The Practice of Formal Methods in Safety Critical Systems*

Shaoying Liu, Victoria Stavridou, Bruno Dutertre
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey
TW20 0EX, U.K.

Abstract

By describing several industrial-scale applications of formal methods, this paper intends to demonstrate that formal methods for software development and safety analysis are increasingly adopted in the safety critical systems sector. The benefits and limitations of using formal methods are described, and the problems of developing software for safety critical systems are analysed.

Keywords: formal methods, functional requirements analysis, safety analysis, safety critical systems.

1 Introduction

A significant problem of developing software for safety critical systems is how to guarantee that the functional behaviour of a developed software system will satisfy the corresponding functional requirements and will not violate the safety requirements for the associated overall system. In order to solve this problem, it is important to analyse thoroughly the safety properties of the overall system, to achieve accurate software functional requirements and to verify properly the implementation of the software.

Formal methods are perceived by the community as a way of increasing confidence in software for safety critical systems. They are mathematically-based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design and analyse computer systems [Abrial *et al* 1989],[Bowen, Stavridou 1993a]. Formal methods have attracted the attention of many authorities. An example of this is the interim standard 00-55 on the procurement of safety critical software in defence equipment, published by the UK Minstry of Defence [MoD 1991]. 00-55 mandates the production of safety critical module specifications in a formal notation. Such specifications must be analysed to establish their consistency and completeness in respect of all potentially hazardous data and control flow domains. A further fundamental requirement is that all safety critical software must be subject to validation and verification to establish that it complies with its formal specifications.

Interestingly, and despite the lack of any documented factual evidence as to their efficacy, formal methods are clearly considered desirable, particularly in relation to safety critical systems. While we share the view that such methods can be a very important analytical tool [Bowen, Stavridou, 1993b], we also believe that hard factual evidence relating to the impact of formal methods on the dependability of safety critical systems is highly desirable. To this end, we are currently participating in an effort to quantify and understand the dependability implications of formal methods [Littlewood 1993].

This paper describes some applications of popular formal methods in industry, especially for requirements analysis and specification. Since there is a close connection between the functional requirements of software and the safety of an overall system, safety analysis techniques are also reviewed.

*Work is partially supported by SafeIT (SERC/DTI) grants ((IED4/1/90/3)) and ((IED4/1/93/4))

Evidence shows that effective ways of using formal methods for safety critical systems are still an open problem [Austin, Parkin 1993][1]. This is because during the process of software development for safety critical systems, not only the functional behaviour of software has to be considered carefully, but we must also demonstrate that the developed software satisfies the overall safety requirements. This paper analyses this problem and presents a possible way forward.

The remainder of this paper is organised as follows. Section 2 describes the applications of formal methods in functional requirements analysis and specification construction. Section 3 analyses the benefits and limitations of formal methods. Section 4 presents several popular safety requirements analysis techniques. Section 5 addresses the problems of existing software development methods. Finally, we give our conclusions and outline further research.

2 Applications of Formal Methods in Requirements Analysis and Specifications

Requirements analysis is a key activity in the process of software development for achieving satisfactory systems. The quality of requirements analysis determines the quality of requirements specifications which directly affects the quality of the developed system.

Evidence shows that the use of formal methods increases confidence in software systems, especially for safety-critical systems [Austin, Parkin 1993]. One of the important reasons for this is that formal methods can assist people to do requirements analysis thoroughly and to express precise requirements specifications.

Many formal notations and methods have been used so far in industry for the purpose of requirements analysis and functional specification of safety critical systems. They include VDM, the B-Method, RAISE and HOL. In this section we present several examples of the industrial application of these formal methods. The lessons learned in practice are described. For the sake of readability, we divide these cases into groups which have had the same (or similar) formal method applied to them.

2.1 VDM

VDM (Vienna Development Method) is a denotational model-based approach [Bjϕrner, Jones 1982],[Jones 1990]. In VDM, specifications are constructed around abstract states which are models defined in terms of data objects such as sets, maps and lists. Operations on these state-like objects are specified by pre- and post-conditions. The precondition is a predicate over the initial state of the operation and can be used to limit the cases in which the operation has to be applicable. The post-condition is another predicate that specifies the "input-output" relationship between the initial and final states of the operation.

Some of the industrial-scale projects which used VDM for requirements analysis and specifications are as follows:

(1) Ammunition Control System

The interim United Kingdom Minstry of Defence (MOD) Defence Standard 00-55 [MOD 1991] on the procurement of safety critical software is expected to force the pace of change in the civil as well as the defence sector. It advocates the use of formal techniques in various stages of system development.

An existing ammunition control system (ACS) is to be replaced and the appropriate MOD authority intends to produce a formal specification of their own regulations which would be incorporated in the Operational Requirements document for the planned ACS replacement in accordance with the spirit of 00-55.

The formal methods group at RHUL (Royal Holloway - University of London) examined the ACS and the explosives regulations. They then con-

[1] This is a report of a literature survey and a survey of industry to discover the current situation of formal methods exercises in both academia and industry. The surveys were conducted through questionnaires.

structed a formal model of the system and proceeded to formalise the associated safety requirements as well as to prove some properties of the specification using VDM [Mukherjee, Stavridou 1993b]. The model reflects two main aspects of the specification which are the storage of explosives in a particular magazine (a magazine is a building in which explosives are stored) and the construction of new buildings. The safety requirements include storing objects, adding magazines and composing operations.

Furthermore, the VDM specification is translated into the language OBJ3 [Goguen, Winkler 1988] for the purpose of rapidly constructing a prototype explosive storage model in order to allow animation of the specification. It is believed that a prototype at such an early stage of the software development process could help demonstrate the quality of the formal specification and find potential problems in the final implementation.

The application described above has suggested several practical benefits and difficulties in using formal methods. First, the use of formal methods can help clarify the documented informal requirements (e.g. UN Orange book regulations in the above case) and provide an unambiguous safety statement for the replacement ACS system. However, it has been demonstrated that wholesale adoption of formalization is not necessary for improving system quality. Rather, it is more appropriate to use formal methods for complex procedures and safety-critical parts of the system, and as a communication medium between customers and contractors. Second, formal specifications can be used as an aid to site planning and prediction of further system development. Third, formal methods can be used effectively not only in their traditional role of improving product quality or reducing product maintenance costs, but they can also facilitate modification of system documents. Fourth, it is shown that as the scale of the applications increases, maintaining the advantage gained from short, abstract specifications becomes progressively more difficult. For example, specifications can become as complex and voluminous as programs. It is believed that handling this complexity represents a challenge that needs to be ultimately addressed through further research. Finally, there is no hard evidence to guarantee the dependability of the system because of using formal methods, but as with all formal specification and verification exercises, the most that can be safely asserted is that certain kinds of logical errors have been made far less likely within the credibility bounds of the human specifier and the human/machine prover.

(2) Medical Instruments Control System

The cardiology business unit at Hewlett-Packard's McMinnville Division is responsible for the production of medical instruments, some of which have life-critical functionality and require high reliability [Cyrus *et al* 1991]. These instruments are used in a high-tension environment by medical personnel who are not necessarily computer literate and do not use the instruments on a daily basis. A project team from the cardiology business unit is responsible for developing one of these life-critical instruments.

Since the correctness of previous generations of the product (software) is not formally verified and the code in these earlier instruments which was written in assembly language is hard to understand and not reusable, the project team decided to adopt formal methods to improve the quality of product.

In collaboration with the applied methods group from HP's Bristol laboratories, the project team produced a complete formal specification of the safety-critical part of the software used in a medical instrument. The specification is derived from the product's external specification (ES) - a natural language description of the product requirements, and is written in the formal language HP-SL, which is a VDM-like language [Rush *et al* 1990], in conjunction with data flow diagrams. The formal specification provided a process model of the system, specifying the relationship of inputs and outputs over time. This model uses the HP-SL history specification, which associates data values with time, thereby capturing timing constraints and specifying a data object's value for all time. The specification is illustrated by a variant of data flow diagrams in which the data flows between processes corre-

spond to HP-SL histories, and the dashed lines are used to indicate optional data which are only present in some product family members.

The formal methods exercise in this application has led to the discovery of several major benefits. First, a well planned project schedule has proved to be necessary to use formal methods successfully for the first time. Such a schedule should allow sufficient time at the outset for the learning curve of both formal notation and problems to be addressed. It is observed that the benefits of a complete external specification, early requirements decisions, complete interface specification, and a good software design compensates for the time needed to produce the formal specification. Second, the formal specification affected the structure and content of the ES. It exposed ES ambiguities and incomplete product definition. For example, the ES described normal system functionality, but did not define system behaviour under abnormal situations. Third, formal specification identified important test cases and boundary conditions. Some processes have preconditions or invariants that define corner cases, which had to be tested. Fourth, working on the formal specification helped new team members learn the product requirements by exposing areas of misunderstanding or ambiguity.

On the other hand, the project team also found some limitations of using formal methods. First, it was very hard to stop thinking in programming terms when HP-SL was used for specifications. This is because while learning HP-SL, people focused on the notation, but not the abstractions or modelling techniques required to use it effectively. Second, some parts of the formal specification were intellectually pleasing but not understandable by other members of the team or reviewers. Finally, no early estimates of code size and dynamic behaviour of the system could be reached from the formal specification.

(3) The London Air Traffic Control System

As part of its advanced program to expand and develop the air traffic control system over the south-east of England, the Civil Aviation Authority is building a major new operation centre, the Central Control Function (CCF) facility, at the London Air Traffic Control Centre. Praxis Systems Ltd., which is a software engineering company located in Bath, England, has been selected to conduct a design study for the CCF Display and Information System (CDIS) which forms part of this programme [PRAXIS 93],[Craigen 93]. CDIS forms a vital component of the data entry and display equipment used by air traffic controllers.

Since the requirements analysis phase of the CDIS project was critical to its overall success, Praxis used VDM in conjunction with *entity-relation analysis* (ERA) and *real-time Yourdon* techniques to define the requirements for the system in detail. The requirements were documented in three complementary models: a world model of the system in its environment, based on ERA, a model of the processing requirements based on real-time Yourdon techniques, and a formal specification of the CDIS data and functions built using VDM. The formal VDM specification is complementary to the ERA and Yourdon models, and is both abstract and unambiguous.

It has been suggested that VDM is difficult to be used effectively to capture the real requirements for interface design, while prototyping is extremely helpful in this respect. In order to define the details of the user interfaces (in terms of screen layouts, layering of windows, 'look and feel' etc), a number of prototype user interfaces were constructed for the major classes of user. These prototypes were demonstrated to users and their effectiveness evaluated. As a result of this exercise, the details of the actual interfaces were documented, as the third key component of the specification.

2.2 B-Method

The B-Method is a formal software development process used for the production of highly reliable, portable and maintainable software which is verifiably correct with respect to its functional specification [Abrial *et al* 1991]. The method uses the Abstract Machine Notation (AMN) as the language for specification, design and implementation within the process.

Specifications are organized and presented as Abstract Machines. Abstract Machine clauses provide a list of state variables, an invariant constraining and relating the state variables, and operations on the state variables. Operations are specified in a pre/post condition style: state variable changes are abstractly specified as substitutions of new values for old, under stated pre-conditions. Abstract Machines may be parameterised so that instances of machines can be reused in the incremental construction of more complex machines.

The variables of a constructed machine include the collection of variables from each of the used machines. The invariant includes the conjunction of the invariants from each individual machine. New variables can be introduced and new invariant conditions can be imposed. The initialisations from the used machines are inherited.

An operation from a used machine may be promoted, in which case it becomes an operation of the new machine. Also new operations can be constructed from existing operations.

The method is supported over the entire spectrum of activities from specification to implementation by a set of computer-aided tools.

A case of applying the B-Method is as follows:

Railway Signalling System

With the objective of increasing traffic movement by 25% while maintaining the safety levels of the conventional system, a computerized signalling system for controlling RER (regional subway) trains in Paris, called SACEM system (partly embedded hardware and software), was developed by GEC (General Electric Company) Alsthom, Matra Transport and RATP (the Paris transportation authority) in 1989, and has since been controlling the speed of all trains on the RER Line A in Paris [Guiho, Hennebort 1990],[Silva *et al* 1992].

The SACEM software consists of 21,000 lines of Modula-2 code. 63% of the code is safety-critical and has been subjected to formal specification and verification. The specification was written using Abrial's B-Method and the proofs were done manually using automatically generated verification conditions for the code and Hoare's logic. The validation effort for the entire system took about 100 man years and therefore, this exercise represents a substantial investment in the application of formal methods.

The experience shows that the SACEM work has primarily benefited from formal specification which enabled precise and unambiguous statements about the system to be made. It is believed that the system is safer as a result of the formal specification and verification exercise. Howevere, a difficult problem which the project team met during the development of the system was communication between the verification personnel and the signaling experts who were not familiar with the formal notations used. They overcame this problem by providing the signaling experts with a natural language description derived manually from the formal specification.

2.3 RAISE

RAISE (Rigorous Approach to Industrial Software Engineering) is a systematic development method [Bjϕrner *et al* 1985],[Prehn 1987], which is a combination of useful aspects of VDM with well-researched areas of algebraic specification techniques and CSP [Hoare 1985]. It provides two languages, RAISE Specification Language (RSL) and RAISE Development Language (RDL). RSL is intended to support multiple styles of specification and levels of design refinement, whilst RDL is for implementation of specifications.

Currently RAISE is successfully being used on two applications:

- The Bell and LaPadula security model. The purpose is to investigate modelling security in RSL and to examine the issues of combining the functional and security refinements. Approximately two person-months have been spent on this project and a draft report on the work has been produced.

- The Safe Monitoring and Control System. This system is a distributed controller, for

use in mining and other industrial applications. Six person-months have been spent. RSL has been used to formulate the requirements, which have evolved during this time.

The reason for RAISE being chosen for the projects was that the project teams believed that formal methods of specification are useful for a large class of systems, partly because the need for formulation forces people to consider the requirements carefully. The experience with formal development exercise in the projects supported this belief. However, one problem is that, because RSL is a large language, it is not obvious which style(s) of RSL to use at the various stages of development. It is suggested that this problem should be addressed by separating the concerns of formal specification and formal development. That is, the first RSL specification should be constructed at requirements capture, without any regard for development. Then the development route should be outlined and a new top-level specification suitable for refinement should be produced. Then the formal development can be carried out based on the new specification as in the RAISE method.

2.4 HOL Logic and System

HOL system is a proof assistant based on high order logic [Gordon 1988]. The HOL logic can be used as a formal language for system specification and verification. A HOL specification consists of a collection of *theories*. Each theory consists of a set of types, type operators, constants, definitions, axioms and theorems. The definitions describe all the functions introduced in the theory; the axioms present properties of the types or functions which do not need proving; and the theorems express more complex properties which need to be proved based on the definitions and/or the axioms. Furthermore, a theory can build on other theories in the way that all the types, constants, definitions, axioms and theorems of those theories can be used in the theory being built.

An example of using the HOL logic and system is as follows:

Embedded Microprocessors

The Viper (Verifiable Integrated Processor for Enhanced Reliability) is a microprocessor that has been specifically developed for safety-critical applications [Cullyer, Pygott 1987]. The HOL system has been used to verify parts of the processor. However the method used and the claims about the correctness of this processor have caused some controversy in industrial and even the formal methods communities [Brock, Hunt 1990],[Cohn 1989].

Viper was the first "real" microprocessor produced using formal proof technique and intended for serious use. The proofs focus on the top level functional specification with the host machine (state machine) and the block level description of Viper (register transfer level description). The proofs relating to the top level specification uncovered a type error, a confused definition and an incomplete check [Cohn 1988] while the (only partially completed) block level proof did not expose any problems.

The Viper project team has learned some lessons from the formal verification exercise. First, the dependability of the chip was enhanced at a price. Second, it is difficult to imagine that the formal verification produced a cheaper chip. Third, HOL can be used for large scale projects. This work has also raised the interesting question of whether such proofs should be carried out by mathematicians or digital designers.

3 Benefits and Limitations

The experience of formal methods in industry and academia suggests that although formal methods can bring benefits to software development, many limitations exist. We summarize below some survey results on these two aspects of formal methods, based on [Austin, Parkin 1993] and the practical applications provided previously, and analyse these in terms of safety critical systems applications.

3.1 Perceived benefits

- **Requirements and Specifications are unambiguous.**

The main reasons for this are twofold. The first is that all the variables used in formal specifications are typed and each type definition is based on mathematical objects (e.g. natural number, real number, boolean value) that have precise semantics. The second reason is that every operation in formal specifications is defined precisely in the sense of its precise input and output relationship.

- **Errors due to misunderstandings are reduced.**

  As formal specifications are unambiguous, communication between people involved in requirements analysis, specification construction, design and implementation via the formal specifications is enhanced. Therefore, errors due to misunderstandings are reduced.

- **Implementations based on formal specifications are usually easier than those based on informal ones.**

  This is because formal specifications usually present precise tasks for implementation, whereas informal ones cannot easily do so.

- **Correctness proofs can be carried out, especially for safety-critical properties.**

  Correctness proof has been recognized as a powerful approach to verifying implemented software systems against their specifications. It is especially necessary for safety critical applications. Since formal specifications adopt mathematical notation, correctness proofs become possible.

- **Validation of requirements specifications becomes easier.**

  Because of the precision of formal requirements specifications, every task specified can be precisely interpreted thus enhancing the clients' ability to scrutinize the correctness of formal requirements specifications.

3.2 Perceived limitations

- **Formal specifications are difficult to read.**

  The reasons for this limitation are twofold. The first is that the majority of people working in the computing industry at present are accustomed to traditional informal methods and are not well trained in formal notations.

  The second reason is that mathematical notations are usually more difficult to understand than informal descriptions. Two elements contribute to this difficulty. First, mathematical notations are concise and the information described by them is therefore compact. Second, the language in which these descriptions are expressed is necessarily terse and populated with abstractions.

- **Formal methods cannot help model all aspects of the real world.**

  The difficulty is that the real world includes static and dynamic aspects while formal methods are a static technology for dealing with modelling and abstraction. Dynamic aspects may be modelled using formal methods, but the model produced cannot really demonstrate the dynamic behaviour of the desired systems.

- **Correctness proofs are resource-intensive.**

  This is because considerable time is required to produce formal specifications. Furthermore, since there is intrinsic difficulty in performing correctness proofs automatically (e.g. assertion construction, associated knowledge management and efficient use), proofs have to be done manually or interactively with machines, which is resource-intensive.

- **Development costs increase (for some companies and projects).**

  The main reason for this limitation is that many companies and projects need to invest more money for training their staff in formal methods technology.

- **Formal specifications can still have errors.**

  As we mentioned previously, formal methods can help reduce errors due to misunderstand-

ing. However, this is no guarantee that people will not make mistakes in formal specifications (e.g. syntactic errors and semantic inconsistency). No formal method so far can provide automatic support to semantic consistency checking for formal specifications (only some tools for syntax and type checking are available).

- **There is no mechanism available in many popular formal methods for describing time constraints on a proposed system or a component of the system.**

  The reason for this is that the initial purpose of some formal methods (e.g. VDM, Z) is not for developing time-critical systems but for developing non time-critical systems. However, when they are used for time critical applications, it is difficult or impossible to describe timing behaviours.

- **Environments to support the use of formal methods are not available.**

  Tools to support the use of some formal methods do exist (e.g. *CADIZ* for Z notation was developed at the University of York, *mural* for VDM was built at the University of Manchester [Jones *et al* 1991]). However, none of them are powerful enough to support the whole activity of using formal methods, such as consistency checking of specifications, specification refinements, correctness proofs, software testing etc.

- **It is not yet clear how to incorporate formal methods into the whole life cycle of software development.**

  To solve this problem, we need to answer the following questions: How to construct good quality formal specifications (understandable, consistent, structured)? How to refine formal specifications? How to do software verification (including correctness proofs and software testing) in an efficient way? How to do documentation? how to manage a software project and so on. There have been some research results which answer some of these questions for some formal methods (e.g. VDM, see [Liu 1993],[Bear 88]), but there is still a long way to go for many other formal methods (e.g. Z, RAISE).

4 Safety Requirements

Safety is a property of an overall system, which depends on both hardware and software in embedded safety critical systems. A safety requirement can be expressed in terms of knowledge about the possible causes of system safety failure (e.g. the airplane crash is caused by the engine failure). The result of this analysis can help engineers understand the safety problem concerned and make decisions on eventual system design, including hardware and software. Several techniques for safety analysis have been used by industry for decades, and some have attracted great attention in the research community. They include *Fault Tree Analysis*, *Failure Modes, Effects and Criticality Analysis*, *Failure Propagation and Transformation Notation*, and *Toulmin Argument Form*.

We believe that identification of appropriate safety requirements is a prerequisite for any useful safety critical application of formal methods. Therefore, safety analysis methods must be incorporated in the lifecycle of formal methods applications. It is important to realize that formal methods are not alternatives to safety analysis; the latter gets as close to analysing physical reality as possible, while the former deals in models and abstractions.

In this section we focus on the introduction of four techniques. Their principles and possible connections with formal methods are described.

4.1 Fault Tree Analysis (FTA)

Fault Tree Analysis has been used for the assessment of system reliability and safety [Chelson 1971] for decades and has been developed into a well-understood, standardised method with wide applications throughout the discipline of safety and reliability engineering. A comprehensive introduction to fault tree analysis is the extensive and authoritative Fault Tree Handbook [Veseley *et al* 1981].

Traditional fault tree analysis is a probabilistic

method in which potential causes of some failure ("top event") are organised in a tree structure reflecting causality – causality is a crucial notion underlying all safety analysis techniques. High-level events can be caused by various combinations of lower-level events, with the principal logical connectives used in the tree being **AND** and **OR** gates, which have meanings analogous to those traditionally used in electronic circuit design. Priority-**AND** gates, exclusive-**OR** gates and **INHIBIT** gates (which generate a true output if some input representing an event in the system is true, and some external "conditioning event" has occurred) are also available for use. Examples of applying fault trees to system safety analysis are described in [HSE 1987],[Bowman *et al* 1991].

Leveson and her colleagues were the first to apply fault trees to the safety analysis of software [Leveson, Harvey 1983a],[Leveson, Harvey 1983b] at the statement level. Software fault trees are derived from the software (programs) based on the semantics of statements (e.g. sequential, conditional and iteration statements). Since the statements may be either concrete or abstract (e.g. **if** $x > 1$ **then** $y := x - 1$ **else** $y := x + 1$ is a concrete conditional statement, while **if** $x > 1$ **then** S_1 **else** S_2 is an abstract one because S_1 and S_2 are not defined yet), software fault tree analysis can be applied on both software design and code levels. The goal of software fault tree analysis is to show that the logic contained in the software design will not produce system failures, and to determine environmental conditions which could lead to the software causing a safety failure.

Clark and McDermid propose a more traditional view of the application of fault trees to software [Clark, McDermid 1993]. It is suggested that weakest preconditions are used for program specification and validation, and software fault tree analysis is employed for a system-wide analysis of hazards. The scope of software fault trees can be increased to include, for example, compiler errors, control errors, and memory errors, as well as logical errors. Thus a more realistic view of the software's role in system hazards can be given.

Hansen *et al* have recently developed fault trees into a notation for describing software safety requirements for design specifications [Hansen *et al* 1993]. Specifications are given in a real-time, interval logic, based on a conventional dynamic systems model with a state changing over time. Fault trees are interpreted as temporal logic formulae giving a cause effect relationship between states. It is shown how such formulae can be used for deriving safety requirements for design components. Similar work on formalisation of fault trees is also described in [Bruns, Anderson 1993].

4.2 Failure Modes, Effects and Criticality Analysis (FMECA)

FMECA is a quantitative and qualitative method which is used to analyse the effects of a single failure on a system. In particular, it is useful as a tool for summarising the failure behaviour of a system and can be used "pro-actively" as a tool for reliability growth [Raheja 1990] if well integrated into the development lifecycle. FMECA is also often used as the basis for analysing maintainability and related requirements.

The principle of FMECA is that it attempts to evaluate the effects of a single failure of a component on the system as a whole. The mapping from failure modes to effects may be a direct one, or hierarchies of failure modes and effects may be necessary, with combinations of failures required to result in particular output states. FMECA and related techniques are of great pragmatic importance as one of the most common requirements in many safety critical systems is that there should be no single point of failure which can lead to a hazard.

FMECA is well-understood at the systems level. Sound methodologies and standards have existed for many years [APR 1967] and equipment suppliers and users have developed much experience in using the method. However, it is necessary to formalise FMECA activities if it is to be used in conjunction with formal methods for the development of safety critical systems. It seems that not much research has been done on this topic.

4.3 Failure Propagation and Transformation Notation (FPTN)

FPTN is a graphic method for expressing the failure behaviour of systems with complex internal structures [Fenelon, McDermid 1993]. Since failure propagation can be viewed as a form of data flow, FPTN was initially intended to resemble data flow based methods such as CORE and Mascot. It is a modular and hierarchical notation which allows decomposition based on system architecture where the conventional concept of data flow is replaced by failure propagation between modules.

A software module in FPTN is represented by a box with a collection of input and output failure modes. The box contains a set of predicates which express the relationship between the input and output failure modes of the module. These predicates in fact correspond to the sum-of-products form of the minimal cutsets of the fault tree (a minimal cutset being any set of conditions necessary and sufficient to cause the loss event described at the top of the tree) for each output failure mode. In effect the FPTN box contains a forest of fault trees laid on their sides.

FPTN may be used in conjunction with structured and formal methods based on data flow analysis [Liu 1993],[Liu 1994] for safety critical systems. The functional specifications can be the basis of failure behaviour analysis of the system using FPTN and FPTN analysis can result in more safety critical information for the refinement of the functional specifications.

4.4 Toulmin Argument Form (TAF)

Toulmin Argument Form was proposed by the English philosopher Stephen Toulmin in 1950's [Toulmin 1958] and was initially designed as a method and tool for expressing the structure of arguments. It is usually expressed graphically and has six parts as follows:

- Claim: the claim is an assertion describing the conclusion we wish to establish (e.g. the safety of the process of producing PETN can be guaranteed, where PETN stands for pentaerythritol tetranitrate);
- Data: the data provides some facts to support the claim (e.g. the decomposition can be prevented);
- Warrant: the warrant presents a justification for deriving the claim from the data (e.g. as long as the decomposition can be prevented, the safety of the process of producing PETN can be guaranteed);
- Backing: the backing provides some facts to support the warrant and to provide evidence that it is trustworthy (e.g. the corresponding chemical principles);
- Qualifier: the qualifier indicates the degree to which data and warrant support the claim (e.g. probably or certainly);
- Rebuttal: the rebuttal is a statement describing an exceptional condition under which data and warrant do not support the claim (e.g. the reaction container has ruptured)

Quite recently the Toulmin Argument Form has attracted the attention of the safety critical systems research community. In the ASAM (A Safety Argument Manager) project which was funded by SERC and DTI, and conducted by the University of York in collaboration with Logica Cambridge Ltd. and the Civil Aviation Authority [Forder *et al* 93], the Toulmin Argument Form was developed and applied to express the safety case of a safety critical system.

5 Discussion

There are several problems with the existing methods and principles for developing the software of safety critical systems, but we believe the following three are most critical.

First of all, there is no well-designed language and principle to describe mappings from physical environments (including physical model, operation principles and properties of the physical systems) to safety requirements. Therefore, formal validation of safety requirements cannot be conducted. The ASAM project has made some efforts in this

area [Forder *et al* 1993]. In SAM (Safety Argument Manager) which is a prototype produced by the ASAM project to support the management of safety arguments, system modelling is thought to be a first step in the construction of safety requirements. Its purpose is to map a physical model of a physical system to a logical system model which is expected to reflect all the safety related knowledge about the physical system. However, there is no principle on how to do this mapping, and the language used to express the logical system model is a collection of binary relations, which is not powerful enough for the purpose.

Second, safety requirements are usually expressed in natural language (although some graphical notation may be used to layout the structure of text, such as Toulmin Argument Form). Such requirements are not very helpful to software developers, and cannot serve as a firm basis for proving whether a developed software system satisfies the corresponding safety requirements. There has been little research so far on the application of formal methods to expressing safety requirements. The benefits and disadvantages of this kind of application are not clear yet.

The third problem is that the precise relationship between safety and the functionality of a system is not consistently used as a framework for software development for safety critical systems in the whole software life cycle. The safety of an overall system and the functionality of the software used in this system are analysed, designed, implemented and verified separately. There is no systematic approach so far to link them together properly.

Formal methods can be an effective tool in increasing the confidence of software implementations for safety critical systems. However, it is not clear how to use formal methods in order to ensure that safety and functional requirements are properly captured and implemented.

We believe that safety is a property of an overall system (including software and hardware) and is completely reflected by the functional behaviour of the software system and hardware. In order that the functional behaviour conforms to safety requirements, safety constraints must be considered and enforced during the whole process of software development including functional requirements analysis, design, implementation and verification, rather than after the software system is implemented.

In order to realise this goal, a safety-oriented approach to software requirements analysis and implementation is necessary. Without such an approach, even if we precisely specify the functional requirements using formal methods, we are still not able to guarantee that the implemented software satisfies the corresponding safety requirements.

6 Conclusions

This paper describes the current practices of formal methods technology in safety critical systems in industry, and analyses the benefits and problems of using formal methods. The evidence shows that formal methods can help increase confidence in software. The most successful role which formal methods can play is to assist developers to clarify requirements, to allow formal reasoning about specifications and to allow correctness proofs of implemented programs against their formal specifications. In order to support the application of formal methods, efficient tools and efficient environments need to be developed and staff training must be provided.

In particular for safety critical systems, there is an absence of the principle for enforcing safety constraints during the whole process of software development, and no appropriate language and method for supporting this principle exists. We believe that a proper combination of causal tree analysis, temporal logic, and pre- and post-conditions style notation is a step in the right direction.

7 Acknowledgment

We would like to thank all our colleagues in the formal methods group for their contributions to this research in various ways. David Outteridge reviewed the draft of this paper and discussed many relevant issues. Anne E. Freitag and Janet Hales

helped us correct the English. Their comments and suggestions have helped us improve the quality of this paper. The paper has also benefitted from the comments of two anonymous referees.

8 References

[Abrial *et al* 1991] J.-R. Abrial et al, "The B Method", in VDM'91 Formal Software Development Methods (ed. S. Prehen and W.J. Toetenel). Springer-Verlag, 1991. pp. 398-405.

[APR 1967] APR926, "Design Analysis Procedure for Failure Modes, Effects and Criticality Analysis (FMECA)", Aerospace Recommended Practice (APR) 926, Detroit, USA, Society of Automotive Engineerings (SAE), 15th September, 1967.

[Austin, Parkin 1993] Stephen Austin and Graeme I Parkin, "Formal Methods: A Survey", Published by National Physical Laboratory, Teddington, Middlesex, U.K., March 1993.

[Bear 1988] S. Bear, "Structuring for the VDM Specification Language", VDM'88, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, pp. 2-25, 1988.

[Bjϕrner, Jones 1982] D. Bjϕrner and C.B. Jones, "Formal Specification and Software Development", Prentice-Hall International Inc., 1982.

[Bjϕrner *et al* 1985] D. Bjϕrner *et al*, "The RAISE Project - Fundamental Issues and Requirements", RAISE/DDC/EM/1, Dansk Datamatik Center, 1985.

[Bowen, Stavridou, 1993a] Jonathan Bowen and Victoria Stavridou, "Safety-Critical Systems, Formal Methods and Standards", Software Engineering Journal, **8(4)**, July 1993, pp. 189-209.

[Bowen, Stavridou, 1993b] Jonathan Bowen and Victoria Stavridou, "Formal Methods: Epideictic or Apodeictic?", to appear in the Software Engineering Journal, 1993.

[Bowman *et al* 1991] William C. Bowman, *et al.*, "An Application of Fault Tree Analysis to Safety-Critical Software at Ontario Hydro", Proceedings of the Conference on Probabilistic Safety, Assessment and Mangagement, April 1991.

[Brock, Hunt 1990] B. Brock and W.A. Hunt, "Report on the Formal Specification and Partial Verification of the VIPER microprocessor", Technical Report No. 46, Computational Logic Inc., Austin, Texas, USA, January 1990.

[Brownbridge 1989] David Brownbridge, "Using Z to Develop a CASE Toolset", in Proceedings of the 1989 Z user meeting, Workshops in Computer Series, Springer-Verlag, December 1989.

[Bruns, Anderson 1993] Glenn Bruns and Stuart Anderson, "Validating Safety Models with Fault Trees", SAFECOMP'93, Proceedings of the 12th International Conference on Computer Safety, Reliability and Security, Springer-Verlag, 1993.

[Chelson 1971] P.Q. Chelson, "Reliability Computation Using Fault Trees", Technical Report (NASA-CR-124740), NASA Jet Propulsion Laboratory, 1971.

[Clark, McDermid 1993] Stephen J. Clarke and John A. McDermid, "Software Fault Trees and Weakest Preconditions: A Comparison and Analysis", Software Engineering Journal, July 1993, pp. 225-236.

[Cohn 1988] A.J. Cohn, "A Proof of Correctness of the Viper Microprocessor: the First Level", in VLSI Specification, Verification and Synthesis, Kluwer Academic Publishers, 1988.

[Cohn 1989] A.J. Cohn, "The Notion of Proof in Hardware Verification", Journal of Automated Reasoning, May 1989, 5, pp. 127-139.

[Collins *et al* 1987] B.P. Collins, J.E. Nicholls and I.H. Sorensen, "Introducing Formal Methods: the CICS Experience with Z", IBM Technical Report TR 12.260, December 1987.

[Craigen *et al* 1993] Dan Craigen *et al*, "An International Survey of International Applications of Formal Methods Volume 2 Case Studies", NISTGER 93/626, 1993.

[Cullyer, Pygott 1987] W.J. Cullyer and C.H. Pygott, "Application of Formal Methods to the VIPER microprocessor", in IEEE Proceedings, Part E, Computers and Digital Techniques, May 1987, 134, (3), pp. 133-141.

[Cyrus *et al* 1991] Judith L. Cyrus, J. Daren, Paul D. Harry, "Formal Specification and Structured Design in Software Development", Hewlett-Packard Journal, December 1991.

[Delisle, Garlan 1990] N. Delisle and D. Garlan, "A Formal Specification of an Oscilloscope", IEEE Software, September 1990, pp. 29-36.

[Dyer 1992] Michael Dyer, "The Cleanroom Approach to Quality Software Development", Wiley Series in Software Engineering Practice, 1992.

[Ehrig, Mahr 1985] H. Ehrig and B. Mahr, "Fundamentals of Algebraic Specification 1", Springer-Verlag, Berlin Heidelberg, 1985.

[Fenelon, McDermid 1993] Peter Fenelon and John A. McDermid, "Safety Case: An Integrated Toolset For Software Safety Analysis", The Journal of Systems and Software, 1993: 21(3), pp. 279-290.

[Forder *et al* 1993] Justin Forder, Chris Higgins, John McDermid et al, "SAM - A Tool to Support the Construction, Review and Evolution of Safety Arguments", Proceedings of Safety-critical Systems Symposium 1993, Bristol, Springer-Verlag, 9th-11th February, 1993.

[Garlan, Delisle 1990] D. Garlan and N. Delisle, "Formal Specifications as Reusable Frameworks", VDM 90: VDM and Z!, Springer-Verlag, New York, 1990, pp. 150-163.

[Goguen, Tardo 1979] J.A. Goguen and J.J. Tardo, "An Introduction to OBJ: a language for writing and testing formal algebraic program specifications", Proceedings IEEE Conference on Specification for Reliable Software, IEEE Computer Society, pp 170-189, 1979.

[Goguen, Winkler 1988] J.A. Goguen and Timothy Winkler, "Introducing OBJ3", Technical Report, SRI-CSL-88-9, SRI International, August 1988.

[Gordon 1988] M.J.C. Gordon, "HOL: A Proof Generating System for High Order Logic", in G. BIRTWISTLE and P.A. SUBRAMANYAM (Eds): VLSI Specification, Verification and Synthesis", (Kluwer, 1988) pp. 73-128.

[Guiho, Hennebort 1990] G. Guiho and C. Hennebort, "SACEM Software Validation", International Conference on Software Engineering, 1990.

[Hansen *et al* 1993] K.M. Hansen, A.P.Ravn and V. Stavridou, "Linking Fault Trees to Software Specifications", Proceedings of Colloquium on Analysis of Requirements for Software Intensive Systems, Malvern, U.K., 19-20 May, 1993.

[Haykin, Warmar 1988] M.E. Haykin and R.B.J. Warmar,"SmartCard Technology: New Methods for Computer Access Control", NIST Special Publication 500-157, Sept. 1988.

[Hill 1989] J.V. Hill, "The development of high reliability software - RR and A's experience for safety critical systems", Second IEE/BCS Conference, Software Engineering 88, Conference Publication No. 290, July 1988, pp. 169-172.

[Hill 1991] J.V. Hill, "Software development methods in practice", Proc. 6th Annual Conference on Computer Assurance (COMPASS), 1991.

[Hoare 1985] C.A.R. Hoare, "Communicating Sequential Processes", Prentice-Hall International, 1985.

[Houston, King 1991] I. Houston and S. King, "CICS project Report: Experiences and Results from the use of Z", VDM'91: Formal Software Development Methods", Vol. 551, pp. 588-96, Lecture Notes in Computer Science, Springer-Verlag,

December 1991.

[HSE 1987] Health and Safety Executive, "PES – Programmable Electronic Systems in Safety Related Applications", Crown copyright, 1987. pp. 80-99.

[Haykin, Warmar 1988] Martha E. Haykin and Robert B.J. Warmar, "Smartcard Technology: New Methods for Computer Access Control", NIST Special Publication 500-157, Sept. 1988.

[Jones 1990] C.B. Jones, "Systematic Software Development Using VDM", Prentice-Hall International, Second Edition, 1990.

[Jones *et al* 1991] C.B. Jones, K.D. Jones, P.A. Lindsay and R. Moore, "*mural*: A Formal Development Support System". Springer-Verlag, 1991.

[King 1990], S. King, "Z and the Refinement Calculus", VDM'90, Lecture Notes in Computer Science, Springer-Verlag, pp. 164-188, 1990.

[Kuhn, Dray 1990] D.R. Kuhn and J. F. Dray, "Formal Specification and Verification of Control Software for Cryptographic Equipment", 6th Computer Security Applications Conference, Phoenix AZ, December 6-8, 1990.

[LACOS 1991] LACOS, "Experiences from Applications of RAISE", LACOS project reports, LACOS/CRI/CONS/13/V1 and LACOS/CRI/CONS/20/V1, June 1991 and March 1992.

[Leveson, Harvey 1983a] Nancy G. Leveson and Peter R. Harvey, "Analyzing Software Safety", IEEE Transactions on Software Engineering, Vol. SE-9, No. 5, September 1983, pp. 569-579.

[Leveson, Harvey 1983b] Nancy G. Leveson and Peter R. Harvey, "Software Fault Tree Analysis", The Journal of Systems and Software, 1983, pp. 173-181.

[Linger, Mills 1988] R. Linger and H. D. Mills, "A Case Study in Cleanroom Software Engineering: the IBM COBOL Structuring Facility", CompSac, IEEE Computer Society, 1988.

[Linger *et al* 1991] R. Linger, H. Mills and B. Eitt, "Software Engineering Laboratory (SEL): Cleanroom Process Model", NASA Goddard Space Flight Centre, SEL-91-004.

[Littlewood 1993] Bev Littlewood, "The Need for Evidence from Disparate Sources to Evaluate Software Safety", Proceedings of the Safety-critical Systems Symposium, Bristol 1993, pp. 217-231.

[Liu 1993] Shaoying Liu, "A Formal Requirements Specification Method Based on Data Flow Analysis", The Journal of Systems and Software, May 1993, pp. 141-149.

[Liu 1994] Shaoying Liu, "Internal Consistency of FRSM", to appear in The Journal of Systems and Software, 1994.

[MoD 1991] MoD91, "The procurement of Safety Critical Software IN Defence Equipment", Interim Standard 00-55 Issue 1, Minstry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow G2 8EX., 1991.

[Mukherjee, Stavridou 1993] Paul Mukherjee and Victoria Stavridou, "The Formal Specification of Safety Requirements for Storing Explosives", Formal Aspects of Computing, **5(4)**, pp. 299-336, 1993.

[Nix, Collins 1988] C.J. Nix and B.P. Collins, "The use of Software Engineering, including the Z notation, in the Development of CICS", Quality Assurance, Vol. 14, No. 3, pp. 103-110, September, 1988.

[PRAXIS 93] PRAXIS, "A Technical Overview of CDIS", PRAXIS CDIS Technical Overview, Bath, England, 1993.

[Prehn 1987] S. Prehn, "From VDM to RAISE", VDM– A Formal Method at Work, Lecture Notes in Computer Science 252, Springer-Verlag, pp 141-150, March 1987.

[Raheja 1990] Dev Raheja, "Software System Fail-

ure Mode and Effects Analysis (SSFMEA) - A Tool For Reliability Growth", IRSM'90, Tokyo, pp. IX-1 - IX-7 (1990).

[Rush *et al* 1990] T. Rush *et al*, "Case Studies in HP-SL", Software Engineering Department, HP Laboratories Bristol, HPL-90-137, August 1990.

[Silva *et al* 1992] C.Da Silva *et al*, "Formal Specification in the Development of Industrial Applications: The Calcutta Subway", April 1992, submitted for publication.

[Spivey 1988] J.M. Spivey, "Understanding Z: a specification language and its formal semantics", Number 3 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1988.

[Spivey 1989] J.M. Spivey, "The Z Notation", Prentice Hall International, 1989.

[Toulmin 1958] Stephen Toulmin, "The Uses of Argument", Cambridge University Press, 1958.

[Veseley *et al* 1981] W.E. Veseley *et al*, "Fault Tree Handbook", Division of the System Safety Office of Nuclear Reactor Regulation, US Nuclear Regulatory Commission, Washington DC, 1981.