The SHIP Safety Case Approach

P.G. Bishop and R.E. Bloomfield,
Adelard,
London, E3 2DA, England

Abstract

This paper presents a safety case approach to the justification of safety-related systems. It combines methods used for handling software design faults with approaches used for hazardous plant. The general structure of the safety argument is presented together with the underlying models for system failure that can be used as the basis for quantified reliability estimates. The approach is illustrated using plant and computer based examples.

Introduction

The SHIP project was sponsored under the EU Environment Programme (Major Industrial Hazards). The objective of the project was to assure plant safety in the presence of design faults but it was tackled from a novel standpoint. In software, all faults are design faults so techniques developed for software might well be applicable to the design of complete systems.

This paper describes a central element of this research—the SHIP safety case. The concept of a "safety case" grew out of work in the nuclear industry and is now a familiar term in many industries. For example, in the UK the CIMAH regulations implement the EC Directive on Major Hazards (the Seveso Directive) and require a report on the safety of the installation that addresses the dangerous substances involved; the installation itself; the management system; and the potential for major accidents. In SHIP, the safety case concept has been formalised and extended to cover both hardware and software systems.

Elements of a safety case

We define a safety case as:

> *"a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment"*

The development of the SHIP safety case is based on: a simple model of the system behaviour; the available evidence; and an argument which translates evidence into claims about behaviour.

General Safety Case Structure

The safety case should be developed in parallel with the design. It will evolve and become more detailed as the system is developed. At each stage, the basis for the safety arguments should be clear. The safety case should:

- make an explicit set of claims about the system
- provide a systematic structure for marshalling the evidence
- provide a set of safety arguments that link the claims to the evidence
- make clear the assumptions and judgements underlying the arguments
- provide for different viewpoints and levels of detail

There is much work on the structuring and representation of arguments in mathematical logic [Gentzen69] and in formal methods [Hoare69, Jones90]. In the safety field we have the traditional representation such as fault trees [Vesely81] and the safety arguments in the Safety Argument Manager (SAM) [McDermid94]. The safety case structure developed in SHIP draws on this work. In addition we have formalised the basic argument structure, so that the safety argument could in principle be checked, supported and maintained by tools if the approach is further developed. Within this structure we consider the basic types of argument that can be deployed.

Within the SHIP model, a safety case consists of the following elements: a *claim* about a property of the system or some subsystem; *evidence* which is used as the basis of the safety argument; an *argument* linking the evidence to the claim, and an *inference* mechanism that provides the transformational rules for the argument. This is summarised in the figure below.

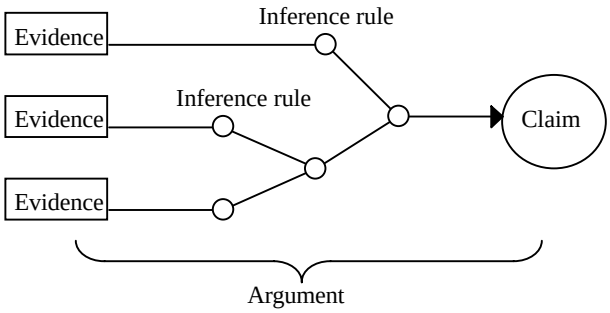

Figure 1: Argument Structure

The actual nature of the argument and the inference mechanism can vary depending on the system design and the safety case strategy. For example an argument could be:

- *Deterministic*, where the evidence can be axioms, the inference mechanism is the rules of predicate logic, and the safety argument is a proof using those rules.
- *Probabilistic*, where the evidence could be component failure rates and assumptions of independence, and the inference mechanism is statistical analysis.
- *Qualitative*, where the evidence might be adherence to standards, design rules, or guidance. The inference mechanism is some form of acceptance criterion based on this.

In addition the overall argument should be *robust*, i.e. the argument can be sound even if there are uncertainties or errors.

Structuring a Safety Case

In practice it is unlikely that any safety case will be entirely deterministic or probabilistic. It may consist of a number of claims about specific properties of the system which may not necessarily be the same type. In addition it needs to be viewed at various levels of detail. It is proposed that a safety case can be structured as a hierarchy of claims as shown below:

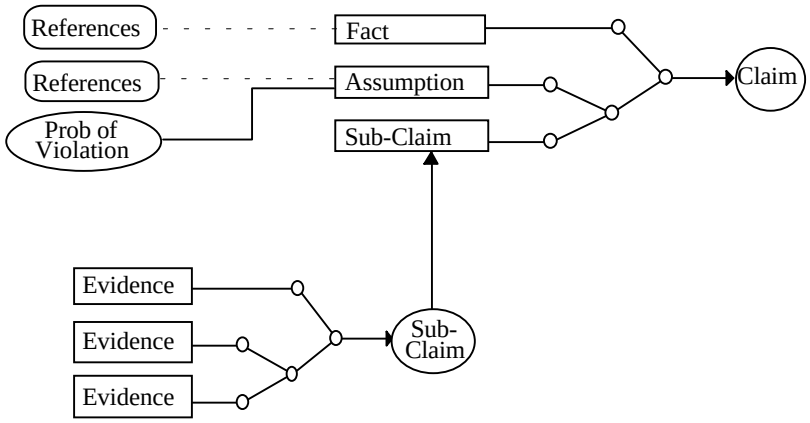

Figure 2: Hierarchic Argument Structure

In this model, the evidence used at one level of the argument can be:

- facts, e.g. based on established scientific principles and prior research

- assumptions, which are necessary to make the argument, but may not always apply in the "real world"
- sub-claims, derived from a lower-level sub-argument

This is a recursive structure which can represent arguments at successively finer levels of detail. This structure could evolve over the lifetime of the project. Initially some of the sub-claims might actually be *design targets*, but as the system develops the sub-claims might be replaced by facts or more detailed sub-arguments based on the real system. Deviations in implementation can be analysed to see how this affects a sub-claim, and how changes in sub-claim "ripple through" the safety argument.

In order to simplify the evaluation of this structure, it is proposed that:

- any argument tree must be consistent in type using a single consistent set of inference rules
- the evidence from sub-arguments must be consistent in type, or it must be possible to transform the type

For example, if the top-level argument is probabilistic, and there is a deterministic argument that some subsystem is free of design faults, the zero fault result is transformed into a zero failure rate in the top-level evidence. Equally if the top-level argument is deterministic, lower-level probability calculations can be linked to probabilities of violating the assumptions. For example it might be possible make a deterministic argument that a system is safe provided that "at least two feed pumps will always be available". A separate lower-level computation might calculate the chance that this assumption will be invalid. This information might then be used to qualify the claim (e.g. there is 0.999 chance/year of safe operation).

Deterministic Argument

A deterministic argument supports a claim or sub-claim by showing that, given some assumptions and a model of the real world, certain hazardous behaviours are "incredible". A simple example for a chemical plant may be that the inventory of two chemicals is not sufficiently large for a critical explosive mass to be present. An example from the computer area would be a proof that a communications protocol cannot deadlock.

In addition to deterministic claims about the behaviour of the system, weaker deterministic arguments may be made about the faults in a system. These arguments require evidence of the complete absence of certain classes of faults for a particular system function or component. For example, the use of a CAD system may exclude some forms of translation fault or the typing mechanisms in a high level computer programming language may exclude certain errors.

Deterministic arguments would normally require a formal model of the system and a proof that the system is safe with respect to its safety requirements (the proof could be a rigorous style argument rather than a machine-checked proof). The supporting evidence could include:

- explicit validation of the model assumptions
- an independent check of the formal argument

In addition to arguments based on formal models, it may also be possible to claim "fault-freeness" on the basis of exhaustive test coverage of all required behaviours.

Probabilistic Argument

Probabilistic argument combines parameter estimates to obtain an estimate of the probability of some top level property (e.g. dangerous failure). The inference structure might be some form of Bayesian combination, evaluation of some stochastic model (e.g. Markov model or a fault tree), or simple statistical combination. It should be noted that probabilistic arguments also make use of an underlying model of system behaviour and the relationships between the various forms of evidence.

Qualitative Argument

Most safety cases will include important qualitative claims. In some ways they are similar to deterministic arguments in the sense that some particular property exists. While deterministic arguments might be binary, qualitative arguments might be more fuzzy (e.g. refer to a rating such as "good", "indifferent" or "bad") and the ratings may be assigned by expert judgement. Qualitative assessment can also be binary where some "tick-list" of criteria must be satisfied to demonstrate acceptability (e.g. conformity to standards, construction criteria, or design rules). This may be a valid approach if the design tick-list encapsulates past experience which has been shown to achieve safety.

Dealing with Uncertainty

Any safety argument is susceptible to error (e.g. in the evidence, the argument or the assumptions) so there should be strategies for limiting the risks associated with such errors. One simple strategy is to design the overall argument so that can withstand a single flaw, i.e. we adopt a qualitative defence in depth approach to the argument. In this case it might be structured as follows.

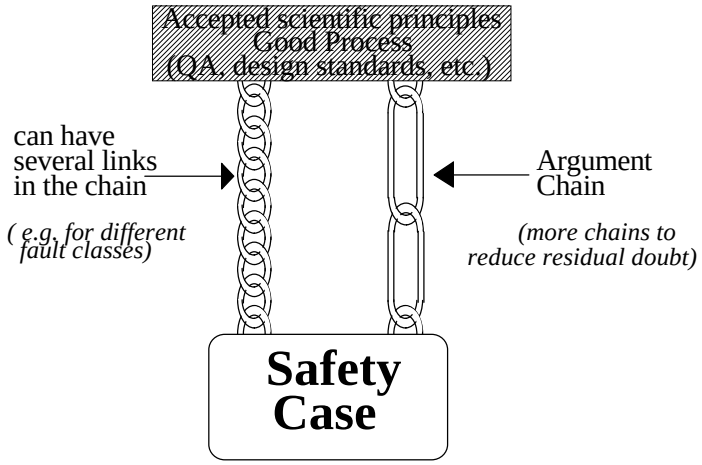

Figure 3: Example of a Safety Argument that Minimises Residual Doubts

Either chain would be sufficient to support the claim, and care should be taken to avoid any common links (e.g. common assumptions) between the two chains.

More sophisticated arguments could also be deployed which take a more quantified approach to such factors as: the confidence in specific assumptions, common mode failure probability, and numerical limits for claims made on any single leg

Approach to Quantification

While the safety case structure can accommodate qualitative claims for system safety, the main objective of SHIP was to use a more quantified approach. In this section we outline the basic concepts that can support a quantified safety argument.

Underlying Models

Our approach to reliability quantification in a safety case is based on two simple underlying models. The first is based on a standard model for software failure—and since software failures are due to design flaws, the same theories should be directly applicable to plant or computer hardware failures caused by design flaws. The second model considers the overall response of the system when a failure occurs, and demonstrates that the safety argument has to take into account the design methods, development process, and existing field experience.

Failure model for software

In trying to understand and predict the observed reliability of a software based system we need an underlying model for software failure. The reliability of a system is based on three factors:

1) the number of faults
2) the size and location of faults
3) the input distribution (operational profile)

This is illustrated in the figure below.

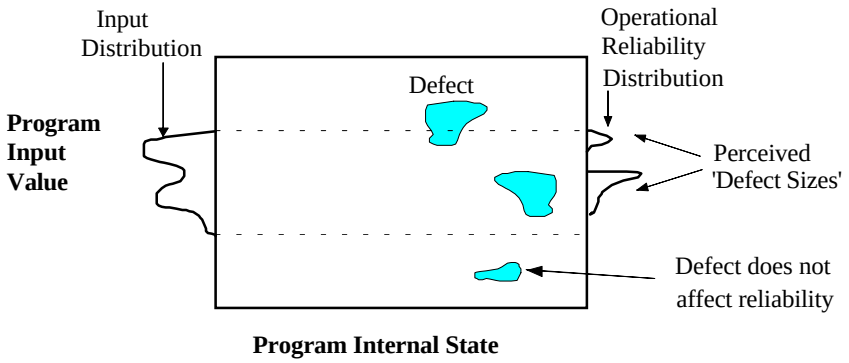

Figure 4: Illustration of the Software Failure Process

It is clear from the diagram that an alteration of the input distribution could radically alter the operational failure rate of the system. Where there is a single copy running in a fixed environment or where there are very many copies of the software running, the input distribution is likely to be effectively stable. Under such a stable input distribution, the faults are likely to have a fixed "perceived size" (which may be zero if a fault is not covered by input values).

In practice the number of faults within an item of software will not remain static. As operating experience is gained, faults will be revealed and corrected so the reliability of the software should grow with increasing execution time.

We considered that this model could be applied directly to hardware systems, so the related software reliability assessment methods could also be deployed. These methods include reliability growth modelling, testing and formal methods.

System failure behaviour

In safety related systems, we are not just concerned with reliability in general; we also need to distinguish between dangerous and safe failures. This leads to the underlying model of behaviour shown below.

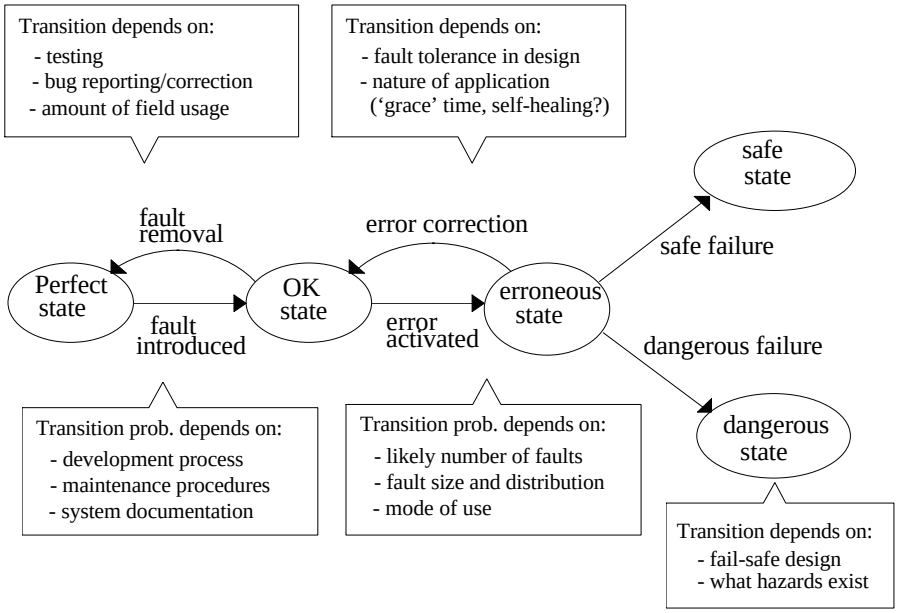

Figure 5: Model of System Failure Behaviour

This follows the standard fault-error-failure model for software. A *fault* is a defect in the program code and is the primary source of the failure. After development, the program could be perfect or faulty. However, even if it is faulty, the program may still operate correctly most of the time (i.e. stay in the OK state) until some triggering input condition is encountered. Once triggered, some of the computed values will deviate from the design intent (an *error*). However the deviation may not be large enough (or persist long enough) to be dangerous, so the system may recover naturally from the "glitch" in subsequent computations ("*self healing"*). Alternatively explicit design features (e.g. diversity, "firewalls", etc.) can be used to detect such deviations and either recover the correct value (*error recovery*) or override the value with a safe alternative (*fail-safety*).

If the failures are reported back then, in the longer term, the software can be corrected and a new version issued. Each version should (hopefully) contain fewer faults than the previous one and could potentially result in a "perfect" program. This is the principle underlying software reliability growth modelling.

These concepts should be equally applicable to faults in plant designs. The only difference in physical systems is that faults can occur spontaneously (e.g. random failures due to deterioration or stress) without any external intervention, and these faults can be fixed using a new part of the *same* design. However this aspect is already covered in conventional system reliability analyses.

Given that the basic concepts are valid, we still need to select which concepts will be deployed to support the specific safety argument. The strategy for developing a safety case is discussed below.

Using the Models to Develop a Safety Case Strategy

The overall approach to generating the safety case involves:

- characterising the safety case arguments in terms of the transitions of the model
- ensuring the implementation strategy is compatible with the safety argument(s)
- determining and evaluating the evidence to support the claims made about the transition probabilities in the model

Characterising the safety case

As noted above one of the primary claims in developing the safety case is about the probability of dangerous failure. In developing the safety case arguments, it is useful to consider the mechanisms that determine the dangerous failure probability as indicated in the annotations of Figure 5. However, this is a general model, and a particular safety argument may focus on quantifying particular transition arcs. The main approaches are listed below:

1) A fault elimination and quantification argument can increase the chance of being in the "perfect" state and can also would reduce the probability of the *OK* → *erroneous* transition. An extreme example would be an argument of correctness. This would imply that the error transition rate was zero, and this would be sufficient to bound the dangerous failure rate.

2) A failure containment argument that would strengthen the *erroneous* → *OK* or *erroneous* → *safe* transition. An example would be a strongly fail-safe design which quantifies the fail-safe bias. This, coupled with test evidence bounding the error activation rate, would be sufficient to bound the dangerous failure rate.

3) A failure rate estimation argument that would estimate the *OK* → *dangerous* transition. The whole system is treated as a "black-box" and

probabilistic arguments are made about the observed failure rate based on past experience or extensive reliability testing.

It is also possible to apply the arguments selectively to particular components or fault classes, e.g.:

1) A design incorporates a safety barrier which can limit dangerous failures occurring in the remainder of the system. The safety argument would then focus on the reliability of the barrier rather than the whole system.

2) Different countermeasures might be utilised for different classes of fault. Each fault class then represents a separate "link" in the argument chain, and all fault classes would have to be covered to complete the argument chain. For example, design faults might be demonstrated to be absent by a deterministic argument, while random hardware failures are covered by hardware redundancy.

Implementation supports the safety argument

The previous discussion illustrates the key and closely coupled roles of the development processes and the design in formulating the safety case. Sometimes, the design and development approach is geared toward implementing the operational requirements; the need to *demonstrate* safety is only considered at a later stage. This can lead to considerable delays and additional assessment costs. The safety case should be an integral part of the design methodology and the feasibility and cost of the safety case should be evaluated in the initial design phase. This "design for assessment" approach should help exclude unsuitable designs and enable more realistic design trade-offs to be made.

Sources of evidence and types of argument

The arguments themselves may be either probabilistic or deterministic and utilise evidence from the following main sources:

- the design
- the development processes
- field experience

In considering the construction of a safety case, there are a range of options open to the designer at the preliminary design phase, as illustrated in Table 1 below. This is not a comprehensive list, but it serves to illustrate the basic approach to designing a system and safety case. Consideration of the readily-available evidence could have a strong influence on the economics of different design solutions.

Type of Argument	Implementation Options/Evidence		
	Development process	**System design**	**Field experience**
Fault elimination and quantification Maximising the probability of a "perfect" state	Procedures, Standards, Documentation, Config. control, Testing, Reviews, Design tools Formal methods	Design simplification Formal proof of system properties Use of standard components	Prior operating history as evidence of correctness Fault reporting, Design correction
Error activation Minimising OK→erroneous	Testing according to expected usage		Avoid changes in the usage Avoid known problem areas
Failure containment Strengthening erroneous → OK erroneous → safe		Fault Tolerant designs Fail-safe designs	Fault injection tests
Failure Estimation Estimating OK → dangerous	Reliability testing		Operational failure reports. Reliability growth models

Table 1: Arguments and Evidence

Given a list of possible implementation options, the designer then has to produce an overall system architecture which uses some cost-effective subset of these arguments. This may entail using different types of evidence for different components and may use different types of argument (e.g. fault elimination, failure containment and failure estimation). The safety case may also include diverse argument chains to allow for uncertainty. Some examples of different safety arguments are given in the following section.

Illustrations of the Safety Case Approach

The work on incorporating some current software concepts within safety cases has also been beneficial in the reverse direction. The structuring concepts in safety cases (e.g. those for dealing with residual doubt) can be equally beneficial to

software, especially where software is critical to the safety of the overall system. In SHIP the safety case approach was applied to both plant and software examples. Some examples of the different types of safety case for both plant and software-based systems are described below.

Nuclear Pressure Vessel

We examined a pre-existing safety case for a nuclear pressure vessel [Hirsch87] and found that the arguments could be mapped on to the proposed argument structure as shown below.

Transition	**Cause**	**Safeguards**
"Sound" → faulty	cracks grow due to normal ageing or abnormal transient	cracking minimised by: production processes, sound design, QA, avoidance of past problems detected by: pre-service tests, on-line inspections.
faulty → erroneous	crack grows large enough to leak	minimised by periodic inspection of vessel
erroneous → safe	reactor trips before the vessel fails	on-line water leak detection initiates trip
erroneous → dangerous	catastrophic failure of vessel	judged incredible

Table 2: Safety Case Arguments for a Nuclear Pressure Vessel

The safety case for the pressure vessel can be represented using the basic transitions of the model. The safeguards given in the final column show how the transition is minimised or eliminated. The top-level arguments are predicated on sub-claims that fast fractures cannot occur and that a vessel always leaks before it breaks. These sub-claims are supported by a large body of scientific evidence from fracture mechanics and metallurgy.

Errors in the argument can be tolerated because there are two forms of protection (periodic crack detection and on-line leak detection) either of which should be sufficient to maintain reactor safety. In addition there is a separate argument leg based on field experience where it is shown that the required level of reliability has been achieved on similar vessels in the past and that all known pressure vessel design flaws have been avoided.

Boiler System Control

The Boiler System Control Specification study [Bishop93] looked at the use of formal specification methods in constructing a safety case. In this example the claim is essentially a deterministic argument that design faults are absent, coupled with probability estimates for random hardware failures. A top-down approach is used where the boiler dynamics are formally modelled using Temporal Logic Algebra [Lamport91]. Boiler safety constraints were identified (which were basically that the water level had to remain within upper and lower limits). The control and safety functions were then modelled and shown to preserve the safety constraints. Successive design iterations were made which identified additional component failure modes. The final software design specification was shown to satisfy the safety requirements provided certain assumptions were made about the failure behaviour of the components and the diagnostic capabilities of the software. By computing the likelihood of violating these assumptions the dangerous failure rate can be estimated.

This is quite an effective and systematic method of eliciting the underlying assumptions, and also for deriving an associated fault tree for random failure probability calculations. There is no independent "second chain" which could protect against flaws in the assumptions, so ideally an independent safety system would be needed to cater for residual doubt. This illustrates that there is a duality between safety arguments and system architectures. If diverse systems are used, a single argument can be used for each one. If only a single system is used, diverse arguments are needed to support the safety claim.

Analysis of Industrial Controller Field Data

Field data can be used to provide supporting evidence of "perfection", or at least to give some lower bound on the expected level of reliability. This could be used either as the main safety case argument or as one leg in an overall safety argument. In SHIP we examined field data on industrial computer based control systems from both public and private sources. In one study of fault reports for a small industrial controller we observed the following pattern of fault discovery after a new industrial control system was released.

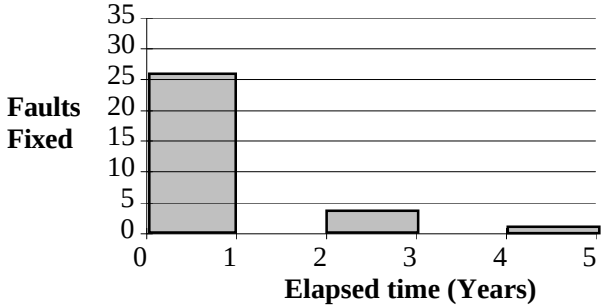

Figure 6: Fault Fixes over Time (Small Industrial Controller)

Faults reported by customers are recorded, and fault fixes are incorporated in later revisions of the design. This can happen several times in a year. It can be seen that in the second year no faults were fixed at all. Since these controllers are mass-produced with thousands being sold every year, this represents an extremely high reliability level. It is suspected that the subsidiary peaks in later years were actually due to *new* faults introduced when new features were added to the system. So it might be argued that after a year of fault fixing the design may be effectively "perfect".

When we looked at a large, complex, industrial controller, a different pattern emerged where there was little change in the fault fixes for successive years. We think this is due to a combination of factors. Firstly fewer units were sold so faults were not detected so rapidly in operational use. Secondly the system was more complex so it probably contained more faults. Finally there is more scope for introducing faults when new features are added.

A safety case argument based on observations of field reliability would therefore have to take into account the complexity of the system, the stability of the design (i.e. the rate of addition of new features) and the amount of field usage (which affects the rate of fault removal). A theory which models these effects is being developed in another research project [QUARC].

Summary and Conclusions

The SHIP project set out to improve the state of the art in assessing the safety of systems containing design faults. We think that we have succeeded to the extent that we have identified an overall safety case structure, and the different forms of argument and evidence that can be used. We have also shown examples of different forms of safety case that can be constructed based on both probabilistic and deterministic design arguments, and evidence from field experience. Other examples were developed in SHIP using a range of approaches (including qualitative methods), but there is insufficient space to present them in this paper.

Currently most safety-related software standards focus on a development process which will minimise faults, and this does not lead to a quantified reliability estimate. The safety case approach proposed in SHIP takes a more global view and provides a framework for estimating the safety and reliability of the software and the associated systems. More work is needed to make this work routinely applicable to industry, and further research into the application of these concepts is being undertaken [QUARC]. We expect to see the SHIP work influencing relevant industrial standards. Indeed, some of the safety case concepts developed in SHIP have already been utilised by the authors in producing the forthcoming revision to the Ministry of Defence standard for safety-critical software [MOD91].

Acknowledgements

The SHIP project (ref. EV5V 103) was carried out with financial support from the European Union in the framework of the Environment programme, sub-theme: Major Industrial Hazards.

We wish to acknowledge the contribution of the SHIP project partners to this work. The project partners are: Adelard, UK; the Centre for Software Reliability (CSR), UK; Objectif Technologie, France; Ente per le Nuove Tecnologie l'Energia e l'Ambiente (ENEA), Italy; Istituto di Elaborazione dell' Informazione (IEI-CNR), Italy; and the Finnish Technical Research Centre (VTT), Finland. Associate partners are: the Franco Polish School of New Information and Communication Technologies (Poland); and the Institute of Computer Systems (Bulgaria).

References

[Bishop93] P.G. Bishop, G. Bruns, S.O. Anderson, "Stepwise Development and Verification of a Boiler System Specification.", *Int'l Workshop on the Design and Review of Software Controlled Safety-related Systems*, National Research Council, Ottawa, Canada, June 1993

[Gentzen69] G. Gentzen, *The Collected Papers of Gerhardt Gentzen*, North Holland, 1969

[Hirsch87] Hirsch Study Group, "An assessment of the integrity of PWR Pressure Vessels", Addendum to the Second Report of the Hirsch Study Group, HL/087, 1987

[Hoare69] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, Vol. 12, No. 10, 1969

[Jones90] C.B. Jones, *Systematic Software Development using* VDM, Prentice-Hall International, London, UK, Second edition, 1990

[Lamport91] L. Lamport. "The temporal logic of actions", Technical Report 79, Digital Systems Research Center, 1991

[McDermid94] J.A. McDermid, "Support for safety cases and safety argument using SAM", *Reliability Engineering and Safety Systems*, Vol. 43, No. 2, 111-127, 1994

[MOD91] "The Procurement of Safety Critical Software in Defence Equipment", Ministry of Defence, Interim Defence Standard, IDS 00-55, 5 April 1991

[QUARC] Quantification of Reliability in Computer Based Systems, UK Health and Safety Executive Nuclear Safety Programme, Scottish Nuclear Contract: 70B/0000/006384

[Vesely81] W.E. Vesely et al., *Fault Tree Handbook*, NUREG 0942, US Nuclear Regulatory Commission, Washington DC 20555, 1981