General Development Framework and Its Application Method for Software Safety Case

Fuping Zeng
School of Reliability and System Engineering, Beihang University, Beijing, 100191, China
Email: zfp@buaa.edu.cn

Minyan Lu[a] Deming Zhong[b]
School of Reliability and System Engineering, Beihang University, Beijing, 100191, China
Email: [a] lmy@buaa.edu.cn [b]timothy@gmail.com

***Abstract*—Safety case has already been adopted and developed across many industries because it is a good means to demonstrate whether software safety is acceptable. Despite the wide requirements for safety cases across many industries, it is a major challenge to construct compelling and general software safety arguments. The general development framework for software safety case(GDFSSC) and its application method has been discussed in this paper. Firstly, construction principle for software safety case from the view of hazard is given. Secondly, the general development framework for software safety case is proposed. Then the application method for the GDFSSC based on GSN pattern is elaborated, and braking control software is chosen as experimental example for proposed approach. The experimental results preliminarily show the proposed approach is feasible and more effective to develop a safety argument for demonstrating the acceptability of software with respect to safety.**

***Index Terms*—software safety, safety case, GSN pattern, development framework**

I. INTRODUCTION

Software is becoming increasingly important in assuring the safe operation of defense, aerospace, nuclear and railways and so on. Software and its associated computing systems (computer system hardware and firmware) are used in on-board and ground systems to support safety-critical functions such as guidance, navigation, and health monitoring. Software is also used to produce safety-critical data and to assist in mitigating system risks. Therefore the risk associated with the use of such software must be identified, characterized, analyzed and mitigated until the risk is reduced to the public. It is an important and disturbing problem for managers and developers how to demonstrate whether software safety is acceptable. Currently, safety case is a good means to solve this problem.

The concept of presenting safety-related information and arguments in a formal report initially came from the nuclear industry, but the notion of 'safety cases' is originated in major industrial accident control regulations introduced in the process sector in the UK in 1984[1]. Lord Cullen, in his report on the Piper Alpha accident in 1990[2], recommended the introduction of a safety case regime as part of the regulation of oil and gas facilities and operation. The purpose of a safety case is to "communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context" [3].

Safety case has already been adopted and developed across many industries (including defence, aerospace and railways) and is mandated to use in many safety standards[4][5]. Extensive work has been done in the area of safety case. Adelard developed 'Assurance and Safety Case Environment(ASCE)' and 'Safety Case Manual' presents the safety case structure as a set of claims which, using an argument, are supported by evidence[6][7]. Kelly defined and demonstrated a coherent approach to the development, presentation, maintenance and reuse of the safety arguments within a safety case[8][9]. Wagner constructed the safety case for a cruise control system describe in a case study in the automotive domain with a special consideration of existing domain-specific models[10]. Yuan discussed a more recent, argument-based approach to achieve and demonstrate computer system safety[11]. The challenge of evaluating confidence in safety cases is explored in Ref [12].

Despite the wide requirements for safety cases across many industries, it is poorly understood how to construct a safety case. On the one hand, many of previous approaches primarily focused on the presentation, reuse, confidence and tool of safety case, and the studies about the development of safety case are not much. On the other hand, some example of safety case is applied to a particular software product and the features specific to a particular software cannot be taken into account in a generic application. Thus, it is a major challenge to construct compelling and general software safety arguments. This is the subject of this paper.

In this paper, we explore the challenges of providing a general framework for making and justifying decisions about the arguments and evidence required to assure the safety of the software. After construction principle for software safety case is given, the general development framework for software safety case(GDFSSC) begins to be presented. Based on which, the method of applying the GDFSSC is proposed and the corresponding GSN safety

doi:10.4304/jsw.8.12.3262-3268

case pattern libraries are developed. An overarching motivation for this work is eventually to advance a framework which is possessed of stronger applicability and generality in order to develop software safety case effectively.

The rest of the paper is arranged as follows. In section 2, the related knowledge about safety case and its graphical presentation notation is introduced. In section 3, the general development framework for software safety case(GDFSSC) is proposed. In section 4, the application method for the GDFSSC based on GSN pattern is elaborated, including GSN safety case pattern for GDFSSC, process for constructing software case and experiment application. Finally, our work of this paper is summarized in the last section.

II. Related Knowledge

A. Safety Case

The definition from Defence Standard 00-56[13] is that '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 environment'.

The core concept is that a safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context, and context-free safety is impossible to argue. A safety case consists of explicit safety claim, the evidence that the claim has been met, and the argument linking the evidence to the claim. The relationship between these three elements is depicted in Figure 1. Both argument and evidence are crucial elements of the safety case that must go hand-in-hand. Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained--it can be unclear that (or how) safety objectives have been satisfied [14].

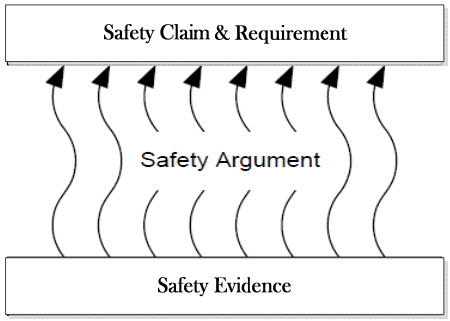

Figure 1. Structure of safety case

B. GSN & GSN Pattern

Currently, the most well-known notations for describing safety cases are graphics-based. The Goal-Structuring Notation (GSN) is a prototypical example of such a notation. The principal elements of the notation are shown in Figure 2. These elements are placed together to form a goal structure. The purpose of a goal structure is to show how **goals** are broken down into sub-goals, and eventually supported by evidence (**solutions**) whilst making clear the **strategies** adopted, the rationale for the approach (**assumptions**, **justifications**) and the **context** in which goals are stated. For further details on GSN see [13].

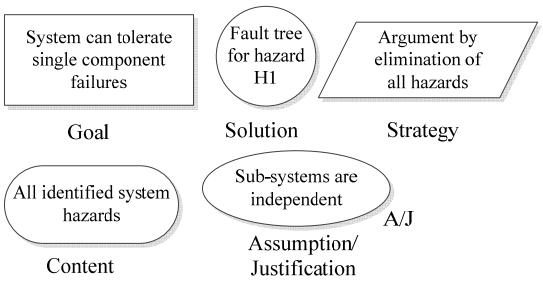

Figure 2. Principal elements of GSN

Safety cases tend to be huge and complex, and thus are hard to write and verify. The concept of safety case patterns in GSN is introduced in order to reuse successful safety cases patterns. Figure 3 shows a simple goal structure pattern that uses these extensions. In this structure, the top-level goal of system safety (G1) is reexpressed as a number of goals of functional safety (G2) as part of the strategy identified by S1. In order to support this strategy, it is necessary to have identified all system functions affecting overall safety (C1) e.g. through a Functional Hazard Analysis. In addition, it is also necessary to put forward (and develop) the claim that either all the identified functions are independent, and therefore have no interactions that could give rise to hazards (G4) or that any interactions that have been identified are non-hazardous (G3).

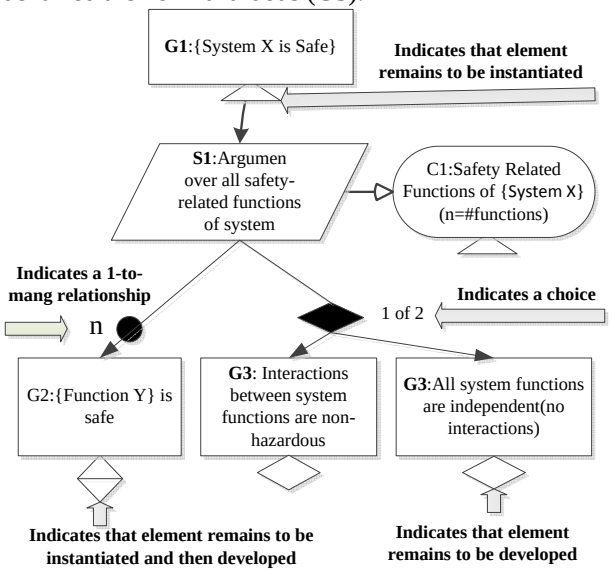

Figure 3. GSN Extensions for Pattern Description

III. General Development Framework for Software Safety Case (GDFSSC)

A. Construction Principle for Software Safety Case

It is the process for safety engineer to fight against hazard. Hazard comes from the hazardous scenarios, which are caused when the interactions between system components deviate from the normal behavior. The unexpected results that may be dangerous arise from such hazardous scenarios. Some of hazardous scenarios would involve software because some of software failures are the hazard reasons or make the hazard control no action. Such software failures are called hazardous software failures in this paper.

The system requirements that prevent system into the hazardous scenarios are called system safety requirements, and for software, are called software safety requirements(SSRs). There are different solutions or measures to avoid hazardous software failures of SSRs. Such solutions or measures are called software safety control, which is the corrective action for hazardous software failures.

We will have confidence in the delivered software as long as safety controls are correct, sufficient and realized. This is the construction principle for development framework for software safety case, as shown in Figure 4.

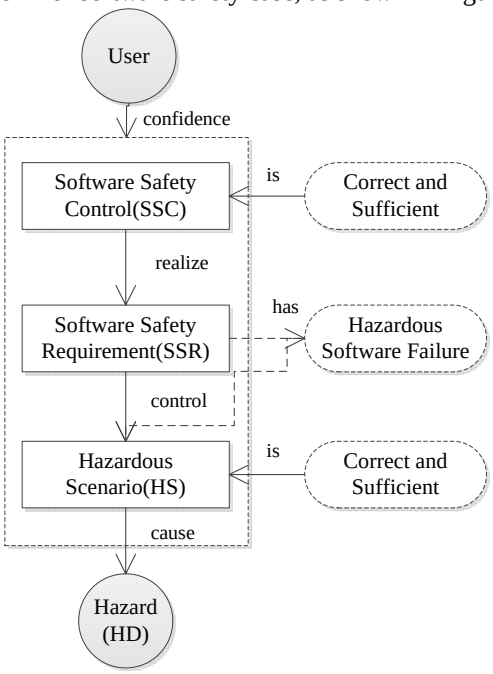

Figure 4. A Sketch Construction Principle for GDFSSC

The above construction principle is further described as follows:

(1) HD_i, i=1,…,n, represents the *i*th hazard that can cause accident.

(2) $HS_{i,j}$, j=1,…, o, represents the *j*th hazardous scenario that can cause HD_i.

(3) $SSR_{i,j,k}$, k=1,…, p, represents the *k*th software safety requirement that is derived from $HS_{i,j}$.

(4) $SRC_{i,j,k,m}$, m=1,…, r, represents the *m*th software safety control that can realize $SSR_{i,j,k}$.

The following gives the conditions on basis of the above definition.

Condition 1: If there is any of an unacceptable hazardous scenario, the system will remain in the hazard condition. In other word, the system will be safe if all of unacceptable hazardous scenarios do not happen. This can be expressed in (1).

$$\forall i \in [1, n], \bigcap_{j=1}^{o} \sim HS_{i,j} \mapsto \sim HD_i \quad (1)$$

Condition 2: Software safety requirement can prevent the hazardous scenarios to appear, which can be expressed in (2).

$$\forall i \in [1, n], \forall j \in [1, o],$$
$$\bigcap_{k=1}^{p} SSR_{i,j,k} \mapsto \sim HS_{i,j} \quad (2)$$

Condition 3: Software safety control can ensure the correct of software safety requirement, which can be expressed in (3).

$$\forall i \in [1, n], \forall j \in [1, o], \forall k \in [1, p],$$
$$\bigcap_{m=1}^{r} SRC_{i,j,k,r} \mapsto SSR_{i,j,k} \quad (3)$$

Corollary 1: The condition 1, condition 2 and condition 3 will be founded at the same time and the HD_i will not occur if all of the software safety controls (i.e. $SRC_{i,j,k,m}$, $i \in [1,n], j \in [1,o], k \in [1,p], m \in [1,r]$) are correct and sufficient. Thus, we have confidence in the delivered software.

B. General Development Framework for Software Safety Case(GDFSSC)

Under the guidance of the construction principle for software safety case, the general development framework for software safety case(GDFSSC) is proposed as shown in Figure 5. The elements in GDFSSC are organized into the package, and there are seven packages altogether. Table 1 gives the relation between the structure of safety case and the packages of GDFSSC.

TABLE I.
THE RELATION OF SAFETY CASE STRUCTURE AND GDFSSC

No	Safety Case Structure	the Package of GDFSSC
1	Safety Claim	1) Software Safety Claim Package
2	Safety Argument	1) Software Safety Requirement Analysis Package 2) Hazardous Software Failure Analysis Package 3) Realization Package for Hazardous Software Failure Alleviation 4) Verification Package for Hazardous Software Failure Alleviation 5) Software Safety Process Package
3	Safety Evidence	1) Software Safety Evidence Package 2) Software Safety Process Package

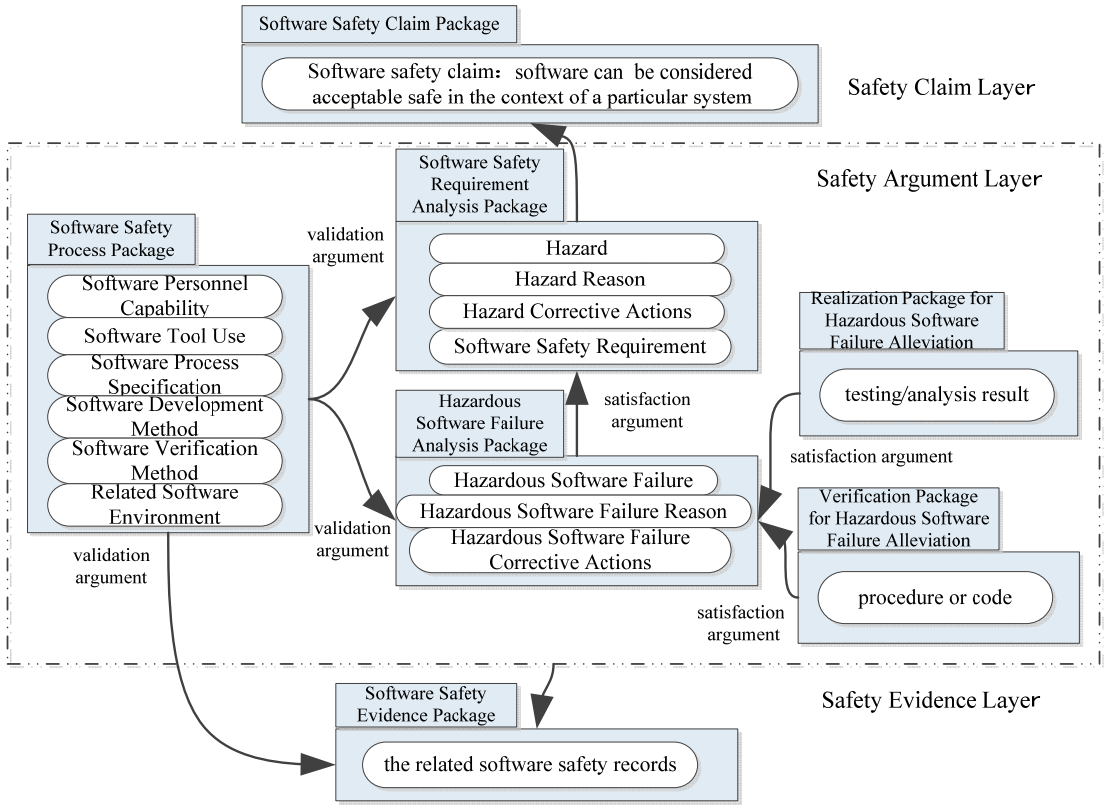

Figure 5. Development Framework for Software Safety Case

The framework considers the construction of the software safety case in terms of the three primary stages:

1) The acquisition of software safety claim

2) The constituent of software safety argument

3) The selection of software safety evidence

From the concept of software safety, software safety claim is set 'software can be considered acceptably safe in the context of a particular system'. Once the software safety claim has been identified, an argument is required to show that the software safety claim has been met. However this is not sufficient to demonstrate the acceptability of the software. In this paper we identify two types of evidence that are required for a complete software safety argument:

(1) **Validation Argument** Demonstration that the set of argument objects is complete and "accurate", e.g. cover all hazards to which the software can contribute.

(2) **Satisfaction Argument** Demonstration that all argument objects have been met.

Satisfaction argument is obtained based on product-based approach, that is, explicit evidence of safety, directly linked to the safety requirements of the system.

First, software contributions to system hazards are acquired from hazard reason and hazard control. Software safety requirements are developed to mitigate the contributions of software to system hazards. Thus, software is acceptably safe if all of software safety requirements are satisfied; Then, that software safety requirements are not satisfied means means software has failed. These failures are called hazardous software failures. Therefore, software safety requirements can be demonstrated to satisfy if hazardous software failures are eliminated or mitigated; Last, hazardous software failures will be not eliminated or mitigated if software has defects(i.e. hazardous failure reason). That is, there are not proper corrective action to avoid the occurrence of hazardous software failures. Program code and testing may verify that software has not defects to contribute to hazardous software failures.

The premise of satisfaction argument is that argument objects are completely and correctly obtained, including software safety requirements, hazardous software failures, code and the result of testing, which can be demonstrated to meet by validation argument. Validation argument is obtained based on process-based approach, that is, recommendation or prescription of development processes and methods, including six aspects that are software personnel capability, software tool use, software process specification, software development method, software verification and related software environment. Safety is determined by an appeal to the quality of the process.

IV. Application Method for the GDFSSC Based on GSN Pattern

GDFSSC has identified the types of evidence that are required and gives guidance on the structuring of the argument, and it is conceptual and independent of approaches that could be used for its implementation. A suitable approach for applying the framework and

constructing software safety arguments based on GSN pattern is presented as followings.

A. GSN Safety Case Pattern for GDFSSC

Safety case patterns are based upon reusable goal structures that can be instantiated to aid the construction of parts of a safety argument. First the GDFSSC is made an abstraction using the connotation of safety case pattern, and then it is described by the GSN pattern. GSN safety case patterns for GDFSSC consist of a collection of highly interrelated patterns which can be combined to form a software safety argument. Dependent on the system being assessed, a selection of patterns can be made. These patterns can then be instantiated and joined together to develop a specific safety argument. There are six GSN safety case patterns for GDFSSC:

1) System Level Safety Argument Pattern
2) Software Contributions to System Hazards Pattern
3) Software Top Level Safety Argument Pattern
4) Software Safety Requirement Satisfaction Pattern
5) Hazardous Software Failure Elimination Pattern
6) Software Corrective Action Satisfaction Pattern

The architecture in Figure 6 shows the interactions of these patterns. When instantiated, a number of patterns have undeveloped goals for which another pattern may provide a suitable decomposition.

Figure 6. Architecture of Safety Case Patterns for GDFSSC

For example Software Top Level Safety Argument Pattern, it is used to identify the argument approach used for demonstrating the acceptability of a particular software safety requirement (SSR). It expands on the undeveloped goal ‘software is acceptably safe in particular system’ from the Software Contributions to System Hazards Pattern.

As at the software level, this pattern identifies the primary claims for developing a software safety argument (validation, satisfaction). For satisfaction, the pattern identifies the individual software safety requirements, and develops an argument that each of SSRs has been satisfied. For each claim about an individual SSR further decomposition of the argument is required before specific items of evidence can be identified. These claims can be developed further using the Software Safety Requirement Satisfaction Pattern. For validation, the pattern identifies the process factors related SSRs, and specific items of evidence can be identified for the claims about process factors.

Figure 7 describes this pattern using GSN pattern, and it contains nine goals, four strategies, two contexts. Two contexts and one goal need to be instantiated, and six goals need to be developed.

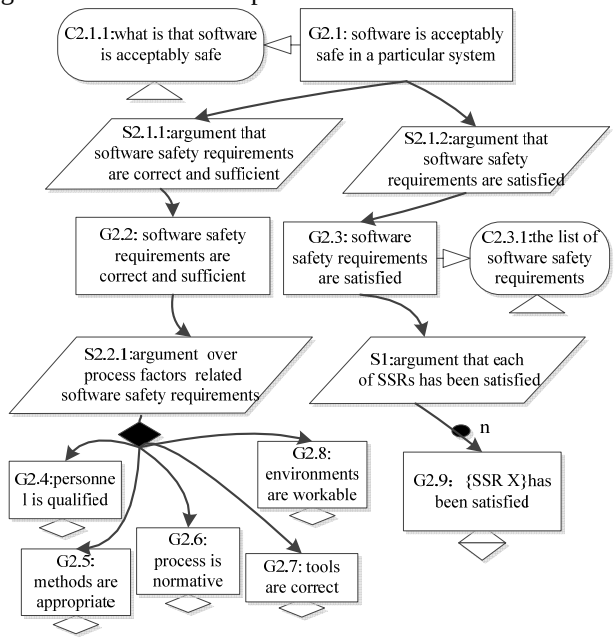

Figure 7. Goal Structure for Software Top Level Safety Argument Pattern

B. Process for constructing software case

To better instantiated, the process for constructing software case based on GSN pattern is given as followings:

Step 1: The list of system hazards needs to be identified, and the process factors related system hazards identification need to be also obtained. Then the severity for each of system hazards needs to be analyzed. Based on which, the system level safety case for specific software can be developed according to System Level Safety Argument Pattern.

Step 2: The hazard reasons and hazard controls for each of system hazards need to be discussed, and then software contributions to system safety case may be constructed in accordance with Software Contributions to System Hazards Pattern.

Step 3: Software safety requirements need to be obtained, at the same time the process factors related SSRs obtainment need to be considered. The safety level safety case can be created on the basis of Software Top Level Safety Argument Pattern.

Step 4: The hazardous software failures, failure reasons and corrective actions need to be analyzed for each of software safety requirements. The safety case for software safety requirement satisfaction can be developed based on Software Safety Requirement Satisfaction Pattern.

Step 5: The safety case for hazardous software failure elimination or mitigation on according to Hazardous Software Failure Elimination Pattern.

Step 6: The program code and the testing result need be obtained. The safety case for the realization of hazardous software failure corrective actions can be constructed based on Software Corrective Action Satisfaction Pattern.

C. Experiment Application

A safety case has been constructed for safety-critical braking control software(BCS) as an experimental example for proposed approach. BCS is part of aircraft braking system(ABS), used to brake the wheels while touching the ground.

According to the process for constructing software case, we first identify the ABS hazards by literature research, historical data, expert interviews and brainstorming. At the same time the severity of ABS hazards is analyzed. Then BCS safety requirements are developed after BCS contributions to the ABS hazards are obtained. And then we analyze the hazardous software failures, failure reasons and failure corrective actions for each SSR using FMEA method. Last we construct BCS safety arguments based on six GSN safety case patterns for GDFSSC. To space limitations, here are just a part of the application result. The ABS hazard and its serverity is shown in table 2. Table 3 gives the BCS contributions and SSR to the hazard 'explosion or fire'. Figure 8 shows the safety level safety case for BCS.

TABLE II.
THE ABS HAZARD AND ITS SEVERITY

Hazard	Severity
explosion or fire	Catastrophic
brake weakness	Hazardous
slip and deviation	Hazardous
oil pollution	Major
false alarm	Minor

TABLE III.
BCS CONTRIBUTIONS AND SSR TO THE HAZARD 'EXPLOSION OR FIRE'

Hazard	Software contributions to hazards	SSR
explosion or fire	BCS should have the function of grounding protection. ABS will not brake if there are the brake instructions when plane is in the air. The brake command is delayed when plane is on the ground.	grounding protection
	BCS should have the function of tire pressure monitoring. The brake is released when a single tire bursts. Antiskid is removed when more than two tires burst	pressure monitoring

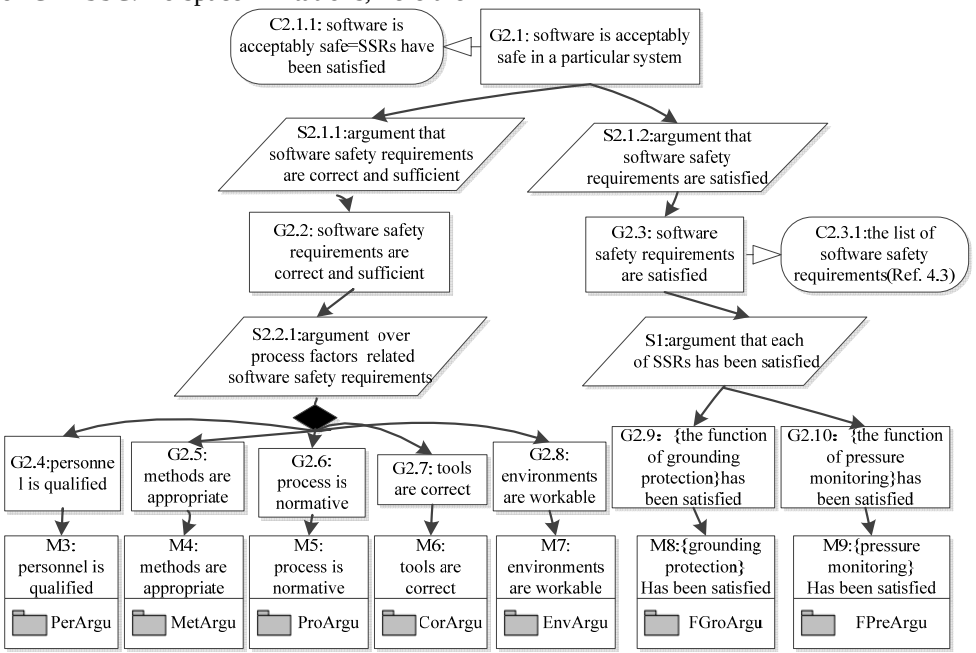

Figure 8. The safety level safety case for BCS

The BCS safety case developed according to the proposed approach in this paper has been approved by some stakeholders, e.g. there are designers, operators, managers and evidence providers. The application result preliminarily shows the proposed approach is feasible and more effective to develop a safety argument for demonstrating the acceptability of software with respect to safety.

V. CONCLUSIONS

Thus, it is a major challenge to construct compelling and general software safety arguments.

This paper has presented a general framework that can identify the types of the required evidence, and its application method for generating software safety arguments. Both the underlying concepts and a method of implementation have been described. This framework including validation argument and satisfaction argument is a feasible approach to demonstrating the contribution of software to system safety, and it would help to

improve the demonstration of software safety. Thus their practical use, it is hoped, will help to produce safer software.

The following work about this research is to apply this framework and its application method on a real project in order to further illustrate the effectiveness of the proposed approach.

ACKNOWLEDGMENT

This work has been partially project supported by the National Defense Pre-Research Foundation of China (No. 513190801) and has been supported by Beihang University of Foundation, China (No. 501LZGF201211 4074).

REFERENCES

[1] "Control of Industrial Major Accidents Hazards Regulations (CIMAH)", 1984.
[2] Cullen THL, "The Public Inquiry into the Piper Alpha Disaster 2 Volumes: Her Majesty's Stationary Office", 1990.
[3] Kelly TP, "Arguing Safety: A Systematic Approach to Managing Safety Cases", University of York, Dept. of Computer Science, 1998.
[4] "DS 00-55, Requirements of Safety Related Software in Defence Equipment". UK: Ministry of Defence, August 1997.
[5] "JSP 430 - Ship Safety Management System Handbook", UK: Ministry of Defence, January 1996.
[6] Bloomfield R., Bishop P., "Jones C.C.M., Froome P.K.D.: ASCAD – Adelard Safety Case Development Manual. Adelard",1998.
[7] http://www.adelard.com/index.html.
[8] Kelly T P, McDermid J A, "A systematic approach to safety case maintenance. Reliability Engineering and System safety", 2001(71): 271-284.
[9] Bate.I., Kelly T., "Architectural Considerations in the Certification of Modular Systems", In proceeding of SAFECOMP 2003.
[10] Wagner S, Schatz B, Puchnerz S, et al, "A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models", 2010 IEEE 21st International Symposium on Software Reliability Engineering. 2010.
[11] Yuan T, Kelly T, "Argument Schemes in Computer System Safety Engineering", Informal Logic. 2011, 31(2): 89-109.
[12] Zeng Fuping, Lu Minyan, Zhong Deming, "Using D-S Evidence Theory to Evaluation of Confidence in Safety Case", Journal of Theoretical and Applied Information Technology, 2013, 47(1):184-189.
[13] "DS 00-56, Safety Management Requirements for Defence Systems". UK: Ministry of Defence, December 1996.
[14] "GSN COMMUNITY STANDARD VERSION 1", GSN contributors, 2011.

Fuping Zeng is a lecturer in the School of Reliability and System Engineering, at Beihang University, China. Her research interests are mainly software safety analysis, safety design and evaluation.

Manyan Lu is a professor in the in the School of Reliability and System Engineering, at Beihang University, China. Her research interests are mainly software dependability engineering, reliability engineering.

Deming Zhong is an associate professor in the in the School of Reliability and System Engineering, at Beihang University, China. His research interests are software quality evaluation, safety engiering.