The Development of a Commercial "Shrink-Wrapped Application" to Safety Integrity Level 2: the DUST-EXPERT™ Story

Tim Clement, Ian Cottam, Peter Froome, and Claire Jones

Adelard, Coborn House, 3 Coborn Road, London, E3 2DA, UK

pkdf@adelard.co.uk

Abstract. We report on some of the development issues of a commercial "shrink-wrapped application"—DUST-EXPERT™—that is of particular interest to the safety and software engineering community. Amongst other things, the following are reported on and discussed: the use of formal methods; advisory systems as safety related systems; safety integrity levels and the general construction of DUST-EXPERT's safety case; statistical testing checked by an "oracle" derived from the formal specification; and our achieved productivity and error density.

1 Background and introduction

The DUST-EXPERT advisory system was developed by Adelard from detailed requirements [1]—including new engineering theory—produced by the UK Health and Safety Executive (HSE).

DUST-EXPERT advises on the safe design and operation of plant that is subject to dust explosions. It is related to, but not directly based on, the prototype of the system developed by Dr Sunil Vadera and colleagues at Salford University. The Salford software is now referred to as the "research version" to distinguish it from Adelard's commercial version. HSE successfully used the research version for some time but realised that, if the technology was to become more widely used, a fully quality assured version was needed. Similarly, it would be necessary to develop the system to some appropriate SIL—Safety Integrity Level—as defined by IEC 61508 [2]. Of course, this standard does not exactly apply to an advisory system such as DUST-EXPERT, as there is no "equipment under control" as such, but we followed the requirements of Part 3 of the standard so far as practicable.

The DUST-EXPERT application provides:

- *general advice* on preventing dust explosions, via the familiar Microsoft Windows™ Hypertext Help system (both general help and also context dependent)
- *user-extensible databases* containing properties of over a thousand types of dust and a few sample properties of construction materials
- *decision trees* that are used in determining which approaches (such as fitting an explosion relief vent, or suppression) are appropriate for the plant under investigation
- *calculation methods* for quantitative analysis of, for example, the size of vents needed to control the effect of explosions and the strength of typical vessels

The system was originally specified to run under DOS/Windows 3.1 but also runs on Microsoft's 32-bit operating systems, and has been extensively validated for the original Windows 95 release.

2 The Safety Integrity Level of an advisory system

To some people the idea of an advisory system being safety related is akin to saying a book (such as [3] in this case) is safety related. Others observe the way in which numbers produced by computer-based systems are taken as gospel and are rarely double-checked, certainly unlike reading a point off a graph in a reference text.

In order to determine the appropriate SIL, we conducted a fault tree analysis from data of reported explosions, the ratio of injuries to fatalities, etc. (see for example [4]). Briefly, we concluded that assuming the failure of explosion relief vents led to 1 death/100 000 workers/year, the vent is at SIL 1, and DUST-EXPERT—as a potential contributing factor—should be developed to SIL 2.

3 The development process

The project team put together for DUST-EXPERT had the structure illustrated in Fig. 1.

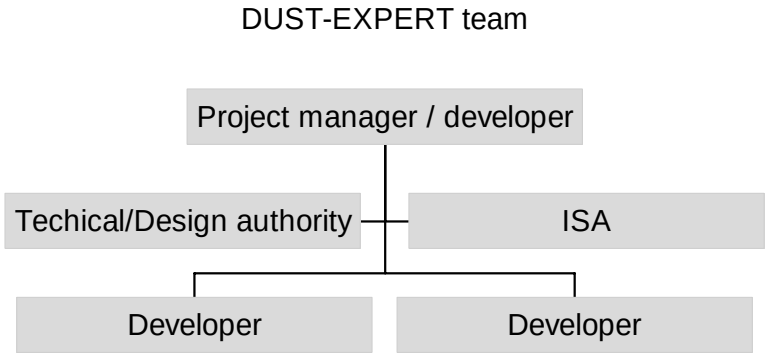

Fig. 1. The team structure

ISA stands for Independent Safety Assessor; he was an Adelard employee but was otherwise independent of the development team. Within the development team there was some sharing of roles: for example, the project manager switched from development to testing as the project progressed, and the technical design authority assisted the two main developers.

The overall shape of the development process was of the type known as the "V with prototyping" model. That is, a traditional approach is taken with diversions, in parallel, to produce input for downstream activities, the main one being to build a dummy system in order to carry out user interface acceptability trials. Prior to Adelard's involvement, HSE had also used the research prototype in order to construct a solid and complete requirement specification. That effort was the main factor in Adelard being able to select a low risk, standard V-model for the development of the commercial, quality assured version.

We chose to build the model of the system in a standard formal specification language (VDM [5]) for the following reasons:

- VDM is a rich and expressive notation for expressing design models.

- Compared with the obvious choice of direct modelling in the main implementation language (Prolog), we gained a strong type system with explicit operation domains, a collection of abstract data types such as sets and maps, and a powerful notation for expressing functions and state changing operations (especially useful for modelling both the database feature and the Windows' interface).

- The IFAD VDM toolbox was available for both type checking of the model and also animating it (where theoretically possible); this latter feature was used both in validating the model and downstream as an oracle against which we could check the results of the pseudo-randomly generated test cases.

- We gained greater confidence in the design than if, say, we had employed some pseudo-code notation without a formal semantics.

- We were able to carry out a software Hazops on the VDM model itself as well as using it for structuring reviews.

- The standard VDM proof obligations allowed us to check the requirements for consistency, although for cost reasons we only carried out these checks informally.

The entire functionality was modelled; the most novel aspect of which is the formal model of the user interface. A separate description of this has recently been published [6].

Once we had an acceptable model of (at least the vast majority of) the requirements, we faced the choice of how to transform this model into code (Prolog) whilst retaining confidence in the validity of the model. We chose to carry out a refinement within VDM to a level where we could define one-to-one transliterations from the VDM features to corresponding Prolog features. This was all done by hand[1] with reviews. Our "VDM implementation level model" has:

- sets and maps implemented as lists

- Windows' states split between explicit and implicit conditions

- the remaining implicit functions given definite algorithms

The transliteration to Prolog was first described in a written procedure and then checked in reviews. This step was probably the most error prone in the development as clerical errors are extremely difficult to spot by review and Prolog compilers are, by definition, forgiving of such mistakes. Still, our final error density was pleasingly low (see Section 4 below).

C++ code was developed by making simple correspondences between, for example, "add element to list" with "add new information line to window".

4 Productivity and other metrics

DUST-EXPERT consists of 15 820 lines[2] written in Prolog and 17 554 lines in C++. The Prolog is relatively complex compared to the C++ code used for the user interface, and yet is shorter. A

---

[1] It is worth noting that in the meantime the IFAD VDM toolbox feature of being able to generate C++ code from (most of) a VDM model has matured, and that that approach, which we rejected because of its immaturity at the time, should be considered for other developments.

[2] All lines of code are measured as non-blank, non-comment lines. The Prolog also follows our VDM style of naming and layout, which produces (perhaps unusually) wide lines of code.

number of factors are at work here: Prolog is at a higher level of abstraction than C++ and much of the C++ code was generated from tools or was simply textually replicated as necessary.

Our productivity was above industry norms with the Prolog being "produced" at approximately half the rate of the C++. This, again, has to do with complexity and the much more rigorous way in which the core Prolog was generated compared with the C++ code; doubtless human and other factors played their parts too.

At the conclusion of statistical and path coverage testing there were 31 known faults, spread across both the Prolog and C++. This amounts to less than 1 per KLOC (thousand lines of code). Of these discovered faults most were minor and only one could be regarded as remotely safety related; through a coding slip from VDM to Prolog it was possible to circumvent the security of the password protected database partitions.

Since release a further small number of faults have come to light, none of which are safety related, and our error density is still just under 1 per KLOC.

There are some 70+ formal reviews recorded in the project database. These are of various kinds: documents, code, progress, etc. They average out at approximately one per week over the development period.

5 User interface prototype

The research version of DUST-EXPERT was a DOS application and had very limited facilities for data entry. This had been identified by HSE as a weakness in the system. As the use of Windows allowed a much richer interface, we proposed to develop a prototype program to demonstrate to the client, and test on potential users, the acceptability of the interface design. Furthermore, we wanted the interface to be as intuitive as possible for users to prevent possibly dangerous misunderstandings.

The user interface prototype had no functionality for actually calculating values, but merely allowed the user to enter values through a calculation screen, navigate a single decision tree and use the Windows' help. We consulted with a user-interface expert on the initial design and then carried out trials with some typical users at our site; other users were sent disks to install the prototype on their own systems and a questionnaire to fill in for feedback. This resulted in several useful changes to the screen layouts, which were then demonstrated to the Project Board.

The DUST-EXPERT user interface design is regarded as very successful. It is believed to have completely changed the attitude towards Windows and Windows-based software of some of the application's champions and early adopters within HSE.

6 Safety Case

The safety claims applicable to DUST-EXPERT are listed in the following table.

Table 1. Safety claims

	Safety claim
1	Functional correctness
2	Accuracy (the results are sufficiently accurate when calculated using finite-precision arithmetic, and numerical instability should be detected)
3	Security (appropriate steps are taken to prevent malicious and accidental changes to methods and data)
4	Modifiability (the chance of maintenance-induced errors is minimised)
5	Fail safety (there is a low probability of unrevealed failures)
6	Usability (the system makes it hard for users to make errors)

A case that would justify Safety Integrity Level 2 was necessary. After some further study, the Adelard project team concluded that we could use several of the techniques usually associated with SIL 3 developments at either manageable or no extra cost.[3] Two examples are the use of formal specification techniques, rather than say structured methods, and a somewhat larger amount of statistical testing than is actually predicted as strictly necessary by the theory for a confidence level of 95%. Provided costs can be kept to acceptable levels, this approach is clearly a good one as it helps to overcome the inevitable level of uncertainty present in the underlying assumptions.

A full Safety Case was produced as a so called "living document" (i.e. there were initial, interim and operational versions) to justify the development, based on:

- a formal specification (the model) of the entire system, including the user interface, in ISO Standard VDM
- identifying and carrying out informal, hand proofs of safety properties on the formal model
- execution of an appropriate subset of the formal model as an "oracle" to aid in verification and validation
- implementation in Prolog where possible (being semantically close to the VDM used in the model) and otherwise in C++ for maximum flexibility in the user interface
- a significant quantity of testing of the integrated system
- a "design for safety" approach where design features were adopted to support the safety arguments

The structure of the safety case follows Def Stan 00-55 [7] recommendations and also Adelard's own (emerging at that time, now published [8]) safety case development methodology.

---

[3] Because, for example, the project team members were all familiar with formal modelling techniques.

Statistical testing is dealt with in some detail (Section 7) below. Here we summarise just a few of the other safety arguments applied to DUST-EXPERT.

6.1 Interval arithmetic

There is a possibility that the underlying hardware—which will vary from user to user—may contain faults and give rise to occasional errors. These may not be detected by the test cases, or may be a feature of the chipset on the PC of a particular user. Furthermore, the calculation methods may be unstable for certain input values. We therefore implemented an "interval arithmetic" virtual machine for DUST-EXPERT. For each method, this carries three values of each variable through the calculations in a systematic way: the actual value, a value slightly below it, and a value slightly above. Generally, the size of the interval is the default precision of the variable, although the user can override this. If the interval at the end of the calculation is greater than 3% of the final value, it is displayed along with the result.

The use of interval arithmetic supports a claim of numerical accuracy of methods, modifiability (since instability in new methods will be detected) and fail safety (many numerical errors should be revealed).

6.2 Design diversity

DUST-EXPERT contains some design diversity that should reveal certain faults.

A hard-copy log is produced of the methods that have been used and their results, including the source of data (by user input, from the databases, or by subsidiary calculation). This is independent of the graphical user interface (GUI), and therefore provides a defence against display errors. Input values are displayed in a particular colour (red by default) when they are first entered on the GUI. When the focus moves (e.g. because the user presses Tab or Enter) the value is read and then all the data values on the current screen are rewritten from the Prolog bindings in a different colour (green by default). This provides a defence against the use of stale data and many types of display error.

This use of design diversity supports a claim of fail safety for the methods and GUI.

6.3 Analytical reasoning

Formal proof is not required by [2] or [7] for SIL 2 software. However, we carried out two sets of hand proofs of safety properties to give confidence that a sample of the subsidiary safety properties hold of the formal specification.

Since the identified safety properties have been shown to hold of the top-level VDM model by testing and by proof, the Prolog must preserve the safety properties provided the translation is correct. For a SIL 4 development, this argument would probably be by means of formal proof. The strength of this argument for lower level systems is a matter for engineering judgement, based on the predefined translation strategy.

The use of analytical reasoning supports a claim of functional correctness of the VDM and Prolog sources, to address residual doubt in the (statistical) testing.

6.4 Desk checking

To enter the calculation methods into DUST-EXPERT, they are first translated to a special-purpose concrete syntax and then loaded into the underlying shell.

The concrete syntax form was checked against the requirements by hand. The internal form was checked by means of a specially written tool that:

- checks calculation method groups for: undeclared variables; variables in the group display lines that do not match the group variables; undeclared methods; and undeclared display line methods

- checks calculation methods for: undeclared variables; variables in the method display lines that do not match the method variables; undeclared options; undeclared display line methods; and local variables that are not defined before they are used

DUST-EXPERT has two databases: a database of dust properties; and a small database of materials properties for use in the Strength of Weak Vessels module.

The materials database was provided on disk to Adelard by HSE and no further checking was carried out. However, in view of its small size, it is reasonable to assume that the HSE would have detected any errors in their reviews of the system.

The following checking of the dust properties database was performed:

- The database was checked by HSE before being provided to Adelard in dBASE III format.

- A specially developed tool was used to print out the internal form of the database in the same format as the source book. This was then checked by hand against the source book. Some 70 errors were found, mostly misregistration of data in columns and single-digit errors. These were marked up on the printed copy.

- These were corrected, following discussions with HSE, in the dBASE III files.

- The tool was then used to print the database, and these were compared with the marked up printed copy. No errors were then detected.

This process also provides assurance of the database module read function. Approximately 1100 records, each containing 16 fields, were checked, giving a level of assurance of better than one error in 17 600 data values.

This use of desk checking supports a claim of accuracy of the calculation methods, databases and warning screens.

6.5 In-Service data recording

Arrangements have been set up to inform Adelard of in-service failures of the software. These arrangements should enable:

- the timely correction of any safety related failures

- the recording of data on the effectiveness of the software engineering process

- long-term assessment of the claims made in the safety case

7 Statistical testing

Probabilistic or statistical testing is a technique for verifying and validating the integrated system, i.e. treating it as a black box. It therefore helps to provide safety assurance in the presence of faults in the various issues of Microsoft Windows and even in lower level software and hardware. Contrariwise, changing one of these components—such as moving to Windows 98 or NT from Windows 95—involves, as a minimum, running all the test cases again in the new environment.

A definition of the statistical approach is:

> "The black box sampling of a system with *sufficient* and *representative* pseudo-randomly generated data to assure correctness to some desired *level of confidence*."

The first issue is how to judge that one's test data is representative. We were in the fortunate situation of having the HSE's experiences with the research prototype, as well as their expert judgements as to which calculation sets (e.g. for vent sizing) are normally used and which only exceptionally. This translated into an order of magnitude more tests for the so-called Basic Calculations Group than for each of the seven more specialised groups, all of which are equally likely and therefore share the same number of generated test cases.

The second issue is how to know if a generated answer is correct. Here we used a subset of the VDM model and executed it under the control of the IFAD VDM toolbox. That is, the test case generator we built (see later) could output a test case in one of two syntaxes: in Visual Basic for Microsoft VisualTest, for testing the application running in its intended environment; and in the VDM command language of the IFAD toolbox, for consulting the oracle. Ideally the oracle and the application should be completely independent to avoid common-mode failures. For the DUST-EXPERT development this was not the case as the application was (partly) the systematic transformation of the VDM model. However, to mitigate potential common-mode failures due to the hardware or software platforms, we made the environments that the oracle and the application ran under radically different (see Table 2).

Table 2. Oracle versus application environments

Oracle	***Application***
486 processor	Pentium processor
Linux operating system	Windows 95
Linux C runtime libraries (e.g. for crucial floating point calculations)	Microsoft C and LPA Prolog libraries (e.g. for crucial floating point calculations)

The question remains, from our definition of this approach, how many test cases are sufficient? A very informal summary of the theory, which can be found in, e.g., IEC 65108, is:

- we know from our safety analysis that DUST-EXPERT must be responsible for no more errors than 1 in (a little under) 1000 demands

- 1000 successful test cases would then tell us that the application was equally likely to be right as wrong (50% confidence)

- applying the formula that the statisticians have given us [9] tells us that 3500 test cases executed successfully would mean that we could have 95% confidence that the application was generating the correct numbers

- to compensate for the risk that out underlying assumptions were less than perfect[4] we increased this figure slightly to 3800

We pseudo-randomly generated 2400 tests for the Basic Calculation Group and 200 each for the remaining seven specialised groups, making our total of 3800. In addition, over 1000 directed tests were executed, some from HSE's supplied acceptance suite and some generated independently by Adelard. In the case of these tests the oracle varied, from the VDM model, to prototype models in C (for an iterative ducting calculation), through the research prototype (where the requirements had not changed significantly), to the use of a conventional desk calculator. The vast majority of these directed tests are also coded in Visual Basic for VisualTest and thus can be rerun with little inconvenience. Statistically speaking they are biased (we chose them all for some reason), but their successful execution certainly added to the development team's confidence in the system's correctness. They also covered the entire system rather than just the critical calculation parts.

Pleasingly, all 3800 statistical tests gave the result the oracle predicted. We did uncover one fault: if approximately 160 calculation windows were opened in one session the application crashed. The problem was diagnosed as an overflow of an integral subrange variable in the user interface code. There were also two surprise results where the application produced negative answers. This was diagnosed as predictable—our oracle told us to expect them—from the poor choice of constraints on the calculation methods concerned.

It is worth noting that the fabled UNIX toolset still has a place in our Windows dominated world. The generators and checkers needed were constructed in a few days as UNIX-based Shell scripts, and were small enough to have their validity checked by inspection.

8 Conclusions

We produced a complex advisory system, on top of a reusable shell supporting constraint-based programming, and demonstrated its conformance to Safety Integrity Level 2. Several aspects of the safety case are at the high end of the SIL 2 band; this both strengthens our argument at this integrity level (because some assumptions are based on engineering judgement) and means that we could rack up the safety case to SIL 3, should it ever be necessary, at only a fraction of the overall development cost.

DUST-EXPERT is not claimed to be error free, but it is demonstrably safe with an enviably low fault density.

References

1. ITT for Development of DUST-EXPERT, ITT reference: version 5, issued 13-Feb-95, HSE.

---

[4] An example is the assumption that any value from the possible range of input values (as defined by HSE) was equally likely. In some cases we knew that HSE had made the input ranges infeasibly wide.

2. International Electrotechnical Commission, Functional safety of electrical/electronic/ programmable electronic safety-related systems. IEC 61508.

3. Lunn G., Guide to Dust Explosion Prevention and protection, Part 1 – Venting, 2nd ed., Institute of Chemical Engineers, 1992.

4. Schoeff, R.W., News Release & Summary for 1994. Dust Explosion Statistics—USA. Europex Newsletter, May 1995.

5. Information technology—Programming languages, their environments and system software interfaces—Vienna Development Method—Specification Language—Part 1: Base language, ISO/IEC 13817-1, 1996.

6. Clement, T., The formal development of a Windows interface, Proceedings of the Northern Formal Methods Workshop, Ilkey, 1998.

7. The Procurement of Safety Critical Software in Defence Equipment, Def Stan 00-55 (Parts 1 and 2) / Issue 2.

8. Adelard, ASACD: the Adelard Safety Case Development Manual, Adelard, ISBN 0-95337710-5, 1998.

9. Littlewood B., Strigini, L., Validation of ultra-high dependability for software-based systems, Communications of the ACM, Vol. 36, No. 11, pp69–80, November 1993.