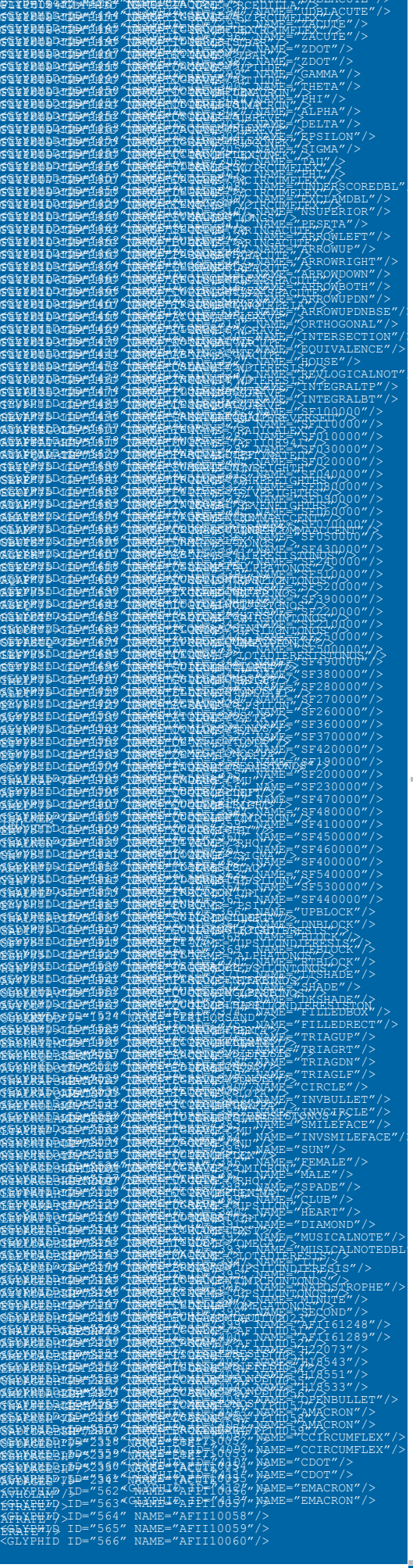


Good Practice Guidelines for Risk Management of Software-based Systems

March 2007

IPENZ
ENGINEERS NEW ZEALAND

IET
The Knowledge Network

The New Zealand Computer Society Inc.

The Institute of Measurement and Control NZ Inc.

CONTENTS

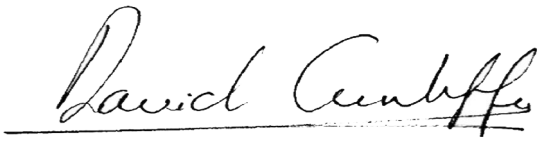
MINISTER'S FOREWORD

New Zealand's future economic development and international competitiveness is reliant upon improving our digital capability, infrastructures and skills – across government, the business sector and communities. These aims are brought together in the Labour-led Government's Digital Strategy.

Underpinning many of these areas is reliance upon international good practice and conformance with internationally-recognised standards. In the area of software engineering standards IPENZ has taken a leadership role and developed *Good Practice Guidelines for Software Engineering in New Zealand*. Increasingly, New Zealand software development companies will need to conform to accepted international benchmarks in order to remain internationally competitive. These guidelines mean that New Zealand companies will have a point of reference for the standards required by the global software market when developing new products.

The Government recognises that these guidelines are important. Industry-led initiatives, such as the development of these guidelines, assist in New Zealand's transformation into an innovative, knowledge-based, internationally-competitive economy.

Therefore, I commend the Institution and its collaborators for undertaking this valuable work.



Hon David Cunliffe

FOREWORD

New Zealand's economic development relies on good practices for developing and operating software-intensive systems. Moreover, to strengthen New Zealand's export competitiveness – both in terms of cost structures and delivered product quality – software development organisations must adopt the good practices already being used elsewhere around the world.

The financial transaction component of our economy has been largely "digital" for many years now. We are also rapidly implementing "digital government" through initiatives flowing from the Government's Digital Strategy. But these critical infrastructures for our 21st century economy fundamentally rely on trustworthy software-intensive systems.

Already our society is highly dependent on software intensive systems, and in turn their definition, development, implementation and operations. Obvious everyday examples include:

- electronic cash registers used for virtually all purchasing – whatever the final payment mechanism might be
- stock control systems
- monitoring and control systems for public utilities, transportation systems and health care
- rapid and easy web-based access to government-held information
- industrial control systems

Building and operating these kinds of critical systems are dependent on good practices across the whole software-intensive system lifecycle, such as those codified by international software engineering standards and maturity frameworks. While slavishly following good practices won't guarantee reliable or cost-effective development and operations, poor or mediocre practices make reliability and effectiveness a matter of chance rather than good management.

There are many other reasons for referring to documented good practices. They:

- foster traceability of features delivered to meet customers' requirements
- describe ways of avoiding risks caused by the failure of software-intensive systems
- provide checklists of procedures and practices that improve the overall efficiency of software development processes
- encompass considerations for building in and maintaining security, privacy and resilience to malicious attacks

In short, good practices as embodied in international software standards and frameworks are the roadmaps towards a disciplined approach to software systems development and operations.

INTRODUCTION

In a hardware-based engineering project, managing risks is one facet of overall project management. A risk assessment should be carried out in the initial stages of the project, followed by a number of related activities and reviews during the development of the design. The risk management process identifies areas that need special care during the operational life of the project (from initial concept development, through implementation and operation, to final disposal) and will ensure the use of suitable procedures to minimise risk and ensure that overall project management takes risk into account.

While risk management is often concerned with issues related to the physical safety of people who will use the final product, other areas of risk exposure such as financial viability and environmental effects are often considered at the same time. In many cases, the overall risk may mostly depend on these other areas.

With software-based systems, there is the same need to identify risks associated with the project, and ensure the use of appropriate management techniques. This document examines standard engineering practice for risk management, and how it applies to a project that involves large software engineering content.

RISK CONCEPTS IN THE ENGINEERING CONTEXT

In this context, the term “risk” has a slightly different meaning from that used in normal conversation. In everyday use, “risk” refers to an element or effect that has some potentially damaging consequences. In engineering work, the risk associated with a project is a measure of the potential cost associated with undesirable effects, and considers both the probability of a damaging event and the severity of the damage done if that event should come to pass.

A useful parallel is to consider a hazard on a golf course. In this case, the damaging event is a ball landing in the hazard, and the average number of extra strokes needed to get out of the hazard can measure the potential severity. While this measure indicates the potential damage to a scorecard if a golfer happens to hit the hazard, it only gives a partial indication of the potential impact of the hazard. To look at the relative significance of a hazard compared to others, we also have to look at the likelihood that a ball will actually land in the hazard. A given design of hazard may be relatively benign if located well away from the hole and off the fairway, but cause many problems if located in a critical location.

We can assign a relative cost to golfing hazards by counting the number of additional strokes they impose over a period, or converting this to an average cost per player. Players could potentially use this information to develop a strategy for a particular hole to minimise the total cost to their scorecards, by managing club and shot selection. The impact of landing in a hazard on a particular player’s scorecard may be reduced if the player develops skills in removing the ball from a sand-trap – this is an example of what we will refer to as “mitigation”.

In a similar fashion, engineering risk looks at a number of identified hazards associated with a project over a period and the total cost they may cause. This information is then used to identify the most significant contributions to the overall risk budget and develop appropriate strategies to minimise the potential losses. These measures focus on hazards that have the potential to cause the most damage, and include techniques to remove hazards or minimise the potential damage of those that cannot be removed, and take precautions that reduce the severity of any resulting damage.

A complete study of the risks associated with a hardware-based design project must look at all possible hazards throughout the operational life of the system. This “lifecycle” covers activities from development, construction and commissioning through the productive period to final decommissioning, demolition and disposal. In a project involving software, the physical activities associated with construction and commissioning may be relatively minor, but there may be hazards associated with introducing new software into a system already in operation, for example. Care must be taken when removing software to avoid affecting programmes that remain in use, and there are

also problems that have arisen with "dead" software that is left in a computer but is not actually being used. Data stored in computers marked for disposal must be deleted.

Software hazards can also arise through the way the system is operated. For example, one of the problems associated with the Therac particle accelerator that overdosed six patients was due to an operator who was very familiar with the system and made a number of keystrokes in rapid succession[1].

In the notes that follow, the following terminology is used:

- Hazard – a condition that has the potential to cause unwanted costs or damage.
- Hazardous event – an event whereby a hazard actually causes unwanted costs or damage.
- Likelihood of a hazardous event – the expected frequency of the event occurring, measured as a number of events in a specified time. Alternatively, this is the probability that the event will occur within a specified time.
- Severity of a hazard – the potential cost that might arise from the hazard. It is expressed in a consistent way for all hazards assessed, to allow meaningful comparisons of the relative impact of each hazard.
- Mitigation – a measure or measures that reduce the severity of a particular hazardous event, once that event has occurred.
- Risk associated with a hazard – the expected cost of a hazard over a period. It is determined numerically by multiplying the severity of a hazard by the sum of the likelihoods of the hazardous events.

Failure Types – Random, Systematic or System

Hardware failures are generally random in nature. Standard models are based on a well-defined pattern of failure, described in terms of predictable failure rates and related information. Random failures of this kind can be reduced by increasing safety margins between the capability of the target system and the demands made on it by the application.

While hardware factors may contribute to the failure of a programmable system, problems arising from defects in software are generally not random and cannot be described by a predictable failure rate. They arise from particular combinations of circumstances, and are perfectly repeatable if the same circumstances recur. These failures are systematic or functional – the latter term indicates that they are due to the incorrect functioning of the system in some cases.

Simply increasing safety factors will not reduce the incidence of systematic failures. These failures generally arise through oversights in specification, invalid assumptions made during design and development, and mistakes during detailed design.

Control of systematic failures requires that processes carried out during the project lifecycle are carefully managed. They are minimised by factors such as good communications between individuals or groups involved in different activities, making sure that all parties involved are aware of assumptions or limitations in scope, or clearly conveying the intention of the developers to end-users. Systematic failures in computer-based projects may also occur in the interface between hardware and software.

The differences between random failures, which are effectively treated with well-developed mathematical methods, and systematic or functional failures, must be remembered in any exercise to manage risk in software-based systems.

In situations involving multi-component systems, failures are often not the result of problems with hardware elements or malfunctions in software items. They are more likely to arise from incorrect interactions between the elements of a system, or between the system and the environment (Leveson, 2002). A "failure" resulting in loss may involve organisational or management aspects of the design or operational system rather than specific identifiable hardware or software elements, or the commonly blamed "human error" at the point of application. Systematic failures can occur at any stage of the project lifecycle, from initial concept development through to actual operational or productive activities.

1 An extract discussing the Therac-25 accidents is available at sunnyday.mit.edu/therac-25.html

RISK ASSESSMENT

AS/NZS 4360: 2004 Risk Management gives a well-established and widely recognised process for managing risk in a general business environment. The steps set out here cover the first four stages of the AS/NZS 4360 process, covering the identification and evaluation of risk. Other stages included in this document cover treatment of hazards and ongoing monitoring and review activities.

The first stage of the risk management process is to establish the context for the project, and identify and assess the risks to which the project could be exposed. Where safety issues are concerned, identification of hazards is an explicit requirement placed on the owner or operator of a project by occupational health and safety legislation. There are also related obligations on designers and others involved in the development, implementation and operational phases.

The risk assessment process has four stages. These can be summarised as:

- What could go wrong? Identify the hazards.
- What happens if it does? Analyse the consequences.
- What can we do to avoid it? Identify possible preventive measures.
- What can we do if it does happen? Evaluate mitigation measures.

A major benefit of the risk assessment activity is that it makes people involved with the project aware of the possibility that things could go wrong, and that there is a need to take some special precautions. It should be a team activity with all the groups involved in the final project, and treated as an educational or training exercise as well as an engineering activity.

One of the outcomes from the risk assessment should be an indication of the overall cost that hazards will have on the project during its lifetime – including the cost of prevention and mitigation measures, and a reasonable estimate of the potential losses.

A formal risk assessment process needs to be carried out in a controlled fashion, and the results fully documented and made available to all interested parties. However, risk assessment should also become an activity that is built in to the decision-making process at a very low level – team members should be aware of the possible impact of their decisions and act accordingly. No golfer will systematically analyse the risk associated with bunkers – this becomes an automatic reaction, affected on the day by other influences such as wind direction.

Context

A product designed for experienced adults may be dangerous if used by children or those without specific training. This is an example of how the different uses for a product may affect the overall risk.

The same applies to systems that include software components. If a software package is to be used for limited purposes by people who are very familiar with it, the associated risk may be very small. However, if the package is adopted for use in a different environment, or made available to less experienced users, the likelihood of error is much greater.

Context also applies to the total system in which a software package may be used. If designed for use on a stand-alone computer, there is no need to consider the possibility of malicious damage by unauthorised users accessing the system via the Internet. However, if the package is used on a connected system, this problem must be addressed.

Context may involve the hardware platform intended to run the software, and may be affected by the hardware configuration of the platform. Many industrial software packages were originally written using the standard PC serial port to communicate with target hardware. With the replacement of the serial port in laptops with USB ports, some of these packages can no longer be used.

Context may also be concerned with the size of the associated application. An example of a factor affected by size is the time to access information in a large database. This may also be adversely affected by the number of users trying to access the system at one time.

Details of the context in which a software component may be used should be set out in a User Requirements Specification or similar document.

Hazard Identification

The first step in any engineering activity should identify the hazards associated with the proposal. It is important to list not only the specific hazards associated with the normal day-to-day running of the project, but to consider those that come about during the implementation and disposal phases, and any extraordinary activities during operation.

In a hardware-based engineering project, the hazards are directly associated with the activities being monitored or controlled, and a hazardous event is triggered by something outside the monitoring, control or protective system. Some examples of typical hazards are high pressures or temperatures, moving machinery, or corrosive chemicals. Hazardous events involve conditions outside the normal working range, malfunction of machinery or injury due to incorrect operating procedures, or unexpected releases.

These hazards tend to be well-documented and are often identified using check-lists created during similar exercises.

Hazards peculiar to a specific context and not generally applicable should be specified in the project brief. In software terms, they should be spelt out in the User Requirements Specification. However, the design team must also act to establish whether hazards other than those specified are likely to be involved. Formal procedures such as a HAZOP review attempt to force the review team to consider a wide range of possible hazards.

It is increasingly being recognised (Leveson, 2005) that physical events such as component failure have less effect on the total project risk than system failures, often involving management issues during project development or operation. As stated above, these types of failure are the main contributors to software-related risks. Because they are less tangible and poorly recognised (if recognised at all) in incident reports, they are often overlooked during hazard assessment. However, it is important to identify any assumptions made during a risk assessment that involve factors relating to the risk – one example is the extent of training or other competencies required of those working with the system.

Problems usually arise in hardware systems when there is an abnormal event of some kind, so that the system deviates from its normal operating regime. Codes of Practice set out acceptable procedures to follow during design or operation. Legislation may require a design to comply with these Codes or provide evidence that work is to a similar or higher standard.

Where software is involved in a protective system, a software failure could affect the ability of the system to respond correctly when required. In this context, a software failure is not often regarded as a hazard in its own right – it will be considered along with the underlying physical hazard. However, it is also possible for a software fault to trigger an incident without an external event – the software failure itself is the trigger event that is directly responsible for loss. No outside event initiates the problem.

Some generic headings for pure software hazards could be (Howard et al, 2005):

- data corruption – for example, buffer overflow
- deliberate misuse or malicious intervention (hacking)
- invalid data entry
- errors in specification, program design, or development activities
- hardware failure
- incompatible software modules in a multi-tasking system

Other more specific hazards must be itemised for individual projects.

It needs to be emphasised that many hazards are introduced by failings in the wider management system rather than specific problems arising from either hardware or software. For critical systems, project managers must take care to use control or monitoring practices that are commensurate with the degree of risk if a failure occurs.

Identify Consequences

A hazardous event associated with any of the identified hazards may give rise to a number of different undesirable outcomes. These are even more context-dependent than the hazards. Some consequences to consider are:

> physical safety issues – where failure of software or hardware may directly result in injury or adversely affect health
  - These apply particularly to programmable systems used in industrial applications to provide protection or supervision of dangerous processes.
  - There have been several cases where failings in medical computer systems have been responsible for patients receiving incorrect dosages of medication.
  - Software used to make engineering calculations needs to have proper testing, verification and validation to make sure the calculations are correct.

> economic damage
  - Financial effects are probably the major potential consequence of a software system failure.
  - There are a number of ways this could arise – from errors in accounting applications to failure to account for specific factors.

> legal consequences
  - Where software is used to comply with legal obligations, incorrect operation might leave an organisation open to prosecution.

> environmental damage
  - This is another consequence that may apply mainly to process control or supervision applications.
  - Records such as Material Safety Data Systems may cause damage if, for example, the properties of a substance are wrongly recorded.

> public opinion
  - While not directly related to hard engineering or financial damage, this topic needs to be specifically addressed.
  - Events that appear to be relatively minor from an engineering or financial viewpoint may attract a disproportionate level of publicity compared with the measurable consequences.
  - Failings in this area may cause loss of future

business, additional costs to an organisation because of increased problems obtaining planning permission, or a need for increased advertising or other expenditure to overcome the damage.

- Bad publicity may also result in indirect financial loss through a drop in the organisation's share price.

In a software context, the consequences of a failure are not often physical. While physical damage to hardware must be considered, for example by developing disaster recovery plans, much of the harm arising from a software system failure is likely to be financial.

Evaluate Consequences

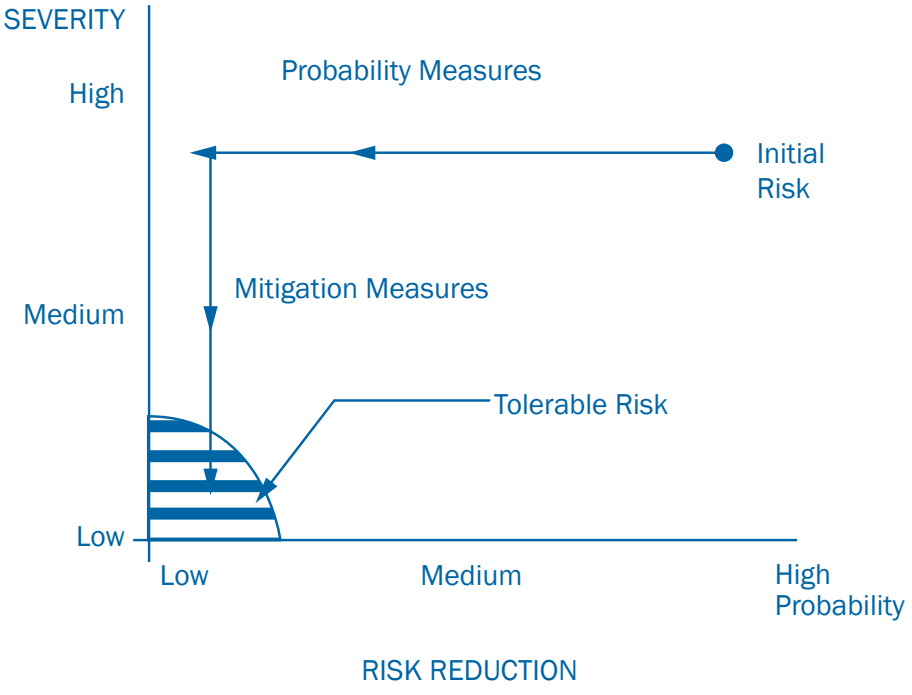
Once the consequences have been identified, they must be ranked in some way to make sure that the resources available for risk control are used to best effect. While public opinion may be a good reason to devote resources to high-profile outcomes that may in fact have a low impact and are very unlikely to occur, this needs to be properly justified rather than be an automatic reaction. There is always a danger that a major hazard may be overlooked or neglected because resources are used to deal with a hazard that is much less significant.

A number of methods have been used to evaluate the consequences of failure in hardware systems and are well described in the literature[2]. Applying these methods to software may not work so well. The traditional methods use expected probability as one of the parameters, but as we have seen this may not be relevant to software systems where functional or systematic failures are more likely to be significant.

A number of tools can rank hazards based on the possible consequences. A very simple approach is to assign hazards a ranking from 1–4 based on the severity of consequences, with another ranking of 1–4 based on the expected probability. The overall ranking is found by multiplying the two rankings, giving a value of 1–16. While this method is simple and relatively easy to implement, it is artificially precise in its results. For example, the ranking of a hazard with severity of four and probability of two is eight, while a hazard with probability ranking of three and severity of three has a relative risk which is one point higher. In reality, these situations need further analysis to sort out their relative significance, or they would be given the same priority for treatment.

Some qualitative methods are given as examples in *IEC Standard 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems* (International Electrotechnical Commission, 1998–2000). In each of these, factors affecting the overall risk level are assigned to classes or ranges, which are then used to indicate a requirement for the integrity of the resulting system. While specific to issues relating to personnel safety, these techniques can be easily amended to other aspects of risk. We will look at these in more detail in a later section.

FIGURE 1: REDUCING PROBABILITY OF FAILURE



2 See for instance: Smith D J 2005, *Reliability, Maintainability and Risk*, Butterworth-Heinemann.

RISK REDUCTION

The ideal target for risk for any project is zero. However, it is not practical to expect to reach this, and in the real world, there will always be some risk attached to any activity, regardless of the resources used to deal with it. The practical engineering objective is to reduce the risk associated with an activity to a level that can be tolerated in the context where that risk arises.

Tolerable Risk

One of the initial decisions that should be made in any project is the level of risk that is tolerable. This may depend on the size of the organisation that will use the product – a large multinational organisation has many more resources than a small start-up firm, and may be prepared to underwrite a much higher level of costs arising from hazards. If the total overall risk cannot be reduced below the tolerable level, the project viability must be reconsidered.

The tolerable risk is a performance figure that that has to be taken into account when making any engineering decisions. A design with a level of risk well below the tolerable level may have been over-engineered, while one that does not meet this target will be unacceptable on safety or related grounds.

Tolerable risk is usually defined in terms of the total risk arising from all hazards associated with the project. When dealing with individual hazards, the tolerable risk associated with a single hazard must be much less than the overall tolerable value. In general, the tolerable risk associated with a single protective function should be between 10 and 100 times less than the overall tolerable risk for the operation as a whole (Timms, 2006).

In general, the overall risk obtained during the risk assessment phase will be well above the tolerable level. If the raw risk associated with any hazard is well below the tolerable level, no further action is needed in relation to that hazard. However there will be some hazards that carry a risk that is much higher, and these need to be reduced in some way. Initially, resources used for risk reduction will be most effective if applied to the hazards having the highest risk, and the risk assessment exercise will target these areas.

The ALARP principle is a guide to the amount of risk reduction that is necessary.

ALARP

ALARP stands for "As Low As Reasonably Practicable". It expresses the concept that risks need to be reduced to a level that is low enough, but not too low. The terminology introduces a couple of significant points to take into account in any risk evaluation exercise.

Any risk evaluation exercise is inherently uncertain. In a well-engineered project, hazardous events should have a very low probability of occurring during the total life of the project. However, it is not possible to guarantee that they will not arise. You can never state that there is "no risk" associated with an activity of any sort. When those with a high probability have been attended to, low-probability hazards will remain.

However, a level of risk that is "too high" makes the project unsustainable. Because of this, as part of the overall project objectives, there should be some statement regarding what is an acceptable risk for the project. The "reasonable" component of ALARP is very vague and generally not regarded with favour by lawyers. However, it does reflect that there is a need for engineering judgement as to what is "reasonable" in a given context. Generally, if a risk prevention measure costs more to implement than the amount it reduces the overall risk to the project, it is "unreasonable" to expect the measure to be applied.

The ALARP concept divides the range of risks into three regions. Unacceptable, high-risk levels are classed as "intolerable", with very low risks classified as "broadly acceptable". The United Kingdom Health and Safety Executive has published guidelines relating these classes to the probability of fatality per person per year in their publication *Reducing Risks Protecting People* (2001). In this document, the threshold for a risk to be classed as intolerable is a probability of fatality of 1×10^{-3} per person per year if the affected person is an employee. For a member of the public, the threshold is 1×10^{-4}. Risks less than 1×10^{-6} are considered as "broadly acceptable" while less than 1×10^{-7} is "negligible".

Reducing Probability of Failure

Risks can be reduced in a number of ways. If the probability of the initiating event can be reduced, this will have a direct impact on the overall risk. With some hazards, the probability of the initiating hazardous event is outside the control of the engineer – an example in civil engineering is the frequency of earthquakes in a particular location. However, with other hazards, the initiating event is a failure of some sort, and here the frequency of failure can be controlled to some extent by design decisions. In designing a car, engineering measures such as anti-lock brakes are intended to reduce the likelihood that the driver will lose control and go into a skid. (See Figure 1 on page 8.)

Reducing Severity – Mitigation

Mitigation measures act to reduce the severity of the outcome, given that a hazardous event has occurred. Where the underlying frequency of hazardous events cannot be reduced, as with earthquakes, engineering measures such as seismic isolation of buildings reduce the impact of the event, and are a form of mitigation.

Mitigation measures may be used with protective measures to minimise the overall risk. The airbag in a car is an example of mitigation – it is intended to reduce injury in the event of a loss of control. The combined system of anti-lock brakes and airbag includes elements to reduce the probability of a skid, and elements to reduce the impact of a skid should one occur.

THE EFFECTIVENESS-CASE

The process of assessing risks identifies the problems that have to be addressed during the design and development process in order to meet the performance objectives of the system, whether these relate to safety, reliability or other related measures. Part of the activity during the development phase must ensure that these problems are addressed, and that the system meets the specified performance requirements.

While it is possible to comply with requirements using an informal process and modifying the design as a result of reviews, it is far more effective to consider performance issues at the outset and decide on strategies to deal with these issues. Development of a product, and its subsequent operation and maintenance, are made much easier if the decisions made during initial development are documented in a suitable form. Documentation may also be specifically required by legislation or in the terms of a contract.

A safety-case, in applications involving physical risk, is a document that demonstrates that a system is acceptably safe to operate in a particular context. It must include evidence about the system performance after hazardous events, and set out the arguments linking this evidence to particular requirements of safety performance.

A great deal of work has been done in this area in recent years, particularly by the Department of Computer Science at the University of York in the United Kingdom. Much of this aims at the process to prepare a safety-case and formalising its structure. While aimed specifically at the issue of demonstrating safety, the approach extends easily to cover other areas of risk. A generic term to cover these other aspects is the "effectiveness-case", and this will be used in the discussions below.

Goal Structuring Notation

Much of the work done at the University of York and other institutions uses a graphical format to demonstrate how a system's performance requirements can be met. This is done by defining goals, and then setting out strategies to meet these goals. In many cases, a goal will have a number of associated sub-goals.

This work is based on a PhD thesis by Kelly (1998). There are many parallels between the use of the Goal Structuring Notation (GSN) and the ideas of top-down development, and just as the details of a design are refined during the development process, the effectiveness-case should evolve as well.

The top-level of the effectiveness-case using GSN is a statement, in the form of a logical proposition. This is the goal that is to be proven. For example, a generic goal could be:

"System X meets the requirements for operation as specified in Document 123."

This goal has some specific items that identify the particular context. We are dealing with System X, and the specific document setting out the requirements is also identified.

The basic format of a GSN document would identify strategies to demonstrate this. In our example, we could elaborate on this with strategies to cover the safety requirements, environmental consequences, and financial aspects of the performance specification. Each of these strategies may then result in a series of sub-goals. The process continues until a strategy can be demonstrated to be complete by some action – this is then a basic solution.

Kelly highlights the need for careful attention to syntax in this process. A goal is a logical proposition to be proved, and must be written using a <noun-phrase> <verb-phrase> format. Strategies are not methods of achieving the desired result – they are ways to demonstrate that the result has been met. Kelly gives an example where the strategy "use mechanical interlocks" is incorrect – it refers to a design decision rather than to the way in which that decision can be evaluated. His recommended wording for this strategy is "argument by appealing to effectiveness of mechanical interlocks in design". As an instruction, this becomes "argue by appealing to...".

Dealing with Failure Modes

The standard techniques to assess risk – or demonstrate effectiveness of hardware-based systems – use statistical approaches such as Fault Tree Analysis or Failure Modes and Effects Analysis (FMEA). Because these apply to failure modes that are essentially random, they are not well suited for dealing with systematic failures with software systems, or to system failures at management level.

This difference can be explicitly dealt with using GSN, by identifying different strategies as being suitable to assess different failure modes. For random failures, a suitable strategy could be:

"Argue by using Fault Tree to demonstrate an acceptably low failure rate."

Systematic failures can be minimised by following appropriate procedures in the product, or by using techniques such as inspection or testing during development. The strategy associated with these can be written in a form such as:

"Demonstrate that the design process used is appropriate to the risk level."

System Integrity Level

The strategies used to demonstrate an effectiveness-case must include references to the integrity of any protective measures. It must be possible to show that any features that protect against a hazardous event will in fact be effective if that event occurs.

Another consideration (that is sometimes overlooked) is that some systems must not operate unless the target incident occurs – airbags in cars are a good example of this.

IEC 61508 defines safety integrity for protective systems as measured by the probability of failure of the system under consideration. It categorises integrity into four Safety Integrity Levels (SIL 1–4), with SIL 1 the least rigorous and SIL 4 the most severe. For safety systems, which operate under exceptional circumstances, the required SIL is based on the probability that the system will fail to operate when a legitimate demand occurs – the Probability of Failure on Demand (PFD). For protective systems with a high demand rate, such as machine guards, the SIL is determined by the reliability of the equipment expressed in failures per hour.

For political reasons, *IEC 61508* is concerned solely with safety systems. It is not concerned with the effects of a spurious activation of the safety system, although these may have some adverse effects. However, the SIL is useful in areas where the critical nature of the system is not specifically safety-related, and in these cases it can be thought of as "system integrity level" rather than "safety integrity level". It will be used in this sense in this document.

FIGURE 2: SYSTEM INTEGRITY LEVEL

	Low Demand	High Demand
System Integrity Level	Probability of failure on demand	Failures per hour
1	10^{-2} – 10^{-1}	10^{-6} – 10^{-5}
2	10^{-3} – 10^{-2}	10^{-7} – 10^{-6}
3	10^{-4} – 10^{-3}	10^{-8} – 10^{-7}
4	10^{-5} – 10^{-4}	10^{-9} – 10^{-8}

The SIL required for a particular protective function can be found by assessing the risk to the enterprise arising from the hazard with no modifications, and comparing this to the specified tolerable risk for that hazard. For instance, if the expected rate of a high-pressure event in a steam system is once every two months, and the tolerable frequency of a pressure-related system failure is once in 200 years (or a 10 per cent chance of a pressure failure over the expected life of 20 years) the protective system can be expected to work 1,200 times. The failure rate must be better than 1 in 1,200, or less than 0.0008. This is a PFD of 8×10^{-4} and corresponds to SIL 3.

Required failure rates can be evaluated quantitatively from calculations involving numerical assessments of severity and probability. However, these assessments often involve a lot of work using statistical data that are not sufficiently well defined for this purpose. Failure rates of new and highly reliable equipment cannot be specified with any degree of confidence, and failure rates of installed equipment depend very much on maintenance routines, adverse factors in the environment, and the availability of suitably trained staff. A more useful approach to assess the system integrity requirements is based on a qualitative analysis.

Figure 3 shows a chart from *IEC 61508* that can be used to determine the required SIL for a protective function. There are four categories of consequence, ranging from C_a (minor) to C_d (major). Likewise, probabilities are ranked from W_1 (low) to W_3 (high). Other factors modify the requirements depending on the likelihood that people will be exposed to the hazardous event when it occurs, and the possibility that anyone so exposed might be able to avoid injury (because of a slow development or warning noises, for example).

As published, the chart does not give any indication on what is regarded as "minor" or "major" consequences, or "low" or "high" probability. To use these scales, they must be calibrated to match conditions in the target environment. Generally, low probability is a situation arising only once or twice during the expected life of a project, while high can be expected several hundred times. Minor consequences in a physical safety application may be injury to one person resulting in less than one day of lost time, while major may involve several fatalities. The calibration of these charts has to be decided by the customer organisation in line with the overall tolerable risk designation for the activity.

Similar charts may be drawn up for other sets of consequences. The diagram below shows possible arrangements for charts relating to production loss and environmental damage, together with possible details for the calibrations.

A qualitative assessment of integrity requirements may be made using these charts as a guide to stimulate group discussion. A suitable group might consist of members of the project team, and representatives of the client organisation and end-users. By identifying each hazard in turn, and then evaluating the consequences, the SIL for each component of risk can be identified – required SIL is the highest of those figures. So, if safety considerations result in SIL 2, production losses SIL 3, and environment SIL 1, the system must meet SIL 3.

In the context of the effectiveness-case, the SIL provides a reference that can be used to identify relevant techniques for different phases of the development or design activity. Strategies can then demonstrate that the techniques are in line with an accepted industry standard such as *IEC 61508*.

Meeting SIL Requirements

What is the impact of the SIL on the design process? *IEC 61508* sets out acceptable procedures that will allow the required integrity level to be met if implemented properly. Software

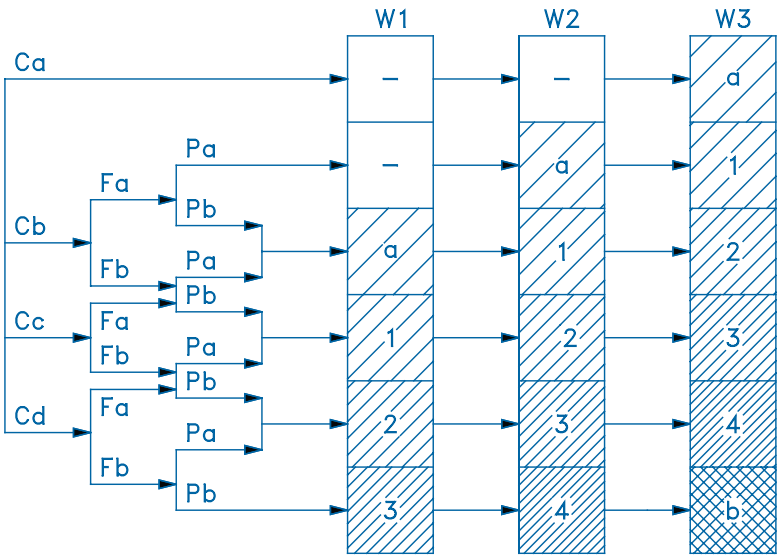
FIGURE 3: IEC 61508 RISK ANALYSIS PROCEDURE

CONSEQUENCES:
Ca Minor
Cb
Cc
Cd Major

EXPOSURE:
Fa Rare to frequent
Fb Frequent to continuous

AVOIDANCE:
Pa Sometimes possible to avoid
Pb Almost impossible to avoid

PROBABILITY:
W1 Low
W2
W3 High



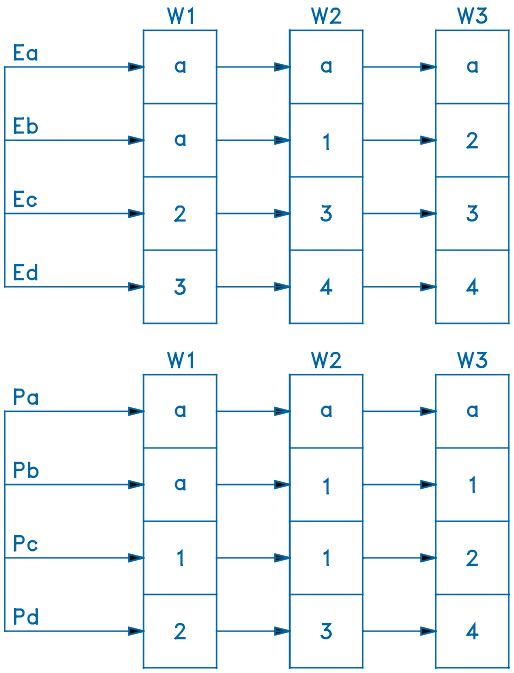
a: No special requirements
1–4: SIL
b: Another approach required

FIGURE 4: ENVIRONMENTAL AND PRODUCTION LOSSES

ENVIRONMENT
Ea Minor Spill/Discharge
Eb Release within fence Limited consequences
Ec Release outside fence Limited Consequences
Ed Release outside fence Major Consequences

PRODUCTION
Pa Minor operational upset
Pb Moderate loss of production Limited damage to equipment
Pc Major loss of production Major damage to equipment
Pd Major damage to non–spared equipment

PROBABILITY:
W1 Less than 1 in 10 years
W2
W3 More than 1 a year



elements are covered in Part 3 of the Standard – these are further expanded in Part 7. As an example, Table A.3 in Part 3 deals with software design and development tools as well as programming language.

The requirements for suitable programming languages are set out in Section C.4.6 of Part 7 of the standard, and recommendations relating to specific programming languages are given in Table C.1 of this Part.

The levels of recommendation are:

HR Highly recommended – if this technique or measure is not used, the reasons for not using it should be detailed during safety planning.

R Recommended as a lower measure to an HR recommendation.

– The technique has no recommendation for or against its use.

NR The technique or measure is positively not recommended for this safety integrity level – if it is used, the reasons for using it should be detailed during safety planning.

From Table A3, use of a suitable programming language is "highly recommended" for all SIL levels. Use of a restricted subset of the language is not required for SIL 1 and SIL 2, but is highly recommended for SIL 3 and SIL 4. Tools and translators must be certified or have demonstrated reliability through extended use.

While standard C and PL/M are acceptable for use at SIL 1, they are not recommended for SIL 3 or SIL 4. However, provided a restricted subset is used, with defined coding standards and static analysis tools, C is acceptable for the higher integrity levels.

These are some of the recommendations given by IEC 61508. Others relate to the need for independent testing and inspection of code, competency requirements for people working on the project, definition of responsibilities for various activities within a project, and factors such as project communications, documentation and control. Those working on software projects with a requirement for high integrity are strongly advised to make themselves familiar with the full set of recommendations.

TABLE A.3 – SOFTWARE DESIGN AND DEVELOPMENT: SUPPORT TOOLS AND PROGRAMMING LANGUAGE (SEE 7.4.4)

Technique/ Measure		Ref	SIL1	SIL2	SIL3	SIL4
1	Suitable programming language	C.4.6	HR	HR	HR	HR
2	Strongly typed programming language	C.4.1	HR	HR	HR	HR
3	Language subset	C.4.2	– –	– –	HR	HR
4a	Certificated tools	C.4.3	R	HR	HR	HR
4b	Tools: increased confidence from use	C.4.4	HR	HR	HR	HR
5a	Certificated translator	C.4.3	R	HR	HR	HR
5b	Translator: increased confidence from use	C.4.4	HR	HR	HR	HR
6	Library of trusted/verified software modules and components	C.4.5	R	HR	HR	HR

*Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

(extracted from IEC 61508-3, 1st edition, 1998)

TABLE C.1 – RECOMMENDATIONS FOR SPECIFIC PROGRAMMING LANGUAGES

Programming Language		SIL1	SIL2	SIL3	SIL4
1	ADA	HR	HR	R	R
2	ADA with subset	HR	HR	HR	HR
3	MODULA-2	HR	HR	R	R
4	MODULA-2 with subset	HR	HR	HR	HR
5	PASCAL	HR	HR	R	R
6	PASCAL with subset	HR	HR	HR	HR
7	FORTRAN 77	R	R	R	R
8	FORTRAN 77 with subset	HR	HR	HR	HR
9	C	R	– –	NR	NR
10	C with subset and coding standard, and use of static analysis tools	HR	HR	HR	HR
11	PL/M	R	– –	NR	NR
12	PL/M with subset and coding standard	HR	R	R	R
13	Assembler	R	R	--	--
14	Assembler with subset and coding standard	R	R	R	R
15	Ladder diagrams	R	R	R	R
16	Ladder diagrams with defined subset of language	HR	HR	HR	HR
17	Functional block diagram	R	R	R	R
18	Functional block diagram with defined subset of language	HR	HR	HR	HR
19	Structured text	R	R	R	R
20	Structured text with defined subset of language	HR	HR	HR	HR
21	Sequential function chart	R	R	R	R
22	Sequential function chart with defined subset of language	HR	HR	HR	HR
23	Instruction list	R	R	R	R
24	Instruction list with defined subset of language	HR	HR	HR	HR

(extracted from IEC 61508-7, 1st edition, 2000)

SUMMARY

Engineering projects involving software components, or projects involving only software, involve risks similar to those found in hardware-oriented engineering activities. Where these risks cannot be reduced by other measures, the engineering processes used during the development and design phases must be suitable to ensure that the system will operate with the required degree of integrity. High-integrity may be required to ensure personnel safety, or to minimise exposure to financial risk or other risk areas.

The risk management process starts with identifying hazards and assessing the associated risk, involving the probability of a hazardous event and the severity or cost of the consequences. Where risks are excessive, they can be reduced either by reducing the probability of a hazardous event or by adding measures to reduce the severity. In either case, evidence must be produced to show that the measures effectively meet the target risk levels. Additional equipment (involving hardware and/or software) must be of sufficient integrity to provide confidence that it will function as and when required.

Risk management requires the development of a case showing that the developed system meets the specified integrity requirements, which may involve reliability as well as safety issues. This case should include evidence related to the integrity, as well as an argument setting out how the evidence demonstrates effectiveness. A suitable format for this is Goal Structuring Notation.

For software systems, integrity cannot be guaranteed simply through statistical failure rates. Most software failures are systematic or functional rather than random, and can be avoided only by adopting appropriate measures during design and development. Suitable measures for safety-related systems are set out in *IEC 61508*. While this is concerned with safety-related systems, its recommendations can be extended to cover other risk areas.

ACKNOWLEDGEMENTS

These guidelines were prepared by Bruce Durdle, based on notes for a course introducing the basic concepts of *IEC 61508*. Comments on these notes were made by Dr E Scharpf and C Feltoe. A number of people have commented on or reviewed this document. They include:

D A Hall
A Holt
C Skinner
J Moore
B MacDonald
P Voldner
T McBride
A Clark

REFERENCES

Health and Safety Executive 2001, *Reducing Risks, Protecting People*, United Kingdon Government, London. Retrieved from http://www.hse.gov.uk/risk/theory/r2p2.pdf.

Howard et al 2005, *19 Deadly Sins of Software Security*, McGraw-Hill, New York.

International Electrotechnical Commission 1998–2000, *Functional safety of electrical/electronic/programmable electronic safety-related systems*, IEC 61508 parts 1-7.

Kelly, T 1998, *Arguing safety – A systematic approach to managing safety cases*, PhD Thesis, Department of Computer Science, University of York.

Leveson, N G 1995, *Safeware: system safety and computers*, Addison-Wesley Professional.

Leveson, N G 2002, *Model-based analysis of socio-technical risk*, Engineering Systems Division, Massachusetts Institute of Technology.

Leveson, N G 2005, A systems-theoretic approach to safety in software-intensive systems, *IEEE Transactions on Dependable and Secure Computing,* 1(1), p66–86.

Standards New Zealand 2004, *New Zealand standard: Risk management,* AS/NZS 4360:2004.

Timms, C R 2006, Achieving ALARP with safety instrumented systems, *The First Institution of Engineering and Technology International Conference on System Safety*, p8.

APPENDIX 1: CRITICAL SOFTWARE SYSTEMS

RISK ASSESSMENT CHECKLISTS

These lists show items that may need to be considered during a risk assessment exercise. They are not specific recommendations, but form a base to develop the specific needs of each individual project. In particular, the items listed may or may not be relevant, and further entries will be needed to meet the requirements of a specific case.

Checklists can summarise vital information for a project development team, and can also record issues that arise during a project for future reference – either by the end-user or during later development exercises. They can also be used to record common data for use throughout the project.

1. CONTEXT

The items on the left of this table are issues to consider during the initial concept development. Any requirement for the item should be indicated under the "Required" column, while the "Specified" column should contain a reference to this issue in the User Requirements documents.

	Required	Specified
> User capabilities		
application experience		
computer literacy		
physical requirements[1]		
> Access		
physical		
internet		
controls		
> Operating system		
> Number of users		
normal		
peak		
> Interfacing		
> Hardware requirements[2]		

1 For example, colour blindness.

2 Is there a need to use a specific platform or special hardware features? For example, many industrial software packages rely on the availability of a "standard" RS232 serial port to interconnect to target hardware – users of these packages are seriously inconvenienced by the absence of such ports on modern laptops.

2. CRITICAL OBJECTIVES

This section identifies and records the primary objectives for the project. It also allows space for the tolerable failure rates for each objective to be set down.

Sections such as “Physical safety” and “Legislative” will generally need to be expanded further to identify particular areas of concern.

	Target	Tolerable Failure Rate
> Performance		
response time		
accuracy		
availability		
> Physical safety		
> Legislative		
> Environmental		
> Economic[3]		
> Timing[4]		
> Community		
acceptance		
public relations		

3 Project budget limitations as well as required economic returns need to be considered.

4 Failure to meet deadlines for delivery of the design may have major consequences.

3. HAZARDS

The Hazards checklist provides an opportunity to identify the particular issues that might arise from failure of the system to operate as intended. The "Possible" column should be used to indicate whether or not a particular item is an issue – if it is, assessment is needed to determine how often it might arise. The SIL column allows the results of an integrity review to be recorded alongside the relevant hazard.

Again, some of the item entries may need to be expanded to give finer detail, and the item list should be treated as an example only – some items may be irrelevant while others may need to be added.

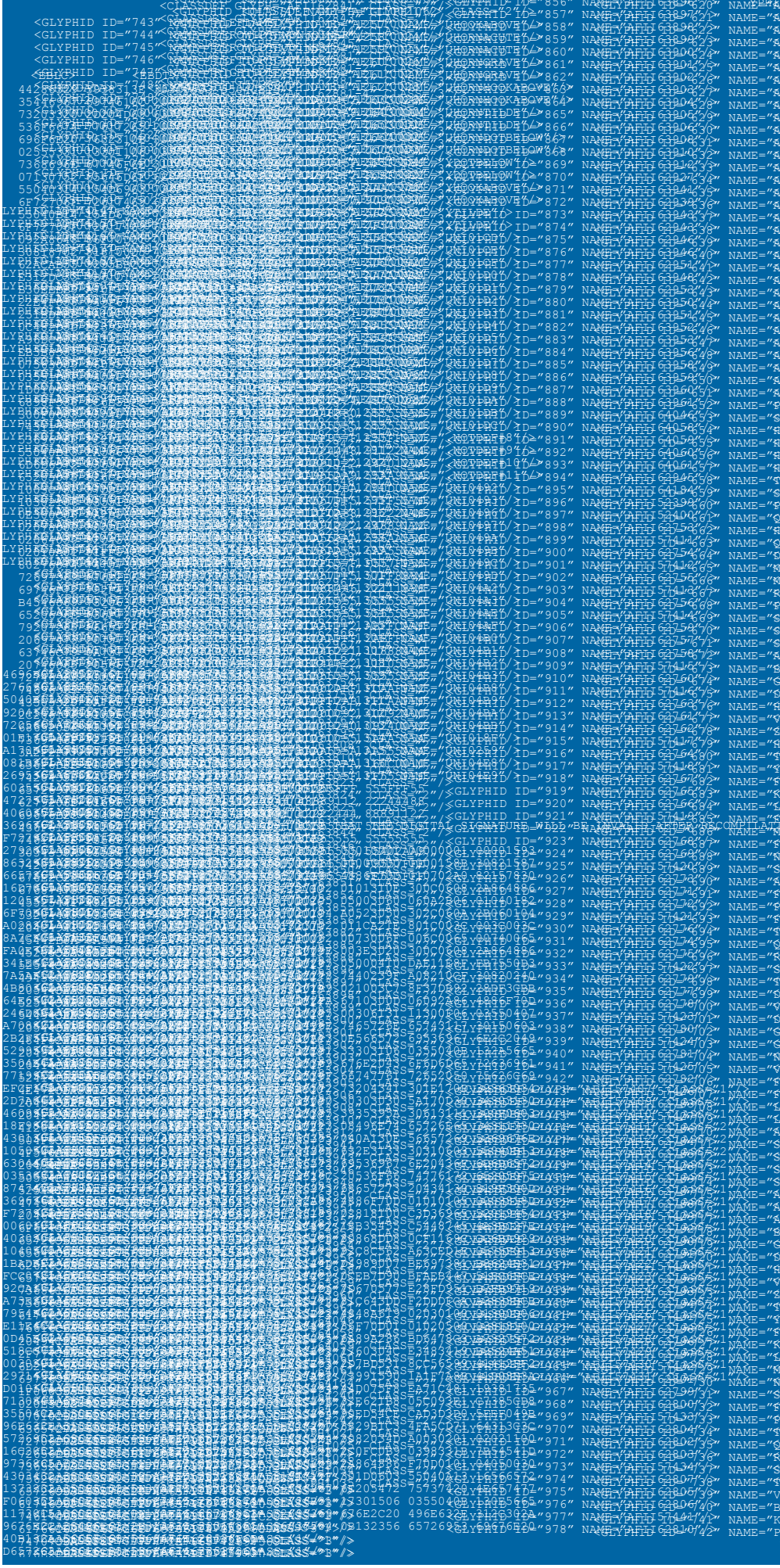
	Possible	Frequency	SIL
> Data entry			
maximum item length			
out of range			
timeout			
> Associated equipment			
failure			
> Unauthorised access			
physical			
electronic			
> Other software[5]			
> Misuse			

5 "DLL Hell" is an example where loading a later version of software can impact on the operation of programs already installed.

4. CONSEQUENCES

This list summarises possible areas where failure could result in a cost to the user or developer. It provides an indication of the areas of concern, and a basis for assessing the relative costs associated with each area.

	Assessment Method	Cost
> Service to users		
degraded		
halted		
> Health and safety		
> Environmental		
> Economic		
> Legal		
> Public relations		



For more information, contact:

IPENZ
ENGINEERS NEW ZEALAND

National Office
PO Box 12 241
Wellington 6144
New Zealand

T 64 4 473 9444
F 64 4 474 8933
E ipenz@ipenz.org.nz
W www.ipenz.org.nz