Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP)

Prepared by **Adelard**
for the Health and Safety Executive

CONTRACT RESEARCH REPORT
337/2001

Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP)

C Jones, R E Bloomfield,
P K D Froome and P G Bishop
Adelard
Coborn House
3 Coborn Road
London
E3 2DA
United Kingdom

This report was produced for the HSE project on 'Assessment of Software Components for use in IEC 61508-Compliant Safety-Related Applications'. The main focus for this project is 'software of uncertain pedigree' (SOUP) used in safety-related applications. This document reviews current assessment methods for SOUP and summarises the evidence required for their use according to IEC 61508 and other relevant standards.

This report and the work it describes were funded by the Health and Safety Executive (HSE). Its contents, including any opinions and/or conclusions expressed, are those of the authors alone and do not necessarily reflect HSE policy.

HSE BOOKS

First published 2001

ISBN 0 7176 2011 5

FOREWORD

HSE recently commissioned research into how pre-existing software components may be safely used in safety-related programmable electronic systems in a way that complies with the IEC 61508 standard.

Two reports resulted from this work:

a) *Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP)* CRR337 HSE Books 2001 ISBN 0 7176 2011 5

b) *Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications* CRR336 HSE Books 2001 ISBN 0 7176 2010 7

The first report summarises the evidence that is likely to be available in practice relating to a software component to assist in assessing the safety integrity of a safety function that depends on that component.

The second report considers how the available evidence can best be used within the framework of the IEC 61508 safety lifecycle to support an argument for the safety integrity achieved by a safety function.

Whilst these reports are the opinions of the authors alone and do not necessarily reflect HSE policy, HSE offers this work as an illustration of a principled approach to:

a) gathering evidence on the performance of pre-existing software components;

b) applying that evidence within the IEC 61508 framework; and

c) constructing a systematic and transparent argument for the safety integrity of a specified safety function.

HSE proposes to issue guidance on good practice in the use of software components in safety-related systems. HSE invites comments on the practicality and effectiveness of the recommended approach to achieving the above three goals, and on any other significant aspect of the safety integrity of software components that is not addressed by this work.

Please send your comments by 27 July 2001 to:

Dr E Fergus
Technology Division
Electrical and Control Systems
Magdalen House
Stanley Precinct
Bootle
Merseyside
L20 3QZ

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

The standard IEC 61508 [1] was specifically developed for industrial applications (e.g. process and manufacturing). In practice it has been adapted for use more widely in other sectors such as rail transport and in air traffic control. For pragmatic reasons, safety-related systems often make use of "software of uncertain pedigree" (SOUP), e.g. commercial operating systems, user interfaces, system libraries, etc. The software might have been designed specifically for safety-related tasks or be a product that was used in non-safety applications. This approach can reduce development time and offers the potential for higher reliability than a "bespoke" system provided the SOUP has been extensively used in previous applications. However, the use of SOUP can present severe problems in demonstrating compliance to IEC 61508, i.e. that the safety integrity of the system containing the SOUP can be *shown* to be acceptable for a given Safety Integrity Level (SIL).

This research study on the "Assessment of Software Components for use in IEC 61508-Compliant Safety-Related Applications" [2] was undertaken for HSE to address the issue of safety-related SOUP. The objectives are:

a) To survey practical and robust technical methods for assessing the safety integrity of SOUP.
b) To recommend criteria and evidence to justify the use of safety-related SOUP in a manner that complies with the principles of IEC 61508.
c) To consult relevant industry sectors to establish that the recommended approach is technically sufficient and capable of practical application.

This document covers the first objective.

To establish the overall context for this document, the remainder of the section discusses of what constitutes SOUP, the motivation for using SOUP on safety-related systems and introduces the main part of the document which examines assessment techniques for SOUP in relation to the requirements of IEC 61508.

1.1 SOUP CHARACTERISTICS

SOUP comes in a variety of forms:

- software components that form part of a program (such as libraries for graphics or numerical calculation and device drivers)
- standalone programs and utilities (e.g. compilers and stress analysis packages)
- high-level services that interact with multiple programs (e.g. operating systems kernels, networking, web servers and databases)
- complete systems where the hardware and software are integrated (such as PLCs, distributed control systems, medical devices and alarm systems) In practice, any safety-related system could contain several types of SOUP at different levels in its architecture, including cases where one SOUP component uses another SOUP component.

The main characteristics of SOUP are:

a) It already exists and cannot be re-engineered.

b) It is generic and hence contains functions that are unnecessary for the system application.

c) It is often subject to continuous change. A mass market SOUP will evolve to meet consumer demands and to match the competition.

While SOUP might be viewed as "field-proven", they are certainly not fault free. Some typical examples of problems encountered when using SOUP are:

- A bug in the timing algorithm in Microsoft Windows 95 and 98 that caused the computer to stop working (hang) after 49.7 days. The actual "hang-time" was 232 milliseconds. Pure black box statistical testing might not have found this problem or it could be masked by more frequent failures.
- The USS Yorktown used a network of NT machines for its control systems. There was a common mode failure and the ship was left powerless for several hours. The fact that the computer network failed is not surprising, both SOUP and bespoke systems fail. What it does emphasise is the need for a hazard analysis and a fault management strategy.
- One of the difficulties with SOUP is the problem of assessing the impact of additional functionality that is not used by the particular application. Normally the additional functionality of the SOUP can be determined from the documentation, but there are cases where entirely undocumented features are included in such products. This is graphically illustrated by the presence of "Easter eggs" in commercially available software. There are over 1821 known Easter eggs in a wide variety of products. One example is the flight simulator game hidden in Microsoft Excel.

Such problems are harder to deal with than in "bespoke software" because there can be severe limitations on independent scrutiny and assessment of the software. There could well be limitations on access to:

a) descriptions of development processes

b) design documentation

c) source code

d) fault histories

However the degree of access to such information is variable—it could be a "thick SOUP" or a "clear SOUP", e.g.:

- Commercial suppliers might provide information on development processes (e.g. compliance to ISO 9001) but refuse access to source code and design documents. However some commercial suppliers do grant restricted access for assessment purposes, and more often fault histories or lists of known faults for product versions are provided.
- The "Open Source" community produces widely-used products like the Linux operating system, the GNU C compiler, the Apache web server and the Perl scripting language. In these cases the source code is open for inspection and so are the fault histories but, due to the collaborative nature of these developments, documentation on the development process and design is sparse.

These characteristics of SOUP pose additional problems when attempting to justify the safety of systems containing SOUP:

- It is difficult to demonstrate compliance to best practice and applicable safety standards.
- Changes in SOUP may be inconsistent with the application software and hence cause new failures. This can occur especially when the change is not evident (e.g. change of SOUP within another SOUP component with no change in version number).

- The additional features in generic SOUP may affect safe operation. For example, a control system that permitted on-line modification could be open to deliberate or accidental introduction of erroneous control programs.

1.2 MOTIVATION FOR USING SOUP

While there are disadvantages in using SOUP, mass market SOUP can reduce the cost of development—indeed, it may be the only way of producing certain systems in a practicable time. Perhaps more significantly from a safety viewpoint, there are good theoretical and empirical reasons for believing that extensive use of a SOUP product will result in increased reliability (as faults are reported by users and corrected by the developers). This is illustrated in the following figure taken from our earlier research and published in the SOCS report [3] to the HSC.

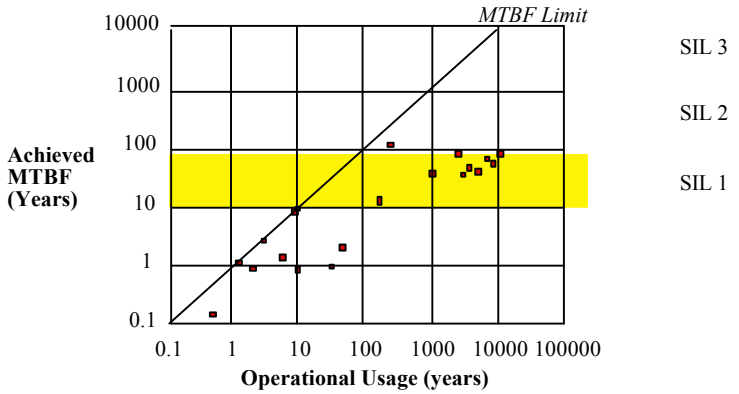

Figure 1: Software MTBF vs. usage for different industrial systems

The figure summarises software failure data from nuclear, chemical and aerospace (control and protection) industries. The claimed MTBF is limited to the total usage time. It can be seen that long term usage is strongly correlated with MTTF. Some systems exhibit growth rates 100 times less than the best case limit. It also seems possible to achieve IEC 61508 SIL1, and possibly SIL2 degree of reliability. Also, many safety-related systems do not need to achieve the reliability levels of SIL1 and for these SOUP is even more attractive.

Apart from the empirical evidence, a recent reliability growth theory [26] shows that times-to-failure can increase at least linearly with usage time, although growth can be reduced by a range of factors such as program size, quality of fault reporting and changes in product functionality. The theory predicts that worst case mean time to failure after a usage time t is:

$$MTTF(t) \geq e^x \, t \, / \, N^x \, d$$

Where N is the number of residual faults at initial release, d is the number of times the software fails before it is fixed, and e is the exponential constant (2.7181).

Ideally faults should be diagnosed and fixed immediately (i.e. d=1); poor diagnosis (d>>1) has the effect of "scaling up" the failure rate contribution of each fault.

If the software is upgraded with new functions this introduces an entirely new set of faults ΔN. As these faults will initially have relatively little usage time, the failure rate will be dominated by the new faults, i.e. the MTTF bound for a software upgrade approximates to:

$$MTTF(t + \Delta t) \geq e^x \, \Delta t \, / \, \Delta N^x \, d$$

where t is the time of the last upgrade, Δt is the usage time since the upgrade, and ΔN is the number of new faults introduced by the upgrade. So while reliability improves as "bug-fix" versions are introduced, reliability falls at the next major release when new functions are added, and there is no overall growth in reliability for a continuously changing SOUP product (indeed there can be a long-term decline).

The implication of this theory is that SOUP can be highly reliable if the following conditions apply:

- *Small programs*. Small programs have fewer faults (small N).
- *Good quality development process*. Reduces N.
- *Extensive field experience*. Large t.
- *Good fault reporting, diagnosis and correction infrastructure*. Reduces d.
- *Stable product*. Avoids "upgrade" effect that limits reliability growth.

In principle therefore, it is possible for SOUP to be reliable; the technical challenge is to identify well founded methods of assessment for *demonstrating* reliability in a safety-related application.

1.3 STRUCTURE OF THIS DOCUMENT

In the reminder of this document, we will examine the current assessment techniques that can be applied to SOUP for safety-related application. In Section 2 we review the assessment methods that can be applied to SOUP, while Section 3 discusses the approach to SOUP taken in IEC 61508 and the subset of recommended techniques which may be applied to SOUP. Section 4 describes the approach to SOUP taken in other relevant standards and compares the requirements for SOUP in 61508 with these other standards. In Section 5 we summarise the main results of the review and identify areas that need to be addressed.

2 ASSESSMENT METHODS FOR SOUP

It is important to control the risks posed by SOUP. To use SOUP, we need to establish that:

a) it correctly implements its stated functionality
b) there is no impact from the added functionality
c) it has no hazardous side-effects (e.g. writing to other memory locations, failing to release resources, etc.)

In this section we examine the main types of assessment that can be carried out on SOUP. These are:

- process assessment
- assessment of previous use
- "black-box" assessment
- "white-box" assessments
- third party assessments

2.1 PROCESS ASSESSMENT

2.1.1 Requirements from Standards

One indirect method of assessing SOUP quality is assessment of the software production process. Current standards regard process as an important factor in achieving software quality. In the context of the theory outlined in Section 1.2, a good process should have the effect of reducing the number of faults. A commercial SOUP product might contain 3 faults per 1000 lines of codes (3 faults/kloc) when initially released, while a more rigorous development (e.g. for aerospace) might achieve 0.1 faults/kloc.

There is heavy emphasis on acceptable process and techniques in IEC 61508, including a quality management system (QMS) such as one that complies to ISO 9000 [14], [15].

IEC 61508 does not take into account more advanced "process improvement" approaches to quality management. The CMM process improvement model [17] identifies 5 possible levels for process management; a similar scheme, SPICE, is being developed as an international standard that enables an organisation's process to be assessed [16]. However at this stage, there is very little empirical data to show that this has any impact on delivered fault density.

The process required by IEC61508 (and other safety standards) includes:

- A defined life cycle.
- Design and code reviews.
- Testing planned and performed.
- Work performed under a QMS
- Configuration management used.
- Safety analyses performed at appropriate point in the life cycle, and appropriately documented.
- Coding standards and procedures should be defined and adhered to.

Assessment of compliance to the requirements of IEC61508 by competent independent assessors can provide evidence of good process and this can be assured by certified assessors using CASS [6] or Factory Mutual [5] assessment processes. The process qualification could be used as indirect evidence of the integrity of a safety-related product.

2.1.2 Process evidence available for SOUP

Process evidence may be available for certain types of SOUP, but it is likely to be limited. Typically, generic commercial products are unlikely to meet IEC 61508 requirements even partially. Greater adherence may be possible for a product developed for a safety-related market (e.g. developed to IEC 880, DO178B). More commonly SOUP suppliers might be able to demonstrate compliance to QMS standards like ISO 9000 (and more rarely to one of the maturity levels in the CMM scheme). QMS compliance does indicate that some consistency in software development is being achieved and also gives some confidence that the software is under configuration control, but it is still relatively weak evidence.

Alternative evidence of good process is supplier "track record", i.e. whether the supplier has produced similar products over an extended period that have satisfactory performance. This provides direct evidence that the supplier's process is capable of producing adequate SOUP (regardless of compliance to standards). In practice, supplier reputation is often a key factor in the choice of SOUP.

2.2 ASSESSMENT OF PREVIOUS EXPERIENCE

2.2.1 Requirements from Standards

The major advantage of pre-existing SOUP over new software is that prior experience is available. Evidence from previous use is recognised in a range of safety standards. In IEC 61508, Annex C.2.10 states that:

A component or software module can be sufficiently trusted if it is already verified to the required safety integrity level, or if it fulfils the following criteria:

- unchanged specification;
- systems in different applications;
- at least one year of service history;
- operating time according to the safety integrity level or suitable number of demands;
- Demonstration of a non-safety-related failure rate of less than
- 10^{-2} per demand (year) with a confidence of 95% requires 300 operational runs (years),
- 10^{-5} per demand (year) with a confidence of 99.9% requires 690 000 operational runs (years);
- all of the operating experience must relate to a known demand profile of the functions of the software module, to ensure that increased operating experience genuinely leads to an increased knowledge of the behaviour of the software module relative to that demand profile;
- no safety-related failures.

NOTE 3 A failure which may not be safety critical in one context can be safety critical in another, and vice versa.

To enable verification that a component or software module fulfils the criteria, the following must be documented:

- exact identification of each system and its components, including version numbers (for both software and hardware);
- identification of users, and time of application;
- operating time;
- procedure for the selection of the user-applied systems and application cases;
- procedures for detecting and registering failures, and for removing faults.

These are quite strict conditions but, if they are complied with, it is reasonable to expect a SOUP to exhibit similar performance for the new application. The observed reliability of a system containing design faults is based on three main factors:

a) the number of faults
b) the size and location of faults
c) the input distribution (operational profile)

This is illustrated in the following diagram.

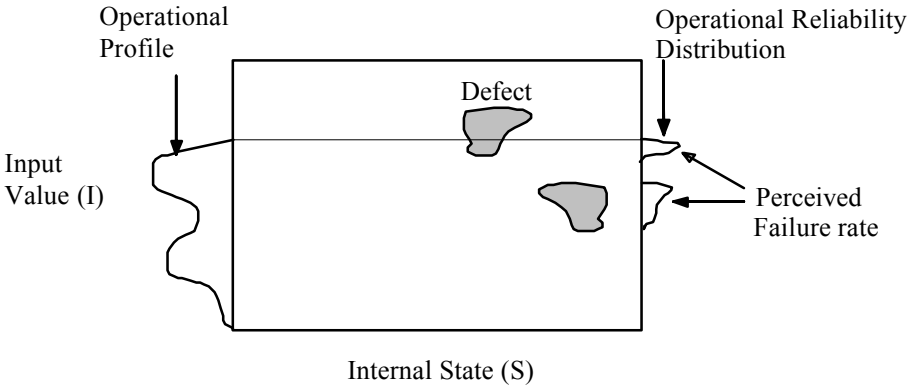

Figure 2: Illustration of the Software Failure Process

It is clear from the diagram that an alteration of the input distribution (I) (also known as the operational profile) could radically alter the operational failure rate of the system. This explains the stated criteria in 61508: the first three conditions reduce the chance of faults remaining in the program:

- unchanged specification – This implies there are no new software functions and therefore no new faults.
- systems in different applications – The use of different profiles reduces the chance of having undetected faults in unused parts of the input space.
- at least one year of service history – This gives time for time-sensitive faults to be revealed and time for any reported faults to be corrected

The fourth condition is a statistical justification of observed reliability from direct observation of its field performance:

- operating time according to the safety integrity level or suitable number of demands

2.2.2 Operating History Evidence for SOUP

There are a number of problems in using past information for SOUP to derive valid estimates of performance, and to adequately estimate reliability we need to ask the following questions for a SOUP system:

- *Are failures reported every time they occur?*
  In our experience with safety-related systems, the first failure of a particular type is reported, but not subsequent ones. Quite often the manufacturer records each fault (and possibly when it was first reported), rather than *how frequently* the fault is activated. This can be an important practical obstacle to justifying SOUP.
- *Is the data available to the user?*
  Quite often the fault reports and user problem reports are confidential to the supplier.
- *Is the* past *mode of use similar to the current application?*
  The operational profile will determine how often defective parts of the software are seen to fail. Quite small changes in operational profile can result in major changes in failure rate. Generally speaking there is little information on mode of use. The only reliable source might be from within the user organisation if the SOUP is used in a range of different applications.

In the face of these difficulties it may be necessary to resort to more indirect methods of assessing the likely reliability using the worst case bound formula in [26] and as outlined in section 1.2 above i.e.:

$$MTTF(t) \geq e^{x} t / N^{x} d$$

This approach requires less data and this makes estimation more feasible. Let us apply the approach to the new Windows 2000 product. There are reputed to be 6400 bugs in Windows 2000. In fact with the reported 50M loc we would expect many more than this (i.e. around 150 000, but of course not all bugs have been discovered yet). The operating system has been beta tested by 750 000 users. If each copy of Windows 2000 was used for 6 months (and only 1 machine), and the code had a generic density of 3 faults/kloc and Microsoft could only diagnose a fault after 10 bug reports, the MTTF from formula (2) would be:

$$\text{MTTF} \geq (750{,}000 \times 2.7) / (50{,}000 \times 3 \times 10) \sim 8 \text{ months}$$

This would support their claim that it is the most reliable Microsoft OS yet, although such estimates would require additional evidence to back up the fault density estimates, and the fault diagnosis probability. Typically real-time programs are hard to diagnose so the reliability could be lower than the prediction derived above.

The prediction method does however give a “ball park” figure that can be used to check the credibility of any operational reliability claims derived from field experience.

2.3 ASSESSMENT WITH BLACK BOX TECHNIQUES

2.3.1 Applicable Techniques

The black box techniques that are applicable for assessment of SOUP software are basically varieties of testing. They include:

- statistical testing
- interface testing

- boundary value testing
- stress testing

For most testing it is necessary to have a specification that is sufficiently precise that the expected results of the test are defined. For complex systems it may be extremely difficult to compute the result of a given test without some diverse automated mechanism which may be difficult or impractical to obtain.

Statistical testing gives a strong argument that a particular SIL is obtained, but typically requires a knowledge of the operational profile and a way of determining the expected results from a large number (at least 1000) test cases. Statistical testing is also useful in assessing a SOUP product in association with the intended hardware.

Interface testing is most relevant when a SOUP component is used in part of a modular design. It is useful to test the response to unexpected or out of range input values. It is also relevant for GUI applications.

Boundary value testing is relevant where the design or documentation of the SOUP defines input sets where discontinuities in the results are expected. At the most detailed level, where source code is available, boundary value testing could exercise all points of choice (if or case statements for instance) with values for each alternative. Boundary value tests can also establish that functions cannot be executed outside a defined range of validity: this may be important depending on the application.

Stress testing aims to examine the behaviour of a program under conditions of low resources. This may explicitly test potentially dangerous failures of the application depending on its intended use or it may be done as a "quality check". If the software behaves well under extreme conditions we may infer that it was well-engineered and is of good quality.

2.3.2 Limitations

The types of testing in current use are all forms of functional testing that help to demonstrate one top-level safety property, i.e. that the software implements the intended function.

However there are other attributes of the software such as:

- timeliness
- throughput
- robustness (to values outside specified range)
- fail-safety/fault tolerance
- security
- usability

Performance factors such as timeliness and throughput can be critical to safety, but are not considered explicitly in IEC 61508. Similarly, it would be feasible to derive black box tests at the PES level for many other attributes that affect safety.

In addition we need to assess whether the SOUP component has any "side-effects" such as writing outside its own memory space. As yet there are no accepted black box methods for checking the "isolation" of different software.

2.4 ASSESSMENT WITH WHITE BOX TECHNIQUES

The following techniques may be used in cases where the source code is available. Source code may be available because software is “open source”. Some companies (e.g. Sun Microsystems) will licence source code to users. Companies may supply source code for assessment, typically of suitability for a specific application.

It may also be possible in some cases to reverse engineer the code and documentation. This is an active and well-researched area driven in part by the continuous need to deal with legacy systems and the past challenges of Y2K [32]. A review of reverse engineering tools is provided at [30]. One class of reverse engineering tools is decompilers that translate an object file into a compilable source file. Decompilers exist for a range of languages such as C, Java, Foxpro databases, and Windows help files [31].

If a safety argument relies also on field experience or it is not possible to correct any errors found in the source code for some other reason, then white box analysis is less useful. Therefore we would expect some preconditions to hold before further analysis is carried out.

- The code should have significant experience (likely to be at least a few years) of use in a similar domain as that intended.
- Faults should have been reported and corrected in later versions of the software.
- The original development process should have been sufficiently good that not too many faults overall have been found.

2.4.1 Static Analysis

Available static analysis tools tend to be targeted at Ada and C/C++ software. In practice SOUP is predominately written in C or C++, although from a safety perspective, Ada is a more strongly-typed and better-defined language and would be a better basis for producing safety-related software. However the deficiencies in C can be compensated for by additional checking tools (e.g. to check for data type consistency).

The following subsections summarise the various categories of static analysis, together where appropriate with information on the tools available to support them. There is some overlap between the categories.

Language subset / programming standards compliance

Various constraints on languages (e.g. C language subsets like SaferC [24], and MISRA [25]) have been proposed to reduce the dangers of misuse in critical applications. These are more effective when mechanically enforced. In SOUP, violations do not necessarily indicate errors, but suggest places for further investigation (including the use of more sophisticated static analysis tools).

Tools that use a strict rule set are likely to find a lot of non-compliances in pre-existing SOUP unless it has been written to a specific standard. Some tools that perform such analyses are listed below.

Table 1 : Tools for checking language subsets

Name	Description	Platform	Comment
Lint	C compliance checker	All Unix	Aims at detecting major breaches of C style and sense, hence fairly liberal.
PC lint	C compliance checker	Windows, Unix version available	Also does some type and data flow analysis, including uninitialised variable detection.
Assent	MISRA compliance		
Safer C	MISRA and "Safer C" compliance	Linux/Windows/ SPARC Unix	Covers the two main safe subsets.
Logiscope	User definable rule set	Windows 95/NT, Solaris, HP-UX, AIX, Digital Unix	Primarily checks software complexity.

Control and data flow analysis

Control and data flow analysis is aimed at detecting dead code and the use of variables without initialisation. This can also be done by the code slicers. Some of the subset compliance checkers may need to do some code and data flow analysis too.

Table 2 : Tools for code and data flow analysis

Name	Description	Platform	Comment
LDRA	Code and data flow analysis	PCs and Unix	Supports Fortran, C and Ada
MALPAS	Performs code and data flow analysis. Also capable of extracting semantic description from code, and comparison with abstract semantic definition.	VAX VMS. Port to Windows NT some time this year	Supports C and Ada and other languages (given a suitable "front end" converter)

Code slicers

These tools analyse the program into control and data flow structure graphs. The tool uses the program structure graph to extract a "slice" of a program that shows how a selected variable value is derived at a particular point in the program. Related analyses allow the subsequent uses of a variable to be shown, or the connections between one variable and another to be traced. Side effects include location of dead code.

Unlike some of the other analyses, this is not an end in itself. These tools are sold mainly as code understanding aids, perhaps more relevant when dealing with long and unstructured routines. The related academic work describes particular uses of the tools, for instance to detect flows from secure variables to insecure variables and so detect security flaws. We would need to think carefully about what questions we would want to ask such a tool.

Table 3 : Tools for slicing code

Name	Description	Platform	Comment
Code surfer	Code slicing and related analyses.	Solaris. Futher Unix platforms and PC versions to follow soon	C language only Relatively new tool. Window based, fully automatic
Unravel	Code slicing for C code.	Unix/Posix with ANSI C and X11 or later.	C language only. Coverage of C appears fairly complete

Type analysis

Type analysis checks that operations receive operands of expected types. This is primarily targeted at C code, since Ada is already strongly typed. Tools that perform this analysis include:

Table 4 : Tools for type analysis

Name	Description	Platform	Comment
PC Lint	C compliance checker	Windows, Unix version available	Checks type consistency
ProQA	C compliance checker	Windows, Unix version available	Checks type consistency

Value range checking

If the possible range of values for variables can be identified by static analysis, then the information can be used to checking for arithmetic overflow, and addressing errors in arrays and structures. This is a relatively new field but one tool is available.

Table 5 : Tools for checking for range errors

Name	Description	Platform	Comment
PolySpace	Identifies potential range overflows	Windows, Unix version available	For C and Ada

2.4.2 Test Coverage Analysis Techniques

Where source code is available, code can be instrumented and tests run in such a way that the code statements executed by the tests are recorded. Tools exist for this (e.g. LRDA Testbed), and some compilers include tools which can be used (perhaps indirectly) to test coverage.

Coverage can be by function: i.e. test to see if every function likely to be used by a given application is executed by a given test suite.

Coverage can be of every statement. It is notoriously difficult to obtain 100% statement coverage as products often contain code which is "dead" (i.e. impossible to activate). It may also contain "error recovery code", which may only be executed if an external value is outside the expected range of operation, or if an internal error occurs, i.e. perhaps a processor error or a code error introduced by program maintenance.

Test coverage analysis can determine how comprehensive an existing set of tests are and hence how much confidence can be gained from the passing of these tests. It can also be used to validate that a certain amount of test coverage is attained by a specially devised set of tests. If very close to 100% statement coverage can be achieved this gives quite high confidence in the correctness of the program although errors activated by particular values cannot be ruled out.

There are many vendors of test coverage tools and many languages are supported, e.g.:

Table 6 : Tools for measuring test coverage

Name	Description	Platform	Comment
Cantata	Test coverage (block, branch, condition)	Windows, Unix version available	For C language
Adatest	Test coverage (block, branch, condition)	Windows, Unix version available	For Ada
LDRA Testbed	Test coverage (block, branch, LCSJ)	Windows, Unix version available	For Fortran, Ada and C

See [32] for a more comprehensive set of test tools.

2.4.3 Fault Injection Techniques

Fault injection techniques generally assess indirectly the effectiveness of a given set of tests by deliberately creating code with known faults. The resulting source is re-compiled and re-tested. If a high proportion of the introduced faults are detected by the testing, then this shows indirectly that the testing is thorough. If the introduced faults are statistically similar to the likely faults existing in the code, then a statistical estimate of the number of faults in the unchanged code can be made.

A similar form of testing is perturbation testing [27], [28] where computed data values are perturbed with specific fault injecting code. This can be used to assess the quality of the test process, but can also be used to assess the fault detection and fault tolerance capabilities within the software or external PES hardware.

2.5 THIRD PARTY ASSESSMENT

Third party assessment is another method for gaining assurance about SOUP. In this section we examine the specific conformance to IEC 61508 that might be obtained as well as more general product and process type-approval.

2.5.1 Conformance to IEC 61508

CASS

CASS (Conformity Assessment of Safety-related Systems) is a scheme to provide a structure so that third-party accredited certification bodies can offer conformity assessment certification for safety-related products to IEC 61508. There are five assessment types:

- Type 1—Application independent (component assessment)
- Type 2—Application specific products
- Type 3—Operations and maintenance assessment
- Type 4—Safety requirements assessment
- Type 5—Functional safety capability assessment

Functional safety capability assessment is the first to be launched, and is a process assessment of an organisation's functional safety capability.

Accreditation will be to EN45011/ISO guide 65, by UKAS and potentially other accreditation bodies. Individual assessors will operate through accredited certification bodies, but registration will apply to individuals not organisations, and assessors need not be employed by a certification body. There will be one grade of assessor. Assessment will initially be by interview, although a course and examination will be introduced later.

At the time of writing, there are several stages to go through before UKAS accreditation can be set up, including developing a certification framework and working with a certification body to refine the approach. The first certificates may be awarded towards the end of 2000.

Factory Mutual

Factory Mutual Research will issue certificates for compliance with IEC 61508. It expects applicants for certificates to use a library of either trusted or verified software components. Evidence to support a trusted claim should include data on the number of hours or demands as

appropriate, and the times at which failures have been noted. Detailed formal or semi-formal analysis, using a well-established model of software reliability, is also required.

Factory Mutual Research has issued certificates for two configurations of Moore Process Automation Solutions' Quadlog/Prosafe, one "fit for use in a SIL 2 safety application", and one "fit for use in a SIL 3 safety application".

2.5.2 Product Type-Approval

Approval of application software

TÜV Rheinland carries out quality testing and certification of application software packages against the manufacturer's declaration in accordance with DIN 66285, leading to "DIN Approved" and "RAL Software" quality marks. It will also test usability against ISO 9241 (leading to an "Ergonomics Approved" mark) and readiness for business application (leading to a "Qualified Software" mark).

Approval of embedded software

Type-approval of PLCs covering software and hardware is carried out by TÜV Rheinland and TÜV Bayern in Germany. Approvals are carried out to the safety requirements in DIN V 19250 [7] and DIN V VDE 0801 [8] and range from either AK 1 (lowest) to AK 8 (highest). Alternatively, fault tolerance can be checked, when the approval ranges from SK 5 (lowest) to SK 1 (highest) (see [9]).

Testing can be carried out concurrently with product development, or after completion of the design. Reliability, availability and quality assurance are taken into account during the certification process, which includes a theoretical analysis (FMECA, worst-case analysis and simulation) and a practical qualification (e.g. vibration and EMC).

The following manufacturers have TÜV type-approved PLCs:

Table 7 : Manufacturers of TÜV type-approved PLCs

ABB	ABB Adtranz	ABB August
ABB Procontrol	GE Fanuc	Hima
Honeywell	ICS	Kongsberg Simrad
Moore	PILZ	Schoppe & Faeser
Siemens	Triconex	Yokogawa Prosafe DSP
Yokogawa Prosafe PLC		

Details can be found at http://www.isep.de/plclist.htm

2.5.3 Process Assessment

Capability Maturity Model

The Capability Maturity Model® for Software was developed by the Software Engineering Institute at Carnegie Mellon University [10]. It is intended to help software developers improve the maturity of their processes from ad hoc, chaotic processes to mature, disciplined ones. The CMM is organised into five maturity levels:

1. *Initial*—The software process is ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
2. *Repeatable*—Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. *Defined*—The software process for both management and engineering activities is documented, standardised, and integrated into a standard software process for the organisation. All projects use an approved, tailored version of the organisation's standard software process for developing and maintaining software.
4. *Managed*—Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
5. *Optimising*—Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organisation should focus on to improve its software process.

The emphasis in the CMM is on measuring the details of the software engineering process, and not on the adequacy of the results of the process. One of the criticisms of the approach is the lack of any empirical evidence correlating the maturity levels with objective measures of behaviour of the products. It is also criticised for being too prescriptive in its definition of the processes to follow (e.g. in requiring a software engineering process improvement group) and in the design of the marking scheme used to assess organisations.

The fact that many software developers have a QMS assessed against ISO 9001 may reduce the importance of the lower levels of process maturity. Indeed, the requirement for ISO 9001 or equivalent QMS in safety-related projects may mean that all developers should be at CMM levels 2–3.

A more detailed critique and discussion of the strengths and weaknesses of the CMM is given in [11][12], and a rebuttal of these criticisms in [13]. Despite these objections, the CMM has had a major impact on the awareness of process maturity, and the assessment business is booming, with extended and revised models being developed.

SPICE

SPICE (Software Process Improvement and Capability Determination) [16] is an international ISO/IEC JTC1 standardisation project to develop a harmonised approach from the current variety of process improvement approaches.

In June 1995, Version 1 of a draft standard was released for international ballot. Following this ballot, the documents have been carried through the international standardisation process and have been published as ISO/IEC TR 15504:1998—Software Process Assessment. Work has

now commenced to revise the TR with a view to ultimate publication as a full International Standard.

The SPICE Trials are now the principal focus of the project. More information about the trials is available at http://www.sqi.gu.edu.au/SPICEtrials

A SPICE Assessor Training Syllabus has also been released.

3 IEC 61508 COMPLIANCE FOR SOUP

In this section we examine the specific requirements in IEC 61508 for SOUP and also how to apply the generic requirements of the standard in a situation where a SOUP component is being considered.

3.1 PROVEN IN USE

IEC 61508 defines the notion of "proven in use". This is to be applied both to the use of pre-existing components, packages and operating systems which will be directly executed in the use of a system, and to software tools such as compilers, testing tools and configuration management tools where a fault in the tool may result in an incorrect software system.

It is necessary to first assess the impact of likely faults in SOUP on the safety functions of the system, and then to choose a system where evidence is available.

IEC 61508 requires that a component or software module can be sufficiently trusted if it is already verified to the required safety integrity level (i.e. the development process followed IEC 61508), or if it fulfils the following criteria:

a) The specification is unchanged.
b) It has been used in different applications for at least one year.
c) The software should exhibit a rate of non-safety-related failures appropriate to the SIL required (e.g. 300 years for SIL2 with 95% confidence).
d) The operating experience must relate to a known demand profile.
e) No safety-related failures occurred.

The exact identification of systems for which operating experience is claimed should be documented including version numbers for software and hardware. Also required is documentation of users, time of application, operating time and procedures applied for detecting, reporting and correcting faults.

IEC 61508 stresses the distinction between "merely pre-existing" and "pre-existing and also proven-in-use". A pre-existing package or component can be regarded as proven-in-use only when there is adequate documentary evidence, based on previous specific use, to demonstrate that the likelihood of any failure due to systematic faults is low enough to achieve the required safety integrity level of the safety function in which the package is reused. In short, it is not sufficient to claim that operating system X is suitable for a safety-related application because several million copies have been sold over the years.

3.2 APPLICABLE TECHNIQUES FROM IEC 61508

The following tables list the techniques from IEC 61508 which may be applied to SOUP. It is not necessary to apply these techniques if a piece of software can be said to be "proven in use". It is unclear how to interpret IEC 61508 if the conditions for "proven in use" are not met and it is desired to apply some of the techniques recommended as appropriate for the SIL of the system component.

We first list the small number of software engineering techniques and design strategies which can be applicable to any SOUP.

Table 8 : Engineering techniques and design strategies for SOUP

Section	Technique	Comments on the application to SOUP
C.3.4	Safety bag	A safety monitor may be used in conjunction with SOUP
C.3.5	Software diversity (diverse programming)	Could theoretically use diverse SOUP (but need to be sure they are diverse and it is not clear how much improvement is gained). Could use a diverse bespoke component to support a critical part of SOUP
C.5.1	Probabilistic testing	Applicable to SOUP, but difficult to do sufficient testing to assure higher levels. Also need reliable alternative determination (oracle) for large number of test cases
C.5.2	Data recording and analysis	Gives feedback on failures occurring in field operation. Applicable to SOUP, but needs to be of high quality to demonstrate reliability
C.5.3	Interface testing	Essentially focused testing, needs reasonably precise knowledge of interface specification
C.5.4	Boundary value analysis	Needs detailed knowledge of specification (when software is a black box). In white box testing requires analysis of the code
C.5.5	Error guessing	Needs expert judgement and knowledge of application
C.5.19	Process simulation	Essentially testing in simulated operational situation. Provides a realistic operational profile, can be valuable for continuously operating systems (e.g. process control). Hard to accumulate sufficient tests to get high degree of confidence in reliability
C.5.21	Avalanche/stress testing	Could be applied to SOUP, helps to demonstrate robustness to overload
C.5.24	Software configuration management	Essential in applications using SOUP, should record exact versions of SOUP tested, installed etc. If manufacturers' configuration management is doubtful, it is important to save and label original installation files.

Obviously any technique referenced may have been used in the design and development of SOUP, and a developer may be able to make documentation available to demonstrate this use. However some further techniques may be applied to "Clear SOUP" where source code is available.

Table 9 : Techniques for "clear" SOUP

Section	Technique	Comments on the application to SOUP
C.4.3	Certified tools and certified translators	Source code may be re-compiled with an assessed compiler to produce a trusted version.
C.5.6	Error seeding	Possible, by seeding errors and recompiling source code.
C.5.8	Structure based testing	Access to source code can make testing more comprehensive or efficient.
C.5.9	Control flow analysis	Technique may find errors missed from testing, perhaps unusual execution paths, can increase confidence in quality of code.
C.5.10	Data flow analysis	Technique may find errors missed from testing, perhaps unusual execution paths, can increase confidence in quality of code.
C.5.11	Sneak circuit analysis	Technique may find errors missed from testing, can increase confidence in quality of code.
C.5.12	Symbolic execution	Potentially can "test" whole classes of input values. For relatively simple programs where tool support is available, this could be very valuable.
C.5.13	Formal proof	Unlikely to be possible unless formal specification is also available, difficult anyway with most languages, and with code not developed with proof in mind. Only likely to be appropriate for SIL 4.
C.5.15	Fagan inspections	Can increase confidence in quality of the code, i.e. that the design implements specification, complies with good practice, etc.
C.5.16	Walkthroughs/ design reviews	Similar to above.
C.5.17	Prototyping/ animation	Animation may be more appropriate and/or convincing than testing for some SOUP.

There are also generic safety engineering strategies that help identify/limit the risks of SOUP (e.g. Hazops, partitioning the system according to safety criticality of components).

3.3 DISCUSSION

IEC 61508 identifies a range of techniques that are applicable to SOUP and also what SIL they are recommended for. However the guidance is just a list with no indication of what techniques will form a sufficient set to justify safety.

In particular it seems to focus mainly on quality control and functional correctness. In practice there are a range of safety-critical attributes like timeliness, security etc. that are needed to assure safety.

A better approach would be to have an identified set of safety attributes, and set of techniques that contributed to each attribute. This would enable a coherent justification to be made where the evidence from specific techniques is linked to top-level attributes.

There is also a problem with using a single SIL value for all the software in the computer. If there is sufficient isolation for the SOUP, the SOUP could be of lower integrity than the "front-line" safety-related software. There is no mechanism in IEC 61508 for assessing the criticality of SOUP components.

4 SOUP IN OTHER STANDARDS

In this section we compare the approach to SOUP adopted in a number of other standards:

- FDA 1252, for software medical equipment
- IEC 60880, for software in nuclear plant
- Def Stan 00-55 and 00-56, for software-based defence equipment

Finally, we discuss the requirements of a recent publication from the CEC setting out the common position of European nuclear regulators.

4.1 FDA 1252 "OFF-THE SHELF SOFTWARE USE IN MEDICAL DEVICES"

4.1.1 General

The guidance states that SOUP (or OTS software in the guide) is commonly being considered for incorporation into medical devices. The use of OTS software in a medical device allows the manufacturer to concentrate on the application software, but the guide warns that software intended for general purpose computing may not be appropriate for a medical device, and the medical device manufacturer using OTS software generally gives up software life cycle control, but still bears the responsibility for the continued safe and effective performance of the medical device.

4.1.2 Overall Approach

The guide is basically risk-based, but takes the position that software failure rates cannot easily be predicted. It therefore takes a consequence-based approach, using the term hazard analysis rather than risk analysis to reinforce this.

Risk is described as a minor, moderate or major level of concern depending on whether there is expected to be, respectively, no injuries, injuries, or fatalities, arising from failures or design flaws.

The overall approach of the guide is to make recommendations on a basic level of documentation needed for all OTS software used in medical devices, and provide a detailed discussion on additional (special) needs and responsibilities of the manufacturer when the severity of the hazards from OTS software failure become more significant. The decision diagram employed is reproduced in Figure 3.

4.1.3 Basic Documentation

The OTS software basic documentation should answer the following:

a) What is it? (Provide title, version, etc., and state why appropriate for this device.)
b) What are computer system specifications? (Specify hardware, OS, drivers, etc., including version information.)
c) How will you ensure appropriate actions are taken by the end user? (Specify training, configuration requirements, and steps to prevent the operation of any non-specified OTS software.)
d) What function does the OTS software provide in this device?

e) How do you know it works? (Describe testing and lists of faults.)

f) How will the OTS software be controlled? (This should cover installation, configuration control, storage, and maintenance.)

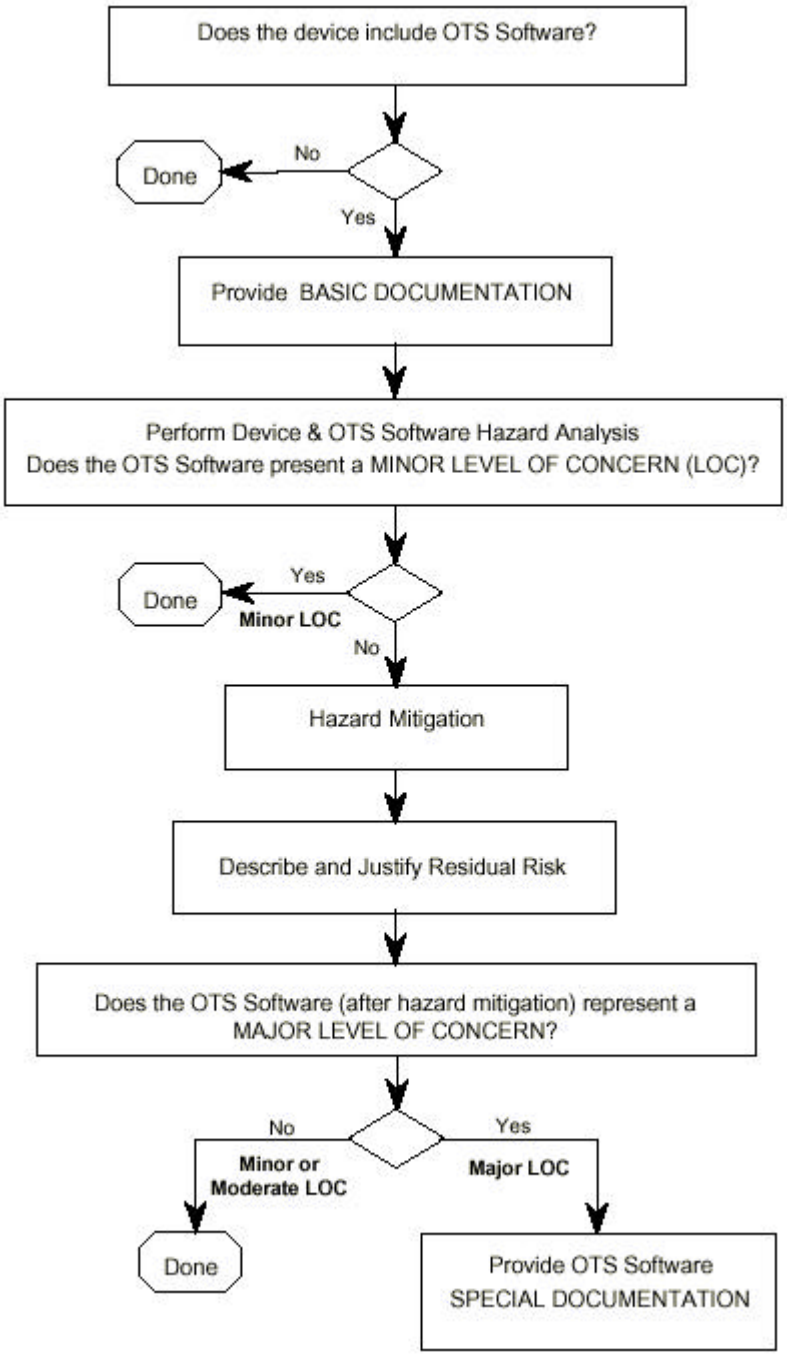

Figure 3: Decision diagram for FDA 1252

4.1.4 Hazard Analysis and Mitigation

The manufacturer is expected to perform an OTS software hazard analysis as a part of a medical device (system) hazard analysis. This should produce a list of all potential hazards identified, the estimated severity of each identified hazard, and a list of all potential causes of each identified hazard.

Where the hazard analysis identifies the need for it, mitigation should be considered by means of (in order of preference) design (or redesign), protective measures (passive measures), and/or warning the user (labelling). The guide contains a list of injury reduction countermeasures running from "prevent accumulation of the energy" and "reduce the amount of the energy delivered" to "provide rapid emergency response to injury" and "improve medical care and rehabilitation after the injury".

A detailed discussion of the residual risk after mitigation should be provided. Operational experience with the use of the OTS software can be submitted as part of the justification of residual risks.

4.1.5 Special Documentation

Special documentation, where required, should:

a) Provide assurance that the product development methodologies used by the OTS software developer are appropriate and sufficient. The guide recommends an audit of the OTS software developer's development methodologies, and states that OTS software may not be suitable for a medical device application if it represents a major level of concern and an audit is not possible.
b) Demonstrate that the verification and validation activities are appropriate and sufficient. Verification and validation activities include those performed by the OTS software developer and the medical device manufacturer.
c) Demonstrate how maintenance and support of the OTS software will be continued should the original developer terminate their support.

4.1.6 Device Master Files

The FDA operates a scheme whereby SOUP software vendors who wish to make their software available for use in medical devices, but who do not want to share confidential or proprietary details with medical device manufacturers, may provide information on development, validation and known faults to the FDA in a *device master file*.

The SOUP vendor can then grant permission to specific device manufacturers to reference the master file in their safety submissions.

4.2 IEC 60880, FIRST SUPPLEMENT

4.2.1 GENERAL

This Supplement to IEC 60880 provides requirements for the software for computer-based safety systems in nuclear power plants. It contains a section on SOUP, which it calls "pre-developed software" (PDS). It acknowledges that the use of PDS can be beneficial to productivity and the reliability of the system when these items are of suitable quality and introduced in a proper manner; benefit from similar operating experience can be claimed and the reuse of validated PDS can increase confidence in the reliability of the system.

4.2.2 Overall Approach

The PDS evaluation and assessment process includes:

a) An evaluation of the functional and performance features of the PDS and existing qualification documentation.
b) A quality evaluation of the software design and development process.
c) An evaluation of operating experience if needed to compensate for weaknesses in demonstration gained from (1) and (2) above.
d) A comprehensive documented assessment of the evidence from the above evaluations, and associated complementary work, which will enable the PDS to be accepted for use in the system.

The overall approach is illustrated in Figure 4.

4.2.3 Evaluation of Suitability

This element of the evaluation compares the system specification with the PDS specification and user documentation at a "black box" level. Analysis or test is required if the functional, interface or performance characteristics of the PDS are not explicitly defined. If the PDS does not meet the requirements of the system specification, it should be used only if it can be modified in an IEC 60880-compliant manner.

The suitability evaluation should identify any additional functions that are included in the PDS but are not needed by the system, and the measures to ensure that these functions do not interfere with safety functions.

The PDS should be under configuration management and its version and configuration should be precisely defined.

4.2.4 Quality Evaluation

This element of the evaluation takes a "white box" approach, based on the design and software quality plan documentation of the PDS, possibly with analysis of its operating history. It compares the importance to safety of the PDS with quality documentation, including the software quality plan, specification, design, coding and maintenance documents, the integration plan, and verification and validation plans and tests.

The level of assurance to be achieved by the quality evaluation will be different for the three safety categories, with category A requiring the highest assurance (see 8.2.1 of IEC 61226).

Documentation of operating experience should be available if necessary to compensate for lack of the above documentation or to justify practices differing from those of IEC 60880.

4.2.5 Evaluation of Operating Experience

This type of evaluation is to provide evidence of suitable operating experience to compensate for deficiencies detected in the quality evaluation. The evidence required is:

a) The methods for collection of data on operating experience, including recording the PDS version's operating time and operating history.
b) The operational history of findings, defects and error reports.
c) The operational history of modifications made for defects or other reasons.

Operating experience should be under conditions similar to the conditions during intended operation. When operating time of other versions is included, an analysis of the differences and history of these versions should be made.

Operating experience should be considered as suitable when the following criteria are met:

a) The PDS has achieved a sufficient accumulated operating time, taking account of the statistical relevance of the data. The rigour of the analysis of operating experience should be consistent with the safety category of the system functions.
b) No significant modifications have been done and no errors have been detected over a significant operating time on several sites or applications.
c) The PDS has preferably operated on several installations.

1 Suitability evaluation

System specification documentation	**Required Input documentation**	PDS specification & user's documentation
Comparison of the system & PDS Specifications	**Evaluation requirements**	Identification of modifications and missing points
	Conclusions	
The PDS is suitable	Complementary work is needed	Ought to be rejected

2 Quality evaluation

	Req. Input documentation	(operating history documentation)
Design documentation	Life cycle documentation	
	Evaluation requirements	
Analysis of design	Analysis of the QA	Identification of missing points
	Conclusions	
The quality of the PDS Lifecycle is appropriate or	Additional test and documentation is required or	The PDS ought to be rejected
The needed modifications of the PDS are feasible	Operating experience evaluation required	

3 Evaluation of operating experience

	Req. Input documentation	
Collection of data	Operating time	History of defects
	Evaluation requirements	
	Conclusions	
Sufficient operating experience	Operating experience not sufficient yet	The PDS ought to be rejected

4 Comprehensive assessment

The quality of the PDS is appropriate	The needed modifications are done

5 Integration in the system and maintenance

Figure 4: Outline of the qualification process in IEC 60880 Supplement 1

4.3 UK DEFENCE STANDARDS 00-55 AND 0056

4.3.1 General

Defence Standard (DS) 00-55 addresses safety critical (SIL 4) software, with guidance on how to modify the requirements for software of lower SIL. DS 00-55 contains a clause on SOUP, which it calls "previously developed software" (PDS). It recognises that the appropriate reuse of well proven software can be of substantial benefit to the integrity of safety-related software.

DS 00-55 is used within the context of DS 00-56, which addresses safety management. DS 00-56 includes a procedure for the allocation of SILs, which is consequence based for the principal source of mitigation, and risk based for other mitigation. In DS 00-56, SOUP is covered by the requirements on non-developmental items (NDIs). The standard requires a safety case for NDIs, and provides guidance on retrospective application covering safety planning, production of a hazard log, and safety analysis, including evaluation of existing safety analysis information and the use of service history.

4.3.2 Overall Approach

DS 00-55 is basically targeted at the most safety critical software, and taken at face value adopts an uncompromising approach to the relaxation of the requirements for new software. It requires that:

a) All PDS should be identified, and justified in the software safety case. The justification should include a safety analysis of the PDS.
b) PDS to be used in the final delivered equipment should conform to the requirements of the standard for new software.
c) Or, reverse engineering and V&V activities should be carried out on any PDS that has not been produced to the requirements of the standard. Reverse engineering means the conduct of retrospective activities covering specification, design, verification and validation to the same standard as new software, and requires access to the source code, design and test documentation.
d) All changes to PDS made as part of its incorporation in the safety-related software should be to the requirements of the standard.
e) Unreachable code should only be allowed to remain in the final application where it can be shown that the risks of leaving it in are less than the risks of modifying the code to remove it.

The PDS should be provided with documentation equivalent to the rest of the safety-related software. However, in cases where the PDS is justified on the basis of in-service history or extensive V&V and is treated as a "black box", it may be acceptable for design information not to be provided as long as a comprehensive requirement specification for the software is provided.

4.3.3 Allowable Reduction in Reverse Engineering

However, the extent of the reverse engineering and V&V activities may be reduced on the basis of a safety analysis, taking account of the rigour of the PDS's development process, its extent and functionality, and its in-service history. The reverse engineering activities should concentrate on the areas most notably inadequate in satisfying the objectives of the standard; the problem report history can be taken into account when identifying inadequate areas.

In-service history may only be taken into account where reliable data exists relating to in-service usage and failure rates. Quantified error rates and failure probabilities should be derived, taking into account:

- the length of the service period
- the operational hours, allowing for different operational modes and the numbers of copies in service
- the definition of what is counted as a fault/error/failure

4.3.4 Relaxation for Lower SIL

The guidance allows unreachable code to be left at SIL1 and SIL 2, even if the risks of leaving it in are more than the risks of modifying the code to remove it, if a strong justification can be made for this approach. All the other requirements on PDS are independent of SIL.

4.3.5 Interpretation of the Defence Standards

The MoD's Equipment Safety Policy (ES Pol) Directorate has recently produced an interpretation of DS 00-55 and 00-56 [22]. It states that, in applying the standard to NDI, too much emphasis has been put on the "full compliance" aspects of a software module of a particular SIL, rather than conducting safety analysis to determine the safety properties of the NDI and gathering appropriate evidence to demonstrate these safety properties.

It recommends using a hazard-directed framework in which the selection of safety analysis and assessment methods is determined, not through a prescriptive process formulated from subjective levels of integrity, but by the safety properties of the software that need to be demonstrated. This approach provides a means by which the application of additional, retrospective assessment methods can be better justified.

The elements of the hazard-directed approach are:

> *System Safety Model*—System safety analysis should be carried out so that hazards can be propagated down through line replaceable units into the software elements.
>
> *Software Safety Requirements*—From the system safety model, the way the software can contribute to the system hazards should be identified and safety properties agreed. Categories for software requirements should be developed, in order to group requirements according to the kinds of evidence that will probably be required.
>
> *Evidence-Base*—It is necessary to show that the requirements have been validated and the implementation satisfies its requirements. Evidence may be direct, may back up direct evidence, or provide the basis for engineering judgements.
>
> *Consensus*—A consensus should be built involving all interested parties and stakeholders. A greater reliance is placed on indirect evidence for NDI and so the subjective assessment of this data demands a consensus of opinion.

4.4 COMPARISON OF OTHER STANDARDS WITH IEC 61508

The following table compares the main recommendations of the defence, medical and nuclear sector standards.

Table 10 : Comparison of standards

	IEC 61508	FDA 1252	IEC 60880 Supplement 1	Def Stan 00-55/56
Evaluation against specification	Not explicitly if "proven in use" argument used.	Basic documentation states function SOUP provides and evidence for correct operation.	Explicit evaluation step; analysis or test required if SOUP specification not explicitly defined.	Carried out as for new software, or by reverse engineering informed by safety analysis and operating data.
Quality evaluation	Not explicitly if "proven in use" argument used.	Process audit required for high SIL.	Required for all SILs but may be compensated for by operating experience.	Carried out as for new software, or by reverse engineering informed by safety analysis and operating data.
Operating experience	Can be used for "proven in use" argument.	Operational experience with the use of the SOUP can be submitted as part of the justification of residual risks.	Can be used to compensate for weaknesses in functional, performance and quality evaluation.	Can be used to reduce extent of reverse engineering.
Variation with SIL	Built with techniques appropriate for SIL. Or, operating time commensurate with SIL ("proven in use").	For high SIL, need special documentation including: audit of developer's process; description of SOUP developer's and device manufacturer's V&V; plans for maintenance and support if original developer terminates support.	Level of assurance to be achieved by the quality evaluation different for the three safety categories. Rigour of analysis of operating experience consistent with safety category.	Requirement to remove unreachable code relaxed for SIL 1 and SIL 2.
Risk assessment	Risk based. Considerable detail on SIL determination in general; nothing specific for SOUP.	Consequence based. Process defined combining hazard analysis and mitigation.	Not covered in Supplement.	Covered by DS 00-56, which combines consequence and risk based approaches. Use of PDS, extent of reverse engineering and use of operating data should all be subjected to risk assessment.

Table 10 : Comparison of standards

	IEC 61508	FDA 1252	IEC 60880 Supplement 1	Def Stan 00-55/56
Documentation	No specific requirements for SOUP.	Basic documentation for low consequence hazards; special documentation for high consequence hazards.	Documentation required on evaluation of functional and performance features, quality evaluation, and evaluation of operating experience.	Documentation required as for new software, except that design information may be omitted if PDS justified as "black box".
Configuration management	Exact identification of each system and its components, including version numbers, required for "proven in use". No other specific requirements for SOUP.	Basic documentation covers installation, configuration control, storage, and maintenance. Installation of non-specified SOUP should be prevented.	SOUP should be under configuration management and its version and configuration should be precisely defined.	Configuration management as for new software, and configuration details required as part of operating history.
Organisational support	—	FDA operates device master file scheme for submitting developer's confidential information.	—	—

4.5 COMMON POSITION OF EUROPEAN REGULATORS

The CEC is about to publish a document entitled *Common position of European regulators for the licensing of safety critical software for nuclear reactors* as a "consensus document" [33]. This has been developed by the European Commission's Advisory Experts Group, Nuclear Regulators Working Group. It makes the point that licensees may wish to make use of *pre-existing software* components (PSW) as these may not only be beneficial for productivity but may also increase safety if introduced in a proper way. The benefit stems from the fact that PSW components have often been used in many applications, and their operating experience, when assessable and representative, can be taken into account. Reusable software components may have been developed to suitably high standards in other industries for use in safety critical applications and, therefore, may be reusable in the nuclear industry.

The document sets out the following as a common position:

a) The functions that have to be performed by the PSW components shall be clearly identified, and the impact on safety of these functions shall be evaluated.

b) The PSW components to be used shall be clearly identified, including their code version(s).

c) The interfaces through which the user or other software invokes PSW modules shall be clearly identified and thoroughly validated. Evidence shall be given that no other calling sequence can be exercised, even inadvertently.

d) The PSW shall have been developed and shall be maintained according to good software engineering practice and QA standards appropriate to its intended use.

e) For safety systems (category one), the PSW shall be subjected to the same assessment (analysis and review) of the final product (not of the production process) as new software developed for the application. If necessary, reverse engineering shall be performed to enable the full specification of the PSW to be evaluated.

f) If modifications of PSW components are necessary, the design documentation and the source code of the PSW shall be available.

g) The information required to evaluate the quality of the PSW product and of its assessment and development processes shall be available; this information shall be sufficient to assess the PSW to the required level of quality.

The document states that for acceptance the following actions shall be taken:

a) Verify that the functions performed by the PSW meet all the requirements expressed in the safety system requirement specifications and in other applicable software specifications.

b) Verify that the PSW functions that are not required by the safety system requirement specifications cannot be invoked and adversely affect the required functions, for example through erroneous inputs, interruptions, and misuses.

c) Perform a compliance analysis of the PSW design against the applicable standards requirements (e.g. IEC 60880).

d) The PSW functions intended for use shall be validated by testing. The tests may include tests performed by the vendor.

e) Ensure that the PSW functions cannot be used by the safety system, by other software or by the users in ways that are different from those which have been specified and tested (if necessary through the implementation of pre-conditions, locking mechanisms or other protections).

f) If credit is given to feedback experience in the licensing process, sufficient information on operational history and failure rates shall be available. Feedback experience shall be properly evaluated on the basis of an analysis of the operating time, error reports and release history of systems in operation. This feedback experience shall also be based on use of the PSW under evaluation in identical operational profiles. This operating experience shall be based on the last release except if an adequate impact analysis shows that previous experience based on unchanged parts of the PSW is still valid because these parts have been unaffected by later releases.

g) If the available information of the type required by position 6 above is not sufficient, then an analysis (risk assessment) of the impact on safety of a failure of the PSW shall be performed. Special attention shall be paid to possible side effects and to failures that may occur at the interfaces between the PSW and the user and/or other software components.

h) Errors that are found during the validation of the PSW shall be analysed and taken into account in the acceptance procedure

.

5 CONCLUSIONS

Whilst use of SOUP is highly attractive to software system developers and some use of SOUP looks unavoidable for most industrial applications, it is clear that assessment of SOUP is a difficult problem. There are a number of European and national research projects investigating the problems (e.g. Cemsis, Quarc and other IMC projects) and the commissioning of at least two studies by major stakeholders (HSE, MoD).

The difficulty in justifying SOUP is a symptom of the general technical problems of justifying safety-related software. Leaving aside issues of intellectual property and access to information, SOUP would be a solved problem if the standards for safety critical software provided a framework that defined:

- methods for the discovery of the safety properties and their assignment to software components
- the type of arguments that could be used to justify that the safety properties hold
- the range of applicable evidence to support these arguments

If this was provided by existing standards all that would be needed would be to examine the problems of achieving these for SOUP: one would just need to look at the availability of techniques and the feasibility of evidence production. Therefore in addressing the SOUP problem we are actually tackling the underlying technical problems of how to justify any safety-related software system.

Requirements in standards differ considerably, and typically either require considerable interpretation in their requirements, or are so rigorous as to be unachievable in most applications. This is especially so for lower reliability levels. However the existing standards, especially IEC 61508, are important because they:

- provide a safety management framework within which the technical aspects of justifying SOUP can be positioned (e.g. systematic approach, document production, independent scrutiny).
- the SIL concept, and its extension to consider the quantification of the risk from a component, is very important in trying to establish how a SOUP component contributes to safety and what risks it introduces.

We therefore propose that the justification for SOUP to be developed in the next deliverable should be based on a safety case approach with a number of additions to an IEC 61508 approach to:

a) take into account the need for a safety case (i.e. describe the approach and how it interfaces to the safety life cycle)

b) cope with the presence of SOUP through out the safety life cycle (architecture development, risk analysis, software engineering techniques, operation etc.)

6 REFERENCES

[1] IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems. 1997.

[2] HSE ITT for "Assessment of Software Components for use in IEC 61508-Compliant Safety-related Applications", RSU ref: 4005/R38.026, 1 Sept. 1999.

[3] HMSO, The Safety of Operational Computer Systems, HMSO 1998.

[4] IEC 60880 Amd.1 Ed. 1.0, "Amendment to IEC 60880—Software for computers important to safety for nuclear power plants – First supplement to IEC 60880", Committee draft, 1999.

[5] Factory Mutual 61508 Product Certification, see: http://www.fmglobal.com/research_standard_testing/product_certification/reliability_certification_services.html

[6] CASS, Conformity Assessment of Safety-Related Systems, http://www.eutech.com/cass/

[7] Grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen (basic safety examinations for PMC protective equipment). DIN V 19250.

[8] Grundsätze für Rechner in Systemen mit Sicherheitsaufgaben (principles for computers in safety related systems). DIN V VDE 0801, 1989.

[9] Dr Rader, Microcomputer in der Sicherheitstechnik (microcomputers in safety techniques). Hölscher, 1984.

[10] Carnegie Mellon University, Software Engineering Institute (Principal Contributors and Editors: Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis), The Capability Maturity Model: Guidelines for Improving the Software Process, ISBN 0-201-54664-7, Addison-Wesley Publishing Company, Reading, MA, 1995. See also http://www.sei.cmu.edu/activities/cmm/

[11] W Humphrey, T Snyder, R Willlis, Software Process Improvement at Hughes Aircraft, IEEE Software, July 1991.

[12] T Bollinger, C McGowan, A critical look at software capability evaluations, IEEE Software, July 1991.

[13] W Humphrey, B Curtis, Comment on a "Critical Look", IEEE Software, July 1991

[14] ISO 9001, Quality Systems—Model for quality assurance in design/development, production, installation and servicing, 1987.

[15] ISO 9000-3, Quality Management and Quality Assurance Standards, Part 3—Guidelines for the Application of ISO 9001 to the development, supply and maintenance of software, 1990.

[16] SPICE Software Process Assessment Parts 1 to 9, ISO/IEC JTC 1/SC 7, Working Group on Software Process Assessment (WG 10), Version 1, 1995.

[17] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software, Version 1.1", Carnegie Mellon, Software Engineering Institute, CMU/SEI-93-TR-24, DTIC, Number ADA263403, February 1993.

[18] FDA 1252, Off-The-Shelf Software Use in Medical Devices, US Department of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health, 9 September, 1999.

[19] IEC 60880 Supplement 1, Software for Computers Important to Safety for Nuclear Power Plants as a First Supplement to IEC Publication 880, Draft, 1999.

[20] DS 00-55, Requirements for Safety Related Software in Defence Equipment, UK Defence Standard 00-55, Parts 1 and 2, Issue 2, 1 August 1997.

[21] DS 00-56, Safety Management Requirements for Defence Systems, UK Defence Standard 00-56, Parts 1 and 2, Issue 2, 13 December 1996.

[22] Safety Assurance for Non Developmental Safety Critical Software, MoD ES Pol SCS, 23 November 1999.

[23] J. Barnes, "High Integrity Ada, The SPARK Approach", ISBN: 0-201-17517-7, 1997

[24] Les Hatton, "Safer C: Developing for High-Integrity and Safety-Critical Systems", ISBN 0-07-707640-0, McGraw-Hill, 1995

[25] MISRA, "Guidelines for the use of the C Language in Vehicle Based Software", Motor Industry Software Reliability Association, 1998

[26] P G Bishop and R E Bloomfield, "A Conservative Theory for Long-Term Reliability Growth Prediction", IEEE Trans. Reliability, vol. 45, no. 4, Dec. 96, pp 550-560.

[27] J M. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Trans. Software Eng., Vol. 18, No. 8, August 1992, pp 717-727

[28] J M Voas and G McGraw "Software Fault Injection: Innoculating Programs Against Errors". John Wiley & Sons, ISBN 0-471-18381-4, 1997

[29] 4th European Conference on Software Maintenance and Reengineering at http://www.uni-koblenz.de/~ist/CSMR2000/index.html.

[30] Technical University of Vienna, http://www.infosys.tuwien.ac.at/reports/bin/dir.pl

[31] http://www.backerstreet.com/cg/work.htm

[32] Software QA Test Resource Centre http://www.softwareqatest.com/qatweb1.html

[33] European Commission's Advisory Experts Group, Nuclear Regulators Working Group, Common position of European nuclear regulators for the licensing of safety critical software for nuclear reactors, to be published shortly.

Printed and published by the Health and Safety Executive
C1 05/01

MAIL ORDER
HSE priced and free publications are available from:
HSE Books
PO Box 1999
Sudbury
Suffolk CO10 2WA
Tel: 01787 881165
Fax: 01787 313995
Website: www.hsebooks.co.uk

RETAIL
HSE priced publications are available from booksellers

HEALTH AND SAFETY INFORMATION
HSE Infoline
Tel: 08701 545500
Fax: 02920 859260
e-mail: hseinformationservices@roffett.com
or write to:
HSE Information Services
Caerphilly Business Park
Caerphilly CF83 3GG

HSE website: www.hse.gov.uk

CRR 337

£10.00

ISBN0-7176-2011-5

9 780717 620111