An Overview of the SoBP for Software in the Context of DS 00-56 Issue 4

Catherine Menon, Richard Hawkins, John McDermid and Tim Kelly

SSEI, University of York

Heslington, York, UK

Abstract Defence Standard 00-56 Issue 4 is the current contractual safety standard for UK MOD projects. It requires the production of a structured argument, supported by diverse evidence, to show that a system is safe for a defined purpose within a defined environment. This paper introduces a Standard of Best Practice which has been produced by the Software Systems Engineering Initiative to provide guidance for software compliance with Defence Standard 00-56 Issue 4.

1 Introduction

Defence Standard 00-56 (DS 00-56) Issue 4 (Ministry of Defence 2007) presents a goal-based, or evidential, approach to ensuring and assuring safety. One of the major principles of DS 00-56 is the need to demonstrate system safety by means of a compelling safety argument, supported by rigorous evidence. This represents a departure from earlier prescriptive UK MOD safety standards in that DS 00-56 Issue 4 states *what* is required, but not *how* this is to be achieved.

While this approach permits the software contribution to system safety to be evaluated contextually in each situation, there is currently a lack of clear guidance on how to perform this evaluation. The absence of guidance is felt in many projects, e.g. the Chinook Mark 3, where difficulties have arisen for the Integrated Project Team (IPT) in determining safety of both the engine control software and the cockpit display software.

Consequently, the Software Systems Engineering Initiative (SSEI) has been tasked by the MOD with the production of a Standard of Best Practice (SoBP) for assessing software compliance with DS 00-56 Issue 4. The remit of this SoBP is to address all aspects of software contribution to system safety, from the integration of COTS software into safety-critical systems, to the use of civil standards such as DO-178B for military applications.

This paper introduces the first issue of the SoBP (Menon et al. 2009), which was completed in August 2009 and is currently available from the SSEI website.

This interim SoBP applies to the activities of contract assessment, software development, assurance, verification and validation and initial acceptance. It does not consider the in-service phase, nor does it consider in detail the concept and assessment phases. Nevertheless, many of the in-service issues are similar in scope to those presented here, especially considerations such as upgrading systems and the use of COTS products. It is the intent that further work will be performed to provide guidance on through-life safety considerations, including operational safety.

In Section 2 we provide a brief overview of the structure and focus of the SoBP. Section 3 addresses the managerial issues involved with assessing the contribution of software to system safety, while Section 4 describes the technical aspects of assurance. Finally, in Section 5 we conclude and discuss the planned updates to the SoBP.

2 Structure of the SoBP

The SoBP addresses two primary areas of concern for software compliance with DS 00-56: managerial and technical. This is an essential distinction, but is not always clear cut in practice. Some of the 'management' decisions identified in this document could be carried out by the prime, some by MOD, or some (more likely) by the two working together. Further, management decisions may apply at several levels in the supply chain. Consequently, the SoBP aims to be applicable independent of the particular stakeholders in any situation.

The structure of the SoBP is based around a swim-lane diagram (Figure 1 in Section 3) showing how safety-related communication should be managed throughout the project. This diagram identifies three major interested parties, or strands: 'Management', 'Assurance' and 'Ensurance'. Relevant portions of this diagram are enlarged in later sections to enhance readability.

The 'Management' strand corresponds to the activities of managerial personnel and those responsible for project management of the customer-supplier boundary. Management activities are typically concerned with overseeing safety management, facilitating customer-supplier interaction and formally assessing relevant deliverables for acceptance. Section 3 of this paper describes these managerial activities, providing guidance on project decisions which are important when producing software which can be shown to be safe with sufficient confidence. The 'Management' swim-lane of Figure 1 is the most detailed, as it is assumed that managerial input and decisions are a primary driver for any project.

By contrast, the 'Assurance' strand corresponds to the activities of those personnel responsible for demonstrating the safety of the software. Assurance activities are typically concerned with the production of a compelling safety argument, supported by rigorous evidence. Section 4 of this paper provides a primarily technical perspective on assurance activities. The swim-lane diagram of Figure 1 is intentionally simplified when representing the Assurance strand. This is because de-

cisions about assuring the safety of the software can only be made in the context of a particular project, and consequently cannot be easily generalised.

Finally, the 'Ensurance' strand corresponds to the activities of those personnel responsible for developing the software. We have deliberately avoided providing explicit guidance for ensurance activities, as this would not be in keeping with the goal-based approach of DS 00-56. In practice, the remit of these activities may overlap. For example, performing hazard analysis and deriving safety requirements will require interactions between activities in all three strands.

2.1 Requirements of DS 00-56: Safety Cases

The purpose of the SoBP is to provide guidance for software compliance with DS 00-56. This requires an understanding of the ways in which software can contribute to system safety, and the recommendations of DS 00-56 which ensure that these contributions are acceptable. One such recommendation is the production of a *safety case*.

From Annex A of DS 00-56, a safety case is 'a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment' (Ministry of Defence 2007). A safety case will evolve throughout a project, and the current state of safety should be reflected via regular safety case reports. The personnel undertaking ensurance and assurance roles are responsible for producing these reports, as well as a final safety case report. The acceptability of these reports may be dependent upon input from the Independent Safety Adviser (ISA) or Safety Committee. Each software safety case report must consider all relevant aspects associated with software safety, including the following:

Requirements validity. The argument must demonstrate that all software safety requirements are complete and accurate for the purposes of mitigating the software contribution to system-level hazards.

Requirements satisfaction. The argument must comprehensively demonstrate satisfaction of all the identified software safety requirements.

Requirements traceability. The argument must demonstrate that the high-level software safety requirements are traceable to system hazards, and also down through all levels of development (detailed software requirements, software design, code etc.).

Software quality. The argument must demonstrate that the software and the development processes exhibit the basic qualities necessary to place trust in the evidence presented. For example, the software must be free from intrinsic errors (e.g. buffer overflows and divide-by-zero errors), and adequate configuration consistency and version control must be demonstrated.

All four of the aspects above must be adequately addressed within the safety case. Fundamental to this concept is the idea of the justifiable confidence in the truth of a safety claim. This is referred to as the *assurance* of that claim. A safety case should provide sufficient assurance of all claims to permit the justified use of the software in the proposed role. If sufficient assurance is not provided, we say that there is an *assurance deficit*. This is an uncertainty or lack of information which affects assurance. Assurance deficits are almost inevitable; the question is whether such deficits are *justified*. An assurance deficit can be justified if the cost of addressing the deficit (e.g. by providing additional evidence) is out of proportion to the benefit that would be gained from doing so. Section 4 provides further detail on this.

3 Managerial Issues

This section describes the key management activities and decisions, as well as the inputs that may reasonably be expected from the Ensurance and Assurance activities. The SoBP presents this material to be read in conjunction with technical guidance, which this paper discusses in Section 4.

The Management strand is concerned with the key decisions on a project level, which include issues of supplier selection (where relevant) and acceptance of the safety case. For each decision we identify:

- Inputs to the decision making activity from Ensurance, Assurance or external activities (for example safety case reports or development plans). This list of inputs is intended to be indicative of the minimum information which will be needed, and should not be considered exhaustive.
- Comparator data to allow assessment of the inputs. This data may be available from a wide range of sources.
- Criteria for making the decision (for example, the acceptability of a safety case, or the extent to which risks are shown to be reduced to an acceptable level). It is likely that for each decision, the criteria should be weighted according to their importance.
- Possible outcomes, which in each case will be one of:
  - Proceed without change to plans
  - Proceed with further safety risk management
  - Iterate selected process steps with remedial action
  - Terminate the process and end the project development

In all cases, the guidance is framed so that all reasonable ways of proceeding will have been evaluated before reaching a decision to terminate the development.

Each of these decisions may involve input from the ISA, Safety Committee or external domain experts. These roles are not distinguished and are assumed to support the Management decision-making activities. In each case, the identity of

personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

The SoBP provides detailed guidance as described above on all of the identified management decisions. In this paper we select two such decisions to discuss in detail, and refer the reader to the issued SoBP for further guidance.

3.1 Software Safety Management Phases

Software development processes can depend on the organisation, context of the project, scope of the project, and so on. This guidance identifies four major phases (Initial, Development, Containment and Acceptance) which can usefully be mapped to all software development projects. These identified phases are not intended as a software development lifecycle, but rather to identify the key decision points relevant to DS 00-56 which occur during contractual interactions. They are orthogonal to the swim-lane strands introduced in Section 2. Figure 1 shows this interaction.

3.2 Swim-lane Diagram

To reduce complexity, the diagram in Figure 1 reflects only the major input(s) for each decision. Similarly, there are a number of iterations and ongoing activities in each lane which are not explicitly shown in the diagram for reasons of clarity. While major iterations are shown (e.g. the main iteration of the development phase reflecting ongoing monitoring of safety management and safety case reports), there will be iteration and ongoing activities within each lane. A single activity – as represented in the diagram – may correspond to a number of iterations of that activity, informed by safety dialogues and checkpoints. Relevant portions of this diagram are enlarged in the following sections, which briefly describe the activities within each phase and provide detailed guidance for selected decisions.

3.2.1 Initial Phase

There are a number of activities and decisions which take place prior to establishing contractual arrangements. These include gathering requirements, establishing a project budget, writing an Invitation to Tender, assessing bids against project criteria, and negotiating with selected suppliers.

As DS 00-56 is a contractual standard, it does not strictly apply to these activities and decisions which take place at a pre-contractual stage. However, this phase is important from a safety perspective because of the importance of supplier selec-

tion. The capabilities of potential suppliers – including in-house developers where relevant – must be assessed prior to finalising a contract. Any issues identified during this assessment may then inform the contractual negotiation. Consequently, adequate assessment of suppliers and bids can enhance the likelihood of eventual delivery of software which is compliant with the requirements of DS 00-56. Full details of the decisions and activities in this phase are provided in the complete SoBP.

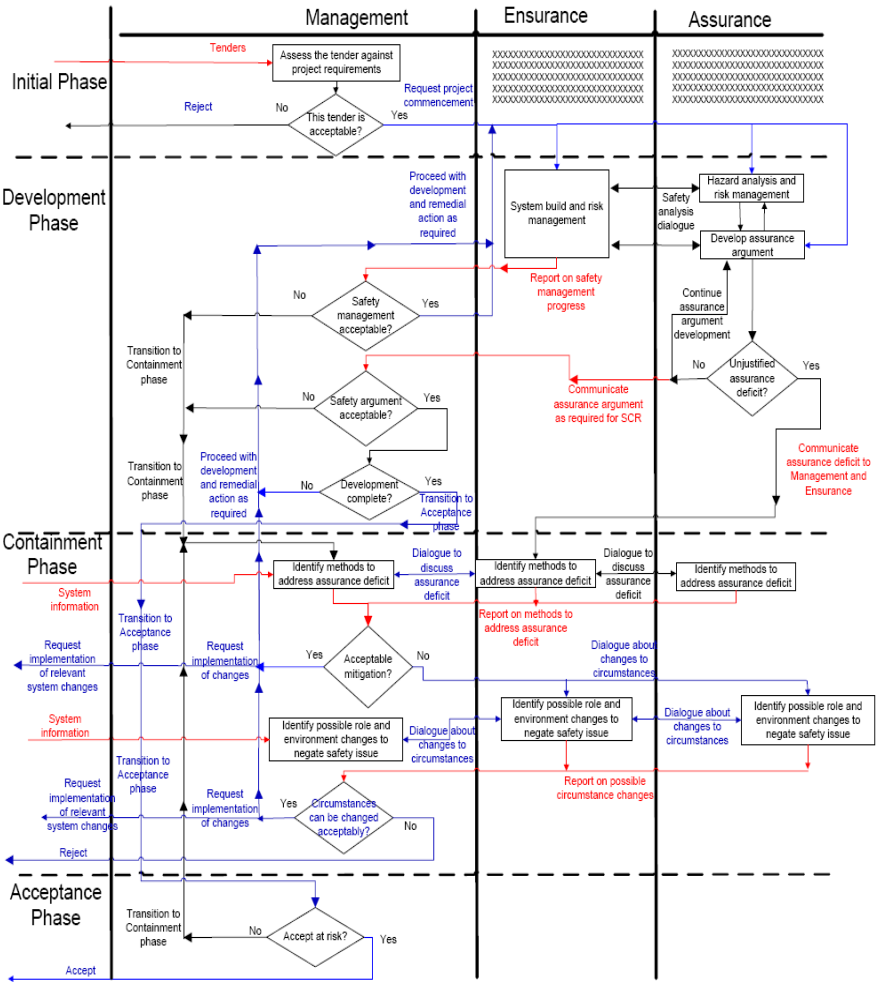

Fig. 1. Swim-Lane Diagram

3.2.2 Development Phase

The development phase is concerned more directly with achieving software safety than the Initial phase. Specifically, the managerial decisions in the development phase are intended to confirm that the software is being developed in an acceptably safe manner according to the requirements of DS 00-56. Figure 2 shows the development phase from the swim-lane diagram. There are three decisions in the phase, one of which (*Safety Argument Acceptable*) is discussed in detail here.

The development phase is iterative as shown in the swim-lane diagram, and consequently all decisions in this phase may be encountered multiple times.

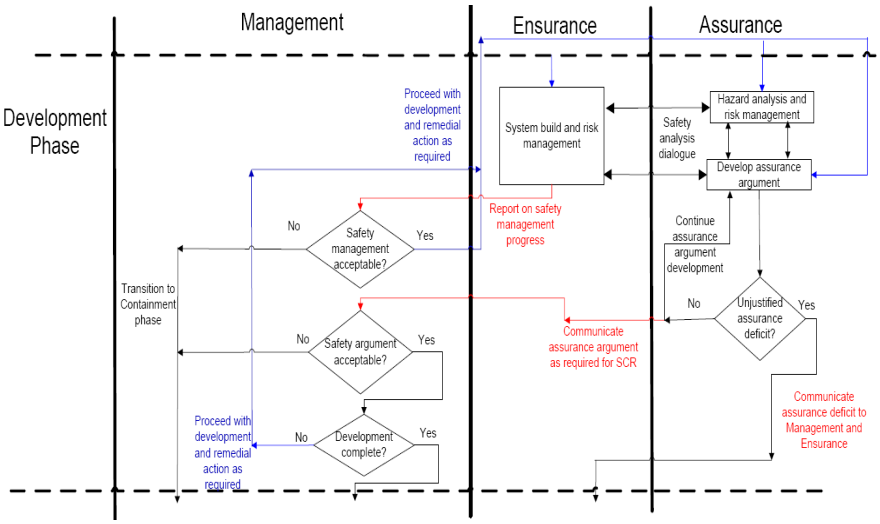

Fig. 2. Development Phase

3.2.2.1 Decision: Safety Argument Acceptable

Management decisions relating to the safety argument will typically require input from domain and safety experts. These experts (such as the ISA) can judge the technical sufficiency of the evidence presented, but the final responsibility for making the decision may be considered a managerial concern. This decision may be made multiple times throughout the software development process; for example when receiving regular safety case reports.

It is worth noting here that this decision will be encountered only where the developers undertaking assurance activities consider that all assurance deficits identified thus far are justified as far as is possible at this stage of development, or that these deficits are likely to be justified by planned future processes. If an assurance deficit is considered by developers to be unlikely to be justified given future de-

velopment, the Containment phase will be entered instead, and this decision will not be encountered.

Inputs

Report on safety management progress including:

- Development of the safety argument.
- Production of evidence to support the safety argument.

Comparator data

- Safety arguments and evidence for similar projects.
- Software safety argument patterns which illustrate typical successful patterns of argumentation.

Criteria

The safety argument should satisfy the following criteria:

- It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements.
- It should be sufficient to provide adequate assurance with respect to all safety requirements, or indicate how this assurance will be obtained.
- It should identify and justify all assurance deficits
- All assumptions should be identified and where appropriate justified, with references to supporting documentation where relevant.
- Evidence of a search for counter-evidence should be presented, and the effect of relevant counter-evidence upon the argument should be assessed.

The evidence provided to support the safety argument should satisfy the following:

- The evidence should adequately support the relevant safety requirements
- The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided.
- The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project

documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

- **Proceed with development.** This is represented as iteration in the development phase of the swim-lane diagram, and occurs when all the above criteria are met. There is no unjustified assurance deficit.
- **Proceed with development where this includes specified further safety management.** This outcome reflects that there is currently an unjustified assurance deficit, but this can be justified by means which have been identified and which will inform future safety argument development.
- **Iterate (repeat) process steps, with remedial action.** For this decision, this outcome reflects that an unjustified assurance deficit is present and can be addressed only by modifying or repeating activities in the development of the safety argument.
- **Terminate the process.** For this decision, this represents an exit to the containment phase. This occurs when there is an unjustified assurance deficit and no identified strategy for sufficiently reducing this deficit.

In practice management of information flows across contractual or organisational boundaries can be problematic. It may be the case that shortfalls in the (demonstrated) safety of the system are related to such boundaries and interfaces. Consequently, it is desirable that management explicitly consider this flow of information. It may also be the case that the flow of information down from the system level to the software is inadequate. In order to assess the potential for the software to contribute to system hazards, a degree of information is needed about the system context. In some cases, this may mean providing information to the supplier about the wider system in order to ensure that safety requirements are satisfied.

It should be noted that as shown in the swim-lane diagram, the only way to proceed to the acceptance phase is by judging the safety argument to be acceptable. This is in keeping with the requirement for an adequate safety case (Ministry of Defence 2007).

3.2.3 Acceptance Phase

While the SoBP provides guidance for assessing the completed software against the requirements of DS 00-56, assessment of the safety case is not the only activity necessary for acceptance of the software. Consequently, safety considerations must be balanced against the other acceptance criteria which are relevant for this project. If the requirements of DS 00-56 are not met (that is, if there is an issue of safety), then the containment phase is entered to attempt to remedy this problem. The issued SoBP contains further guidance on this topic.

3.2.4 Containment Phase

The containment phase is entered only on encountering a significant problem during development which cannot be remedied. Figure 3 shows the containment phase from the swim-lane diagram. Entry can be triggered in one of two ways. Firstly, personnel undertaking assurance activities may note that they are unable to adequately justify all assurance deficits, and that future development is unlikely to provide information which will justify these deficits. Secondly, management personnel may consider that significant problems are exhibited by ongoing safety management, by ongoing safety case development or by the final safety case. These problems may result in a lack of information which has the potential to affect assurance – an *assurance deficit*.

We present one of the decisions made during the containment phase in further detail here, summarising the guidance available in the SoBP.

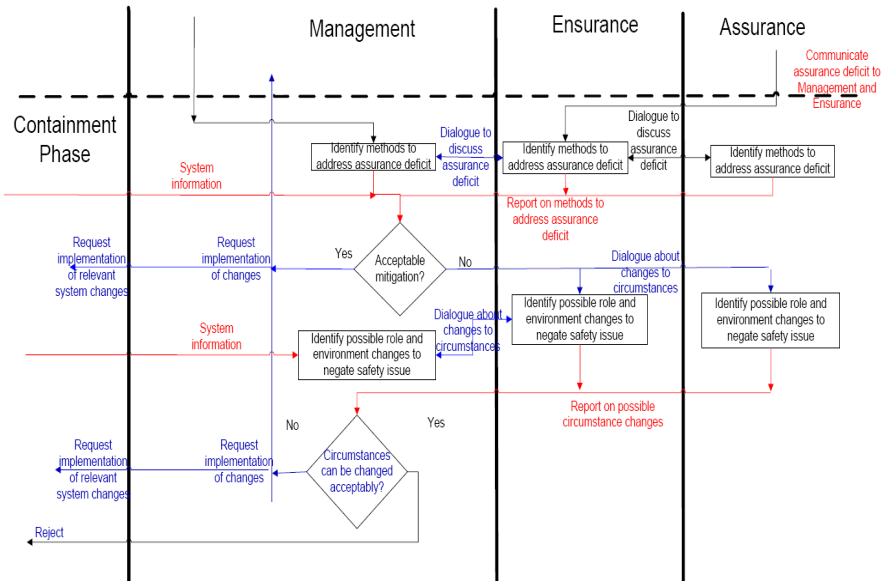

Fig. 3. Containment Phase

3.2.4.1 Decision: Acceptable Mitigation for Assurance Deficits

The decision is encountered when there is – or is likely to be – an unjustified assurance deficit, which is unlikely to be remedied within the bounds of the original safety management plan and proposed safety case structure. The (potential) presence of this assurance deficit should be communicated to Management in a timely manner. In addition to this communication, activities should be undertaken to identify possible methods of addressing the assurance deficit. These activities are

undertaken across all three strands of Ensurance, Assurance and Management, and in some cases external personnel may also be involved to identify methods to address this deficit. Once these methods have been identified, Management must determine whether they represent an acceptable solution to reduce or justify the presence of this assurance deficit.

Inputs

Safety case report including:

- The safety argument
- Evidence to support the safety argument
- A report on the unjustified assurance deficit

Report on proposed methods for addressing the assurance deficit including:

- Input from Ensurance/Assurance/external personnel as relevant

Comparator data

- Safety arguments and evidence for similar projects
- Software safety argument patterns which illustrate typical successful patterns of argumentation
- Information on techniques for resolving assurance deficits

Criteria

The supplied safety argument should satisfy the following criteria, with any discrepancies addressed by the proposed methods for resolving the assurance deficit.

- It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements.
- It should be sufficient to provide adequate assurance with respect to all safety requirements, or indicate how this assurance will be obtained.
- It should identify and justify all assurance deficits
- All assumptions should be identified and justified, with references to supporting documentation where relevant.
- Evidence of a search for counter-evidence should be presented, and the effect of relevant counter-evidence upon the argument should be assessed.

The evidence provided to support the safety argument should satisfy the following, with any discrepancies addressed by the proposed methods for resolving the assurance deficit:

- The evidence should adequately support the relevant safety requirements.

- The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided.
- The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse

The assurance deficit report should provide the following information:

- An assessment of the local and system effects of this deficit, where known.

The report on methods for addressing the assurance deficit should include the following:

- Identification where possible of techniques to address this deficit, with consideration of how these may fit into the safety management plan.
- A comparison of these techniques to demonstrate how they will provide additional assurance.

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

- **Proceed with no change.** Not applicable.
- **Proceed with further risk management.** For this decision, this outcome is applicable in two cases. Where the identified assurance deficit can possibly be remedied with further software safety management (there are no scheduled future assurance tasks which could address this deficit, but some may be added), the identified remedial actions should inform the future development of the safety argument. Where the deficit cannot be remedied (project constraints mean that it is not feasible to add further assurance tasks to address this deficit) development may proceed provided that system-level risk management techniques are identified to justify this deficit. This latter choice will require the cooperation of external developers and approval across the entire system.
- **Iterate (repeat) process steps, with remedial action.** For this decision, this outcome reflects that further development cannot proceed until this assurance deficit is reduced. Alternative verification processes must be undertaken, as this assurance deficit could render nugatory all further development activities.
- **Terminate the process.** This outcome reflects that there is no identified strategy to reduce this assurance deficit, and the next step is to consider a possible change to the circumstances and environment of this software.

3.3 Managerial Summary

This section has summarised some of the guidance provided for managers in the SoBP. In addition to the material presented here, the full SoBP contains detailed discussions of all activities and decisions. This includes explicit listing of criteria on which decisions are made, the input which is expected for the decisions, the potential outcome of each decision, and examples to illustrate how these situations are managed on different projects. This guidance is intended to be read in conjunction with the technical guidance of the SoBP, which we summarise in the following section.

4 Technical Issues

This section provides an overview of the technically-focussed material of the SoBP. It is intended to support the managerial perspective which was discussed in Section 3 of this paper.

DS 00-56 requires the production of a safety argument which is commensurate with system risk:

> 'The Safety Case shall contain a structured argument demonstrating that the evidence contained therein is sufficient to show that the system is safe. The argument shall be commensurate with the potential risk posed by the system...' (Ministry of Defence 2007)

The SoBP provides guidance on how to comply with this requirement when considering the software components of systems. The aim is to provide guidance for the developers of software safety arguments (both Ensurance and Assurance personnel) on how to construct arguments which are sufficiently compelling, and how to justify the sufficiency of those arguments. In addition, the guidance should help those involved in *assessing* software safety arguments (Management personnel) to determine whether or not the arguments provided are sufficiently compelling.

A software safety argument must demonstrate that the software under consideration is acceptably safe to operate as part of the embedding system. This requires a demonstration that the potential contribution made by the software to the identified system hazards is acceptable. To be compelling, the software safety argument must provide sufficient confidence in claims which support this objective. It is inevitable for the software aspects of a system that there will exist inherent uncertainties that affect the assurance with which it is possible to demonstrate the safety of the software. The reason for this is that the amount of information potentially relevant to demonstrating the safety of the system is vast. This may be information relating to the software itself, or to the system within which the software operates. There will also be information relating to the environment and operation of the system, all of which potentially has a role in demonstrating that the software is acceptably safe.

It is simply not possible therefore to have complete knowledge about the safety of the software. This leads to uncertainty – for example, due to the presence of assumptions or known limitations in the integrity of the evidence provided. For this reason it is not normally possible to demonstrate with absolute certainty that the claims made in a software safety argument are true. For a software safety argument to be compelling it must instead establish *sufficient confidence* in the truth of the claims that are made.

It is worth noting at this point that such uncertainties in demonstrating the safety of the software are always present, but are often left implicit. Adopting a safety argument-based approach, as is required by DS 00-56, facilitates the explicit identification of such uncertainties, which makes them easier to reason about, and therefore justify. Reasoning explicitly about the extent and impact of the uncertainties in a safety argument aids in the successful acceptance of the argument as part of a safety case.

The assurance of a claim is the justifiable confidence in the truth of that claim. A useful approach to ensure that a software safety argument is sufficiently compelling is to consider assurance throughout the development of that argument. The approach defined in the SoBP is split into two main parts: a software safety argument pattern catalogue and an assurance based argument development method.

4.1 Pattern Catalogue

The software safety argument patterns introduced above are used to capture good practice for software safety arguments. These patterns can be instantiated with specific claims and evidence to create a software safety argument for any system under consideration. The SoBP provides a pattern catalogue, containing a number of patterns which have been constructed based on existing software safety argument patterns, and an understanding of current practice for software safety arguments. The following argument patterns are currently provided in the SoBP:

1. **High-level software safety argument pattern.** This pattern provides the high-level structure for a generic software safety argument. The pattern can be used to create the high level structure of a software safety argument either as a stand alone argument or as part of a system safety argument.
2. **Software contribution safety argument pattern.** This pattern provides the generic structure for an argument that the contributions made by software to system hazards are acceptably managed. This pattern is based upon a generic ‘tiered’ development model in order to make it generally applicable to a broad range of development processes.
3. **Software Safety Requirements identification pattern.** This pattern provides the generic structure for an argument that software safety requirements (SSRs) are adequately captured at all levels of software development.

4. **Hazardous contribution software safety argument pattern.** This pattern provides the generic structure for an argument that potentially hazardous failures that may arise at each tier are acceptably managed.
5. **Argument justification software safety argument pattern.** This pattern provides the generic structure for an argument that the software safety argument presented is sufficient.

A primary consideration during the development of these patterns was flexibility and the elimination of system-specific concerns and terminology. Consequently, these patterns can be instantiated for a wide range of systems and under a variety of circumstances. To be compelling it is necessary to be able to justify that the instantiation decisions taken in constructing the argument result in a sufficiently compelling argument for the system under consideration (such as why particular claims are chosen whilst others are not required). Guidance for justifying such decisions is provided in Section 4.2.

It is intended that the software safety argument pattern catalogue will be updated and expanded over time to ensure that it reflects current understanding of good practice.

4.2 Assurance-based Argument Development Method

As discussed earlier, there exist many potential sources of uncertainty in demonstrating the safety of the software. Any such residual uncertainty can be considered to be an *assurance deficit*.

It is possible to identify how assurance deficits may arise by explicitly considering how information may be lost at each step in the construction of the argument. As an argument is constructed, decisions are continually being made about the best way in which to proceed. Decisions are made about how goals are stated, the strategies that are going to be adopted, the context and assumptions that are going to be required, and the evidence it is necessary to provide. Each of these decisions has an influence on what is, and is not, addressed by the safety case. The things that are not sufficiently addressed are referred to as assurance deficits.

The SoBP introduces an approach for systematic consideration of how assurance deficits may be introduced at each step of software safety argument development. By identifying where potential assurance deficits may arise, this approach can be used to inform the decisions that are made on how to construct the argument. In order to produce a sufficiently compelling software safety argument, all identified assurance deficits must be satisfactorily addressed, or justification must be provided that the impact of the assurance deficit on the claimed safety of the system is acceptable. Section 4.2.2 discusses how such justifications may be made.

4.2.1 Counter Evidence

DS 00-56 states, 'Throughout the life of the system, the evidence and arguments in the Safety Case should be challenged in an attempt to refute them. Evidence that is discovered with the potential to undermine a previously accepted argument is referred to as counter-evidence.' (Ministry of Defence 2007). Since an assurance deficit corresponds to a lack of relevant information, an identified assurance deficit reveals the *potential* for counter-evidence. That is, there is the possibility that in addressing the assurance deficit (i.e. gaining the relevant information) the information gained would reveal previously unidentified counter evidence. Reasoning about assurance deficits can therefore be helpful in identifying areas in which counter evidence may exist. Conversely, where there is knowledge of existing counter evidence, this can be used to help determine the potential impact of assurance deficits. For example, if other similar projects have identified counter evidence which relates to a particular identified assurance deficit, then the observed impact of this counter evidence on the safety of the other project can be used to indicate the expected impact that such an assurance deficit may imply.

4.2.2 Addressing Assurance Deficits

The discussion above illustrates how assurance deficits may be systematically identified throughout the construction of a software safety argument. The existence of identified assurance deficits raises questions concerning the sufficiency of the argument. Therefore where an assurance deficit is identified it is necessary to demonstrate that the deficit is either acceptable, or addressed such that it becomes acceptable (for example through the generation of additional relevant evidence).

There will typically be a cost associated with obtaining the information to address an assurance deficit. In practice the benefit gained from addressing each assurance deficit does not necessarily justify the cost involved in generating the additional information. In order to assess if the required level of expenditure is warranted, the impact of that assurance deficit on the sufficiency of the argument must be determined.

To determine the impact of an assurance deficit, it is first necessary to assess the software safety argument. Such an argument will make certain claims about the hazard identification, risk estimation, and risk management of the software contribution to system hazards. Since assurance deficits have the potential to undermine the sufficiency of the argument, the impact of any assurance deficit should be assessed in terms of the impact it may have on these claims. For example, an assurance deficit may be sufficient to challenge the completeness of hazard identification, or may be sufficient to challenge the estimated residual risk.

In assessing the software safety argument, it is possible to prioritise some claims as being more important to safety than others. For example claims regarding the behaviour of an architectural component (such as a voter), which carries a greater responsibility for risk reduction than other components, are more impor-

tant to the overall software safety argument. Therefore claims relating to those components would require a greater degree of assurance (more confidence must be established). This is exemplified in DS 00-56: ‘An example of a way of defining the variation of the degree of rigour with potential risk is the specification of a safety integrity requirement for the system’. The document then goes on to state, ‘In setting safety integrity requirements, it is therefore important to consider how much confidence is needed.’ (Ministry of Defence 2007). Where safety integrity requirements have been defined, they can be used as a way of determining the importance of the software safety argument claim to which they relate.

The method introduced in the SoBP to determine the impact of an assurance deficit has two stages. In the first stage, we analyse the claim to which the identified assurance deficit relates; the importance of the truth of that claim to the overall safety argument must be determined. Secondly, we determine the extent to which the identified assurance deficit affects the confidence achieved in this particular safety claim. Not all information relevant to a claim leads to the same increase in confidence in that claim. It is therefore necessary to assess the extent to which any information provided to address the assurance deficit might increase confidence in the truth of the claim.

Knowing the importance of the truth of the claim to the safety argument, *and* the relative importance of the assurance deficit to establishing the truth of that claim, it then becomes possible to determine the overall impact of the assurance deficit. In a similar manner to risks in the ALARP approach (Railtrack 2000), the impact of the identified assurance deficits may be usefully classified into three categories. An ‘intolerable’ deficit is one whose potential impact on the claimed risk position is too high to be justified under any circumstances. A ‘broadly acceptable’ assurance deficit is one where the impact of this assurance deficit on the safety argument is considered to be negligible. In such cases no additional effort to address the assurance deficit need be sought. Finally, a potentially ‘tolerable’ assurance deficit is one whose impact is determined to be too high to be considered negligible, but which is also not necessarily considered to be intolerable. A potentially ‘tolerable’ assurance deficit may be considered acceptable only if the cost of taking measures to address that assurance deficit is out of proportion to the impact of not doing so. The greater the impact of the assurance deficit, the more system developers may be expected to spend in addressing that deficit.

Making decisions relating to the acceptability of residual assurance deficits should, where necessary, involve personnel undertaking management and ensurance activities as well as those involved in assurance. If unable to form a judgment on the acceptability of an assurance deficit, then it is advised that expert assistance should be sought.

Note that the impact of an assurance deficit can only be determined on a case-by-case basis for a specific argument relating to a particular system. The same type of assurance deficit (such as a particular assumption) whose impact is categorised as broadly acceptable when present in the software safety argument for one system, may be considered intolerable when present in the argument for a different system. This is because the impact of an assurance deficit considers its impact

in terms of the overall safety of the system. It is for this reason that particular argument approaches (such as the software safety argument patterns discussed in Section 4.1) cannot be stated as sufficient for particular claims, but must be adapted on each use to be appropriate for the particular application.

Addressing an assurance deficit requires 'buying' more information or knowledge about the system relevant to the safety claims being made. There will typically be a cost associated with obtaining this information. For those assurance deficits categorised as tolerable, the value of the information in building confidence in the safety case must be considered when deciding whether to spend that money. In theory it is possible to do a formal cost-benefit analysis based on a quantitative assessment of the costs associated with the available options for addressing the assurance deficit, and the costs associated with the potential impact on the claimed risk position (such as the necessity to provide additional system level mitigations). However, in many cases a qualitative consideration of these issues will be more beneficial.

In all cases an explicit justification should be provided as to why the residual assurance deficit is acceptable and, wherever appropriate, an argument should be used to provide this justification. The software safety argument pattern catalogue (discussed in section 4.1) contains an argument pattern for constructing an argument to justify that the residual assurance deficits are appropriate.

The approach described above, although similar to ALARP, rather than considering the necessity of adopting measures to directly decrease risk, instead considers measures intended to increase the confidence that is achieved. As such the framework could be considered to help establish a claimed risk position in the software safety case that is ACARP (As Confident As Reasonably Practicable).

5 Conclusions

This paper has introduced the Standard of Best Practice (Menon et al. 2009) for software compliance with DS 00-56 Issue 4. In Section 2 we described the basic structure of the SoBP, emphasising the distinction between the managerial and technical perspectives. Section 3 then summarised the managerial guidance provided in the SoBP. Four development phases (Initial, Development, Containment and Acceptance) were identified, and information was provided about the decisions and activities of each phase. Section 4 introduced the technical guidance which is provided in the SoBP. In this section we described the software safety argument pattern catalogue, which contains patterns or 'blueprints' for constructing safety arguments. Additionally, this section described the assessment of assurance deficits to determine whether these can be justified, or whether they must be addressed by further work on the safety argument. Section 3 and 4 should be read in conjunction, as they represent different perspectives upon the same issues of software contribution to system safety.

It is anticipated that this SoBP will continue to be updated regularly. One of these planned updates will consist of an examination of the issues involved with using other standards – such as DO-178B – to comply with DS 00-56. Another planned update will further refine the technical guidance on assurance deficits by discussing the advantages and limitations of different types of software safety evidence. The SoBP is not intended to be a static document, but rather to represent current best practice. Consequently, further updates, refinements and validation of the results will be anticipated throughout the life of DS 00-56 Issue 4.

Acknowledgments The authors would like to thank the UK Ministry of Defence for their support and funding. This work is undertaken as part of the research activity within the Software Systems Engineering Initiative (SSEI), www.ssei.org.uk.

References

Menon C, Hawkins R, McDermid J (2009) Interim standard of best practice on software in the context of DS 00-56 Issue 4. Technical Report SSEI-BP-000001. Software Systems Engineering Initiative, York. https://ssei.org.uk/documents/. Accessed 5 October 2009.

Ministry of Defence (2007) Defence Standard 00-56 Issue 4: Safety management requirements for defence systems

Railtrack (2000) Engineering safety management – Yellow Book 3, volumes 1 and 2 – Fundamentals and guidance. Railtrack PLC