SOFTWARE VERIFICATION RESEARCH CENTRE

SCHOOL OF INFORMATION TECHNOLOGY

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 99-46

A Process for Derivation and Quantification of Safety Requirements for Components of Complex Systems

Peter Lindsay, John McDermid, David Tombs

December 1999

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533

A Process for Derivation and Quantification of Safety Requirements for Components of Complex Systems

Peter A. Lindsay(1), John A. McDermid(2), David J. Tombs(1)

(1) Software Verification Research Centre, School of Information Technology, University of Queensland, Australia
email: {pal,tombs}@svrc.uq.edu.au

(2) High Integrity Systems Engineering Group, Dept of Computer Science, University of York, United Kingdom
email: jam@cs.york.ac.uk

Abstract

This report describes a formal approach to verification and validation of safety requirements for embedded software, by application to a simple control-logic case study. The logic is formally specified in Z. System safety properties are formalised by defining The paper develops a theoretical basis for assigning safety requirements for components of complex systems, including software, in a form that integrates well with established system-safety engineering processes. Safety requirements here include integrity targets, in the form of maximum acceptable failure rates. We propose a process for flowing safety requirements down to components, by integrating system hazard analysis techniques in a manner that enables functional and quantitative analysis. The originality of the process lies not so much in the individual steps as in the integration of their results within a single coherent framework.

In outline, the steps of the proposed process are:

1. Definition of an appropriate system conceptual design model.
2. Identification and classification of “system hazards”: roughly, states of the system that can contribute to accidents and mishaps. Functional Failure Analysis is the suggested technique.
3. Identification of protective measures and system safety functions, by consideration of possible accident sequences arising from unsafe actions and possible component failures. Event Tree Analysis is the suggested technique.
4. Construction of a detailed cause-and-effect model, to record how faults propagate through the system, and to identify possible common causes. Fault Tree Analysis is the suggested technique.
5. Allocation of functional and quantitative safety requirements, using the models from previous steps to justify quantitative targets.

    A simple running example is used to illustrate the concepts, terminology and features of the process. The terminology of IEC 61508 is used where possible.

Keywords: safety integrity, system hazard analysis, risk assessment

1. Introduction

1.1 Complex critical systems

Information Technology is enabling the creation of new systems, and integration of existing systems into ever-more complex systems of systems. Examples of complex, critical computer-based systems include: command and control systems in defence applications, wireless distributed control systems in the power and process industries, electronic flight-strip systems in Air Traffic Control, proactive signalling systems (e.g. Automatic Train Protection) in railways, and GPS-based systems in emergency-services dispatch systems. As a result of the increased complexity, design errors – which includes all software errors – are increasingly coming to dominate physical failures as the cause of system failures [12], [18].

System safety standards such as IEC 61508 [10] use the concept of “safety integrity level” to capture the fact that some system components are more relied on than others for safety and risk reduction. However, the definition of safety integrity differs widely between standards and, as we have argued elsewhere [13], the variation undermines the credibility of the concept.

This paper takes the broad, simple view that component safety integrity has two fundamental aspects:

1. a *functional* aspect, relating to what the component is supposed to do or not do; and
2. a *quantitative* aspect, relating to the degree to which the system can tolerate failure of the function and yet operate with acceptable risk.

This view is explained in more detail below. We introduce the term *safety integrity requirement* for a functional safety requirement together with a quantified (maximum acceptable) failure rate for that function.

The purpose of the paper is to develop a conceptual framework for analysis of system risk due to component failure, and to describe a process for derivation of safety integrity requirements for components of complex systems. The focus here is on safety, but the approach is expected to generalise beyond safety to other dependability attributes. Similarly, the focus is on software requirements, but the approach extends to other technologies and to operational procedures.

The idea is to assume by default that complex components can fail, due to causes that may not necessarily be known in advance, and to try to ensure that the overall risk of system failure is tolerable. The paper develops a process that integrates system hazard analysis techniques so as to support “integrity allocation”, based on quantitative analysis of the contribution of component failures to system hazards and accidents. The framework makes it possible to set out a “budget” for component failure rate targets and, assuming the components meet their targets, to assess whether the system will operate with tolerable risk.

The focus is on system safety engineering, and how to structure the argument that a given system-as-designed will operate within tolerable risk levels, given certain assumptions about component failure rates. The originality of the approach lies not so much in the individual steps of the process, as in the integration of their results within a single coherent framework.

The scale and complexity of many systems makes it impossible to model them at the highest level of detail, or to describe all conceivable accident scenarios. This is especially true during design, when failure analysis and budget setting takes place. Our approach facilitates progressive modelling of the system and enables designers to focus on the highest areas of risk. Leveson [12] notes that calculating failure rates to high degrees of precision while ignoring the structure of the safety argument can be dangerously misleading. In our process, numbers are used to support the argument, rather than as ends in themselves.

The paper does not address how to assess the failure rate achieved by particular technologies, including software. IEC 61508 and other system safety standards establish guidelines for the level of management and technical rigor to be applied in development, analysis and assurance of software for different safety integrity levels. We note, however, that current standards quote widely differing processes, and there is little agreement what is the “best” process. We believe that this “process prescription” is inappropriate, and it is better, instead, to seek to supply evidence of safety which is as objective as possible. Treatment of such issues is outside the scope of this paper. However it is clear that, since safety is an important cost driver in system and software development, integrity targets are a vital input to scoping, costing and planning of development activities, and integrity assessment is an essential process for optimising software engineering quality and productivity.

1.2 A conceptual framework for risk modelling

The approach is based on modelling of the target system as a collection of components whose functions combine to achieve system functions. The term component is used here in its broadest sense, and may include anything from major subsystems through to operating procedures. No assumptions are made here about the level of granularity of components and component functions. In theory the approach could be taken down to the level of different software modules running on the same processor for example (see [15] for an analysis of software partitioning). In practice however the level will be determined by the extent to which independence of component failures can be established.

The paper combines into a single framework the risk-analysis features from a number of established techniques, including Fault Tree Analysis [20], Failure Modes and Effects Analysis [20], Event Tree Analysis [12], and Computer Hazard & Operability Studies [7]. The framework covers the following factors:

- External and non-software failures, including hardware failures and operator errors
- The possibility of detection of component failure and the effectiveness of mitigation
- System and environmental co-effectors, including external risk mitigations
- Fault propagation mechanisms and (conversely) protection mechanisms
- Common mode failures

The aim of the framework is to enable software functional requirements, and software failure modes, to be classified according to their criticality in the target system, and to provide traceability of safety features and risk reduction mechanisms.

The approach is thus expected to improve targeting of assurance (V&V) effort in system and software development by focussing on critical requirements. It provides the ability to assess robustness of system design with respect to software failures, and to answer questions such as “what are the system-risk implications if a given component fails in such-and-such a way and at such-and-such a rate?” It provides a foundation for software assessment techniques to be better integrated into quantitative system risk assessment. It also provide a basis for comparing designs and for making trade-offs in design.

The modelling framework is sufficiently rich to capture safety issues for a wide variety of systems and component types – including hardware, software and operational procedures – and does not presume any particular safety architecture.

1.3 A process for integrity allocation

We have previously suggested that Cause Consequence Analysis (CCA) [9] should be used to analyse such situations and to set safety targets [13]. CCA is a combination of FTA and ETA that enables both the causes and consequences of hazards to be investigated within a single framework. Thus CCA potentially gives a way of setting derived requirements for mitigating functions, as well as on equipment that can cause hazards. The purpose of this paper is to refine this concept, and to supplement the rather general recommendations presented in the earlier paper [13] by defining more precisely a process by which safety requirements and targets can be derived using a CCA-like approach. The body of the paper is written in the form of a "top-down" iterative process for allocating and refining safety integrity requirements from system level down to component level. The terminology of IEC 61508 [10] is used where possible.

In outline, the steps of the process described in this paper are:

1. Construction of a System Conceptual Design model which defines the scope of the system and the functions of its components (Section 3).
2. Identification and classification of "system hazards": roughly, states of the system that can contribute to accidents and mishaps (Section 4). Functional Failure Analysis is the suggested tool for this step.
3. Identification of protective measures and system safety functions, by consideration of possible accident sequences arising from unsafe actions and functional failures (Section 5). Event Tree Analysis is the suggested tool for this step.
4. Construction of a detailed "cause and effect model", to record how faults propagate through the system from component level to system level, and to identify possible common causes (Section 6). Fault Tree Analysis is the suggested tool for this step.
5. Allocation of a "budget" of safety integrity requirements, using the models from previous steps to justify quantitative targets (Section 7).

In simple terms, the steps address respectively: how does the system work; what can go wrong; how might it go wrong; why might it go wrong; how likely is it to go wrong. No assumptions are made about when in the system development life-cycle this process would be applied.

A simple running example is used to illustrate the concepts, terminology and main features of the process. However, because of space limitations, the example is very simple, and to avoid any possible confusion it is important to be clear about the exact purpose of the example. It is not intended to be representative of the complexity of the systems we are targeting: in fact, the process will seem like overkill for the example. Nor do we claim that it is a safe design. Similarly, it does not illustrate the process in full: for example, completeness of coverage is an important goal of the process that cannot be demonstrated here, even on this simple example. However, the example includes sufficient features to illustrate many of the subtleties of analysis.

2 Background and Literature Survey

2.1 System safety

Before introducing a working definition of safety integrity, this section briefly outlines the basic precepts on which the paper is based. System safety engineering is concerned with assuring that systems will operate with tolerable risk. System risk is typically quantified in terms of severity and likelihood of mishaps. In some approaches the length of exposure to the hazard is separated out as a third dimension of risk, but in many cases it can be treated as an aspect of severity or likelihood.

Safety integrity is a system-specific, function-specific notion. A component used in one system may have quite different safety integrity requirements when used in a different system. Moreover, allocation of safety integrity is a system design and engineering issue. Typically a system is designed to operate with risk “as low as reasonably practicable”. Some technologies are more reliable than others but also more costly, especially when measured in whole lifecycle terms, including the cost of assurance and maintenance [17]. It is usually not feasible, or justifiable, to engineer all parts of a complex system to the highest reliability standards. Thus allocation of integrity is an engineering compromise between cost, functionality and reliability.

In developing safety-critical systems one of the important early activities is the derivation of safety requirements, based on the results of system hazard analysis. It is common to derive two different sorts of safety requirement, as we have to address two classes of failure mechanism: *random failure*, arising from physical causes such as mechanical breakdown, and *systematic failure* arising from design error. Safety integrity requirements for protection against random failures are usually handled by setting failure rate or probability targets for hardware components. Protection against systematic failure is usually addressed by defining the level of rigour to be applied in analysing, developing and assuring the design and implementation; failure rates cannot normally be assigned since there is no established manner of assessing them.

2.2 Related work

Few attempts have been made to develop a generally applicable theory for the establishment of system safety requirements. The literature has many examples of specific analysis methods, but very little on their assembly into a coherent and credible safety assessment process. Many international and sector-specific standards provide guidance on derivation of safety requirements [23], which is almost entirely based in engineering experience rather than academic theory.

A number of texts exist on system safety engineering and its older cousin, reliability engineering. Leveson [12] covers safety management issues and identifies the characteristics that distinguish safety from related engineering disciplines such as reliability. She presents an overview of safety management processes, discusses the principles of hazard and risk assessment at an abstract level and introduces various hazard assessment techniques and their pros and cons but does present a methodology for their use.

Henley [9] outlines a safety process with phases of system definition, preliminary hazard analysis (meaning Functional Failure Analysis, with emphasis on intra-system interfaces), accident sequences (FTA and ETA) and consequence analysis (FMEA, HAZOP, CCA, Markov modelling and statistical measures). There are good descriptions of these techniques, but the overall process is not justified and there is no means of apportioning risk.

Bayesian Belief Networks [13] provide an alternative method for describing and quantifying failures of a system. Application of safety techniques to different domains of dependability (reliability, maintainability, etc) is discussed in [8] and [19]. Finally, a number of studies have been made on particular systems, including a demonstration mine fuze [24], a mass-transit railway [26], and a model of an industrial press, adopted in this paper [1].

2.3 System safety standards

IEC 61508 (Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems) [10] is a generic international safety standard that covers all aspects of safety development, including management, requirements, design, verification and validation. It is intended to act both as a general industry-independent framework for functional safety to be tailored to specific industries, and as a stand-alone standard.

The IEC 61508 approach is based on separating a Safety Related System (SRS) from the Equipment Under Control (EUC). This is a common approach in the process industries, where the EUC carries out a process and the SRS monitors the process. Two modes of SRS operation are considered: continuous and on-demand. (An example of the former is a continuously varying flow control system, and of the latter is emergency shutdown system.) The IEC 61508 definition of safety integrity level is “a measure of the likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time”. The standard places limits on the integrity that can be claimed for the EUC, and concentrates on safety development of the SRS. Based on how far EUC risk must be reduced, a qualitative Safety Integrity Level (SIL) is derived for the SRS, ranging from 1 (low) to 4 (high).

The standard does not consider other safety architectures and does not adequately address the residual risk of a partially complete design. That is, there is no uniform way of balancing achieved risk against target risk and apportioning the residue to software and hardware elements. The methods of IEC 61508 work best when there is a clear separation of control from protection, when the hardware architecture has been finalised, and when component failure rates are known. Little guidance is provided for other situations.

Aeronautics industry standard SAE ARP 4754 [21] defines safety requirements and processes for civil aircraft and airborne systems. SAE ARP 4761 [22] is a related standard that provides guidance on implementation of the processes. Much of the latter document is devoted to a substantial worked example (a wheel braking system) using fault-trees, Markov models and dependency diagrams. The Preliminary System Safety Analysis (PSSA) part of their process is the closest equivalent to the safety process outlined in this paper. A previous study [4] has observed that a proper PSSA is difficult to perform effectively because it analyses the design as it simultaneously derives requirements on it. It also points out apparent inconsistencies in the worked example, where dependencies have been overlooked.

US defence standard MIL-STD-882C [16], UK defence standard Def Stan 00-56 [6] and Australian defence standard Def(Aust) 5679 [5] describe similar collections of activities to be undertaken during development of a safety system. The activities known as Preliminary Hazard Analysis (PHA) and System Hazard Analysis (SHA) are closest to the process in our paper. 882C has a phase known as Safety Requirements Criteria Analysis, which establishes safety requirements on components within a specified risk framework but does not describe technical processes to achieve its goals. 00-56 demands risk estimation during PHA and later during SHA but provides little guidance on what is expected of the earlier effort, where we would expect budgeting to happen. Risk is partitioned into systematic and random aspects at an early stage and integrity levels are derived separately for each aspect. 00-56 suggests that the PHA should be performed through a HAZOP (Hazard and Operability Study), according to Def Stan 00-58 [7]. 5679 demands a safety integrity assessment be undertaken and that SILs are assigned to component safety requirements according to the “level of trust” allocated to them at system level. Levels of trust are allocated according to the severity of the outcome of a system failure and the likelihood it will lead to an accident. While perhaps closest in spirit to our approach, 5679 does not countenance quantitative arguments for system safety design features.

Railways industry standard CENELEC ENV 50129 [3] treats systematic failure differently from random failure and also continuous and demand modes of operation. ENV 50129 admits its tolerable failure rates are based on speculation, although they are perhaps more realistic than IEC 61508 for continuous mode.

2.4 Tool support

It is difficult to assemble and manage a safety argument for a large, complex system without tool support. At present there exist general requirements management and traceability tools (e.g. DOORS, CORE, Statemate) and editors for individual methods (eg. FTA editors, or the SafeCalc tool which calculates SILs for an IEC 61508-style protection system). The Goal Structuring Notation (GSN) [25] structures a safety argument and assemble evidence from diverse sources. Kelly [11] has developed a method to evolve a GSN description as the system design evolves.

3 System Conceptual Design

The first step in our safety process is to develop a suitable model of *System Conceptual Design (SCD)* as an initial basis for integrity allocation. The model should clearly define system scope and operational concept, and describe system functionality at a suitable level of abstraction. It should identify the main system components and their functions, and explain how system functions are allocated to components (sometimes known as the system conceptual architecture).

3.1 Choice of model

For the purposes of the safety process, the SCD model should meet the following criteria:

1. The relationship between the system and the environment is defined sufficiently clearly that it is possible to identify who is at risk from the system, and how they might be affected by its failures (i.e., the system boundary is delineated).
2. The operation of the system is defined sufficiently clearly to indicate how normal or abnormal operation might cause system failure, and how the consequences of system failure depend upon the operational state or mode.
3. The system's design is modelled sufficiently accurately (although not necessarily in great depth) to identify the major components that may fail, and how their failure can affect the rest of the design. The model is sufficient to identify chains of cause and effect that represent the failure behaviour of the system.
4. The system and its interaction with the environment are modelled sufficiently accurately that it is possible to identify which mitigation functions may be triggered by a failure. The model is sufficient to identify chains of cause and effect that represent the failure and recovery behaviour of the system.

The model should not be too detailed, for manageability and breadth of coverage. Functions chosen must not be too deep. There is an issue of judgement as to the inclusion of human operators in the model.

Ideally the model should be produced by the designers; however our experience is that it may be necessary to refine existing design models to enable effective hazard and safety analysis. If this is done, the model should be referred back to the designers and ideally adopted by them; at minimum it should be reviewed and accepted by them. A diagram that indicates clearly the system boundary, its major components and their interconnections is most helpful.

3.2 Running example: an Industrial Press

The (hypothetical) example concerns a large industrial press such as might be used to mould body parts for motor vehicles [1]. The press consists of a heavy (50 tonne) "plunger" which is raised seven metres above the "press bed" by means of a hydro-mechanical plunger-drive mechanism (see Figure 1). Locks hold the plunger at the top with the assistance of the plunger-drive force.

An operator loads sheet metal (an "unformed work-piece") onto the press bed, and then moves away to the press-control panel and pushes a button which causes the press to close: the plunger is released and falls to the bottom under gravity, thereby pressing the work-piece into shape. The press then reopens, by means of the plunger drive. As a safety feature, the press includes a physical guard, which blocks operator access to the press bed while the plunger is falling. Finally, at times the operator may decide to abort operation prematurely in order to adjust the work-piece: the plunger drive activates to slow and reverse the plunger's fall and return it to the top. That completes our brief operational description of the system.

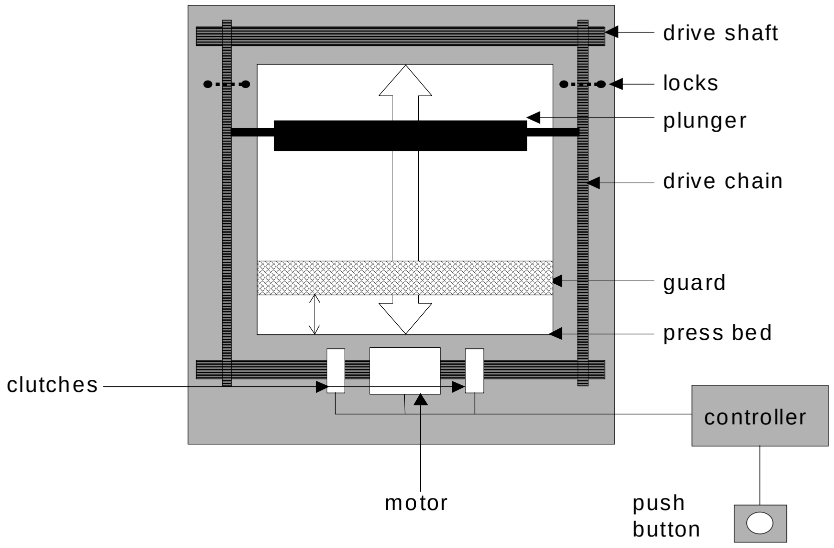

Figure 1 Hydro-mechanical structure of the Press

An abstract SCD model will be developed by fleshing out system design detail below. Recall that what follows is for illustrative purposes only, and is not necessarily the most appropriate design for such a system.

Four main press-operation phases will be considered here: Open (press is held fully open while the operator loads and unloads work-pieces); Closing (press is closing); Opening (press is reopening after closing) and Abort (press is reopening after closing is halted prematurely). A full safety analysis would also consider phases for start-up, emergency shutdown, maintenance, and so on.

The operator will not be considered to be part of the system, except in as much as he/she initiates the 'close press' and 'abort closing' system functions, and his/her location is important in scoping the hazards. We note that, in a full analysis, human-factors issues would have to be considered in more detail, but space doesn't permit us to discuss the details here.

The main system functions for the four phases are as follows:

- Open: the plunger is held at the top.
- Closing: the guard closes and the plunger is released and allowed to fall.
- Opening: when the plunger is at the bottom, the guard opens and force is applied to drive the plunger back to the top.
- Abort: force is applied to the falling plunger to slow and reverse its descent, and the guard opens.

Further detail will be introduced when needed below.

4 System Hazard Identification

4.1 *Accidents and system hazards*

The next step in the process is to identify and categorise hazards and accidents. For our process it is important to distinguish between events which are 'internal' to the system and those which are external, using the system scope defined by the System Conceptual Design model.

We define a *system hazard* to be a system condition or state that may contribute to an accident. For our purposes a system hazard is a system failure mode or state of the system, with all contributing system attributes noted – roughly, the "event" where the chain of cause and effect crosses the system boundary. A system hazard is thus "larger" than a component failure but "smaller" than an accident. For our running example, "operator's hand gets crushed in the press" would be termed an accident, whereas "press closes while the guard is open" is an example of a system hazard. This concept of hazard is consistent with the majority of standards but they sometimes use different terminology. For example, IEC 61508 defines a hazard as a "potential source of harm" where harm is physical injury, and Def Stan 00-56 defines it as "a physical situation, often following from some initiating event, that can lead to an accident".

Returning to the process description, methods for identifying possible accidents are commonplace (see e.g. IEC 61508 and Def Stan 00-56). Typically they involve brainstorming and the use of checklists and historical data, such as knowledge of accidents and incidents that have arisen with this sort of machinery previously. Sources of hazard might include energetic materials (explosives and fuels); corrosive materials; electro-magnetic radiation; chemicals and bio-chemicals; electrical power; heavy machinery; and simple things such as obstacles in the working environment.

Identification and classification of system hazards is central to our process. Because they lie on the system boundary, system hazards represent the limit of what can be controlled and are thus the natural place to set and agree tolerable-risk targets. In a full safety process, the system hazard list would form the basis of the hazard log through which subsequent analysis and resolution of hazards would be tracked. The system hazard list should thus be as complete as possible. It is important to keep the interpretation of system hazards broad, to cover as many "reasonably foreseeable" circumstances as possible, including those for which causes might not be immediately apparent.

For manageability however, it is important to group together system hazards having similar causes and outcomes, and to make the system hazards as clear, concise and orthogonal as possible. For the running example, we would probably thus break the "press closes while the guard is open" hazard down into a number of separate system hazards, such as "plunger released before reaching top (and guard open)" and "guard opens early (before press closes)". In the first case, the operator could already have their hands in the press and would not be expecting the press to close; the cause of the failure is likely to be plunger-related. In the second case, the operator probably won't have crossed from the control panel to the press bed, and will be expecting the press to close; the failure cause is likely to be guard- or controller-related. The two hazards thus have quite different accident exposures and quite different causes.

At this stage in the process it's important to generate as complete a list of hazards as possible. Section 5 describes a way of refining the hazard list and hazard classifications.

4.2 *Functional failure analysis*

The suggested tool for system hazard identification is *Functional Failure Analysis (FFA)*. FFA is not currently well described in the literature ([22] has an example) but is fairly easy to explain. In FFA the system is considered as a collection of functions, which are each analysed in turn to identify their possible failure modes (ways in which they can fail) and how they contribute to hazards and accidents. In the absence of more detailed information about how functions might fail, it is possible to postulate general modes of failure by using the following "general guide-words":

function not provided; function provided incorrectly; and function provided when not intended.

In our approach, FFA is undertaken for each of the component functions identified in the SCD model. For each functional failure mode in turn, analysts:

- describe the “local effect” of the failure (i.e., how it manifests itself);
- identify any coeffectors (internal or external to the system) needed for a hazard to arise, including the corresponding modes of system operation if appropriate;
- describe the “system effect” of the failure and identify any hazard arising.

It is also common in FFA to describe any controls and protective measures that prevent the functional failure propagating, but these will be treated separately in our process (see Section 5). In the course of the FFA it may be necessary to refine the SCD model.

On completion, the FFA is inspected to identify which events lead to each hazard, and a list of system hazards is extracted. System hazards are generalised and grouped according to the nature of their outcomes and the functional failures that give rise to them. This stage in the process aims to produce a first-cut list of system hazard classifications.

As the running example will show, successful application of FFA relies critically on domain expertise.

4.3 Running example

This section illustrates application of FFA to the system functions identified in Section 3.2. Table shows part of the output of an FFA for the “apply upwards force” function of the plunger-drive mechanism. The general guide-words have been applied to help structure the analysis, together with consideration of the different operational phases of the press. Note that the “not provided” guide-word has been interpreted to include “insufficiently provided” in this case, since the effects are the same.

The PONR concept needs explanation. Recall that press operation can be aborted by activating the plunger drive to slow and reverse the plunger’s fall. Taking the plunger’s momentum into account, however, there will be a point after which it is ineffective – or even dangerous – to try to stop the press closing. This point is referred to as the *Point of No Return (PONR)*. When the plunger is falling from the top, activating the plunger drive beyond the PONR will slow the press’s closing and increase the operator’s exposure to risk, by giving them longer to cross the floor from the control panel to the press bed. It may also have other undesirable or unsafe effects, such as damaging the Plunger Mechanism or leading to wastage of work-pieces.

For the rest of the paper, the SCD model will be refined in various ways. The Closing phase of press operation will be split into two, according to whether the plunger is above or below the PONR. The “abort closing” system function will only be available in the above-PONR subphase of Closing.

Guide-word	**Phase**	**Local Effect**	**Coeffectors**	**System Effect**
Function not provided	Open	Plunger falls unexpectedly	Guard open	Press closes unexpectedly & guard is open
	Open	Ditto	Guard closes & traps operator’s hand	Press closes with operator’s hand trapped
	Closing	N/a		N/a
	Opening	Plunger falls unexpectedly	Guard open	Plunger released before reaching top
	Abort	Plunger falls to bottom	Guard opens	Press closes (& guard opens) despite abort
Function provided when not intended	Closing	Plunger descent slowed but not stopped	Plunger beyond PONR (see explanation)	Plunger drive activated while plunger falling beyond PONR
	Others	N/a		N/a
Function provided incorrectly	All	Plunger drive applied unevenly		Plunger falls asymmetrically

Table 1 FFA for “apply upwards force” function of the Plunger Mechanism

As noted above, the main accident to be avoided is the press crushing the operator’s hands, although it may also be appropriate to analyse accidents associated with the guard or plunger drive. At the broadest level, the most important system hazard is if the press closes while the guard is open. By applying an FFA to the component functions identified in the SCD model in Section 3.2, it is possible to break this down into more specific categories of system hazard. One possible categorisation of (some of) the system hazards is the following:

H1. Press closes without operator command. This hazard covers both the first two cases shown in Table 1. We’ve chosen to group these together because the press closing unexpectedly is a fundamental failure of the system that just shouldn’t happen, whatever the state of the guard.

H2. Plunger released before reaching top. This has been separated from H1 since the locks are not a mitigation in this case. The important difference between H1 and H2 is the operational phase involved (Open versus Opening).

H3. Press closes (and guard opens) despite abort. This case has been separated from the rest since the operator' s exposure is significantly different.

H4. Plunger drive activated while plunger falling beyond PONR. This hazard is a possible coeffector for H3, but may have other outcomes - including longer-term effects as noted above - and so is treated separately.

H5. Plunger drive applied unevenly. Again, this hazard has been separated from the rest since it is essentially a different kind of mishap.

Although far from complete, this list illustrates some of the considerations underlying identification and categorisation of system hazards.

5 Protective Measures and Safety Functions

In the next step of the process, we undertake a more methodical, structured analysis of system hazards. The purpose is to analyse how functional failures contribute to accidents and to identify protective measures in the design. The suggested tool for this step is *Event Tree Analysis (ETA)* [12]. Other forms of consequence analysis could be substituted (e.g. risk graphs) but ETA has the advantage of being more structured and amenable to quantitative analysis.

5.1 Event Tree Analysis

ETA is a graphical technique for analysing and recording "accident sequences" – sequences of events from some "initiating event" through to particular outcomes. Initiating events can be external events (such as human actions) that might lead to an accident, or technical failures of the system. Branch points in event trees correspond to possible interventions by the system to control or mitigate the effects of the initiating event, and to different possible outcomes due to coeffectors. For completeness of analysis, event trees also consider the possibility that protective measures might fail. ETA supports quantitative analysis of the likelihood of accidents, given the initiating event, by quantifying the effectiveness of the mitigations (see Section 7). The reader is referred to [20] for a fuller discussion of ETA; here we just describe how it would be applied as part of our integrated process.

Event trees could be developed for many different initiating events: our process chooses discrete events where the system meets its environment, including the human actions. More specifically, three different kinds of initiating event are treated:

1. Events associated with the operator or other personnel in each operational phase which require the triggering of a protection mechanism to avoid a hazard;
2. Failure events associated with the system in each operational phase which require the triggering of a protection mechanism to avoid a hazard;
3. External events which may be hazardous in any or all phases of operation, and which may be common cause failures, e.g. power loss.

The above may lead to some redundancy in analysis, such as common-cause events that also appear in fault trees during causal analysis (Section 6 below). However this should not require much additional work, and provides a crosscheck on the completeness of the analysis.

In constructing event trees we can only ensure good coverage of the design and operating regime by ensuring that we take the trees through to accidents, not just hazards. In this way we should be able to see how mitigating mechanisms prevent accidents, and the effects of their failure. Initiating events and failures of protection may themselves have diverse causes, which are the subject of FTA, the next step of the process.

The ETA will often result in a refinement of the SCD model to include functions which implement protective measures (sometimes called safety functions) and interlocks (constraints between functions in different components). Thus for example the locks which hold the plunger at the top perform a safety function by protecting against failure of the "apply upwards force" function. An example interlock is a mechanism, independent of the Controller, which keeps the locks closed until the guard is closed.

As a further completeness check, each hazardous functional failure identified in the FFA should appear somewhere in the ETA, either as an initiating event or as a failure of a protective measure. A general safety principle is that there should be no "single point of failure". Thus for example locks should be designed to require energization before releasing the plunger: that way, if power fails and the upwards force consequently fails, the locks provide an independent safety function. The ETA thus aids in identifying functional safety requirements, both of failure prevention ("non-initiation") and failure detection and control ("mitigation").

5.2 *Running example*

This section gives illustrative event trees for the running example. Figure 2 gives the event tree for the "operator aborts press operation while plunger is falling above PONR" initiating event. This is an example of a normal event requiring protective measures, since the operator will cross the "exclusion zone" between the control panel and the press to readjust the work-piece. Under normal circumstances, the operator would be protected by the following features of the overall system design:

- the guard is normally closed while the plunger is falling;
- upon aborting, the plunger drive activates and stops the press closing;
- the control panel is placed sufficiently far away from the press to allow the press to close (if all else fails) before the operator has time to cross the zone;
- finally, the operator is trained to be wary of abort failing (in particular they know that the abort option is not available after PONR).

Note that activation of the plunger drive and prevention of press closing have been treated as different events, since for example late activation or insufficient drive force might mean the press closes anyway. Note also that there is no implication that events in the accident sequences are causally related or temporally ordered. Finally, note that the system is treated as a whole, rather than separating it into equipment-under-control and a safety-monitoring system as in IEC 61508.

Figure 3 gives the event tree for the "application of upwards force fails while Open" technical failure of the Plunger Mechanism. As noted above, the locks are the primary protective measure in such a case. Other mitigating circumstances are that the operator's hands are in the press for only a short part of this phase, and that there is a chance the operator will notice the plunger falling and avoid it. (Note however that the environment in which the press operates might be too noisy to rely on the latter.)

5.3 *Outputs of the step*

We define an *accident cutset* to be a set of events that can lead to a particular accident, including the initiating event, component failures, and internal and external coeffectors. Accident cutsets can be read off from event trees. Thus for example, an accident cutset for "hands crushed in press" is the following set of events from Figure 2: operator aborts operation while plunger falling above PONR; guard opens; plunger drive does not activate; operator crosses zone before press closes; operator puts hand in closing press.

Where there is only one accident outcome in the event tree, it is a simple matter of following the "path" through the tree to obtain a cutset. Where the same outcome can occur in more than one way, then it may be necessary to simplify and rationalise the cutsets. For example, if an accident arises whether or not some mitigating event occurs, then this event should be removed from the cutset: the associated mitigating function has no effect, with regard to that particular accident. (The distinction may however become important at component level – but that's left to later in the process.)

The completeness and appropriateness of the system hazard list generated in the previous step of the process can be checked as follows: For each accident cutset it should be possible to map the set of system-related events to an individual hazard. Thus for example, the system events for the aforementioned accident cutset (abort above PONR, guard opens, plunger drive does not activate) correspond to the "Press closes (and guard opens) despite abort" hazard (H3). Note that the other cutset for the same tree also maps to H3, yet represents quite a different set of events. This suggests that H3 should be split into two distinct system hazards - say H3A “plunger drive fails to activate upon abort and guard opens” and H3B “plunger drive fails to prevent press closing after abort”.

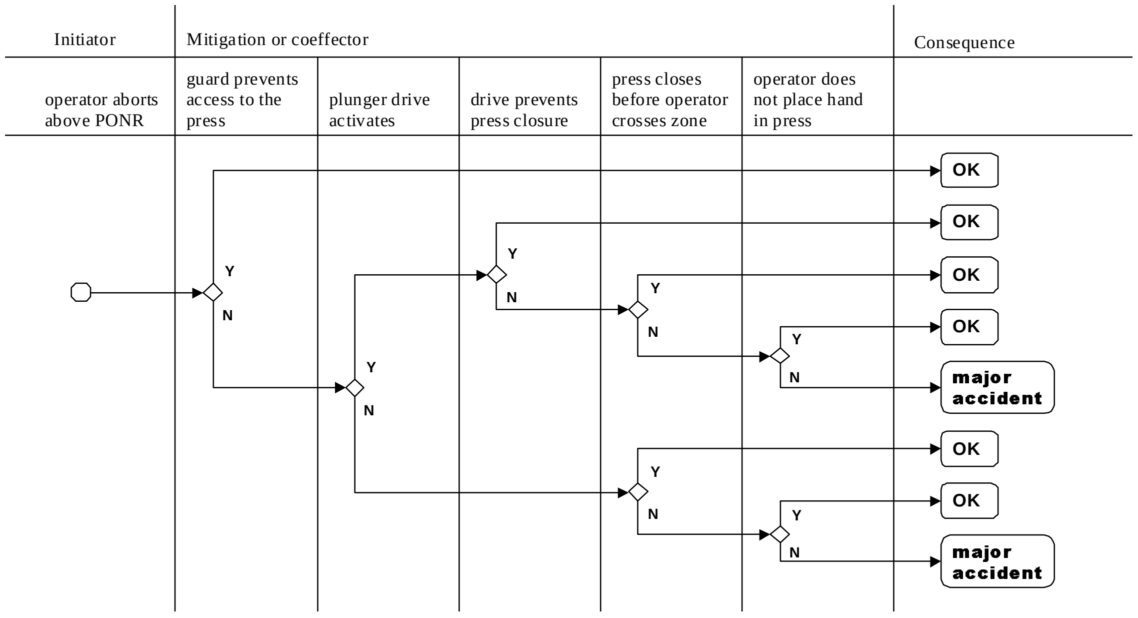
Initiator
Mitigation or coeffector
Consequence
operator aborts above PONR
guard prevents access to the press
plunger drive activates
drive prevents press closure
press closes before operator crosses zone
operator does not place hand in press
Y
N
OK
major accident

Figure 2 ETA for human initiator – operator aborts above PONR

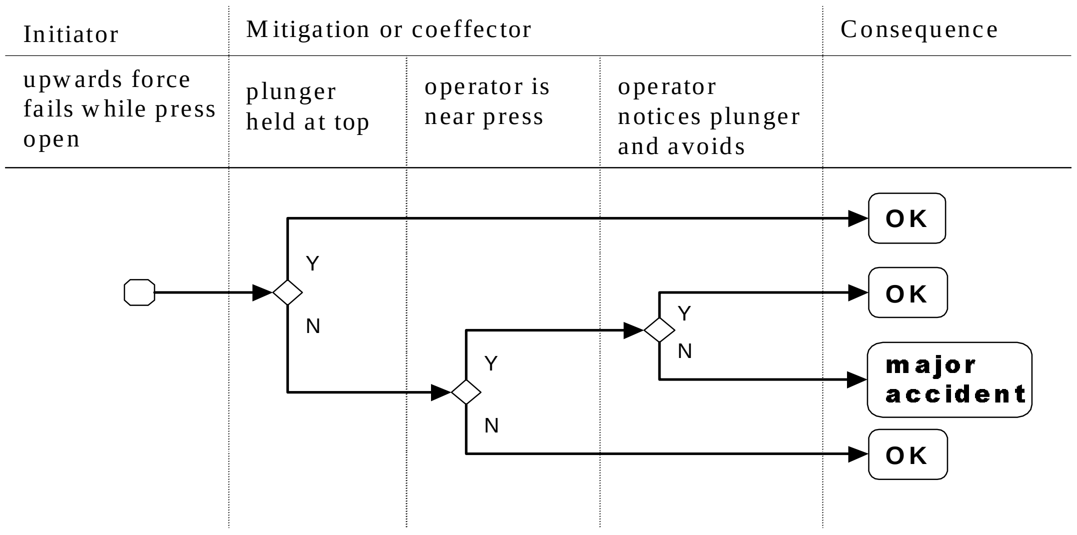

Figure 3 ETA for functional failure – upwards force fails while press open

The ETA leads to identification of safety requirements of the design, by way functionality which implements protective measures. The target allocations at this stage are made to functions, not design components. In some cases it will be straightforward to map functions to components, but this may be relatively uncommon when working at the level of event trees. Ultimately we shall allocate derived safety requirements to components, but in general this will wait until we have carried out fault tree analysis.

The main outputs of this step are: a refined SCD model with safety functions noted; accident cutsets developed to level of system functional failures; a first-cut “cause & effect model” of how system functional failures contribute to accidents.

6 Fault Propagation & Dependent Failures

The next step in the process is to refine the SCD model to component level and to trace the cause of system functional failures down to the level of independent component failures. The analysis follows on from ETA and accident cutset determination, and determines how failures in individual components contribute to the accident sequences previously identified.

Functional failures, such as identified in FFA and ETA in preceding steps, can have complex causes. While many system functions appear to be primarily assignable to specific components, closer inspection usually reveals a more subtle situation. For example, in the running example, the “apply upwards force to plunger” function is primarily the responsibility of the plunger-drive mechanism, the controller plays a role in activating the plunger drive, and it in turn relies on inputs from sensors and commands from the operator via the control panel. A failure of the "apply upwards force" function could be due to a fault in any of these components, or some combination of failures.

Some form of *causal analysis* needs to be undertaken to identify and record the logic of fault propagation, and to break high-level functional failure into “basic events” (component failures and coeffectors). A number of different causal analysis techniques could be applied at this step. We have chosen to illustrate the step using *Fault Tree Analysis (FTA)* since it is widely known and used and can be applied quantitatively, but other techniques could be substituted (e.g. Markov Analysis).

6.1 Fault Tree Analysis

Fault Tree Analysis is a well-understood graphical technique that has been in common use in safety engineering over many years. A description of the technique is found in [20]. Here we simply describe the application of FTA in our process. (Quantitative analysis is deferred to Section 7.)

The step involves applying causal analysis on the functional failures identified during event-tree analysis that play a part in accident cutsets. This means revisiting the System Conceptual Design model developed earlier, and if necessary refining it or developing a component-based alternative to build foundations for our trees.

The role of the component-level system design model is to clearly identify system components and their interfaces. The level of detail in the model determines just what constitutes a basic event. As usual, we are not prescriptive about what level of detail should be chosen: many factors would influence the choice, including the system lifecycle stage and available design detail. To some extent the focus of the model will be determined by the overall safety argument being developed, with emphasis on those aspects of the system which are most critical or which are being relied on most for risk reduction. As usual, it may be necessary to revisit the model at later stages in our safety process, for example when justifying allocation of integrity targets.

A fault-tree analysis would be applied to each hazardous functional failure identified in the ETA. Each fault-tree is developed by breaking down the failure (the so-called “top event”) into component faults and coeffectors. Frequently, a fault in an operational chain of components causes the failure, e.g. “X sends the wrong signal, causing Y to misfire”. Thus although the immediate cause of the functional failure might be the fault of Y, Y might have been behaving to specification, and the root cause might have been a failure of X or some earlier component in the chain of cause and effect. Fault trees are developed by working backwards through the chain of “intermediate faults” until basic events are reached.

We require that basic events be independent as far as possible, and fully traced back to root causes. The goal is to reveal all possible failure mechanisms, and to identify causes which might shared by different failures. Determining independence can be hard, and it is usual to make reference to a model of the physical architecture here. For software components, it can perhaps be achieved by separating common-mode software failures (e.g. due to processor failure) from software functional failures. If no clear argument can be given for independence, two similar events should be assumed to be the same event (i.e., assume dependent until proven otherwise). The importance of having good models and a clear understanding of component failure modes at this level cannot be over-stressed.

Where a fault has been identified that is common to two or more trees, it can be elaborated as a separate subtree, and attached to its parent trees via a “transfer node”. The complete diagram is strictly a Directed Acyclic Graph (DAG) rather than a tree, and should be subjected to qualitative cutset analysis. Potentially, a low-level component “fault” in one tree is closely related to failure of another protective measure. The method tolerates this by separating distinguishing functional failures (in ETA) from component faults (in FTA).

6.2 Running example

For the purposes of illustration, suppose the Press system is divided into the following functional components: plunger-drive mechanism (including the motor, clutches, drive chain, etc); controller hardware (PLC); controller software; guard; locks; button; sensors. (In practice a more precise specification would be needed here, especially of component interfaces, but this informal description suffices to illustrate the technique.)

Figure 4 shows partial fault trees for the two functional failures contributing to hazard H1 in the event tree for “upwards force fails while press open” technical failure of the press in Figure 3. The terminology in the fault trees reflects the switch in perspective from functional failures to component failures and intermediate faults. Note that component failures due to physical causes have been taken into account, such as breakage of the plunger drive mechanism and locks. The trees have been developed back to what could be a common-mode failure, whereby the Controller concurrently commands the locks to open and the plunger to be released (Figure 5).

Component-level cutsets for H1 would be extracted from the trees, and would include for example concurrent mechanical failures of the plunger drive and the locks. Note that in the current design, controller software is a possible single point of failure: a software error might lead to the plunger drive being deactivated and the locks being opened without a command from the operator. Such a design is unlikely to meet the integrity requirements of modern safety standards (see the discussion in Section 7 below). One solution would be to modify the design, and the SCD model, to include an interlock between the control panel and the locks, independent of the Controller, with the safety requirement that the locks can be opened only when the button is pushed. Similarly, the control panel is a possible single point of failure, so an additional protection mechanism is required, such as a sensor to detect person near press, with an interlock to the Controller software.

Finally, Figure 5 shows a second possible software error type, relating to failure to mitigate sensor error. We are assuming software has a responsibility (in this design) for detecting certain sensor errors by checking for logically infeasible combinations, and taking appropriate protective action: see [1] for more detail.

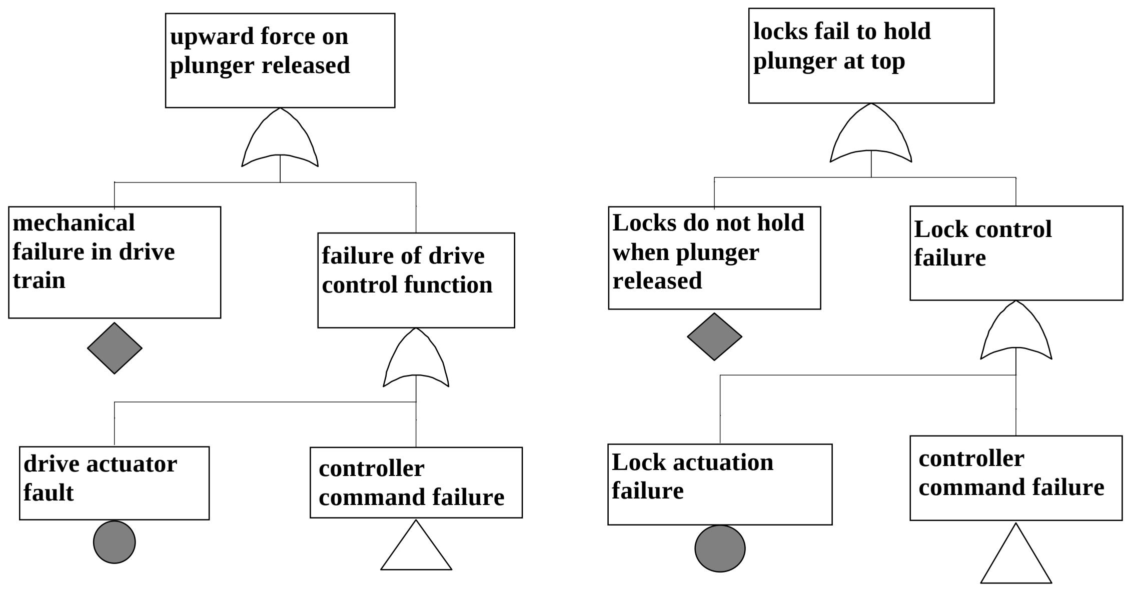
upward force on plunger released
mechanical failure in drive train
failure of drive control function
drive actuator fault
controller command failure
locks fail to hold plunger at top
Locks do not hold when plunger released
Lock control failure
Lock actuation failure
controller command failure

Figure 4 FTA for functional failures: (a) upwards force fails while press open; (b) locks fail to hold plunger at top

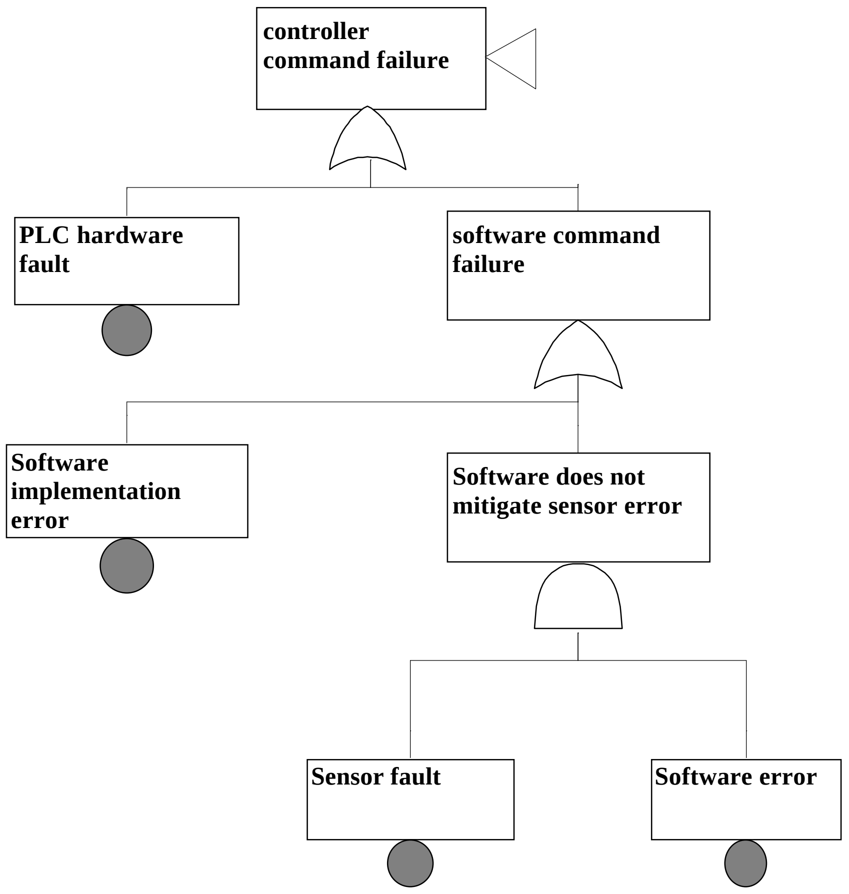
controller command failure
PLC hardware fault
software command failure
Software implementation error
Software does not mitigate sensor error
Sensor fault
Software error

Figure 5 FTA for common fault: controller command failure

6.3 Outputs of this step

The outputs of this step are a detailed cause-and-effect model of the system, which records how component failures contribute to system hazards. The model is a combination of the system design models, event trees and fault trees. The model thus serves a similar purpose to a Cause Consequence Analysis (CCA) model, but the goals and scope of our model have been identified and a systematic development method has been proposed. As usual, the models and fault trees should be validated (e.g. by independent review) and any assumptions noted and discharged during installation, operation, training and maintenance.

As a result of the step, the accident cutsets of the previous step have been refined to component level. By the end of this step, two different kinds of component safety requirement have been derived:

1. Requirements to avoid component failures having hazardous outcomes (“non-initiation requirements”).
2. Requirements to provide protective measures, to control or mitigate the effects of events having hazardous outcomes (“mitigation requirements”).

The statement of each safety requirement is in effect the logical negation of the component failure mode, with context information included. Examples of non-initiation safety requirements are:

- during normal operation, controller software shall deactivate the plunger drive only when the press is open and a close command is received from the control panel;
- controller software shall not activate the plunger drive while the plunger is falling after the PONR.

Examples of mitigation safety requirements are:

- the locks shall be strong enough to support the plunger's weight without the application of upwards force by the plunger drive;
- the control panel shall be placed sufficiently far away from the press that the operator cannot reasonably reach the press before it closes (even if the plunger drive erroneously activates after the plunger is past the PONR);
- the locks shall not be opened until the guard is closed;
- software shall check for physically meaningless combinations of sensor values and stop operation if found.

7 Allocation of Quantitative Targets

The final step in the process is to consolidate analysis and to assign failure rate targets to safety requirements. We have grouped quantitative considerations together in the step for ease of explanation, but quantitative analysis is likely to be interspersed throughout the process in practice, as a sanity check (to check that analysis is focussed on critical issues).

7.1 Safety integrity budgets

Before outlining the theory, we briefly reiterate here the motivation for quantifying safety requirements. For complex systems, it is generally impossible to guarantee that a given requirement will be satisfied 100% of the time. System design and engineering involves trade-offs between cost and reliability: a given requirement can generally be implemented in many different ways, some more reliable than others. Safety-integrity targets are a means of quantifying the criticality of a safety requirement - roughly, how much the system is relying on the safety requirement being performed, recognising that there is a risk that the requirement won't be met. This applies both to requirements of physical components, where there is a chance of random failure, and requirements of logical components, where there is a chance of design failure. (Note however that we are not implying any statistical relationship between probability of design failure and the failure profile of a logical component. Nor does the paper address in any way how failure rates of logical components can be assessed or predicted.)

Our process yields the means for estimating system risk and accident likelihood, once integrity targets have been allocated to component safety requirements. The cause-and-effect model provides the means for determining the system "risk profile" resulting from particular assumptions about likelihood of component failures, common-mode failures, and other accident coeffectors. The hazard analysis techniques described in the preceding steps can guide the apportionment of risk between different failure scenarios.

For the most part, the rest of this section outlines the principles of quantitative risk analysis and the mathematics underlying the hazard analysis tools of previous sections, and indicates some of the practical difficulties applying the theory. In particular, the need for good tool support to automate calculational aspects of the process will become obvious.

Allocation of quantitative targets is most easily explained in top-down terms, from determining tolerable rates for occurrence of accidents, to setting target (maximum failure) rates for system functional failures consistent with the results of consequence analysis (Section 5), through to setting targets for component-level safety requirements consistent with the results of causal analysis (Section 6). The process is far more difficult than this might suggest, however, due to the complexity of cause and effect. Allocations frequently require readjustment along the way, when feasibility of meeting lower-level targets becomes questionable. In what follow we use the metaphor of a *safety integrity budget*, since the problems are akin to balancing a budget. The next section describes the theory and the subsequent section illustrates the theory on part of the running example.

In order to set up a budget, it is first necessary to choose appropriate units for quantifying likelihood. Factors to take into account in choosing units include the type of safety requirement, the analysis tools and types of models used, and the system itself. Thus for example, one might use "probability of failure" for an on-demand function, "mean time to failure (MTTF)" for a physical component, "failure rate" for a human action or a continuously applied function, and so on. Fault trees work best with probabilities; Markov Analysis is better for MTTF calculations. Failure rates might be specified per year, per hour or per second, or perhaps using a time unit directly relevant to the system, such as per operational cycle or per processor cycle. We do not prescribe particular units in what follows, but simply note that they should be used consistently and that a clear justification should be given when converting from one unit to another within the budget.

7.2 Applying the process

The first step of this part of the process is to determine tolerable accident occurrence rates. The step typically involves definition of an appropriate accident severity classification and a policy regarding the acceptable maximum ("target") rate of occurrence of accidents in each category. Accident severity is typically expressed in terms of loss of life or human injury, but may also include damage to the environment. More stringent criteria might apply to members of the public than to operators. Generally however, determination of tolerable accident rates is an industry-specific societal matter and outside the scope of this paper. For the purposes of this paper, we assume that this exercise has been carried out, and that target accident rates have been set.

Recall that the consequence analysis stage of our process (Section 5) yielded cutsets for each accident, consisting of initiating events, functional failures and coeffectors (both internal and external to the system). In theory, a maximum likelihood for a particular accident can be found by summing the likelihood of each of its cutsets. These in turn can be calculated by estimating the likelihood of each of their elements. For the initiating event, an occurrence rate needs to be determined. For subsequent events in the cutset, the probability of the event needs to be estimated, assuming that the preceding events have taken place: in other words, a conditional probability is needed, based on the context in which the event occurs. The occurrence rate for the accident cutset is then the product of the respective event likelihoods.

For example, consider the following accident cutset from Figure 2: a) the operator aborts above PONR; b) the guard opens; c) the plunger drive activates but fails to prevent the press closing; d) the operator crosses the exclusion zone and puts his hands into the closing press. We discuss below how the likelihood of each event would be estimated. For the moment however, we note that the context of the event must be taken into account. Thus for example the likelihood of event d) will be different from the "same" event in the other accident cutset in Figure 2, since in the former case the plunger's descent will be slowed (but not reversed) by activation of the plunger drive, and so the operator has longer to cross the zone.

The likelihood of events outside the control of the system will need to be predicted or estimated, based on assumptions about the environment in which the system will operate and how it will be operated. Examples in this category are human actions (such as deciding to abort operation), external failures (such as power failure) and external coeffectors (such as noisiness of the environment). As far as the integrity allocation process is concerned, the likelihoods of such events are "givens", and together with the tolerable accident rates form inputs to the budget.

The other budget inputs are the likelihoods of internal coeffectors (normal events), which will need to be estimated from knowledge of the design. Thus for example, the likelihood that the operator will cross the zone within the time it takes for the slowed plunger to reach the bottom will need to be estimated. This may involve application of laws of physics, and (depending on the amount of design information available) some simplifying assumptions and error margins.

The variables in the budget are the likelihoods of functional failures and of the effectiveness of protective measures, such as events b) and c) above. In theory, these can be calculated from the likelihood of component failures and coeffectors, using the results of a causal analysis technique which supports quantification, such as Fault Tree Analysis. For isolated and well-understood design elements, such as stand-alone components with well-established failure profiles, it may be possible to provide estimates of component failure rates. But for all other system events, likelihoods will generally have to be provided as targets and flowed down to safety requirement targets.

In fact the situation is more complicated than implied above, since we need to calculate conditional probabilities of functional failures, and failures can share common causes. The likelihood of a functional failure later in the accident cutset may depend on exactly what caused the earlier event. For example, consider the following accident cutset from Figure 3: a) upwards force fails while press open; b) plunger not held at top; c) operator has hands in press and fails to avoid falling plunger. If the cause of a) is a software error which results in the plunger drive being deactivated erroneously, the same error could well result in the locks being opened; under these circumstances the probability of event b) is one.

The implication for our integrity allocation process is that we do not allocate likelihoods directly to functional failures in the accident cutsets. Instead, we drop down to the component level and derive cutsets for component failures. The failure budget is divided between these component-level cutsets. Thus the role of fault trees in our process is to record the logic of fault propagation and to yield component-level cutsets, but not necessarily to directly yield probabilities of functional failures. We do however make use of the same theory as underlies the use of fault trees for quantitative analysis. In particular, the independence of basic events is a key assumption.

There is another important point here. Generally there is little point in setting targets which can't be met. Thus there is an important issue here – that of validating the derived safety requirements. In practice, this can be done for well-understood components, e.g. frequently used pumps or valves. It is much more difficult for "novel" functions, including software. Nonetheless some "sanity check" is required that it is realistic to produce the required functions and, where failure rates are stipulated, that these are achievable.

7.3 Running example

This section illustrates the integrity allocation process on part of the running example. Consider the “hands crushed in press” accident. Let us suppose this is classed as a major accident, and that the tolerability target for such an accident is a rate of occurrence not more than 10^{-2} per year. Since the system hazards are described in terms of press operational phases, we shall use “probability per operational cycle” as the common unit for quantifying likelihood. Let us assume a normal operational cycle takes about a minute and that there would normally be about 10^{5} operations of the press per year. Thus accidents should have a probability of less than 10^{-7} per operation.

The next step in the allocation process is to apportion the budget between accident cutsets. For the sake of illustration, let’s suppose there are 10 accident cutsets and we decide to apportion the budget evenly between them, so that each accident cutset has a target probability of 10^{-8} per operational cycle.

Now consider the system hazard H1 “press closes without operator command”. Suppose that this hazard occurs in exactly one of the accident cutsets, with the following coeffector: operator has hands in press and fails to avoid falling plunger. Suppose it is estimated that the operator’s hands are in the press for approximately one tenth of the time that the press is open, but that the environment is so noisy that the operator might not hear the plunger falling. This means we can assign a probability of 10^{-1} to the coeffector, leaving a budget of 10^{-7} per operation for occurrence of hazard H1. (Note that if the hazard appeared in more than one cutset, its budget would need to be reduced proportionally.)

The budget for H1 would next be apportioned between the corresponding component-level cutsets. Recall from Section 6.2 that software is a single point of failure in the original design. If the design is left unchanged, this implies a software integrity target stricter than 10^{-7} per operation, which is simply not credible [2].

Assuming the design is amended to include an interlock between the button and the locks, the cutset involving controller software error becomes: a) software commands press to close without receiving a valid button signal; b) the interlock fails. Suppose a budget of $2x10^{-8}$ has been allocated to this cutset, and that an integrity target of 10^{-4} per hour has been set for the failure rate of the relay that implements the “don’t open locks unless button pushed” safety requirement. Assuming there are about 50 operations per hour, the probability of interlock failure can be taken as $2x10^{-6}$ per operation. Hence we set an integrity target of 10^{-2} per operation for the “don’t open the locks until you sense the button has been pushed” safety requirement on the Controller. That is, the degree to which the system is relying on the software to not command the press to close before receiving a valid button signal is 10^{-2} per operation.

The outputs of this step are integrity targets for each of the functional safety requirements identified in the previous step, together with documented assumptions about likelihood of external coeffectors. The integrity targets should be consistent with the cause-and-effect models generated in previous steps, and should meet the tolerable accident rates assigned.

8 Conclusions

In summary, the paper describes a process for deriving safety requirements for components of complex systems and a theoretical basis for allocating failure rate targets for safety requirements – "safety integrity requirements" for short. The process is based on the construction of risk assessment models, taking into account the possibility of component failures (random or systematic). The models are constructed in a top-down fashion, from an abstract functional view of the system and its main functional failures and protective features, down to sets of possible causes at the level of component failures and coeffectors. The modelling framework is sufficiently rich to capture safety issues for a wide variety of systems and component types – including hardware, software and operational procedures – and does not presume any particular safety architecture.

The approach integrates well with established safety engineering principles. It is consistent with current system safety standards but improves on them by being applicable to a broader range of application types, and in being better defined and more systematic. In particular, it allows fully integrated treatment of random and systematic failure. It can be applied at different levels of design detail and at different stages of the system development lifecycle. It is well suited to complex systems and structuring of high-level safety arguments, because of the support it provides for abstraction and hierarchical decomposition. This is particularly important for early targetting of high-risk areas and design for safety. The approach also supports iterated balancing of integrity targets during design.

The paper outlines some hazard analysis techniques which have proven effective in building risk assessment models and underpinning the quantitative analysis. The techniques suggested here are Functional Failure Analysis, Event Tree Analysis and Fault Tree Analysis, but other techniques could be substituted.

The process supports risk assessment through quantitative analysis of the consequence of component failures. Note however that it concerns only contingent risk (i.e., what is the effect on the system failure and accident occurrence rates if components fail), and doesn't tackle the really difficult problem of how to assess whether component failure rate targets are achieved. The aim of this paper is instead to quantify how much a system relies on particular safety requirements – or put another way, what is the maximum rate of component failure the system could safely tolerate.

For safety requirements relating to physical features, integrity relates to freedom from random errors, and roughly equates to strength or robustness of the physical component. For safety requirements relating to complex logical functions, however, integrity is a less familiar concept. In engineering terms, it relates most closely to the degree of assurance that should be provided: the stricter the failure rate target, the more rigorous the assurance required. This has important implications in planning the development of safety-critical systems, since safety is a significant cost driver and integrity targets have a significant impact on cost.

We have set out a framework for allocating safety requirements to components of complex systems. The process described here has evolved out of work on a number of systems, and represents our current understanding of a practical approach to setting safety requirements for complex systems. However more work is required to obtain greater confidence in the practicality of the process, in real project settings, particularly dealing with issues of scale and change in complex systems.

Finally, we stress the need for caution when interpreting the figures. They are only as good as the models and assumptions on which they are based.

9 Acknowledgements

The authors gratefully acknowledge the useful feedback from Brenton Atchison, Tim Kelly and Neil Robinson in the development of this paper.

10 References

[1] B. Atchison, P.A. Lindsay and D.J. Tombs, A case study in software safety assurance using formal methods, Technical report 99-31, Software Verification Research Centre, The University of Queensland, Australia, November 1999.

[2] R.W. Butler and G.B. Finelli, The infeasibility of experimental quantification of life-critical software reliability (ACM SigSoft, 16(5), 1991).

[3] CENELEC ENV 50129, Safety-related electronic systems for signalling (European Committee for Electrotechnical Standardisation, Brussels, June 1997.

[4] S.K. Dawkins, T.P. Kelly, J.A. McDermid, J. Murdoch and D.J. Pumfrey, Issues in the conduct of the PSSA, Proceedings of the 17th International System Safety Conference, 1999.

[5] Def(Aust) 5679, The procurement of computer-based safety critical systems (Australian Department of Defence, 1999).

[6] Def Stan 00-56, Safety management requirements for defence systems (U.K. Ministry of Defence Directorate of Standardisation, Glasgow, December 1996).

[7] Def Stan 00-58, HAZOP studies on systems containing programmable electronics, (U.K. Ministry of Defence Directorate of Standardisation, Glasgow, July 1996).

[8] Y. Deswarte, M. Kaaniche, P. Corneille and J. Goodson, SQUALE dependability assessment criteria, in: Proceedings of 18th International Conference on Computer Safety, Reliability and Security (SAFECOMP'99) (Toulouse, 1999); republished in LNCS 1698 (Springer, Berlin, 1999) 27-31.

[9] E.J. Henley & H. Kumamoto, Probabilistic Risk Assessment (IEEE Press, New York, 1981 & 1992).

[10] IEC 61508, Functional Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems, Parts 1-7 (International Electrotechnical Commission, 1999).

[11] T.P. Kelly, Arguing Safety – A Systematic Approach to Managing Safety Cases, PhD Thesis, University of York, 1998.

[12] N.G. Leveson, Safeware: System Safety and Computers (Addison-Wesley, 1995).

[13] P.A. Lindsay and J.A. McDermid, A systematic approach to software safety integrity levels, in P. Daniel, ed., Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP' 97) (Springer, Berlin, September 1997) 70-82.

[14] B. Littlewood and D.R. Wright, A Bayesian model that combines disparate evidence for the quantitative assessment of dystem dependability, in: Proceedings of the 14th International Conference on System Safety (SAFECOMP' 95) (Springer, 1995) 173-188.

[15] J.A. McDermid and D.J. Pumfrey, Safety analysis of hardware/software interactions in complex systems, in: Proceedings of 16th International System Safety Conference (System Safety Society, Seattle, WA, 1998).

[16] MIL-STD-882C, System safety program requirements (U.S. Department of Defense, September 1996).

[17] N. Talbert, Interview with John McDermid: “The Cost of COTS” (IEEE Computer, 31(6): 46-52, June 1998).

[18] P.G. Neumann, Computer-Related Risks (Addison-Wesley, 1995).

[19] D. Prasad, J.A. McDermid and I. Wand, Dependability terminology: similarities and differences, in: Proceedings of the 10th Annual Conference on Computer Assurance (COMPASS'95) (IEEE, Gaithersburg MD, 1995).

[20] N.H. Roberts, W.E. Vesely, D.F. Haasl and F. F. Goldberg, Fault Tree Handbook (Systems and Reliability Research Office of U.S. Nuclear Regulatory Commission, 1981).

[21] SAE ARP 4754, Certification considerations for highly-integrated or complex aircraft systems (Society of Automotive Engineers, Warrendale PA).

[22] SAE ARP 4761, Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment (Society of Automotive Engineers, Warrendale PA, 1996).

[23] A.K. Wabenhorst and B. Atchison, A survey of international safety standards, Technical report 99-30, Software Verification Research Centre, The University of Queensland, Australia, 1999.

[24] S.P. Wilson, J.A. McDermid, P.M. Kirkham, C.H. Pygott and D.J. Tombs, Computer-based support for standards and processes in safety-critical systems, in: P. Daniel, ed., Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP' 97) (Springer, Berlin, 1997) 197-209.

[25] S.P. Wilson, J.A. McDermid, C.H. Pygott and D.J. Tombs, Assessing complex, computer-based systems using the Goal Structuring Notation, in: Proceedings of the 2nd International Conference on the Engineering of Complex Computer Systems (Montreal, 1996) 498-505.

[26] H.A. Zogg, System safety methodology applied to an underground rail station, in: Hazard Prevention, November / December 1985, 16-18.