Creating user manuals for use in collaborative design

Harold Thimbleby
Middlesex University
Bounds Green Road
London, UK
Tel: +44 181 362 6061; Email: harold@mdx.ac.uk

ABSTRACT
User manuals are usually written by technical authors after the design of the device has been committed for production. If the manual's review leads to insight into the design, it is too late. Meanwhile, if the design is modified, the manual may be inaccurate. This paper describes an example language for creating accurate and complete manuals from formal specifications. We show how it can be used to improve part of the Flight Crew Operating Manual for the Airbus A320 fly-by-wire airplane. The technique is easy to implement, can be generalised to other domains, and contributes to concurrent engineering practice—increasing common ground between engineers, users and HCI practitioners.

Keywords
User manuals, formal specification, concurrent engineering.

INTRODUCTION
User manuals are the scapegoat of bad system design. Systems with bad user interfaces are difficult to explain clearly, how ever skillful the manual's technical authors. Many guidelines have been proposed to improve user manuals, notably Carroll's work [2]. It seems, however, that most attempts to improve manuals take the design of the system for granted. Maybe Carroll's minimal manual idea could be used to drive design: if a better manual is shorter, then a better design might lend itself to being explained in a briefer manual [7]. As pointed out in [7], to do this assumes concurrent engineering practice. Writing the manual after the design has been finalised is too late (even if writing proceeds in parallel with system implementation). Insights technical authors and others have reading the manual are too late.

Computer science is increasingly moving towards formal specification, where the design of a system is described mathematically, analysed, and only then transformed into software, firmware or hardware. Formal methods may be *required* in military and safety critical applications. Though this may improve the technical quality of the delivered product, it adds yet another step in the design process, and makes the inital design work even less accessible to users and non-technical experts, including user interface designers. What is required is a fast way to move from formal specification to user manual.

We propose an approach that takes a specification of the device as a set of predicates (as in TLA [4]) and automatically writes a (hypertext) minimal manual. As systems become more complex it is essential to have computer support to guarantee the correctness (and continuing editability) of their documentation. Serious accidents have occurred with very simple manuals done by hand [1]. Our own work has found further flaws in the A320 Flight Crew Operating Manual (FCOM).

PREVIOUS WORK
The work described here builds on earlier efforts: HyperDoc [8] is an interactive device simulator that generates interactive assistance, hypertext manuals, and also supports sophisticated design analysis in *Mathematica*; Manual Writer [9] is a simple program taking Prolog system specifications and managing the technical authors' and engineers' concurrent revision and editing.

These approaches are restricted to finite state machine (FSM) descriptions. This is not as problematical as it may sound, since FSMs need never be represented explicitly by the designers—they can be constructed in *Mathematica* or Prolog. However, FSMs lack structure and the design of a good manual is a hard problem. We have had some success generating statechart-based manuals automatically, but it seems better to use a higher level approach than FSMs.

An alternative approach is to use sophisticated natural language generation, such as IDAS [5]. There is scope for both approaches in technical authoring, but in contrast to us, these approaches typically rely on world knowledge databases, whereas we wish to emphasise accuracy and mathematical tractability, and *simplicity*.

EXAMPLE: ORIGINAL A320 MANUAL
We give a verbatim exerpt from the A320 FCOM Brakes and Antiskid, 1.32.30, Rev 15, Seq 001 (quoted in [3]):

> **ALTERNATE BRAKING WITHOUT ANTI-SKID**
>
> The anti-skid system is deactivated:
>
> - electrically (A/SKID and N/W STRG sw OFF or power supply failure or BSCU failure)
> - or hydraulically (Y+G sys to lo press, the brakes are supplied by the brake accumulators only).

Note unexplained abbreviations such as Y+G, and that the phrase "A/SKID and N/W STRG sw OFF or power supply failure or BSCU failure" has at least three different interpretations.

Ladkin took the A320 FCOM and derived a specification of the braking subsystem [3]. Part of this specification looks like this: (A/S off and N/W STRG off) or power supply failure or BSCU failure or (green lowpress and yellow lowpress).

Ladkin's reverse engineering exposed many problems with the FCOM. Reverse engineering is not necessary where a specification is otherwise available.

THE MANUAL SPECIFICATION LANGUAGE

The manual language allows arbitrary propositional statements using Boolean connectives and other operators, such as [*e*] which is true iff *e* is a contingency. An example statement is: `"Normal without A/S" = Normal & ¬Antiskid`

Explanations are generated in several ways. For example, `?` generates a tabular explanation, as illustrated below.

The explanation operators are not just **print** statements—they are more sophisticated. Explaining `Normalwithout A/S` means explaining `Normal`, which itself needs explaining, involving further terms such as "…`and not BSCUfailure`…" The full explanation has to be *minimised*, and we use the Quine-McCluskey Algorithm to do so [6]. Certain terms are rewritten, so we can explain ¬(`failure`$_1$ `or failure`$_2$…) as "`All systems OK.`" Similarly, the engineers may wish to write in terms of `Y` or `G` (as in the FCOM extract above!) but the user may prefer to read clearer text. Designers require various checks, for instance that all states of the device are covered by the manual; such checks can now be made part of the manual.

This paper does not have space to show how rewrite rules and tables are used to clarify factorisable expressions of the form *A*(*B*+*C*), etc. These simplifications also require computer support to be done correctly and well.

EXAMPLE: AUTOMATICALLY GENERATED MANUAL

ALTERNATE BRAKING WITHOUT ANTI-SKID Alternate braking without anti-skid mode is achieved when:	
	A/SKID switch set OFF **and** N/W STRG switch set OFF
or	BSCU failure
or	power supply failure
or	**both** green **and** yellow hydraulic pressure insufficient **and** autobrake is inoperative

A table is used to unambiguously and clearly present the logical expression (other forms are possible). Some text in the original has been moved to reduce clutter, but is still accessible via hypertext linking, e.g., every mention of BSCU is linked to its explanation, "`The double channel Brake Steering Contol … has modes operative or failure.`"

As described here, there is no scope for fresh editorial contributions from technical authors. If a phrase generated by the process can be improved, once it is, the link from the specification is lost, and with that, all guarantees that it was correct. This problem has been solved elsewhere [9].

The Quine-McCluskey Algorithm finds use in simplifying digital circuits. As well as being minimal (hence fast and cheap), logic is usually designed to eliminate race conditions and other problems. Race conditions occur in user manuals: e.g., "Insert the plug and take care to switch off first"—the user may insert the plug without *first* switching off. We are exploring the use of circuit design tools to analyse user manuals to relate usability requirements back to specifications.

CONCLUSIONS

The approach easily generates an improved extract for the A320 FCOM. Had such a system been available when the A320 was designed, user interface specialists and pilots would have been able to address some of the design issues that are now so obvious. Had the A320 specification design changed as a result of these insights, the manual could have been updated almost instantaneously. This sort of leverage must be a major contribution to the effectiveness and acceptability of iterative design in industry.

Given that the approach is so simple to implement, one wonders why similar methods are not yet available to system designers to improve their effectiveness in *collaboration* with user interface experts. Though the arguments for the approach seem overwhelming in safety-critical applications like the FCOM, the approach can also increase quality in everyday devices such as consumer electronics (cf, [9]).

Acknowledgements

Peter Ladkin contributed enormously to the approach taken.

REFERENCES

1. J. André, "Can structured formatters prevent train crashes?" *Electronic Publishing—Origination, Dissemination and Design*, **2**(3):169–173, 1989.

2. J. M. Carroll, The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill, MIT Press, 1990.

3. P. B. Ladkin, "Analysis of a technical description of the Airbus A320 Braking System," *High Integrity Systems*, in press.

4. L. Lamport, "The Temporal Logic of Actions," *ACM Transactions on Programming Languages and Systems*, **16**(3):872–923, 1994.

5. E. Reiter, C. Mellish & J Levine, "Automatic generation of technical documentation," *Applied Artificial Intelligence*, in press.

6. W. V. Quine, "A way to simplify truth functions," *American Mathematical Monthly*, **62**:627–631, 1955.

7. H. W. Thimbleby, *User Interface Design*, Addison-Wesley, 1990.

8. H. Thimbleby & M. Addison, "Intelligent adaptive assistance and its automatic generation," *Interacting with Computers*, in press.

9. H. Thimbleby & P. B. Ladkin, "A Proper Explanation When You Need One," in M. A. R. Kirby, A. J. Dix & J. E. Finlay (eds), *Proceedings BCS Conference HCI'95*, **X**:107–118, 1995.