Formal and Informal Specifications of a Secure System Component: Final Results in a Comparative Study

T. M. Brookes[1], J. S. Fitzgerald[2], P. G. Larsen[3]

[1] British Aerospace (Systems and Equipment) Ltd., Plymouth, UK
[2] Centre for Software Reliability, University of Newcastle upon Tyne, NE1 7RU, UK
[3] IFAD (The Institute of Applied Computer Science), Odense, Denmark

Abstract. This paper presents the findings from the later phases of a study of the effects of introducing formal specification to the commercial-scale development of a small security-critical system component. The objectives and form of the study are briefly reviewed. Observations have been made of the effort profile across the project, compliance of the developed system with customer requirements and software characteristics. The results of these observations are presented. Conclusions and areas of further work are discussed.

1 Introduction

An important topic in any branch of practical engineering is how to improve the development process with the aims of decreasing time to market, reducing the overall development costs and ensuring that the product meets the customer's requirements. Numerous techniques, for example those of structured and object-oriented design, have been introduced in recent years with claims that they allow these objectives to be reached. Formal specification, the subject of this paper, can be viewed as a complementary technique which allows system requirements to be captured and expressed in a rigorous fashion.

The use of formal specification is currently mandated for high assurance security- and safety-critical systems [ITS91,MoD91a,MoD91b]. In these cases, the perceived expense of using formal specification can be justified as there is no other method of performing the project to the satisfaction of the regulatory authorities. But what are the true costs of using formal specification in such a project? The study described in [FBGL94] and [Fit95] sought to provide evidence on the costs and benefits of a modest degree of formal specification by comparing separate developments of the same system, one development using conventional practice, the other using the same techniques plus specification in VDM-SL supported by the IFAD Toolbox. A previous paper [FBGL94] described the differences in the two development paths' early phase of system design. This paper presents findings from the later software design, implementation and testing phases.

Section 2 briefly reviews the development process and the comparison already made of the early phases. Section 3 presents the new results from the later stages and compares the distribution of effort across the whole process. the overall observations of the project are discussed in Section 4. Section 5 describes the future direction of formal specification work in BASE and Section 6 briefly discusses the value of this kind of small comparative study.

2 Project Overview

2.1 Background

One important area of BASE business is the development of secure message handling systems. Such systems are typically developed to high levels of assurance assessed by a third party. Increasingly, the criteria for achievement of high assurance levels require some use of formal techniques. The study reported here was motivated by a desire to observe the consequences of introducing a modest degree of formal specification into a BASE development process in order to meet the criteria for high assurance levels.

The study was intended to assess:

1. the effectiveness of formal specification in terms of the additional development costs versus any benefits gained by reducing ambiguities, misunderstandings and rework, with a view to its possible introduction to all or part of the life cycle for other projects;
2. the problems and difficulties involved in the use of formal specification, so that future projects may learn from this experience; and
3. the training requirement needed to introduce formal specification into the design process.

It should be stressed that it was not an aim of this study to show the absolute worth of formal methods. In particular, it was not an attempt to prove that some costs or benefits stem directly from the formality of a notation.

2.2 The Trusted Gateway Development

The development of a *trusted gateway* was used as the baseline project for the comparison. The gateway considered is a simple device which is located in the communications path between systems at different security levels. Its purpose is to determine the classification of the messages which pass through it and to ensure that they only pass to a destination at the correct security level. It was an excellent system to consider as its behaviour could be well defined, was sufficiently simple as to be tractable with the effort available for the project, yet was sufficiently complex as to be non-trivial.

The trusted gateway was developed by two separate design teams who did not communicate, although they were aware of each other's existence. The first team used a conventional development methodology, Ward and Mellor [WM86] supported by the Teamwork[1] Computer Aided System/Software Engineering

[1] Teamwork is a Registered Trademark of Cadre Technology Inc.

tool set. The other team followed a similar design process, but used formal specification in VDM-SL [ABH$^+$95] to support this design methodology. The formal specification was developed and tested using the VDM Toolbox from IFAD [ELL94]. The utilisation of tools and the details of the development process followed are described in [FLBG95].

The development was initiated by producing a customer requirement that was submitted to each of the design teams. In the first (system design) phase there was a divergence in the design as each of the groups interpreted the original specification. The two design processes were carefully logged: the results of their comparison are described in more detail in [FBGL94].

The second (software design) phase saw each system design and test plan developed to levels of greater detail. In the implementation and testing phases each version of the system was coded and tested using the test plan devised by its own design team. The test procedures from the other design team were then applied to compare the test coverage achieved by the two design approaches. The software produced was subjected to a final acceptance test by the customer. Areas where the performance was deficient were highlighted and examination of project records was used to pinpoint the decisions in the design process responsible for the introduction of that deficiency.

Training in the use of formal specification and the Toolbox was provided for the engineers involved in the design and monitoring activities. The Toolbox provided a means for preparing VDM specifications in the correct format, performed static checking, and could animate the VDM-SL specifications produced during the course of the design. IFAD and the Centre for Software Reliability at the University of Newcastle also participated in the comparative reviews of the design and progress.

3 Observations

This section presents the results of the observations made in the later design phases and across the project as a whole. The following points were assessed and used to compare the two design processes:

1. effort required to perform the design task (Section 3.1);
2. compliance of the system with the original specification(Section 3.2); and
3. performance of the system and tests in terms of code size, speed, and complexity (Section3.3).

A number of other observations on specification style, training, tool support and specification language syntax were also recorded (Sections 3.4 – 3.7).

3.1 Design effort

The tables below show how the effort expended on the project was distributed. Figure 1 shows the normalised number of hours spent in each phase assuming that the project was 100 hours in duration. The effort was provisionally allocated on a 40:40:20 hour split between systems design, software design and implementation. The column headed ‘Allocated Time’ indicates the time allocated to each phase at the start of the project. The columns headed ‘Conventional’ and ‘Formal methods’ show the time expended in the individual paths.

Figure 2 indicates the distribution of time among the phases in each path (note that the figures have been rounded to two places.)

Phase	Allocated Time	Conventional Method	Formal Method	*Excess F over C*
System Design	40	30	35	+ 17%
Software Design	40	40	33	- 17%
Implementation	20	17	13	- 24%
Totals	100	87	81	- 7%

Fig. 1. Normalised Number of Person Hours Spent in each phase

Phase	Conventional Method	Formal Method	*Ratio*
	% Project	% Project	*(C/F)*
System Design	34%	43%	0.77
Software Design	46%	41%	1.15
Implementation	20%	16%	1.25
Totals	100%	100%	

Fig. 2. Distribution of Project Time within each path

The system design phase required roughly 17% more effort in the formal path than for the conventional path when considered in terms of the overall project (which includes design reviews, etc.). The engineers in this phase had equal skills and experience. When the time spent in this phase is examined as a percentage of the project, there is a larger difference. In the formal path 43% of the time is spent in the system design phase as compared to only 34% on the conventional path; a more significant difference. Neither engineer was limited by available resources as both underspent the total project budget, by 25% in the conventional path, and 12.5% in the formal path.

In the software design phase the engineers were of different skill and experience, the engineer in the formal path having more experience. Although the formal path required less effort to complete, it was felt that this result was biased by the difference in the engineers. After comparing the work of the two engineers, and taking into account their different experience levels, there was not judged to be a significant additional effort which would be incurred if formal specification were used in the software design phase under normal conditions.

In the implementation phase, the engineers were again of similar age and experience. Both paths took the same amount of effort to complete the first version of the system. However, the conventional implementation required re-work to correct a problem discovered when the system was tested with the test suite developed on the formal path. This increased added about 15% to the cost of the phase. As a percentage of the overall project, the formal software implementation took 13% of the effort as compared to 17% in the conventional path.

Comparing the effort for the entire programme, the use of formal specification did not incur an overhead, in fact the overall effort required was slightly less. The difference is not felt to be significant. The effort distribution in the formal path exhibited higher costs in the early parts of the programme where system requirements are being analysed and understood, but that the additional effort is recovered in the later stages of the programme. This change in the effort profile is also typical of the introduction of structured design methods where system understanding is promoted before development.

3.2 Compliance with Customer Requirements

In the system design phase, the formal specification path detected a special condition, implicit in the requirement, which was not identified in the conventional path until the test suite developed for the formal path was run on the implemented system. In normal business, the error would not have been detected by testing, but would possibly have been detected by the customer[2]. Re-work was required in the conventional development process to correct this, adding the 15% extra effort to the implementation phase already discussed. No effort was used to correct the supporting design documentation, so this represents the very minimum additional cost. Had this system been developed for external evaluation, considerably greater time and effort would have had to be expended to correct and re-evaluate the design documentation.

3.3 Code Metrics

Complexity The routines which determine message security classification were compared to evaluate the relative complexity of the code developed on the two paths. This routine was chosen because it forms the kernel of the system, and in particular implements the security enforcing function.

The McCabe Complexity of the code was found to be 74 for this routine on the conventional path and 10 on the formal path. If the formal specification itself were treated as an implementation, its complexity would be estimated at 4.

[2] In BASE, this would be referred to as a *design error* because it led to the development of a product which did not meet the customer's expectations.

These figures would suggest that the formal path produced much simpler code in the main function than the conventional path. An investigation was conducted to see if the change to the code on the conventional path caused by the failure to pass the formal test suite had affected the code complexity. This showed that there had been a significant increase in the complexity (from roughly 10 to the 74 measured) when the code for the main function was re-written to correct the deficiency. The conventional development did not in itself produce complicated code, but problems may have been introduced when the routine was redesigned to correct problems discovered late in the testing process.

Size The number of lines of code in a routine is not a particularly helpful metric, although in this case, as both routines are trying to implement the same function, they are of some use. The code on the formal path was less than one fifth of the size of the code on the conventional path. The ratio of lines of comment to code, which is an indication of the maintainability of the final system as the code is probably better documented internally, was much larger on the formal path, albeit the number of lines of comment was smaller. The differences identified here cannot be attributed solely to the use of formal specification, and are believed to reflect the experience and ability of the software engineer.

Speed The speed of operation was tested by passing a large block of messages through the system. To minimise any machine-related errors, the software was installed on the same machine for each test, and all other software running was disabled. The results obtained are shown in Figure 3.

	Formal	Conventional	
Phase	Implementation	Implementation	*Ratio*
initialisation time (seconds)	70	17	4
processing rate (char per sec)	250	18	13.9

Fig. 3. Code Speed for the Different Implementations

The formal implementation spends roughly four times as long checking the system data, the classification definitions and start and end of message definitions, as the conventional implementation. However, when the system is processing messages it is almost fourteen times faster. Since the trusted gateway is designed to be set up once and then left to operate for a long period of time, the relative speed of initialisation does not matter in assessing system performance. The system developed in the formal path would thus be considered to be much faster than the conventional implementation. Note that speed was not given to either team as a design objective.

3.4 Specification Style: developing an implementation for a memory purging operation

To give an indication of the specification styles used by the BASE engineers, this section illustrates the evolution of the formal specification during the software development phase by considering the simple operation used to purge the trusted gateway's system memory. At the start of the system design phase, `Purge` was defined on the very abstract system design model, with no indication as to how the operation was to be implemented. The message is replaced by an empty sequence which would satisfy the requirements of the security policy model that there should be no remnant of the message left in the system after processing. The definition is shown below as recorded in the design documentation [3]:

```
This represents the clearing of the Message Data areas, which
is not strictly needed for the execution of the formal
specification in the IFAD Toolbox, but is included for clarity
of the overall formal specification.
\begin{vdm_al}

  operations


        Purge : () ==> ()

        Purge()==
                (
                ValidMessage := ""
                );

\end{vdm_al}
```

This specification was refined (in the informal sense) during the software design phase, where a specific algorithm recommended in the original customer specification is described. The algorithm writes characters over the location in the memory occupied by the message. The new definition is shown below, together with the designer's comments:

```
This operation performs a purge on the first 'length' characters
of 'buffer'.  Each element of the buffer has the sequence FF
then 0 written to it eight  times followed by 246 once. This
sequence is repeated 4 times.

Note that the testing of this function in vdmde (the IFAD Toolbox)
is very time consuming due to the loops. It can be disabled if
```

[3] The LaTeX `vdm_al` commands are part of the interface of the IFAD Toolbox, and separate the VDM-SL formulae from the explanatory text.

```
required if the term length is replaced by 0 in the first for
loop. Also note that the characters written to the buffer are
not those required due to keyboard limitations.

Input Parameters    length
                    - The number of characters to be purged.
Returns             None
States Affected     buffer
                    - buffer has a wipe sequence written to it

\begin{vdm_al}

  Purge :  nat ==> ()
  Purge (length)==
  (
        for i = 1 to length do
        (
                for count1 = 1 to 4 do
                (
                        for count = 1 to 8 do
                        (
                                buffer := buffer ++ {i |-> ’A’};
                                buffer := buffer ++ {i |-> ’O’}
                        );
                        buffer := buffer ++ {i |->’B’}
                );
                buffer := buffer ++ {i |-> ’O’}
        ) -- i < length
  );

\end{vdm_al}
```

When this function has to be implemented in a high level language, the second version can be easily translated, whilst the first possesses insufficient detail to be used. The implementation in C is shown below.

```
/************************************************************/
/* Function            : Purge                              */
/* This function will purge the data between the two pointers */
/* passed in                                                */
/*                                                          */
/* Parameters   : (global) plus                             */
/*                LPSTR lpszCommence : point from which to  */
/*                                     commence purge       */
/*                LPSTR lpszFinish : stop purge             */
/*                                                          */
/* Return                : void                                 */
/*                                                              */
/**************************************************************/
void FAR PASCAL Purge(LPSTR lpszCommence, LPSTR lpszFinish)
{
    int iNumberChars, iLoop, iCount1, iCount;

    iNumberChars = lpszFinish - lpszCommence;

        for(iLoop = 1; iLoop <= iNumberChars; iLoop++)
        {
                 for(iCount1 = 1; iCount1 <= 4; iCount1++)
                 {
                        for(iCount = 1; iCount <= 8; iCount++)
                    {
                        _fstrcpy(lpszCommence, "A");
                        _fstrcpy(lpszCommence, "0");
                the}
                _fstrcpy(lpszCommence, "B");
             }
             _fstrcpy(lpszCommence, "0");

        ++lpszCommence;
        }

    lpszCommence = lpszCommence - iNumberChars;

    /* clear */
    _fmemset(lpszCommence, 0, iNumberChars);

        //SendMessage(hMMI, STATUSMSG, IDS_PURGE, 0L);

}
```

The core of the implemented routine and the final VDM specifications are very similar. The code which has been added is primarily concerned with the implementation in the target hardware and operating system. A formal proof of correctness against the specification is hindered by the absence of an appropriate proof theory for the implementation language. However, informal examination of these structures would suggest that the implementation respects the formal specification.

3.5 Training

One objective of the study was to assess the training requirement needed to introduce formal specification into the design process. A basic one week course in the use of formal specification and the IFAD Toolbox was given, and was sufficient for imparting general specification skills. Overall, this is an encouraging result in that it suggests that the training overhead associated with the introduction of formal specification into a design process is typical of introducing a new technology into a company. The engineers, who had no background in formal methods, found it straightforward to apply. Consultancy from expert users was found to be essential when the engineers were starting to apply formal specification. This could be supplied either by experienced practitioners within the company, or by an outside consultant.

3.6 Tool Support

The formal path used the VDM Specification Language supported by a computer-based tool, the IFAD VDM-SL Toolbox, which allowed the specification to be type checked and animated. These facilities allowed engineers who were unfamiliar with the language to quickly learn how to write well-formed formal specifications.

It is important that the introduction of formal design into the development process be supported by computer based tool. Tool support was regarded as a practical necessity for the formal path to be followed at all. For a perspective on the practical use of the Toolbox, see [Muk95].

The use of animation allowed the system design to be examined early in the development process and let the engineers consider if the specified behaviour met expectations. The test cases used at this stage were carried forward through the programme and were eventually applied to the final system. This technique for demonstrating conformance to requirements could be applied by inexperienced engineers who have good domain knowledge but relatively little expertise in formal specification. For this reason it is believed to be a more attractive technique than formal proof.

3.7 Presentation of Specifications

VDM-SL was used in the formal path because of the availability of expertise and tool support for this language. In principle, any other model-oriented formal specification language susceptible to the same degree of type-checking and animation with sufficiently good tool support could have been used.

The Standard version of VDM-SL has an alternative ASCII syntax which does not use the mathematical symbols commonly found in formal specifications. The engineers expressed a preference for the the ASCII syntax, even though it is more verbose. This is probably due to their familiarity with programming languages and the IFAD Toolbox's use of the ASCII version of the language. In discussing formal specifications with colleagues unfamiliar with formal notation, the ASCII syntax was also preferred as a less intimidating alternative to the mathematical syntax.

4 Discussion

As indicated in the introduction, this study did not aim to show the absolute worth of formal specification. However, it does add to the body of evidence in the public domain on the costs and benefits of using formal techniques. In this section, we consider the main points which BASE have felt to be of relevance to them from the study. Recall from Section 2.1 that the study was intended to assess the effectiveness of formal techniques (costs and benefits), the problems and difficulties of adding formal techniques to the development process and the training requirement associated with formal specification. Each of these areas is considered in turn.

Effectiveness

The study indicated that adding formal specification can bring real benefits to a product development without incurring a prohibitive cost overhead when the costs are considered over the entire life cycle, providing it is applied where appropriate, and is applied in conjunction with other design methodologies which cater for large systems. The opportunity afforded for early error detection and resolution are expected to contribute to a reduction in time to market.

The development process clearly illustrated how the detection of errors late in the design process can lead to expense in making corrections and result in code which is poorly supported by the design documentation and is difficult to maintain. This is often stated in the literature as a driver to good systems design as a cost saver. Tests developed from a poor understanding of the system will not necessarily detect design flaws. These may be found for the first time by the customer.

It was felt that the major benefits of adding formal specification are seen in the better definition of the data used, and the identification of exception conditions. Defining data (in the system state and interfaces) in an implementation-independent manner allows this information to be passed through the various design stages without transformation, albeit with the addition of detail, until the system is implemented. The rigour required to define the data using a formal notation also forces the designer to question the customer (or to make and justify assumptions about the data) very early in the design process. It was noted that during the formal design of the trusted gateway considerably more time was spend in elucidating the data types which the system used and writing them down unambiguously (see [FBGL94]), possibly as a consequence of the use of a model-oriented specification language. The resulting formal definition can be included in the data dictionary entries used in the CASE tools.

Defining the required functionality is a more demanding activity because of the skill required to understand which parts of a design should be expressed formally and to be able to abstract the required functionality from requirements which are rarely clear, consistent and unambiguous.

Problems introducing formal specification

Formal specification did not make as valuable a contribution to the software design phase as was expected after the system design phase. This may have been because the example chosen was simple enough for the system design to be easily expressed as code, or it may be inherent in the method. Producing software from a formal specification is not simple even if the specification is executable: constructs used in the formal specification may not translate well into the target language leading to either an inefficient implementation, or a substantial amount of re-work to optimise the code design. One would expect that increasing local experience in designing efficient code from formal specifications would make this task easier, e.g. as well known implementation strategies are recorded for future use.

Training

It was originally felt that the same training would be appropriate for both systems and software engineers. However, after the experience of this project, we feel that short one- or two-day supplementary courses are needed to impart specialist skills. The systems engineers require skills in abstraction, functional specification and data specification. The software engineers require additional skills in refinement.

Training in the use of formal specification is essential for an engineer who is going to implement a formal specification. The ability to read a specification is not sufficient: the training must encompass the concepts behind the use of formal specification. In particular it should be stressed in training that the formal specification of an algorithm does not preclude many implementations from being used providing they are compatible with the formal specification. It is in this area that the skills of a software engineer will still be employed.

5 Outcome of the study

BASE has already applied formal specification to a sub-system in a larger secure system which required a high level of evaluation. The external authority which evaluates designs has indicated that the trusted gateway design using VDM-SL is could be evaluated to the new high level sought.

Guidelines are being developed recommending the use of formal specification techniques in the areas where this study has suggested they are beneficial, principally in the system design phase. However, it was also noted that formal specification should not be employed universally: each function should be considered on its own merits to assess whether or not the use of formal specification, with the attendant overhead, was justified.

Within the company, presentations have been given to the engineering community to disseminate the results of the trusted gateway study. With the existence of a small core of personnel who have some expertise with the use of formal specification, the aim is to spread the use into areas of projects where the use is beneficial. Formal specification will, at least initially, be applied where appropriate by reason of criticality, complexity of data or functionality.

6 Concluding Remarks

We have described the findings of the later stages of a comparative study in the development of a system with and without the use of formal techniques. One might conclude by asking how useful such a study is in increasing the exploitation of mathematically rigorous techniques in the development of software systems.

First, the object of the study was not to provide a "success story" for formalists. The trusted gateway project was aimed at one specific development process in BASE and geared to one area of application where the motivation to experiment with formal techniques already existed through the desire to reach specific levels of assurance. As an outcome of the trusted gateway development, it has become possible to develop some systems to a higher level of assurance than hitherto in BASE. Work is actively being pursued to widen the use of mathematically-based techniques where there is felt to be a genuine benefit in the company.

Although the trusted gateway development was a much more closely defined and monitored process than is usual in commercial software development, the sample size was one and a host of variable (human) factors were unaccounted for in the comparative analysis. To conduct a statistically significant "clinical trial" of formal methods would be beyond the means of most companies or even industrial sectors. We would suggest that adoption of formal methods is more likely to proceed via smaller-scale studies specific to the needs of particular groups, companies or sectors. The results of such studies will always be qualified, and rarely be generally applicable, but they will help in building a larger body of evidence which may be useful to those considering formal techniques for the first time, and certainly more useful than the unsupported claims which abound in some other areas of software engineering.

Acknowledgements

The authors would like to thank British Aerospace (Systems and Equipment) Limited for giving permission to publish many of the details of this work, as well as the engineers who participated in the study. We are grateful for the support of the European Commission (ESSI Grant 10670). JSF is grateful to the United Kingdom Engineering and Physical Sciences Research Council for support under an EPSRC Research Fellowship. All the authors are grateful to the anonymous referees for FME'96 for their helpful comments on an earlier draft of the paper.

References

[ABH+95] D.J. Andrews, H. Bruun, B.S. Hansen, P.G. Larsen, N. Plat, et al. *Information Technology — Programming Languages, their environments and system* *software interfaces — Vienna Development Method-Specification Language Part 1: Base language.* ISO, 1995.

[ELL94] René Elmstrøm, Peter Gorm Larsen, and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.

[FBGL94] J. S. Fitzgerald, T. M. Brookes, M. A. Green, and P. G. Larsen. Formal and informal specifications of a secure system component: first results in a comparative study. In M. Naftalin, B. T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 35–44. Springer-Verlag, 1994.

[Fit95] J. S. Fitzgerald. The ConForm Project Home Page. World-wide web at URL:
http://www.cs.ncl.ac.uk/research/csr/projects/ConForm.html, 1995.

[FLBG95] J.S. Fitzgerald, P.G. Larsen, T.M. Brookes, and M.A. Green. *Applications of Formal Methods*, chapter 14 Developing a Security-critical System using Formal and Convential Methods. Prentice-Hall International Series in Computer Science, 1995.

[ITS91] Office for Official Publications of the European Community. *Information Technology Security Evaluation Criteria*, June 1991.

[MoD91a] United Kingdom Ministry of Defence, Directorate of Standardisation. *Safety Management Requirements for Defence Systems Containing Programmable Electronics*, 1991.

[MoD91b] United Kingdom Ministry of Defence,Directorate of Standardisation. *Procurement of safety-critical software*, 1991.

[Muk95] Paul Mukherjee. Computer-aided validation of formal specifications. *Software Engineering Journal*, pages 133–140, July 1995.

[WM86] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems*, volume 1-3. Yourdon Press, New York, 1985-1986.