

From logic to manuals

[Harold Thimbleby](#)
[Computing Science](#)
[Middlesex University](#)
GB-London, N11 2NQ

[Peter Ladkin](#)
[Technische Fakultät](#)
[Universität Bielefeld](#)
D-33501 [Bielefeld](#)

Abstract

We demonstrate a simple language that combines specifications and manuals. This shows: first, that a user manual can be automatically reconstructed from a logic specification that is effectively identical to the original logic (up to ambiguities in natural language); second, that such an automated process can help detect errors. The process is fast, and suitable for use in participatory design.

29 February 1996; revised 5 September 1996. Published in *Software Engineering Journal*, **11**(6), pp347-354, 1997.

Introduction

To use complex systems safely and efficiently requires appropriate training and documentation, based in turn on an understanding of the systems themselves. Systems are (or should be) based on a *specification* of what they are to do. We will take a requirements or design specification henceforth to be a (logical) predicate in a precise language that a system is required to satisfy. This specification puts conditions on certain state variables of the system. A system satisfies the specification just in case it exhibits no behavior that is proscribed by the specification, and it fails to satisfy a specification just in case it may exhibit behavior that is proscribed by the specification. How do we tell the users how to use the system? How do we ensure that our instructions are logically compatible with the specification?

It is widely recognised that requirements change and systems evolve, amongst other reasons because of design needs, and in response to user feedback [[SwaBal82](#)] [[Boe86](#)]. In systems where users form an essential part, it is common practice to involve the users in the design process. As users increasingly understand a system, they may help change its design for the better. Logically accurate user documentation is desirable in itself, and a reliable, fast method for generating it could improve the design-trial-redesign loop. We describe a simple prototype tool, implemented by the first author, that accomplishes

this.

Aircraft control and flight management is a human-in-the-loop system. We use it as an example in this paper, partly because some actual user manual pages have been reverse-engineered by the second author [Lad95] and now exist in a form on which we can demonstrate the forward-generation of new manual pages; and partly because it is pilot folklore that Flight Crew Operating Manuals (FCOMs) are in constant need of improvement [Buc94]. However, the technique we illustrate is applicable to derivation of accurate documentation in general, and the tool we describe is not limited to particular types of systems.

Deriving documentation from specifications is not new. [LevHeiHil94] and [HeiLev96] describe the provision of full formal requirements specifications for TCAS II avionics systems. [ParMadIgl94] and [ParMad95] describe the documentation of the software design process: "*The validation of design documents [...] is a major part of engineering. [...] Documentation can be used both as a design medium and as the input to subsequent analysis and testing activities.*" The IDAS project [ReiMeLev95] aimed to derive technical documentation in readable natural language, aimed at "*technicians and other experts [rather than] 'the man in the street' (e.g., aircraft maintenance manuals, not VCR operations manuals)*". In contrast with Leveson *et al.*, we do not aim to document requirements. In contrast with Parnas *et al.*, we do not aim to document a development process for engineering purposes. We do aim to support the provision of a complete, consistent, usable end-user manual for a process-control system. We assume the system requirements and development as given. In contrast to the IDAS work, we do not aim to automate the production of natural language instructions, and we are more concerned with documenting real-time systems to enable timely user response in changing situations, such as needed in FCOMs. This does not preclude eventual synthesis between the approaches.

Documenting Avionics systems

With the introduction of so-called fly-by-wire airplanes in commercial transport, avionics now encompasses all aspects of piloting. A pilot of such an airplane is interested in what the specification says about only the state variables that are immediately 'to hand' - say, *position of thrust levers* and its relation to *measured engine power* - and their changes. Other variables such as those describing, say, *engine fuel-flow controller state* are more important for design and maintenance engineers than for flying the aircraft. So the pilot's view can be regarded as an abstraction of the design specification of the whole aircraft, namely a predicate in a restricted vocabulary of state variables that is logically implied by the design specification.

It is important to distinguish the particular needs of an operator from the general need to document the development and design of a system. An operator will in general not

care how, or using what process, the system was developed. A pilot only cares how the airplane functions *right now* and how to control it. A pilot flying a landing go-around will use features of the *design* (position of the controls, for example) in order to command the aircraft to fulfil certain *requirements* (in this case, performance requirements). One can expect an operations manual thus to contain information from both requirements and design.

We are interested in deriving end-user manuals from requirements or design specifications. The meaning we give to the word 'specification' allows a precise, correct statement of what the operator 'sees' also to be described as a specification. A system may thus have many specifications of this type. (We take it as one of the advantages of the logical approach to specification that it handles this form of abstraction so trivially.) The FCOM is an important end-user document recalling essential information for the pilots, and is legally required to be on board an aircraft in flight. It omits information maybe necessary for engineering but of little help in the cockpit, as well as much information on the system design pilots would have learnt in a training course. A crucial feature of the FCOM is that the airplane (and the pilots!) should behave exactly in the way detailed in the FCOM. The FCOM thus should be a specification in our sense, for some part of the total behavior of the aircraft.

The second author [[Lad95](#)] analysed part of the FCOM for the Airbus A320 airplane from this point of view as a logical document. He showed that the FCOM for a particular A320 (the one reported in [[MCAA194](#)]) has shortcomings if treated as a rigorous definition; namely, ambiguities and omissions. He rewrote the FCOM pages concerned as Predicate-Action Diagrams [[Lam94a](#)], a way of describing finite state machines in which states and transitions between them are labelled with predicates with precise semantics in the Temporal Logic of Actions (TLA). TLA is a temporal logic designed for system specification and verification [[Lam94b](#)].

What we regard as logical shortcomings in an FCOM may have origins in the conflicting needs of pertinence, coverage and brevity, as well as in simple error. Some shortcomings may arise from a psychological judgement of how best to promote recall of systems and emergency procedures that pilots have learnt in training but are seldom required. Desiderata for operating manuals may thus conflict. However, we suggest that consistency with specifications is not in conflict with any other desideratum. We call such a manual *accurate*. (It follows that accurate manuals must be specifications in our sense.) To fulfil other desiderata, a method generating accurate manuals should allow optimisation possibilities according to different criteria. Our method does this, starting from specifications whose accuracy we assume is given.

There is a legal requirement in aviation for accurate flight crew operating manuals. Buck [[Buc94](#)] strongly makes the point that manuals frequently fall short of what pilots need in a difficult situation, that pilots in such situations often must improvise, and that this is becoming particularly difficult with the introduction of complex avionics. Avionics

systems can be considered *interactively complex* and *tightly-coupled* [Per84]. A suitable illustration may be derived from a recent accident to a Birgen Air Boeing B757 aircraft in February 1996 [Lad9x]. There are two relevant features of this accident, which illustrate the difficulties of writing quickly-usable manuals when the system is interactively complex.

- The captain, who had suffered and identified a failure of his airspeed indicator (ASI), called for the center autopilot to be turned on. However, the captain's ASI gets its information from the Left Air Data Computer (ADC), as does the center autopilot. The captain's ASI indicated increasing airspeed with increasing altitude (a phenomenon known to all pilots as consistent with a blocked pitot-static system) and the center autopilot, following the false airspeed data, raised the nose, causing the airplane to slow down and stall. Although there are standard pilot workarounds ('alternate' data paths, and a mechanical back-up) which could (one may speculate 'should') have been used, it is hard to imagine that the captain deliberately used an autopilot which he knew to be receiving false data. It could be argued that this is a consequence of interactive complexity: precise information on the center-autopilot/LADC coupling does not appear to be in the Operating Manual (which must be concise and quickly-usable, and thus compromise on the amount of information included), but it is included clearly in the much larger systems description document, which is a training and reference manual not intended for quick reference in an emergency [Anon96].
- The US National Transportation Safety Board (NTSB) suggested an improvement to the Operating Manual. During the accident sequence, the crew appeared to be confused by two advisory messages, *RUDDER RATIO* and *MACH/SPD TRIM*, which appeared on the cockpit Indication and Crew Alerting System (EICAS) display. They are displayed when the *stabilizer trim and asymmetry module computer* compares the several airspeed sensor inputs and discovers a discrepancy of more than 10 knots. The Operations Manual apparently did not inform pilots that these two advisories together indicate an erroneous airspeed indication. The NTSB has recommended to the US Federal Aviation Administration that the manual be revised.

Manual-writing for an interactively complex system is a task that is not easily optimised. The functionality provided by our simple tool could be expected to help. We adopted a re-engineering approach, respecifying existing manual fragments and applying consistency and completeness checks by hand [Lad95]. This was followed by forward development of the new manual using our tool from these specifications. The contents-in-the-large are determined by the domain experts; we analyse these contents for certain formal properties and preserve those properties during regeneration. The tool we describe here is

- simple,

- fully implemented,
- runs on real, if simple, examples,
- and produces a manual that is comparable with the original.

Our tool has been used as part of the *DiDoLog* system [Bar96], a simple experimental forward-generation tool for operating manuals. *DiDoLog* allows finite state machines to be built graphically using TLA definitions annotated to the states and transitions. TLA+ specifications are automatically generated from the annotated state machines, and finally a logical manual is generated with the tool described in this paper. (*DiDoLog* requires Tcl/Tk [Ous94].)

We demonstrate the tool by taking the re-engineered logic specification of the A320 braking subsystem from [Lad95] and construct precise FCOM pages from it, below. Apart from their precision, these pages are comparable in content and structure to the original. Their layout is also comparable, though we chose to use tables rather than bullet lists (as are used in the FCOM). When our automatic construction techniques are applied to appropriate specifications, the result is also a specification, presented in a more human-oriented way.

This entire paper (rather, its electronic version as submitted to this *Journal*) is itself the result of a single actual run of the tool. The example input is included as source, along with the text body of the paper as *meta-comment* (see below). The output of running the tool on the source of the paper then yields the text, the example input, plus the output to the examples, formatted as you read below. We did no cut-and-paste on the examples, nor any other post-editing of the paper.

The system is simple, and so are the examples. But that's partly our point. Specifications of user interfaces of critical systems *are designed to be simple*. We are describing a preliminary tool that handles examples of the appropriate order of complexity. However, in the average FCOM, there will be many such examples. Keeping track of the interconnections can be complex without automated help. This is partly why the traditional 'by hand' approach to manual writing is prone to lose rigor.

The usual benefits of automatic generation include the ability to maintain a correct manual under changes of the specification. This aids in *collaborative design* [Thi96], in which users are involved in assessing designs and hence improving their effectiveness or safety, as practiced with human-in-the-loop systems. We have mentioned that our approach facilitates the design-trial-redesign loop. In principle, this could be extended to concurrent engineering, in which technical authors would maintain documentation and evaluate it with operators *at the same time as* engineers maintain a complete system specification. We have discussed elsewhere techniques to allow technical authors to improve the quality of language used in manuals without compromising the guarantees of automatic generation [ThiLad95].

A language for manuals

Ladkin's preferred FCOM specification is in Predicate-Action Diagrams [*op. cit.*], which have a rigorous formal translation into TLA though use of the temporal-logical operators in his paper is *pro forma*. For the present paper it suffices to use Boolean logic.

Our language has Boolean expressions, using `!`, `&` (or `dot`), `+`, `=>` and `<=>` as the Boolean operators not, and, or, implies, equivalent; others are easy to add. Operator precedence is standard; association is to the left. Two print operators evaluate and print minimised expressions: `>e` prints a minimised equivalent to `e`; and `?e` prints a minimised and factored table in HTML (the hypertext markup language used on the World Wide Web [[NCSA96](#)]).

In addition, rewrite rules can be used to simplify expressions. A rule of the form `r := e` allows that the expression `e` is rewritten by `r`. Since rewrite rules may overlap (as in `r:=a&b; q:=b&c; ?a&b&c;`), their application requires minimisation. Use of rewrite rules is illustrated in the examples below.

Since one may not wish to print an explanation of something that is trivial, there are two more more operators. The slash makes printing using either `>` or `?` conditional; thus `>e/p` prints a minimised equivalent of `e` if `p` is true. Secondly, the expression `[e]` reduces to true iff `e` is a contingency (that is, iff `e` is neither strictly true nor false).

The language also provides for name bindings, including functions. To follow the examples below, we mention the following details: comments are taken from `%` to the end of the line; variable names are either strictly alphabetic strings, or anything between quote symbols (including `\` or `\\` which represent `"` and `\` in the conventional way).

Our language may not be formally the most elegant or complete. It was implemented to prove a concept, and it works. We do not provide a formal definition in this paper for several reasons: the program is available from the authors, we would rather encourage others to develop our work rather than take it as definitive, and anyway - a point already emphasised - it is a simple system that takes only a week or two to implement. We would like to encourage other workers - particularly in industry - to adopt the concepts, not the specific prototype. The next sections give some examples of real runs with it.

Examples

We distinguish *input* and *output* in our examples:

```
% Use a long variable name...
```

```
> "This is a simple example using a function we define";
```

This is a simple example using a function we define

% Define a function equals (one of many ways to do it)

```
equals(a, b)
  = a + b => a & b;
```

```
> equals(a, a);
```

```
true
```

```
> equals(a, true);
```

```
a
```

```
> equals(equals(a, b), equals(b, a));
```

```
true
```

```
> "Equals commutes"
  / equals(a, b) <=> equals(b, a);
```

Equals commutes

Minimisation becomes more interesting when it is given problems that do not reduce so simply, such as Exercise 1.16.A.3 from Zissos's textbook [\[Zis72\]](#) on digital circuit design:

```
zissos
  = !a & d + !b & c & d + a & !b & (c + d) + !b & !c & !d;
```

```
> zissos;
```

```
d & !a + !c & !b + !b & a
```

This result is different from the model answer, as the term $!c!b$ is not given: there is a mistake in the Zissos textbook.

Further, with the automatically generated tabular form, we see that $!b$ can be factored out from two terms:

```
? zissos
  <- "Zissos example";
```

	not <u>b</u>	and	not <u>c</u>
or	"	"	<u>a</u>
or	<u>d</u> and		
	not <u>a</u>		

Zissos example

Note how the $<-$ operator specifies a label for the table. The same label is used in the glossary that is automatically generated at the end of this paper.

Extracts from the A320 FCOM

Following the simple examples, we now turn to the A320 FCOM. Ladkin's paper gives extensive examples of ambiguities in the FCOM. We shall be content merely to show one example, that an unambiguous section of the FCOM can be duplicated.

Extract from original FCOM:

NORMAL BRAKING

Braking is normal when:

- green hydraulic pressure is available
- A/SKID and N/W STRG switch is ON
- PARKING BRAKE is not ON.

Anti-skid is operative and autobrake is available.

There is additional prose in the FCOM that is descriptive rather than logical. It can be directly copied to the manual generator's output using 'long' variable names, as shown in the examples above, or by using meta-comment.

Ladkin defined appropriate logical variables and determined the normal braking condition to be *normal = green hydraulic and a/s and n/w strg and not park brake and antiskid and autobrake*.

Writing Ladkin's formulas together with some explanatory rewrite rules gives the following manual specification:

```

"both power supply and BSCU operational"
  := !("BSCU failure" + "power supply failure")
  -> "Power normal.";

"both green and yellow hydraulic pressure insufficient"
  := !("yellow hydraulic" + "green hydraulic")
  -> "Hydraulics failure.";

"nosewheel steering switch is ON"
  := "n/w strg";

"nosewheel steering switch is OFF"
  := !"n/w strg";

% (some specification details omitted for brevity)

normal
  = "green hydraulic" & "a/s" & "n/w strg" & !"park brake" & antiskid &
  autobrake;

```

% Group lines of input together to get clearer output...

```
(
> "<h4>NORMAL BRAKING</h4>";
> "Normal braking mode is achieved when:";
? normal
  <- "Normal braking mode"
);
```

NORMAL BRAKING

Normal braking mode is achieved when:

<p>parking brake is OFF and green hydraulic pressure available and antiskid mode active and autobrake armed and antiskid switch is ON and nosewheel steering switch is ON</p>

Normal braking mode

We have chosen to display the formula as a table; we believe this to be clear and concise (see also [\[OkuMatHir96\]](#)). We could use different styles of presentation. In HTML, the variable names (possibly rewritten) are automatically-generated hypertext links to additional explanations of the variables or states being specified.

The tabular representation we chose is a minimal disjunctive normal form (DNF) equivalent to the original formula, with each line in the table being one term from the normal form. This representation closely matches the original form of the FCOM. The DNF lends itself to expressing precisely what the original FCOM chose to represent as bullet lists (one difference remarked by Ladkin [*op. cit.*] is that some of the bullet lists in the FCOM had obscure semantics).

We implemented the language with a minimisation algorithm that outputs minimised DNF, because it was useful for us. It is also a criterion that can be implemented precisely and optimally, and not confuse our claims with uncertainty over the exact nature of presentation heuristics. Minimal DNF helps, amongst other things, to check the logical meaning of desired manual entries, and that is what we wanted to try with the A320 manual pages. However, our approach is not beholden to minimal DNF. One can choose other features to minimise. One may prefer to output manual pages which satisfy other structural criteria (for cognitive reasons, say). We could also generate text that is not best presented as DNF. For example, many manuals contain conjunctions of implications, of the form:

((s11 and s12 ...) implies q1) and ((s21 or s22 ...) implies q2) and ...

We do not know what users of manuals would like to minimise under what circumstances, though there is evidence (and supporting argument) that minimisation of

some form is desirable [Car90]. For example, would users prefer to minimise the number of terms in a minimal expression, or the maximum number of factors in any term, or perhaps the greatest number of non-trivial common factors? There is scope for experiment, and we would guess that the results would depend on what users (in the current case, pilots) expect of the documentation. We do not wish to impose a particular answer. Indeed, one might provide a set of answers to accommodate various types of intended use.

Checking logical properties

Consider the following completeness condition: each state in the system should be documented by some mode in the manual, and modes in the manual should (where appropriate) be mutually exclusive. These and other checks are easy to perform and to document.

In the following example, we first define a function `possible(x)` to be true if `x` is possible, that is, if `x` is true or is a contingency. Next, if normal mode without antiskid is possible, we output the simple text "Alternate without antiskid and normal modes overlap when" (i.e., treating it as a simple variable name). Finally, we output as a table the conditions under which normal mode without antiskid is possible. We ran this function as follows:

```
possible(x)
  = x + [x];

normaltwout
  = altwout & normal;

(
> "<b>Alternate without antiskid and normal modes overlap when</b>:";
? normaltwout
)
/ possible(normaltwout)
<- "Overlap example";
```

Overlap example: **Alternate without antiskid and normal modes overlap when:**

	parking brake is OFF and green hydraulic pressure available and antiskid mode active and autobrake armed and antiskid switch is ON and nosewheel steering switch is ON	and	BSCU failure
or	"	"	power supply failure

Overlap example

We discover that the FCOM description does not explicitly account for power supply failure or BSCU failure modes. Thus can the tool be used for completeness checking.

The organisation of higher-level manuals

The FCOM is a much reduced form of material included in pilot training, which one would expect to cover items that are omitted or ambiguous in the FCOM. This suggests that one should be able to pick 'higher-level variables' for comment in the FCOM, but also that one should be able to go deeper (simply by adding variables to the 'domain') and view a specification at a more detailed level. This further suggests a hierarchical tool that could operate down to whichever level is desired by the user.

The current approach does not do this; it assumes the specification is 'flat.' However, extension would be easy, such as embedding it within an interactive hypertext system with some theorem proving capability. It would be important for any such extended tool to include various 'completeness' checkers. In particular, for a user to be able to request an abstract manual, the tool should be provided with a suitable summary of what information is *not* explicitly represented.

Glossary

It is arguable that a manual should contain a glossary and an index of all names used. Our software constructs one with simple hypertext features automatically. We implemented simple features that seemed to us to yield immediate benefits, without making the language unwieldy, but it falls short of what would be required for commercial use. The glossary for the examples in this paper was created automatically by running the source text of this paper through the program.

The glossary appears as follows. Any name, or head token of a rewrite rule, appearing in an explanation is linked to an entry created for it in the glossary. The glossary contains any descriptive text that has been associated with the name or rule, and also links back to the uses of the name. A user of a hypertext version of the manual can click on a phrase occurring in an explanation to obtain a full explanation of that phrase in the glossary, and similarly by clicking in the glossary move back to any use of the phrase in that or other explanations.

Beyond the clear advantages of a complete glossary, our approach closely associates glossary entries with the uses of the names (this is a 'standard' advantage of automatically constructed indexing). Thus it is easier to check the specification as it is being written, and built-in system checks ensure that all entries in the glossary are annotated.

The language provides three mechanisms to support the hypertext glossary. Any name or rule followed by `-> string` has `string` appended to its glossary entry. For example:

```
"switch on"
  := SW
  -> "Mains switch.";
```

```
"switch off"
  := !sw
  -> "Mains switch.";
```

```
"Mains switch."
  -> "The switch is at the bottom of the red console, pull it forwards to make
it 'on.'";
```

Any expression followed by `<- string` is named in the glossary (by the *string*) as the context of use for all of the names in the explanation generated by the expression. As mentioned above, the string also occurs in the explanations generated, as a caption. The sense is that the right-pointing arrow puts text further on in the document, into the glossary, and the left-pointing arrow refers backwards from the glossary to the expression. In the HTML, of course, these forward and backward references are hypertext links.

Finally, since manuals are often structured by sections, the same form without an expression defines a 'global context' for all following explanations - that is, as if they are in the context of a section with that label. The following example shows both forms in use:

```
<- "Mains switch example";
> "<h4>Mains switch example</h4>";
```

Mains switch example:

Mains switch example

```
? sw
  <- "The explanation of sw itself";
```

[switch on](#)

The explanation of sw itself

The glossary for all the examples is at the end of this paper.

Guarantees

All good software should come with a guarantee. In many papers, examples are written by hand, or edited from actual program output. Such massaging can lead a reader to wonder if what is presented is an accurate reflection of what the program is really doing. We guarantee that the output the reader sees in this paper is the actual output to the examples. Here's how.

The features of our language so far discussed allow a manual to be constructed from a specification, but do not allow the manual to 'talk about' the specification as an object in

itself, which (for example) is what the text of this paper does, and what would be required in various specification documents as a system design is negotiated. Thus we built a 'meta-mode' for the language.

In normal-mode, the program processes all specifications precisely as *described* in the examples. In meta-mode, however, it processes specifications and additional text. The text is considered as a 'meta'-comment. We have already mentioned the normal '%' comment, and examples of its use appear in this paper above. Meta-comment has no enforced typographical style, and the comment markers (which are braces) are *not* represented in the output.

This paper was created by meta-mode. The text of this paper, except the examples, was written as meta-comment. The text was input as 'source,' along with the input to all examples as described, but no output to any example. The output is the submitted version of the paper, which is identical up to typography to the version that the reader is currently reading, complete with output and glossary for all the examples. Evaluation in meta-mode enforces the font conventions for distinguishing input and output, to alleviate confusion. The actual source and output is available from both <http://www.cs.mdx.ac.uk/harold/> and <http://www.techfak.uni-bielefeld.de/~ladkin/>.

Meta-mode is invoked automatically for any specification that uses meta-comments. If an author uses meta-comments in a source, to talk about the specification, the system generates a manual representing those comments, the specification, and the explanation. Meta-comments can also be commented out, and even *those* comments can be meta-commented, *etc.*

Implementation issues

TeX [[Knu92](#)], LaTeX [[Lam94](#)] or SGML [[Bry88](#)] would all have been preferable as output markup languages. It is no problem to generate them instead: we just find HTML preferable for collaborating over the World Wide Web. HTML's tables (specifically, Netscape's tables) are fairly basic and restrict the ways in which a minimal expression can be represented. There are many alternatives that could be tried, such as decision trees - especially if we were to take advantage of interactive representations rather than conventional paper-like forms. Extensions to our approach should permit a choice of representations for the application. In turn, this would permit evaluation of metrics on the output; for example, if it is known that the depth of a decision tree relates to its ease or reliability of use, then the system should summarise such metrics.

The manual specification language was first prototyped as an embedded language within Prolog. Not surprisingly, as minimisation, factorisation and rewriting are all NP-complete optimisation problems, the Prolog was too slow for practical use! The program was rewritten in C, and used the standard low-level programming trick of implementing DNF with up to 32 variables as a vector of 32-bit words, which gave it acceptable

performance. Minimisation uses the Quine-McCluskey algorithm. There are many alternative algorithms that could also be used (including professional tools for optimising digital logic circuits).

The C program is limited in the complexity of queries it can handle (because the minterm algorithm can generate huge numbers of minterms). This has not been a practical limitation; one could argue that use of a specification that required more than 32Mb of RAM could involve such a degree of complexity that a mechanically generated manual would not help much!

Conclusions

We have given a proof of concept that a user manual can be produced automatically, using straightforward programming, in such a way as to be comparable with a desired original (modulo typography). This technology shows how easy it is for manual writers to check completeness and correctness, and indicates that, modulo the goals of particular manuals, automatic production of correct manuals is a relatively straightforward task. We have pointed out that the speed-up and quality improvements enable concurrent design, which should result in better systems through involving operators earlier, and more effectively, in the design process.

We have done no research into the 'best' form of FCOM, one meeting all the desiderata in mutually optimal ways. Such questions would need to be addressed in any development of our work. We showed here that we could provide the FCOM material with logic and precision, by using an automatic tool. To do that we did not need to address the human factors questions. Our tool, particularly given its precise functionality, can be used in controlled experimentation.

Acknowledgements

Thanks to Hagen Barth who ported the program and identified and fixed some hardware/compiler dependencies. We are grateful to Paul Curzon and to the referees, who made very useful comments on the paper.

References

[Anon96]: Anonymous Expert Source. [Back](#)

[Bar96]: H. Barth, *DiDoLog: Automatable Generation of Specifications and Formally Correct Manuals from Informal Sources*, Diplom (Master's) Thesis, Technische Fakultät, Universität Bielefeld. Available through <http://www.techfak.uni-bielefeld.de/~ladkin/>. [Back](#)

- [Boe86]: B. W. Boehm, *A Spiral Model of Software Development and Enhancement*, ACM SIGSOFT Software Engineering Notes, 11(4):14-24, August 1986. [Back](#)
- [Bry88]: M. Bryan, *SGML: An Author's Guide*, Addison-Wesley, 1988. [Back](#)
- [Buc94]: R. N. Buck, *The Pilot's Burden: Flight Safety and the Roots of Pilot Error*, Iowa State University Press, 1994. [Back](#)
- [Car90]: J. M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press, 1990. [Back](#)
- [HeiLev96]: M. P. E. Heimdahl & N. Leveson, *Completeness and Consistency Analysis of State-Based Requirements*, IEEE Transactions on Software Engineering, 22(6):363-377, 1996. [Back](#)
- [Knu92]: D. E. Knuth, *The TeXbook*, Addison-Wesley, 1984, 1986, revised 1992. [Back](#)
- [Lad95]: P. B. Ladkin, *Analysis of a Technical Description of the Airbus A320 Braking System*, High Integrity Systems, 1(4):331-349, 1995.
Also available from <http://www.techfak.uni-bielefeld.de/~ladkin/>. [Back](#)
- [Lad9x]: P. B. Ladkin, ed., *Computer-Related Incidents and Accidents with Commercial Airplanes*, Hypertext Compendium of sources and commentary.
Available from <http://www.techfak.uni-bielefeld.de/~ladkin/>. [Back](#)
- [Lam94]: L. Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, 1994. [Back](#)
- [Lam94a]: L. Lamport, *TLA in Pictures*, IEEE Transactions on Software Engineering, 21(9):768-775, 1995. Also available from <http://www.research.digital.com/SRC/tla/>. [Back](#)
- [Lam94b]: L. Lamport, *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, 16(3):872-923, 1994.
Also available from <http://www.research.digital.com/SRC/tla/>. [Back](#)
- [LevHeiHil94]: N. Leveson, M. P. E. Heimdahl, H. Hildreth & J. D. Reese, *Requirements Specification for Process-Control Systems*, IEEE Transactions on Software Engineering, 20(9):684-707, 1994. [Back](#)
- [MCAA194]: Main Commission Aircraft Accident Investigation Warsaw, *Report on the Accident to Airbus A320-211 Aircraft in Warsaw on 14 September 1993*, Warsaw, March 1994. Text body without appendices also available from [\[Lad9x\]](#). [Back](#)
- [NCSA96]: NCSA, *A Beginner's Guide to HTML*. Available from <http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>, 1996. [Back](#)

[OkuMatHir96]: H. Okuno, H. Matsumoto & H. Asai, *TableSpec: Free Format Specification Table and Source Code Generation*, Software - Practice & Experience, 26(2):213-235, 1996. [Back](#)

[Ous94]: J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994. [Back](#)

[ParMadIgl94]: D. L. Parnas, J. Madey & M. Iglewski, *Precise Documentation of Well-Structured Programs*, IEEE Transactions on Software Engineering, 20(12):948-976, 1994. [Back](#)

[ParMad95]: D. L. Parnas & J. Madey, *Functional Documents for Computer Systems*, Science of Computer Programming, 25:41-61, 1995. [Back](#)

[Per84]: C. Perrow, *Normal Accidents: Living With High-Risk Technologies*, Basic Books, 1984. [Back](#)

[ReiMelLev95]: E. Reiter, C. Mellish & J. Levine, *Automatic Generation of Technical Documentation*, Applied Artificial Intelligence 9(3):259-287, 1995. Also available through http://www.dai.ed.ac.uk/daidb/staff/personal_pages/chrism/idas.html [Back](#)

[SwaBal82]: W. Swartout and R. Balzer, *The Inevitable Intertwining of Specification and Implementation*, Communications of the ACM, 25(7):438-440, July 1982. [Back](#)

[Thi96]: H. W. Thimbleby, *Creating User Manuals for Use in Collaborative Design*, ACM Conference on Computer-Human Interaction, CHI'96, Vancouver. CHI Conference Companion, M. Tauber, editor, 279-280, 1996. [Back](#)

[ThiLad95]: H. Thimbleby & P. B. Ladkin, *A Proper Explanation When You Need One*, in M. A. R. Kirby, A. J. Dix & J. E. Finlay (eds), *People and Computers X*, Proceedings of the BCS Conference on HCI'95, 107-118, Cambridge University Press, 1995. Also available from <http://www.techfak.uni-bielefeld.de/~ladkin/> [Back](#)

[Zis72]: D. Zissos, *Logic Design Algorithms*, Harwell/Oxford University Press, 1972. [Back](#)

Appendix

HTML Comments

Although HTML is generated, the system does not interpret HTML itself. Thus HTML tags may be used in variable names and in meta-comments. These tags have side-effects on the mark-up of the manual. One use of this is defining variables with names that are HTML section headings, as in some of the examples. This feature was also exploited in this paper to omit some detail not relevant for our exposition.

We have noted that details of the specification were omitted. This was done by enclosing the specification between HTML start and end comment tags, themselves enclosed inside meta-comment braces, as follows:

```
% material omitted... (this is a comment that does appear)
{<!--}
text that is processed, but does not appear in the paper
{-->}
```

Thus the program processes the complete specification, yet allows us to present a briefer paper uncluttered with the detail in the specification needed for completeness with respect to the original.

A production system would be likely to control such freedom of expression closely. Consider that, though masquerading in the description above as a processed line of text ("*text that is processed...*"), in fact it was not processed.

The prototype provides a very coarse control switch. It is possible to generate a manual with either full copying of input (as in this paper) or with no copying. The former is useful for working with a specification when seeing its manual; the latter is useful for using a manual without its logic specification intruding. The switch, then, allows a single file to be used for both development/debugging and use, hence further helping to reduce errors in manuals.

[Back to top.](#)

Web home pages of authors:

[Peter Ladkin: http://www.TechFak.Uni-Bielefeld.DE/techfak/persons/ladkin/](http://www.TechFak.Uni-Bielefeld.DE/techfak/persons/ladkin/)

[Harold Thimbleby: http://www.cs.mdx.ac.uk/harold](http://www.cs.mdx.ac.uk/harold)

Support: [Peter Ladkin](#), 1996-09-03

Glossary

a

Used in:

[Zissos example](#)

antiskid mode active

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

antiskid

antiskid switch is ON

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

a/s

autobrake armed

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

autobrake

b

Used in:

[Zissos example](#)

BSCU failure

The double channel Brake Steering Control Unit (BSCU) controls the anti-skid system via the alternate servo valves.

Used in:

[Overlap example](#)

c

Used in:

[Zissos example](#)

d

Used in:

[Zissos example](#)

green hydraulic pressure available

NB, check yellow hydraulic available too.

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

green hydraulic

nosewheel steering switch is ON

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

n/w strg

parking brake is OFF

Used in:

[Normal braking mode](#)

[Overlap example](#)

Specification:

not park brake

power supply failure

Used in:

[Overlap example](#)

switch on

Mains switch. The switch is at the bottom of the red console, pull it forwards to make it 'on.'

Used in:

[The explanation of sw itself in Mains switch example](#)

Specification:

SW

Document processed 1:02 PM on Sunday, February 09, 1997. (Source last modified 1:01 PM on Sunday, February 09, 1997.)