of quotations is to surround a word that may be slightly misused.

'The database has "integrity".'

However, English is flexible enough that the metaphorical use of a word is legitimate.

Quotation Marks, Cont'd

So if the word you are about to enclose in quotation marks is the most descriptive you can find, then use the word proudly and do not apologize for it by enclosing it in quotation marks.

'The word "good" has a "w" and is good.'

means that whatever "good" means, the word "good" is good, but the apologetic version,

Quotation Marks, Cont'd

'The word "good" has a "w" and is "good".'

ends up being a logical tautology, X is X, or the ironic version, that

'The word "good" has a "w" and is bad.'

Bottom line: use quotation marks to enclose *only* a quotation, a phrase used as itself, or an ironic use of a phrase.

Quoting Input & Output in Text

In any manual, you will need to quote text that is input to or output from the system being described.

The reader must be able to distinguish the use of a word as itself and as input or output.

Quoting I/O in Text, Cont'd

Usually, one uses quotation marks for this purpose:

'You must enter "enter" to the program.'

However, in any UM, there will be *lots* of instances of these quotation marks.

Quoting I/O in Text, Cont'd

You can avoid all the quotation marks by establishing a typeface, i.e., font, convention for normal text and for input and output text, e.g.:

Normal text: Roman Serif

Normal text definitions: Italics Serif

Normal text headers: Bold Serif

I/O actual text: Roman Sansserif

I/O variable: Italics Sansserif

Quoting I/O in Text, Cont'd

Examples of serif typefaces are:

Times and Times New.

Examples of sansserif typefaces are:

Helvetica and Arial.

Quoting I/O in Text, Cont'd

So,

'You must enter "enter" to the program.'

would be written as

'You must enter **enter to the program.'**

Quoting I/O in Text, Cont'd

Very Important!!!!!! The text you use for I/O text should match what is in any screen diagrams.

If you use a serif typeface in the screen diagrams, make your normal text sansserif.

Specifying “How” Information?

We are admonished to specify What a CBS does, and not How the CBS does it.

Sticking to What gives the implementers the greatest freedom.

Sometimes it is necessary to give some How details, usually under the rubric of “Architecture”.

It is possible to do so in a UM.

How Spec Example -1

E.g. Knuth exposed the line-breaking algorithm for $\text{T}_{\text{E}}\text{X}$ in *The $\text{T}_{\text{E}}\text{X}$ book*, which serves both as a specification of $\text{T}_{\text{E}}\text{X}$ and as a UM, for 2 reasons:

- 1. to ensure that *all* implementations of $\text{T}_{\text{E}}\text{X}$ produce the same formatted output, even down to the line breaks and spacing between words, and**

How Spec Example -2

- 2. to give the user enough smarts about the line-breaking algorithm that he or she can exercise the commands to achieve the desired line breaking and interword spacing.**

That the specification of T_EX is its UM served as a filter to make sure that a How detail showed up in the manual only when the detail is necessary for the user's effective use of T_EX.

Special kinds of UMs

Other Kinds of UMs

The above has talked about UMs as RSs for Information Systems.

There are other kinds of systems with no traditional UMs

- **Embedded, e.g, device controller or aircraft**
- **Platform, e.g. POSIX, X**

So it looks like advice fails

No! Just have to be creative at identifying users

Embedded System E

User of E is another system S with which E interacts.

Consider the programmer of S that uses E .

He or she needs to know what functions E offers under what conditions.

The manual for these functions of E is equivalent to a requirements specification for E .

Platform P -1

User of P is an application A running on top of P .

Consider the programmer of A that runs on top of P .

He or she needs to know what services P offers.

The manual for these services of P is equivalent to a requirements specification for P .

Platform *P*-2

Manual should be written as a system programmer's manual for *P*, e.g., a collection of UNIX manual pages for Sections 3 and 4, programs and data.

Guess what! That's exactly the format of the POSIX and X standards = specification. It is a specification in that anyone can implement these in whatever way he or she wants, so long as the specified external interface is met. These standards are written as the manual for any implementation.

Requirements for Platforms

A computing platform, e.g. operating system, has users that differ from the users of a single application.

A platform user is a sophisticated user who programs applications that run on the platform, for use by others.

Equivalence of RSs and UMs holds even for these platforms.

POSIX Example -1

E.g., the POSIX system is a standard generalization of the various UNIX platforms.

POSIX is specified by collection of UNIX-style man pages describing the kernel routines and data that are available to use to write applications running on any POSIX system.

POSIX Example -2

The man pages

- **describe what an implementation of a POSIX must make available; they give for each kernel routine the interface it must support.**
- **describe what an application running on a POSIX may assume; they give for each kernel routine the interface that can be assumed by invokers.**

UM for Platform

Even for an operating system, a well-written UM can serve as a RS.

Of course, this UM, aimed at the programmer of applications, may not appear well written to the user of an application.

Why UMs Validate Well

UMs seem easier to validate than traditional SRSs.

Why?

One group in the Technion SE studio insisted on writing a traditional SRS rather than the suggested UM.

They did a good job of it, following a standard template to the letter.

Comparing SRSs and Manuals

Thus, Berry had a chance to compare UMs for WD-PIC to a traditional SRS for WD-PIC.

Even though Berry is

- **thoroughly familiar with WD-PIC and**
- **thoroughly computer literate,**

he had a hard time understanding some specifications of some features in the SRS.

SRS vs. UM -1

With SRS, Berry could not see that what was specified was not quite what he wanted.

With UM, Berry had no such problems; He was able to instantly spot specifications that did not capture his desires and to describe what was wrong and how to fix it or at least what the misunderstanding was.

SRS vs. UM -2

The clarity of the two specifications were like night and day.

Berry empathized with customers who report that they understand and accept specifications that they were too embarrassed to admit that they had not understood at all.

Key Difference -1

The normal SRS describes only what the system does.

The UM describes conversations between user and system to achieve user's goals.

The UM was more alive.

Berry could see himself as the user in the manual.

Key Difference -2

He could spot instantly described user behavior that did not correspond to what he would do.

The normal SRS does not describe user's inputs and reactions.

It describes only system's behavior in a vacuum from the user.

Key Difference -3

So, Berry had no idea what user behavior was implied.

Thus, if a behavior bore any resemblance to his preconceived idea of system behavior, he believed that the SRS described what he wanted.

All the Gory Details -1

No requirements specification method that does not force working out the details is going to work.

It is only in working out the details that all the show-stopping exceptions and interactions are going to be discovered.

All the Gory Details -2

These details can be worked out in any of several media:

- **the software itself,**
- **a complete formal specification,**
- **a complete, traditional SRS, or**
- **a complete, scenario-based UM.**

The advantage of UM is that changing the manual consistently is much cheaper than changing either the software itself or a complete formal specification.

All the Gory Details -3

Also, unlike a complete, traditional SRS, a UM is both needed and perceived as needed after the software is delivered.

Thus, the motivation to keep it up to date is higher than that to keep a traditional SRS up to date.

All the Gory Details -4

The advantage of the software itself or a complete formal specification is that it is hard to handwave over the details, to cheat to leave the impression of completeness when details are missing.

If details have been left out, the software will not work or the formal specification cannot be verified to satisfy requirements.

All the Gory Details -5

While it is fairly easy to leave details out of a UM, since the UM is intended to be delivered with the software to help naive users, the incentive is to get those details in.

Thus, it is an issue of finding a right medium for expressing detailed requirements that is both cheap to change, but hard to hand wave one's way to a false impression of completeness.

Requirements Notations -1

As you are writing a RS, you may use whatever organization and notation that helps you achieve these objectives.

Requirements Notations -2

When picking the organization and notation, remember the intended audience.

This means not using a notation that will turn your audience off.

On the other hand, this does not mean to shy away from a notation from fear that the audience will not understand it.

Requirements Notations -3

It is my experience that any notation

- **that suits the situation in which it is used**
- **that is consistently applied**

can be learned by anyone of reasonable intelligence that is in a position to understand what is being conveyed, e.g., a client.

Requirements Notations -4

After all, the software is being developed for the client's application domain, and he or she clearly knows the vocabulary of that application — the professional jargon.

Requirements Notations -5

For example, a state machine is a very natural way to describe reactive systems, but is not a good way to describe calculation of the standard error of the difference between two vectors, and is not a good way to describe the formatting of a paragraph.

For the client of a reactive system, a state transition diagram would make perfect sense, while for the statistician or the word processing specialist, a state transition diagram would appear as mumbo-jumbo.

Requirements Notations -6

A control and data flow diagram is a very natural way to describe an industrial process in which there may be several activities happening concurrently.

For the client of a process control system, a control and data flow diagram would make perfect sense, while for the statistician or the word processing specialist, a control and data flow diagram would appear as gobbledegook.

Requirements Notations -7

My experience has been that a good notation just happens during the elicitation and analysis phase.

It happens because at the time it was introduced, it seemed like the logical thing to use.

Requirements Notations -8

When it is introduced in this way, the notation is understood instantly or with minimum explanation by all involved in the elicitation and analysis.

The notation writes itself!