A Software Safety Process for Safety-Critical Advanced Automotive Systems

Barbara J. Czerny, Ph.D.; Joseph G. D’Ambrosio, Ph.D., PE; Paravila O. Jacob, Ph.D.; Brian T. Murray, Ph.D.; Padma Sundaram; Delphi, Corp.; Brighton, Michigan

Keywords: SW Safety, Safety-Critical Advanced Automotive Systems, By-Wire Systems

Abstract

A new generation of advanced automotive systems are being implemented to enhance vehicle safety, performance, and comfort. As these new, often complex systems are added, system safety programs are employed to help eliminate potential hazards. A key component of these advanced automotive systems is software. Software itself cannot fail or wear out, but its complexity coupled with its interactions with the system and the environment can directly and indirectly lead to potential system hazards. As such, software safety cannot be considered apart from system safety, but the unique aspects of software warrant unique development and analysis methods.

In this paper we describe the main elements of a software safety process for safety-critical advanced automotive systems. We describe how this proposed process may be integrated with an established system safety process for by-wire automotive systems, and how it may be integrated with an established software development process.

Introduction

Expanding demand for further improvements in vehicle safety, performance, fuel economy and low emissions has led to a rapid and accelerating increase in the amount and sophistication of vehicle electronics. Emerging vehicle electronics systems are programmable, with a substantial software component, and are highly complex distributed systems. Increasingly, they are receiving driver inputs and directly controlling essential vehicle functions like braking and steering. They help provide unprecedented improvements in safety, but also have the potential capacity to create safety-related issues when not operating properly.

Such systems are frequently referred to as safety-critical systems (refs. 1-2). Safety is an emergent property of the entire system, so a system safety process is employed to validate that the system operates as intended and does not operate in any way to cause harm (refs. 2-4).

A good system safety process must give careful consideration to the software components of a system. Software is not inherently dangerous. It does not contain or channel energy by itself and it does not fail or wear out. While all subsystems and components of a system must be analyzed for their potential contribution to system hazards, software plays a special role for three reasons. First, for many systems, including the automotive systems under consideration here, the overall functional behavior of the system is defined and determined largely by software. Second, software is known to be complex to analyze since even relatively small programs can have a large number of internal interfaces and interactions. Finally, systems with embedded software tend to rely heavily on software not only for proper functionality, but also for diagnosis and recovery of the non-software parts of the systems, thus adding to size, complexity, and criticality.

Because of its special role in overall system safety, the process of analyzing software for features or omissions that can lead to potential hazards, as well as the process of implementing software hazard control techniques and features, is referred to as software safety. A software safety process assumes that a good underlying software development process is established. The
A software safety procedure based on a set of required high-level tasks, with a corresponding set of recommended methods to implement the tasks, can address the unique aspects of the automotive domain. Without industry-wide agreement, it is not possible to directly incorporate a rigid process standard (e.g., IEC-61508) as a required procedure: any divergence to meet unique project needs will result in a quality audit non-compliance. Therefore, a software safety process must be flexible enough to accommodate different customer desires and requirements and to accommodate the varying needs during the different stages of product development, while at the same time enforcing a structured process that helps ensure software safety. The list of high-level tasks and recommendations in the proposed process comes from identifying best practice elements from existing documents, and integrating these into an adaptable process feasible for the automotive domain.

In this paper we provide an overview of a system safety process applicable to safety-critical, advanced automotive systems. We discuss existing software safety documents and describe some of the elements of a software safety process. We then present a software safety process that may be applied to safety-critical, advanced automotive systems. Finally, we discuss some of our experiences in applying the proposed software safety process and what was learned from these applications.

**System Safety Process Overview**

Application of a system safety program to a product under development can help improve and document the safety of the product. The objectives of a system safety program include, but are not limited to:

1. Identify potential hazards and associated avoidance requirements;
2. Translate safety requirements into engineering requirements;
3. Provide design assessments and trade-off support to the ongoing design;
4. Assess relative compliance of design to requirements and document findings;
5. Direct and monitor specialized safety testing; and
6. Monitor and review test and field issues for safety trends.

A system safety working group is typically established and assigned responsibility for identifying and executing tasks necessary to achieve the safety program objectives (ref. 1). The selected tasks are often based on an established system safety process. Typical process tasks include: system safety program planning, preliminary hazard analysis (PHA), system-level safety requirements development, detailed hazard analysis, detailed safety requirements development, safety verification, and safety case development (ref. 6).

Given that software is a key component of advanced automotive systems, a significant software safety effort is required to help ensure that software-related problems do not result in potential system hazards. Software safety cannot be considered apart from system safety, but the unique aspects of software warrant unique development and analysis methods. In terms of safety, a unique safety process for software is often used. The following sections discuss software safety processes and their elements, and describe a software safety process that may be applied to safety-critical, advanced automotive systems.

### Review of Existing Software Safety Documents

A number of software safety standards and guidelines documents, and methods from various organizations exist today. This section provides a brief overview of several of these standards, guidelines, and methods.

**National Aeronautics and Space Administration (NASA):** NASA-STD-8719.13A provides the requirements to implement a systematic approach to software safety as an integral part of the overall system safety program (ref. 7). The standard is intended to be applied to software that could cause or contribute to the system reaching a specific hazardous state, software that is intended to detect or take corrective action if the system reaches a specific hazardous state, and software that is intended to mitigate damage if a mishap occurs. Safety-critical software is identified during the system and subsystem safety analyses. The level of required software safety effort for a system is determined by its system category and the hazard severity level. The NASA Guidebook, NASA-GB-1740.13-96, provides more detailed information to assist in applying the standard (ref. 8).

**U.S. Department of Defense:** MIL-STD-882C is primarily geared toward system safety, so a detailed software safety process is not addressed (ref. 3). It does, however, provide a software hazard risk assessment process that considers the potential hazard severity and the degree of control that the software exercises over the hardware. The software control categories are based on the level of control the software has over the hazardous function being assessed. It does not provide guidance or recommendations on the tasks or levels of analysis to perform for the determined software criticality.

**Radio Technical Commission for Aeronautics (RTCA):** RTCA/DO-178B provides guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements (ref. 9). This standard provides a means of categorizing software, provides a good description of software development tasks, and links the system safety assessment process with the software development process. No specific safety tasks are detailed and the software categories are not directly translatable to the automotive domain.
Joint Software System Safety Committee (JSSC): The JSSC Software System Safety Handbook, a Technical & Managerial Team Approach, provides management and engineering guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk (ref. 10). The document was a joint effort between U.S. armed forces branches, with cooperation from several governmental agencies, academia, and defense industry contractors. It was intended to capture the “best practices” pertaining to software system safety. The software safety process is integrated with the system safety and software development processes. The process includes identifying generic and system safety-critical software requirements, performing appropriate software safety analyses during each stage of the software lifecycle, verifying that the software was developed in agreement with applicable standards and criteria, and developing a software safety assessment. No specific guidance is provided on determining the level of software safety effort required.

U.K. Ministry of Defense (MOD): MOD DEF STAN 00-55 emphasizes the procedures necessary for specification, design, coding, production, and in-service maintenance and modification of safety-critical software (ref. 11). The standard identifies two categories of software: safety-related software and safety-critical software. Safety-related software is software that relates to a safety function or system and encompasses all Safety Integrity Levels (SILs). Safety-critical software is software that relates to a safety-critical function or system; this is software of the highest SIL (S4), the failure of which could cause the highest risk to human life. It provides guidance and recommendations on the requirements for developing software at the various SILs.

International Electrotechnical Commission (IEC): IEC 61508 part 3 describes the software requirements of the IEC 61508 standard and is intended to be used only after a thorough understanding of parts 1 and 2 of the standard (ref. 12). Part 3 applies to any software forming part of or used to develop a safety-related system as described within the scope of 61508 parts 1 and 2. The level of analysis detail required is dependent on the determined software SIL. The standard includes requirements for safety lifecycle phases and activities to be applied during the design and development of the safety-related software, and requirements for software safety validation. It includes guidance and recommendations for the selection of techniques and measures to use to satisfy the determined SIL. IEC 61508 was intended to be a generic standard from which application specific standards would be developed.

Motor Industry Software Reliability Association (MISRA): MISRA compiled eight detailed reports containing information on specific issues relating to automotive software. The reports are summarized in a single document: Development Guidelines for Vehicle Based Software (ref. 5). The summary report contains information on the software lifecycle and describes three approaches for determining the integrity associated with an ECU (detailed in report 2 on Integrity (ref. 13)). Although the report does not provide an explicit process for software safety that could be directly implemented, it does provide a good overview of issues that need to be addressed when developing vehicle based software, and it contains good recommendations.

APT Research, Inc.: APT’s 15 Step Process for Definition and Verification of Critical Safety Functions in Software was presented at the 2001 ISSC (ref. 14). The 15 steps include identifying system hazards, identifying software safety functional requirements, and tailoring the safety effort to criticality. The method shows the integration of the 15 step process for software system safety into the system safety process and the software lifecycle.

Given the safety-critical nature of some advanced automotive systems, application of techniques above and beyond existing software development techniques must be considered. Suppliers of
automotive systems, and automobile manufactures currently apply various safety analyses in varying degrees to systems being developed. At this point, no single software safety standard has been adopted by the automotive industry. However, there are elements common to most of the identified processes and methods, and this set of common elements can form the basis of a software safety process that provides adequate flexibility. These elements include: software safety planning, requirements analysis, architectural design analysis, detailed design analysis, code analysis, test planning, testing, test analysis, and assessments.

Elements of a Software Safety Process

This section provides an overview of some typical elements of a software safety process. The software safety process proposed in this paper is an integration and adaptation of these elements.

Software Safety Planning: A plan is developed for carrying out the software safety program for the project. This plan identifies the software safety activities deemed necessary for the project, and is developed in conjunction with, and may be part of the System Safety Program Plan. The plan may evolve during the development process.

Software Safety Requirements Analysis: Safety-critical software requirements are identified during the system PHA to eliminate, mitigate, or control hazards related to software. Software safety requirements may also stem from government regulations, customer requirements, or internal requirements. A matrix identifying software safety requirements may be initiated to track the requirements throughout the development process.

Software Safety Architecture Design Analysis: Software components and functions are identified in the software architecture design phase. The software components and functions that implement the software safety requirements or that affect the output of the software safety requirements are identified as safety-critical. A software criticality level is determined for the safety-critical software components and functions. Appropriate levels and types of analyses are identified based on the determined criticality level. This information may be tracked in the software safety requirements matrix.

Software Safety Detailed Design Analysis: The system and software detailed designs are analyzed to help ensure that the designs satisfy the software safety requirements. Subsystem interfaces may be analyzed to identify potential hazards related to interfacing subsystems. Software safety detailed design analysis is performed during the software detailed design phase and during the software implementation and unit testing phases.

Software Safety Code Analysis: The code is analyzed to help ensure that the software safety requirements are satisfied in the implementation. The analysis may check for potentially unsafe states that may be caused by I/O timing, out-of-sequence events, adverse environments, etc.

Software Safety Test Planning: A software safety test plan is developed to help ensure that all identified safety requirements related to or affected by software will be adequately tested. Test procedures should include both nominal and off-nominal conditions. The software safety test plan may be part of the software test plan. Test planning is initiated during the software architectural design phase and continues through the beginning of the software integration and acceptance testing phase. Testing and verification requirements may be included in the software safety requirements matrix. This facilitates the tracking and verification process to help ensure that the software safety requirements are satisfied and appropriately tested and verified.
Software Safety Testing and Test Analysis: Planned software safety tests are performed, and test results are reviewed to help ensure that the appropriate software safety tests have been performed and that safety requirements have been satisfied. Any software problems identified are corrected, and follow-up tests are defined to verify that the identified software problems were corrected and that no additional problems were introduced into the system.

Software Safety Assessments and the Software Safety Case: A software safety assessment is completed at major gate reviews during the project development process. This assessment describes the current state of safety in the software being developed. The assessment indicates any known issues and what will be done to resolve them. Issues from previous safety assessments that have been closed are identified and marked as closed. The software safety case provides supporting documentation and justification as to why the developers believe the software as developed is safe. This documentation is developed from the final software safety assessment. All open issues from the final software safety assessment should be closed. If issues remain open, justification must be provided as to why the open issues are acceptable. Any residual risk associated with software that remains in the system and that has been determined acceptable, is justified in the software safety case. The software safety assessments and the software safety case may be part of the system safety assessments / case respectively.

Software Criticality Levels: Software criticality levels indicate the potential level of risk associated with different software components. Appropriate levels and types of analyses are determined based on the determined software criticality levels.

In the next section we describe the proposed software safety process that combines the elements described in this section.


The software safety process described here contains elements from several existing standards, guidelines, and methods. These elements have been tailored and combined in an effort to make them applicable to the automotive domain; specifically to safety-critical, advanced automotive systems, such as by-wire systems. The main sources used are the JSSC Software System Safety Handbook, NASA-STD-8719.13A, and APT Research, Inc’s. 15 step process.

In addition to the previously discussed criteria that need to be satisfied for the automotive industry, Delphi has certain internal requirements that also need to be satisfied. The internal requirements are that the software safety process must be:

1. integrated into and compatible with the established system safety process for by-wire systems,
2. integrated into and compatible with established software development processes and compatible with established coding standards,
3. adaptable to different projects in different stages of the development process.

Delphi has three main stages of development: Technology Development Process (TDP), Advanced Development Process (ADP), and Production Development Process (PDP). The output of the TDP is a proof of concept technology. During this process we are determining whether a technology concept is feasible. The results of this process are that “we built one and it works”. In the ADP stage the concept that was built in the TDP stage is further developed to make it production ready. In the PDP stage the product is developed for a specific customer application and put into production. The software safety process needs to be useful at varying degrees in each of the three stages of development. Software safety criteria become increasingly
rigorous as we move from the TDP to the ADP, and then to the PDP; the most rigorous criteria being required during the PDP.

Figure 1 shows the software safety process being developed within Delphi for safety-critical, advanced automotive systems. The software safety process is shown in relation to a software development process. Umbrella software development activities shown include: software requirements traceability, discrepancy reporting and tracking, software configuration management, and software quality assurance. An umbrella software safety activity includes tailoring the software safety effort. The lengths of the boxes in the figure show the duration of the tasks and the various phases of development they may be carried out in. The tasks that occur during each of the software safety process steps are the same as those described under the Elements of a Software Safety Process section and will not be further described here.

Integration of the proposed software safety process with the established software development process will not be discussed in detail. Essentially, the integration with the established software development process entails identifying the procedures required to satisfy the process and linking their inputs and outputs with the software safety process. Since any software safety process developed is closely integrated with the software development process, the goal is to ultimately incorporate the software safety process aspects directly into the software development procedures. A detailed list of tasks that may be performed as part of the software safety process will be identified. A tailored subset of the software safety tasks described within the software development procedures are applied if the system being developed is deemed a safety-critical system. Otherwise, they are omitted.

A software safety process cannot be performed apart from a system safety process. Figure 1 also shows the proposed software safety process integrated with Delphi’s system safety process for by-wire systems. Software safety requirements are initially obtained from the system safety PHA when potential hazards are identified that may be related to software. Additional safety requirements may be identified during later system and software process steps. Results affecting the software safety process obtained from system safety analyses are communicated to the software developers during the appropriate stage of development. Likewise, information obtained during the software safety process that affects the system safety analyses results is communicated to the appropriate system developers. The software and system safety assessments start during the early stages of both the system and software safety processes, and continue through the development of the final system and software safety case. A system and software safety assessment is presented at each major gate review, with the final safety assessment forming the basis for the safety case.

The appropriate levels and types of analyses for the high-level process tasks, are determined based on identified software criticality levels, the project’s development phase, and customer requirements. The classification scheme for the proposed software safety process is still under development. The final classification method may be a modification and integration of several techniques.
Experiences With the Software Safety Process

Safety-Critical Technology Development Project: As previously discussed, the goal of a Delphi TDP is to develop concept requirements and to demonstrate proof of concept, which typically involves building a prototype vehicle. As is often common with research and development activities, the development process is characterized by rapidly changing requirements. Operating restrictions can be placed on the use of a prototype vehicle, and in these cases it is not necessary that a safety case and associated validation testing be fully completed. However, the development team may decide that a basic understanding of software safety issues is important during the TDP phase if these issues will have a significant impact on the later phases of product development.

For a recent safety-critical TDP, the software safety planning activity identified that the following tasks should be performed:

- SW Safety Requirements Analysis – Explicitly specify all diagnostic and supervision requirements needed to help mitigate or eliminate potential hazards;
- SW Safety Architecture Design Analysis – Design reviews, software hazard and operability analysis (HAZOP)/FMEA for all SW modules;
- SW Safety Detailed Design Analysis – Design reviews, software HAZOP/FMEA for safety-critical modules;
- SW Safety Code Analysis – No activities required;
- SW Safety Test Planning – Explicitly specify all diagnostic and supervision requirements tests;
- SW Safety Testing – Execute all specified tests on a system bench or in a software development environment;
- SW Test Analysis – Responsible engineers review and approve all test results;
• Verify SW Developed in Accordance with Applicable Standards and Guidelines – Specify coding guidelines, but no formal review of the code; and
• SW Safety Case – Include results of the software safety program in the system safety assessments/case.

The software HAZOP/FMEA for architecture design analysis was effective in providing a timely, broad analysis of the system to confirm which modules were and were not safety-critical and to identify needed software revisions. The software HAZOP/FMEA for detailed design analysis was found to be too time consuming for this TDP, and was ultimately never fully completed. Design reviews were beneficial, but a structured change control/review process was not implemented, so it was not possible to claim at all times that the detailed design reviews were up to date. Overall, the need for a software safety activity was confirmed, but depending on project resources and timing, it may be desirable to postpone the detailed analysis activities until later phases of the development process (i.e., ADP or PDP).

Safety-Critical Product Development Process Project: For a PDP project, Delphi follows a more rigorous system safety process. The safety requirements are tailored to customer requirements. The results of each activity in the safety process are documented in detail and are part of the overall system safety case. Careful attention is given to results of individual analyses, and tests and safety issues are assigned priorities and resolved accordingly.

For a recent PDP project, the software safety planning activity identified that the following tasks should be performed:
• SW Safety Requirements Analysis – Base software safety requirements specification on the results of functional requirements analysis, PHA, software hazard analysis, and hazard testing;
• SW Safety Architecture Design Analysis – Design reviews, software system level FMEA;
• SW Safety Detailed Design Analysis – Design reviews, update the software system level FMEA, detailed software FMEA;
• SW Safety Code Analysis – Code inspections;
• SW Safety Test Plan – Specify all required software safety tests;
• SW Safety Testing and Test Analysis – Test and demonstrate that the safety requirements are achieved. Evaluate compliance with customer required standards; and
• SW Safety Case – Include results from each of the above tasks.

The approach yielded positive results in verifying the software safety design. Since system-level FMEA was performed early in the design process, design improvement recommendations were easy to implement. Given the level of hardware integrity provided by the system design, detailed software FMEA was required and it was very effective in analyzing the integrity of the software design and its potential impact on system hazards.

Summary

In this paper we reviewed existing standards, guidelines, and methods for software safety. In addition, we described some typical elements of a software safety process derived from the reviewed documents, and then presented how these elements are integrated to form the proposed automotive, software safety process. The process is based on a set of required high-level tasks with a corresponding set of recommended methods for implementing the tasks based on the determined software criticality level. The recommended methods are adaptable to the specific
needs of individual projects. Finally, we discussed two project applications, including lessons learned.

To date, the process has had limited application, and additional experience with the process methods is required before consideration can be given to making the process a required procedure. The set of recommended methods will be revised appropriately based on the results of future applications.

References


Biography

B. J. Czerny, Ph.D., System Safety Engineer, Delphi, Corp., 12501 E. Grand River, MC 483-3DB-210, Brighton, MI, 48116-8326, USA, telephone - (810) 494-5894, facsimile - (810) 494-4689, e-mail – barbara.j.czerny@delphi.com.