Original Article
https://doi.org/10.56094/jss.v57i3.206

International System Safety Society

www.jsystemsafety.com

Journal of System Safety

Established 1965 Vol. 57 No. 3 (2022)

Assessing the Software Control Autonomy of System Functions in Safety-Critical Systems

Vu N. Tran[ab], Viet N. Tran[c], Long V. Tran[cd]

[a] *Corresponding author email: vu.n.tran3.civ@us.navy.mil*
[b] *Naval Air Warfare Center - Weapons Division; China Lake, California, USA*
[c] *University of Southern California, Los Angeles, California, USA*
[d] *United States Air Force, Ohio, USA*

Keywords
software safety, software control category, hazard analysis, safety process

Peer-Reviewed
Gold Open Access
Zero APC Fees

Received: 29-Jun-2022
Accepted: 06-Aug-2022
Online: 04-Oct-2022

Cite As:
Tran V.N., Tran V.N., Tran L.V., Assessing the Software Control Autonomy of System Functions in Safety-Critical Systems. Journal of System Safety. 2022;57(3):45-55. https://doi.org/10.56094/jss.v57i3.206

ABSTRACT

Software Control Category (SCC) denotes the degree of control autonomy, command and control authority, and redundant fault tolerance software has over hazardous system functions of safety-critical systems. The use of SCC for determining the software contribution to system risks is a unique feature of the MIL-STD-882E System Safety Standard. A lower SCC designation means that the software system has a greater control autonomy over hazardous system functions, whereas SCC 1 means complete autonomous control. Software with greater control autonomy over hazardous system functions require greater effort to assure reliability and safety. Correct assessment of the SCC level of hazardous system functions is crucial for optimizing the safety property of a system developed under budget, schedule, and resource constraints. Beyond the categorical definitions provided by the MIL-STD-882E Standard, there is little information on conducting an SCC assessment. To close this knowledge gap, we present an SCC assessment method. Our paper will describe in detail the process and rules for assessing SCC. For illustration, we apply our method to assess the SCC of several safety-significant functions of an automobile's brake-assist system.

INTRODUCTION

The two crashes of Boeing airliners model 737 MAX that took 346 lives are a stark reminder of the risk of embedded software in safety-critical systems (Wikipedia, 2022a). As software controllers continue to replace specialized hardware devices in safety-critical systems, the risk of software-induced system failure continues to grow. This trend repeats across multiple industries, including transportation systems, traffic control systems, medical surgery equipment, nuclear power centers, power grid infrastructures, industrial robots, and military weapon systems. Eliminating and controlling the contribution of software to system hazards, i.e., the software risk, is

an objective of system safety engineering. Software system safety, or software safety, is a subdiscipline within system safety that focuses on applying system safety principles and practices to software systems development (Wikipedia, 2022b). Software safety controls the risk of defective embedded software systems triggering consequential system mishaps. Software safety's vital role in system safety engineering continues to grow (Danhauser, 2014).

To determine the software risk in safety-critical systems, the MIL-STD-882E Standard focuses on three risk factors: 1) the system mishap severity, 2) the software control autonomy, and 3) the level of rigor (LOR) of the safety quality assurance process. The mishap severity factor measures the magnitude of the consequence of a system mishap. This factor is well-known in system safety and reliability with established methods for estimating. LOR represents the required level of safety analysis and assurance of hazardous or safety-significant system functions (SSFs). The MIL-STD-882E Implementation Guide (JS-SSA-IG, 2018) provides detailed information on the different LOR levels and tasks. Software Control Autonomy (SCC) expresses the degree of control autonomy, command and control authority, and redundant fault tolerance software has over the SSFs (Safety, 2012). How to assess this risk factor is not discussed in the Standard and its related handbooks and guides. It remains a knowledge gap to be addressed. The need for a systematic and rigorous assessment approach is apparent when dealing with systems with many software controllers supporting many SSFs (JS-SSA-IG, 2018). For instance, a high-end automobile system today can have hundreds of independent processors with software controlling hundreds of SSFs. The code size of such a system can easily exceed one hundred million lines of code (Charette, 2021). A SCC assessment method ensures that the resulting SCC of the SSFs are correct, consistent, and explainable.

We propose a functional approach to assessing the SCC of hazardous system functions. Our approach is aligned with the overall functional approach to hazard analysis in the MIL-STD-882E FHA method. We will provide a detailed description of our SCC assessment method, including the process and rules for systematically and consistently deriving the SCC level of individual SSFs. For illustration, we will use our method for assessing several system functions of an automobile's brake-assist system.

REVIEW OF LITERATURE

To support our review, we started by asking, "How does one approach assessing the control autonomy of the embedded software supporting safety-critical system functions?" In addition to looking through the MIL-STD-882 Standard and supporting documents, we performed a literature search on the IEEE Computer Science Digital and ACM databases, using keywords such as "software control category" and "SCC" "MIL-STD-882", "software control level." The list of relevant papers on the software control category within the last ten years is negligible. Below are notes from our literature review.

The MIL-STD-882 Standard plays a central role in military weaponry, aerospace, and nuclear center system safety. The latest revision E of this Standard emphasizes software system safety by adopting the Functional Hazard Analysis (FHA) method. Within the FHA, the SCC is introduced as a risk measuring factor. The SCC designation describes the software contribution risk associated with an SSF; thus, it is a system function-level designation.

The MIL-STD-882 Standard remains widely adopted within the defense, nuclear, and aerospace sectors. A new revision of the MIL-STD-882, revision F, is under development. We expect that SCC will remain critical in considering the software risk in safety-critical systems.

Several related documents mention the adoption of the SCC but provide little additional information on how to conduct an SCC assessment. The Joint Software System Safety Engineering Handbook (JSSSEH, 2010) provides this concise guidance: "the analyst identifies the software safety-significant functions early in the analytical phase and assigns a mishap severity and software control category to each." The Joint-Services Software Safety Authorities Software System Safety Implementation Process and Tasks Supporting MIL-STD-882 (JS-SSSA-IG, 2018) emphasizes that "accurate assessment of the SCC based upon the complexity of the system, autonomy of the system's functionality, and/or its command-and-control authority is imperative." Both the NATO Guidance on Software Safety Design and Assessment of Munition-Related Computing Systems (AOP-52) (NATO, 2016) and The Nuclear Regulatory Commission's Software Safety Hazard Analysis document (Lawrence, 1996)

adopted the MIL-STD-882 Standard. However, neither discusses how to conduct an SCC assessment.

Literature research in safety requirements engineering reports limited publications on modern software-based system safety methods (Martins and Gorschek, 2017; Martins, L. and Gorschek, 2020). Most system safety methods in practice today are derived from hardware-based methods such as Fault Tree Analysis (FTA), Failure Mode Effect Analysis (FMEA), Failure Mode Effects Analysis and Criticality Analysis (FMECA), and Preliminary Hazard Analysis (PHA). Several reasons for the lack of adoption of software-based hazard analysis methods, including Functional Hazard Analysis, are 1) the lack of software safety teaching in computer science and software engineering curricula, 2) the lack of familiarity with software-based methods among system safety professionals, 3) the lack of published information on the effectiveness of software-based methods, and 4) the slowness in the adoption of software-based methods in safety standards.

One interesting publication is from Robert Smith, where the author uses Fault-Tree Analysis to verify that the different proposed MIL-STD-882E SCC designations are distinct (Smith, 2018). However, the paper does not address the issue we seek to address, which is how to systematically assess the SCC of SSFs.

The Guidelines for Development of Civil Aircraft and Systems (ARP4754A, 2010) outlines a method for assessing the Functional Development Assurance Level (FDAL) of system functions. This method first decomposes individual system functions into subfunctions and allocates these subfunctions to the subsystems that make up the system architecture. The system functions are then assigned an FDAL level based on the structures of their subfunctions. For instance, if a function classified as an FDAL A is decomposable into multiple redundant and independent functions, that function may be reclassified as an FDAL B, i.e., a lower risk function. This FDAL assessment method, supporting the software safety activities in civil aviation, has steps similar to our proposed SCC assessment method.

Our previous papers on software safety (Tran et al., 2016; Tran et al., 2016b; Tran et al., 2016c) describe different aspects of the MIL-STD-882E Functional Hazard Analysis method. However, we did not cover how to conduct an SCC assessment.

SCC AS A RISK FACTOR

The system risk associated with an SSF failure is typically expressed as the product of two risk factors: the mishap severity and the probability or rate of occurrence. For quantitative estimation, historical data, system analysis, and simulation studies are used to quantify the probability of occurrence. For qualitative estimation, likelihood categories are used in place of the numeric probability of occurrence. A risk matrix is then used to combine the two risk factors into a single risk level (Safety, 2012).

SYSTEM RISK = SEVERITY x PROBABILITY (or LIKELIHOOD)

With the use of software to control SSFs, attempts have been made to maintain compatibility between software and hardware reliability calculations for joint system reliability assessment (IEEE-1663, 2016). Software reliability methods such as Software Fault-Tree Analysis and Software Failure-Mode Effects Analysis provide software failure probabilities for use in system risk assessment. The MIL-STD-882E Standard, however, moves away from using probabilistic estimation of software risk. The Standard uses alternative factors to approximate the software contribution to system risk: the degree of software control of an SSF and the level of safety quality assurance rigor. The failure risk of an SSF is deemed higher if the function is controlled by a single software controller (an autonomous control structure) instead of a redundant set of independently built and operated controllers (a redundant fault-tolerant control structure). Similarly, the failure risk of an SSF is deemed higher if the safety quality assurance process employed was less rigorous than required.

SOFTWARE RISK = SEVERITY x SOFTWARE CONTROL CATEGORY x LEVEL OF RIGOR

Figure 1 provides an overview of how the SCC is used in software contribution risk assessment. SCC assessment, i.e., determining the degree of software control autonomy of an SSF, is performed after identifying the SSF as a software-controlled SSF. The output of the SCC assessment is used in conjunction with mishap severity to determine the software safety criticality of the SSF, i.e., the Software Safety Criticality Index (SWCI). The SWCI of an SSF then drives the determination of the initial software risk, the development of risk mitigations, the level of rigor

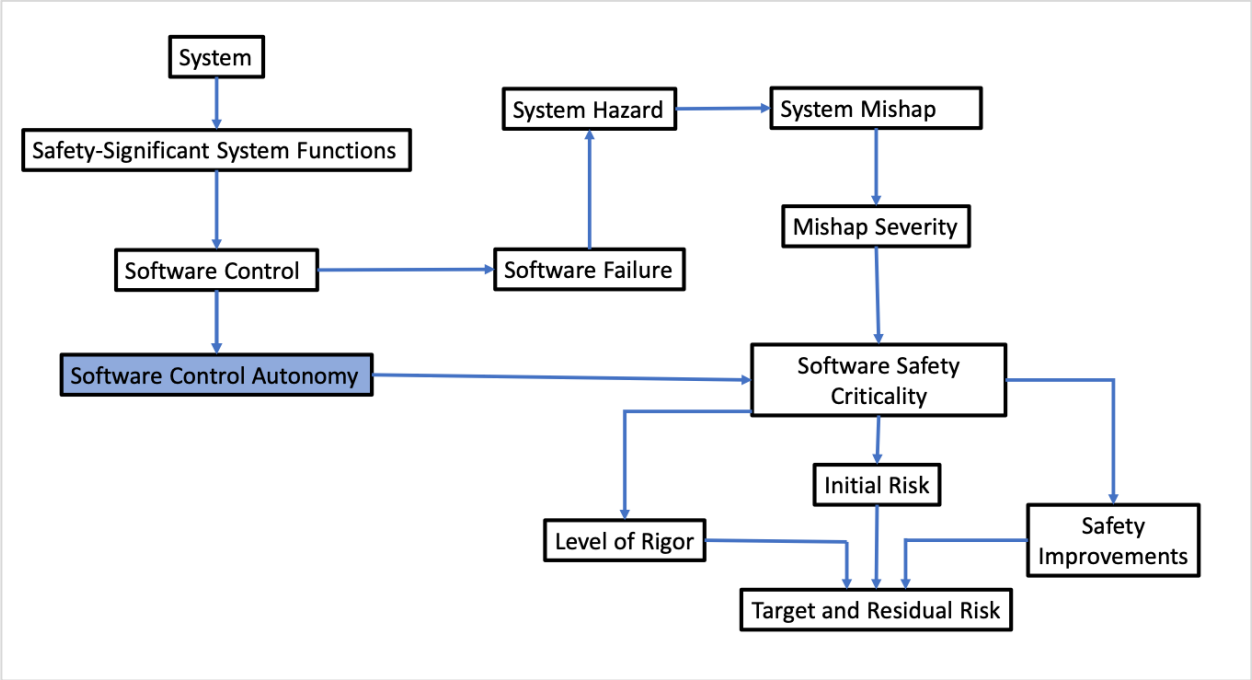

Figure 1: How the SCC is Used to Determine the Software Risk

(LOR) of the safety assurance effort, and ultimately the target and residual risks. Wrong SSC assessment could lead to the erroneous determination of LOR, the third factor of software risk, ultimately resulting in the incorrect software risk assessment. This paper focuses on assessing the SCC of SSFs.

THE SCC CATEGORIES

The MIL-STD-882E Standard defines five Software Control Categories (SCC); some categories have two subcategories (Safety, 2012). The SCC designation is not applied to SSFs controlled strictly by non-software, e.g., hardware or human actions. This section provides our interpretation of each SCC as described by the Standard, emphasizing the distinguishing features of the software control category while adhering to the Standard's definitions. The Standard assumes that the SCC risk factor is only relevant when the software fails to perform its intended control functions.

SCC Level 1: Autonomous (AT)

The AT (SCC 1) designation applies to SSFs controlled by autonomous software functions. Controlling complex system functions could be performed by networked software functions running on independent processors. Failure of any part of the software control puts the system into a hazardous condition leading directly to a system mishap. Autonomy means that there are no external means to detect and intercept the system to prevent a mishap once the software controlling an SSF fails.

SCC Level 2: Semi-Autonomous (SAT)

The SAT (SCC 2a) designation applies to SSFs controlled by software functions that run autonomously, similar to the AT designation. Failure of any part of the software control can put the system into a hazardous condition leading to a mishap. The system, however, is designed to provide a window of opportunity for an independent external actor, e.g., hardware, software, or a human, to detect and intercept the hazardous condition, in a timely manner, thus preventing the system mishap. The external actor does not rely on the faulty software controller to carry out the time-sensitive control action to bring the system back into a safe state.

The SAT (SCC 2b) designation applies to SSFs monitored by software monitoring functions that provide timely safety-significant information to an external actor, allowing the external actor to control a hazardous condition. The monitoring software may or may not control the system function. System mishaps can occur when the monitoring software fails to provide safety-significant information correctly or in a timely way. The independent external actor can also

initiate control actions to prevent a system mishap despite the failed monitoring software.

SCC 3: Redundant Fault-Tolerant (RFT)

The RFT (SCC 3a) designation applies to SSFs controlled by redundant and independent controllers, including hardware, software systems, human actions, and combinations. Independent controllers mean controllers that: 1) receive data from different sources, 2) make independent decisions, 3) take independent actions, 4) are built independently, 5) operate independently, and 6) fail independently. Independence enables redundancy and fault-tolerance. The RFT is assigned when redundancy and fault-tolerance are sufficient to ensure all identified hazardous conditions caused by software failures are controlled. System failure occurred when all redundant controllers failed.

The RFT (SCC 3b) designation applies to SSFs controlled by software functions that depend on an external independent actor's concurrence to initiate control actions. System failure occurs when the software controller accidentally initiates control actions without an agreement. The RFT is assigned when redundancy and fault-tolerance are sufficient to ensure that no control action can be started without external concurrence. Failure of either controller prevents further control actions.

SCC Level 4: Influential (INF)

The INF (SCC 4) designation applies to SSFs relying on software system functions for capturing non-time-sensitive safety-related information. While software functions are responsible for collecting, logging, or displaying safety-related information, they do not control the SSFs, e.g., these SSFs are controlled by hardware. Failure of an INF software function will result in the loss of valuable safety-related information but does not induce a system mishap. The external actor that receives the safety-related information is not expected to initiate any immediate safety action.

SCC Level 5: No-Safety-Impact (NSI)

The NSI (SCC 5) applies to system functions that are not safety significant, i.e., system functions that are not supported by safety-significant software. Failure of the software controlling the system functions will not induce a system mishap.

SCC ASSESSMENT: A FUNCTIONAL APPROACH

SAFETY-SIGNIFICANT SOFTWARE FUNCTIONS (SSSFS)

Each system is functionally composed of a set of system functions. System functions that are of interest to system safety are the safety-significant system functions (SSFs). Software Control Category (SCC) is a system-level property of an SSF. This property is derived from assessing the degree of control software has over the SSF. A software controller could comprise multiple networked software functions running in different subsystems. Each software function could be further decomposed into sub-functions and allocated to different components within a subsystem. Thus, each SSF can be functionally decomposed into a tree where the lowest-level allocated software functions, or safety-significant software system functions (SSSFs), reside at the bottom of the tree. Also, at the bottom are the non-software functions allocated to hardware and human to support the SSF. The assessment of the SCC of an SSF requires understanding the roles and structure of these allocated SSSFs and their relationship with the non-software functions through the lenses of system safety. Figure 2 shows a sample functional decomposition of a system to its SSFs and SSSFs for SCC assessment. Each top-level SSF is decomposed into a set of safety-significant software and non-software functions. A software controller comprises all SSSFs supporting an SSF.

THE SCC ASSESSMENT PROCESS

The SCC assessment process consists of three steps: First, decompose an SSF into a set of software functions and allocate them to the components of the system. Second, assess the SCC level of individual SSSFs supporting the SSFs. Third, adjust the SCC of the SSFs.

In Step 1, Identify the Safety-Significant Software Functions (SSSwFs), each SSSF is decomposed into low-level subsystem functions. System functions that are deemed not hazardous are not analyzed in this step. Decomposing SSSF reviews all the functional elements that support this system function. Multiple SSSFs can share the same low-level subsystem functions. More complex functions

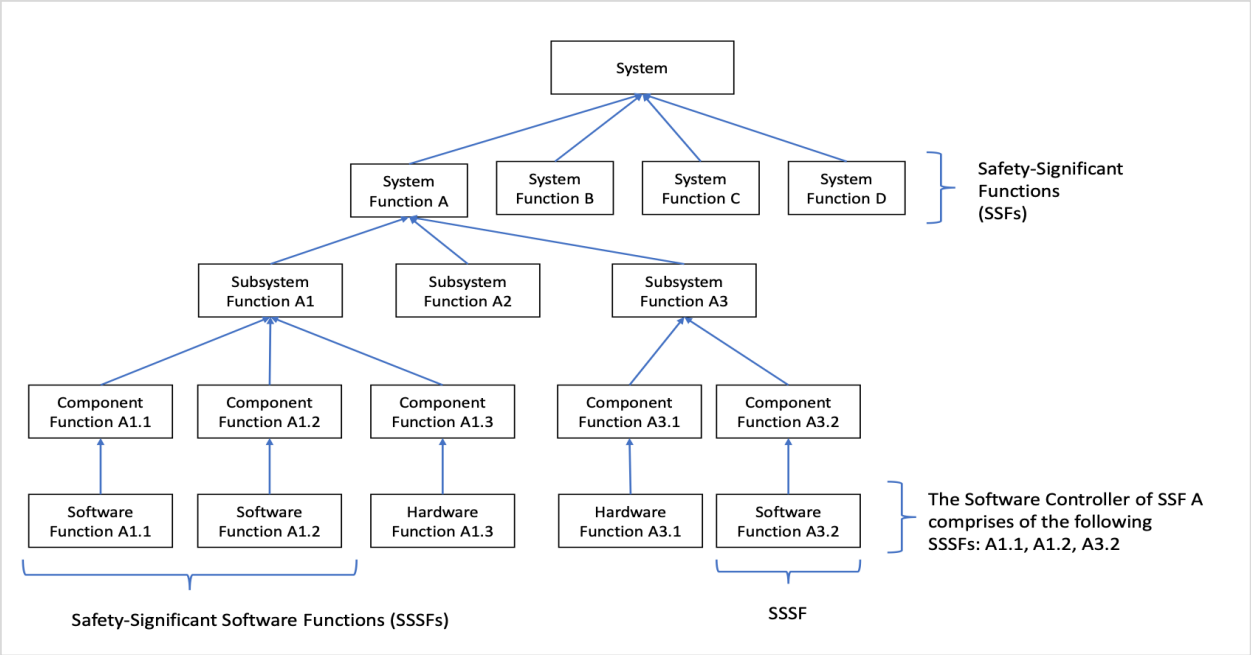

Figure 2: A Functional Decomposition of a System

are further decomposed into smaller functions that could be allocated to the different physical components in the system architecture. Physical components could be hardware, software, or an operator. At the lowest level, the functions allocated to a software component are the SSSwFs that, together with other non-software components, control the SSSF. Step 1 is completed when all SSSFs are functionally decomposed to software and non-software functions allocated to the system architecture components.

In Step 2, Assign SCC to the SSSFs, SCC assessment is performed top-down, starting with the top-level SSF assigned SSC 1, autonomous control. As the analysis progresses down the decomposition tree, each child function inherits the SCC level of its parent function by default. A child function can have a SCC different than its parent function if it is realized by a structure that fits a lower SCC designation. Step 2 is completed when all SSSFs are assigned SCCs.

In Step 3, Reassess the SSF's SCC, the top-level SSF's SCC is reassessed once all the SSSFs supporting it have been assigned SCCs. This reassessment is performed bottom-up until the top-level SSF is reached. At each level, the SCC of a function is reexamined now that the SCCs of its subfunctions are known.

THE SCC ASSESSMENT RULES

There are several rules that guide the SCC assessment of individual functions. These rules can be used in Steps 2 and 3 of the SCC Assessment process.

1. The Top-Level Rule: A top-level SSF is given an SCC 1, i.e., autonomous control, when no other information is available.
2. The SCC Matching Rule: A function whose control structure meets the description of a lower SCC designation is assigned that designation.
3. The Inheritance Rule: A subfunction inherits the SCC level of its parent function by default when no other information is known.
4. The Partition Rule: The SSSFs residing in different physical components can have different SCC levels.
5. The Reuse Rule: An SSSF can have multiple SCCs if it is used by multiple parent functions leading to multiple SSFs. The highest SCC level, i.e., highest control autonomy, will be assigned to the shared SSSF.
6. The Reduced Autonomy Rule: The SCC of a parent function may be lowered if all child functions have a lower SCC than the SCC of the parent function.

In the next section, we will use our method to analyze the rear-ended vehicle collision prevention system for demonstration.

SCC OF REAR-ENDED VEHICLE COLLISION PREVENTION SYSTEMS – A SMALL CASE STUDY

Rear-ended vehicle accidents are among the most common accidents with a risk of vehicle and people injuries (Ryan, 2020). Over the years, many safety braking solutions have been developed to address this problem. This case study will look at a brake-assist system with three safety functions often available in automobiles equipped with the redundant electro-hydraulic brake system. The redundant brake system supports P < 10E-8 probability of braking function loss with a newly introduced electric controlled brake system backed up by a traditional hydraulic brake system. A brake-assist system is designed to augment the vehicle driver, so it remains primarily the driver's responsibility to ensure safe driving. In the rest of this section, we will focus on assessing the SCC level for the SSFs of this simplified brake-assist system.

The three top-level SSFs of our brake-assist system are:

1. The Adaptive Cruise Control (ACC) function controls the vehicle's speed relative to the speed of the front vehicle in a long highway drive without driver interference. While the ACC is more than a brake-assist solution, its Auto-Deceleration function serves to avoid a potential rear-ended collision. Only the Auto-Deceleration function of the ACC will be analyzed.
2. The Autonomous Emergency Braking (AEB) function monitors a potential front collision with obstacles, including another vehicle. When a collision is imminent and there is no driver-initiated braking, the AEB function will initiate braking at full force. It will automatically release the brakes once the vehicle has completely stopped.
3. The Emergency Brake Assist (EBA) function supports the driver in urgent braking. The function monitors the brake pedal to detect a rapid brake attempt by the driver and applies full force to the brake.

Figure 3 provides a simplified illustration of this brake-assist system. The SSFs are identified in the diagram, separated into non-software functions and software control functions, and mapped into a high-level functional architecture. This high-level architectural diagram provides the context for assessing the SCC level of these three SSFs. The assessment will be presented in detail below.

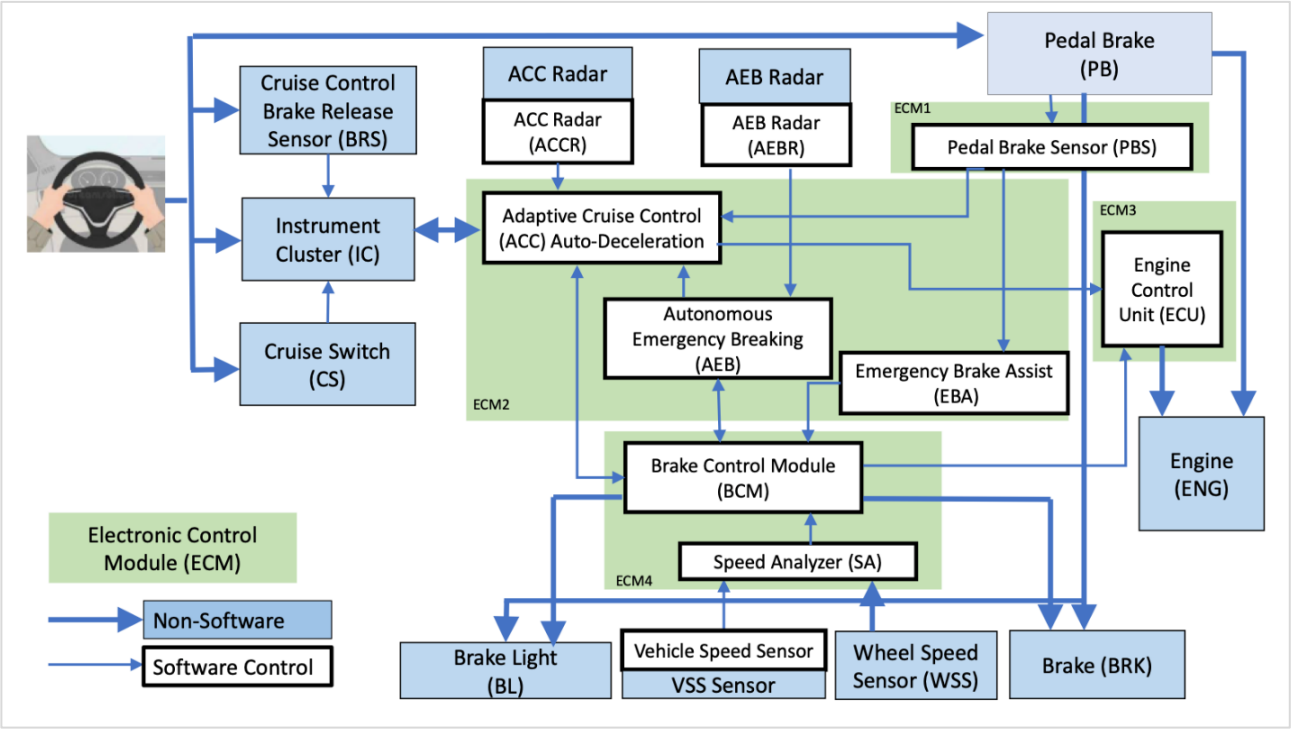

Figure 3: Building Blocks of a Rear-Ended Vehicle Collision Avoidance System

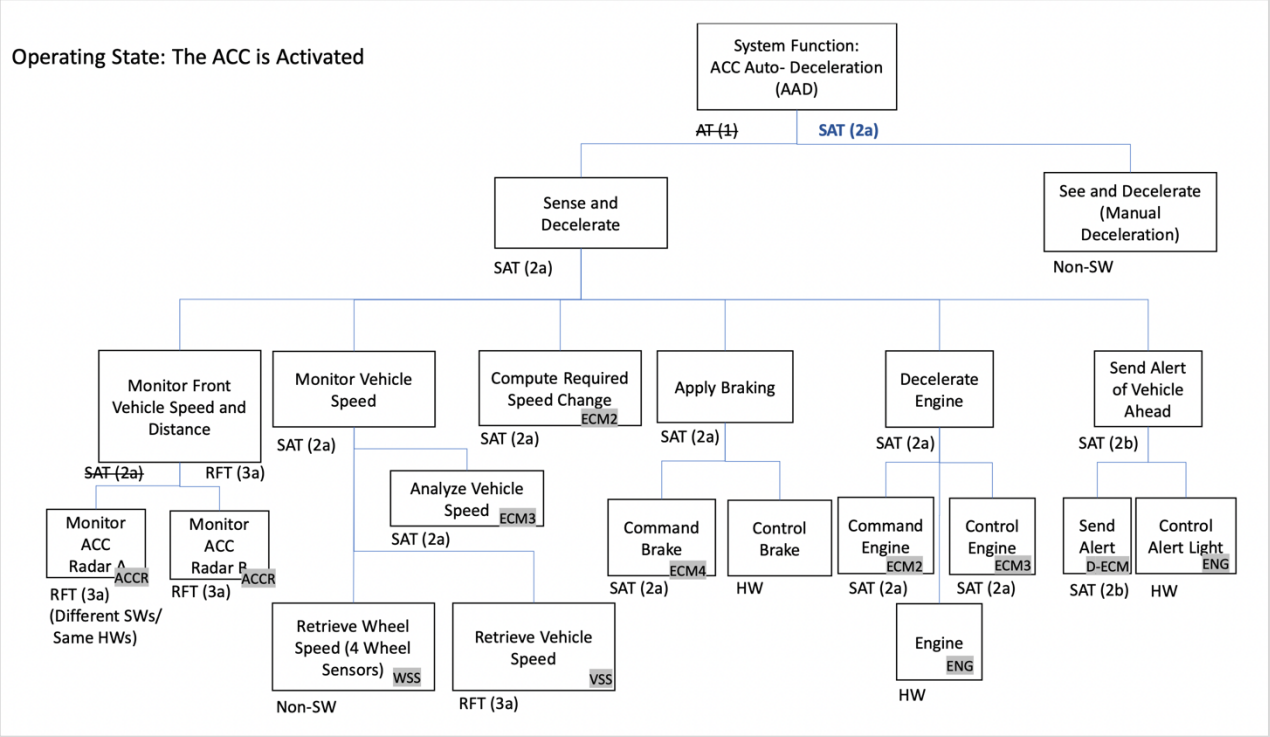

Figure 4: Decomposing Safety-Significant System Function ACC Auto-Deceleration

ACC AUTO-DECELERATION (AAD)

Figure 4 decomposes system function AAD. Function AAD was initially tagged as an AT (SCC 1) function (Rule 1). This function is decomposed into two distinct primary-backup deceleration functions: Sense-and-Decelerate and See-and-Decelerate. Sense-and-Decelerate represents the new computerized function, and See-and-Decelerate represents the manual function performed by humans. The independence of these two functions reflects the requirement that the vehicle driver is responsible for always maintaining visual contact with the front vehicle. The vehicle driver can override the ACC system to slow down the vehicle if needed. Failure of the Sense-and-Decelerate function does not prevent the See-and-Decelerate function from performing the same deceleration. The Sense-and-Decelerate software function thus meets the design criteria of an SAT 2a (Rule 2). The Sense-and-Decelerate function is further decomposed into a set of interacting subfunctions with many to be implemented in software. These software functions inherit their parent's SCC level (Rule 3). The Monitor-Front-Vehicle-Speed-and-Distance function will be realized by a reliable, redundant, fault-tolerant software-controlled radar system, thus qualified for an RFT 3a (Rule 2). Rolling up to the SCC level, the Monitor-Front-Vehicle-Speed-and-Distance function becomes an RFT 3a, according to Rule 6.

Function Monitor-Vehicle-Speed (SAT 2a) comprises three functional components. Function Analyze-Vehicle-Speed determines the vehicle's speed based on the wheel and vehicle sensors. Function wheel speed sensor system is an RFT function with four sensors (Rule 2). The vehicle speed sensor system is redundant to the wheel speed sensor system, thus qualified for an RFT 3a (Rule 2). Function Analyze-Vehicle-Speed, however, is not an RFT function. This function inherits the SAT 2a designation from its parent function Monitor-Vehicle-Speed (Rule 3). Function Monitor-Vehicle-Speed remains an SAT 2a after rolling up the SCC levels.

Functions Control Brake and Control Engine inherit the SAT 2a from its parent according to Rule 3. It may be tempting to classify these functions as AT (1) as they are responsible for controlling the brake and engine systems, respectively. This assignment would be incorrect. Neither function should be given a higher SCC level than its parent function Sense-and-Decelerate (SAT 2a). While the failure of functions Control Brake or Control Engine will fail the function Sense-and-Decelerate, the function See-and-

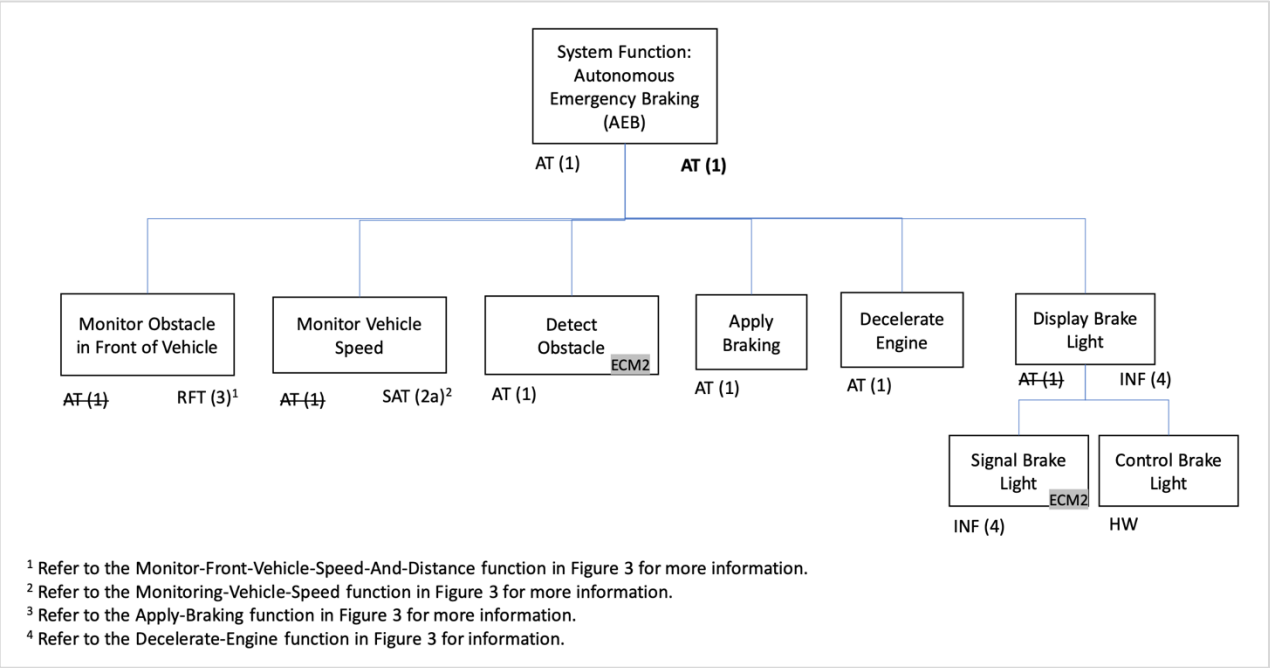

Figure 5: Decomposing Safety-Significant System Function Autonomous Emergency Brake

Decelerate is capable of controlling the hazardous condition. Functions Control Brake and Control Engine are thus SAT 2a functions.

Function Sense-and-Decelerate remains an SAT 2a, making the top-level SSF ACC Auto-Decelerate (AAD) an SAT 2a, according to Rule 6. Function See-and-Decelerate is realized by the vehicle driver and the hydraulic brake system, i.e., non-software.

AUTONOMOUS EMERGENCY BRAKING (AEB)

Figure 5 decomposes function AEB. Unlike function AAD, which is activated manually, function AEB is automatically activated when the vehicle is started. Function AEB function does not assume that the vehicle driver will be responsible for detecting obstacles on the road in front of the vehicle. Instead, the AEB operates autonomously. When function AEB detects an obstacle blocking the vehicle's path, it initiates braking control in full force. Function AEB will not release the brake until the vehicle is completely stopped. Failure of the AEB to detect and brake can lead to a system mishap. The AEB is an AT 1 (Rule 2).

In Figure 6, function AEB is decomposed into subfunctions that scan for obstacles on the road, monitor the vehicle's current speed, analyze the radar signals to detect the obstacles, apply brake and engine control, and display the brake light. Functions Monitor-Of-Vehicle-Obstacles-In-Front-Of-Vehicle, Monitor-Vehicle-Speed, Apply-Braking, and Apply-Engine-Control are decomposed and assigned SCC levels described in system function AAD's assessment. Function Display-Brake-Light is an INF 4 (Rule 2) since there is no expectation that it is used to trigger a safety action due to the short response time required to execute an emergency braking. Function Detect-Obstacles is an AT 1 designed to autonomously initiate emergency braking upon detecting an obstacle on the road (Rule 2). The parent function AEB remains an AT 1.

EMERGENCY BRAKE ASSIST (EBA)

Function EBA relies on the braking action initiated by the vehicle driver to activate emergency braking assistance. The function is thus an RFT 3b (Rule 2) as it cannot start the additional braking action without expressed concurrence for the action by the driver, the independent actor. In addition, failure of the EBA function does not prevent completion of the emergency braking by the vehicle driver who initiated the action as the EBA function is only an assisting function. It is not designed to replace the manual braking action. Further decomposing function EBA into subfunctions shows that function Display-Brake-Light qualifies for an INF 4, Rule 2. The remaining subfunctions retain the RFT 3b SCC level from the parent function (Rule 3).

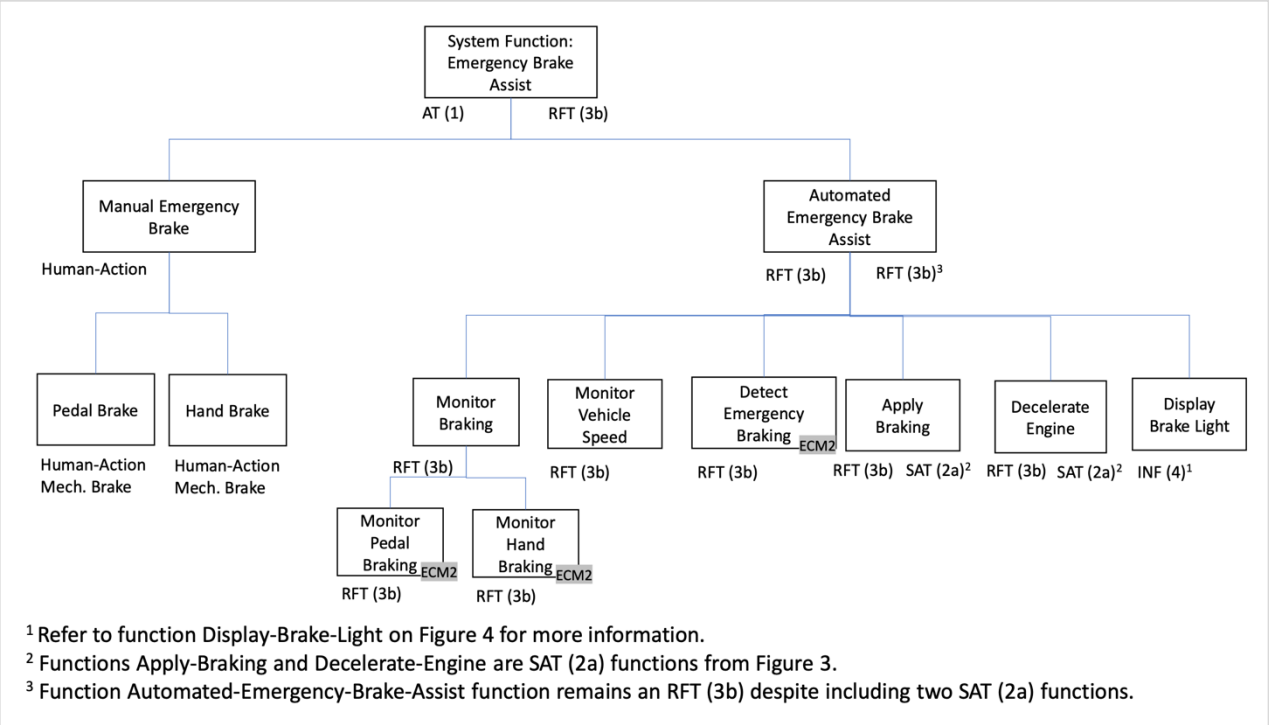

Figure 6: Decomposing Safety-Significant System Function Emergency-Brake Assist

While function Apply-Braking is assigned SAT 2b in Figure 5, it should be noted that it is assigned an RFT 3b here, Figure 6. Applying Rule 5, the SCC level of function Apply-Braking is adjusted as an SAT 2b. We reassess the SCC level of the parent function Automated-Emergency-Brake-Assist once the SCC levels of all its subfunctions have been determined as an RFT 3 due to all the redundant user actions available (Rule 2).

Table 1 presents a mapping of the three SSFs to SCC levels. The map captures 1) the relationship between individual SSFs and the software subfunctions allocated to the components of the system, 2) the degree of control autonomy of the software subfunctions supporting the individual SSFs based on the system architecture, 3) the degree of software control autonomy of the SSFs. There is one column for each SSSF. Below the SSF names are the assigned SCCs. Beneath the assigned SSF SCCs are the allocated software component functions and assigned SCCs. Software component functions supporting multiple SSSFs have multiple assigned SCCs. The rollup of the assigned SCCs to the highest degree of control category for the software component functions is to the right of the table. This SSF-SCC Map provides complete traceability from the SCC levels assigned to the software functions to the SCCs assigned to the SSFs. This traceability simplifies the adjustment of the SCC assignments when an SSF is removed or added.

CONCLUSION

The importance of software system safety as a subdiscipline of system safety continues to grow as software control replaces traditional hardware control in safety-critical systems. The recent high-profile failure of the Boeing 737 MAX systems is a reminder of the software risk in safety-critical systems. The MIL-STD-882E Standard provides a method for determining the software criticality and risk of SSFs. This method relies on the estimation of the degree of control autonomy software has over hazardous system functions. Correct assessment of the SCC level of hazardous system functions is essential for optimizing the safety property of a system developed under budget, schedule, and resource constraints. Presently, little information is available on systematically performing an SCC assessment. Our paper fills this knowledge gap. We presented a functional method for assessing the SCC of the SSFs. For illustration, we provided a detailed description of how to determine the SCC of the brake-assist system functions of an automobile.

Table 1: SSSF – SSC Map

		SSSF			
		AAD	AEB	EBA	
Comp.	**Function**	2	1	3	**SCC**
ACCRA	Monitor ACC Radar A	3a			3a
ACCR	Monitor ACC Radar B	3a			3a
VSS	Retrieve Vehicle Speed	3a	3a		3a
ECM3	Analyze Vehicle Speed	2b	2a		2b
ECM2	Compute Req'd Speed Change	2a			2a
ECM4	Command Brake	2a	1	3b	1
ECM2	Command Engine	2a	1	3b	1
ECM3	Control Engine	2a	1	3b	1
D-ECM	Send Alert	2b			2b
AMBER	Monitor AEB Radar A		3a		3a
AMBER	Monitor AEB Radar B		3a		3a
ECM2	Detect Obstacle		1		1
ECM2	Signal Brake Light		4	4	4
ECM2	Monitor Pedal Braking			3b	3b
ECM2	Monitor Manual Braking			3b	3b
ECM2	Detect Emergency Braking			3b	3b

AUTHORSHIP CONTRIBUTIONS

Vu Tran conceived the presented idea and wrote the paper. Vu Tran and Long Tran co-developed the method. Viet Tran reviewed the literature. Long Tran and Viet Tran co-developed the automobile example. All authors reviewed the paper and contributed to the final manuscript.

COMPETING INTERESTS

All authors declare they have no potential competing interests.

ORCID IDS

Vu N. Tran https://orcid.org/0000-0001-9064-8262
Viet N. Tran https://orcid.org/0000-0002-8078-6925
Long V. Tran https://orcid.org/0000-0003-0343-2768

REFERENCES

[1] Wikipedia (2022a). Boeing 737 MAX groundings. Retrieved Jun 20, 2020, from https://en.wikipedia.org/wiki/Boeing_737_MAX_groundings

[2] Wikipedia (2022b). Software System Safety. Retrieved Jun 20, 2020, from https://en.wikipedia.org/wiki/Software_system_safety

[3] Safety (2012). Department of Defense Standard Practice: System Safety (MIL-STD-882E). Retrieved July 3, 2022, from: http://everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-882E_41682/.

[4] Charette, R. N. (2021). How Software Is Eating the Car. IEEE Spectrum. Retrieved May 8, 2022, from https://spectrum.ieee.org/software-eating-car.

[5] JSSSEH (2010). Joint Software Systems Safety Engineering Handbook.

[6] JS-SSA-IG (2018). Joint Services - Software Safety Authorities - Software Systems Safety: Implementation Process and Tasks Supporting MIL-STD-882E, JS-SSA-IG Rev. B.

[7] NATO (2016). Guidance on Software Safety Design and Assessment of Munition-related Computing Systems. Edition B (Version 1). North Atlantic Treaty Organization. Allied Ordinance Publication.

[8] Lawrence, J. D. (1996). Software Safety Hazard Analysis, NUREG/CR-5430 UCRL-ID-12254. https://doi.org/10.2172/201805

[9] Martins, L. and Gorschek, T. (2017) "Requirements engineering for safety-critical systems: overview and challenges," IEEE Software. https://doi.org/10.1109/MS.2017.265100352

[10] Martins, L. and Gorschek, T. (2020), "Requirements engineering for safety-critical systems: An interview study with industry practitioners," IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2018.2854716

[11] Smith, R. (2018) Verifying Software Control Categories (SCCs) Using Quantitative Fault Tree Analyses (FTAs). Retrieved May 5, 2022, from https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2018/systems/Thurs_21310_Smith.pdf.

[12] ARP4754 (2010). Guidelines for Development of Civil Aircraft and Systems. Aerospace Recommended Practice 4754.

[13] Ryan, T. (2020). "Rear-End Car Accident Statistics," Car Accident Case Law. Retrieved Apr 3, 2022, from .

[14] Danhauser, C. (2022). The Concept of Software Principal for Safety. International System Safety Conference 2022 (Paper in review).

[15] IEEE-1633 (2016). IEEE Recommended Practice on Software Reliability. IEEE Std 1633-2016.

[16] Tran, V., et al. (2021a). Functional Hazard Analysis for Engineering Safe Software Requirements (Extended Version). Presented at the 5th International Conference on Information and Computer Technology – Virtual (CICIT 2021). https://doi.org/10.1109/ICICT52872.2021.00031

[17] Tran, V., et al. (2021b). Assessing the Software Risk Contribution to System Hazards using MIL-STD-882E: Challenges and Recommendations (Extended Version). Presented at the 37th International System Safety Conference – Virtual (ISSC 2021). Retrieved July 10, 2022, from www.linkedin.com/in/vuntran.

[18] Tran, V., et al. (2022c). Functional Hazard Analysis of an Adaptive Cruise Control System - A Software Safety Requirements Engineering Case Study (Extended Version). Presented at the 68th Annual Reliability and Maintainability Symposium. Retrieved July 10, 2022, from www.linkedin.com/in/vuntran.