IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 16, 2020, accepted August 2, 2020, date of publication August 7, 2020, date of current version August 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3015151

An Integrated System Design and Safety Framework for Model-Based Safety Analysis

RAHUL KRISHNAN AND SHAMSNAZ VIRANI BHADA, (Member, IEEE)
Department of Electrical and Computer Engineering, Worcester Polytechnic Institute (WPI), Worcester, MA 01609, USA
Corresponding author: Rahul Krishnan (rkrishnan2@wpi.edu)

ABSTRACT Safety analysis is often performed independent of the system design life cycle, leading to inconsistency between the system design and the safety artifact. Additionally, the process of generating safety artifacts is manual, time-consuming, and error-prone. As a result, safety analysis often requires re- work, is expensive, and increases system development time. Several model-based systems engineering (MBSE) approaches have been developed to automatically generate certain safety artifacts. However, these approaches only cover part of the system design and safety life cycle. To truly leverage the benefits of MBSE, system design must be undertaken together with safety analysis for the entire life cycle, and multiple safety artifacts must be generated from the same model. Moreover, MBSE approaches that require a model transformation between the system design and the safety model suffer from the inability to automatically reflect changes made to a safety artifact in the system and the safety model. This paper presents a framework to integrate the entire system design and safety life cycle using an MBSE approach. Both the system design and the safety data are captured in a single SysML model, from which safety artifacts such as failure modes and effects analysis (FMEA) tables and fault trees are automatically generated. This framework ensures consistency between the system design and the safety analysis by requiring no model transformation, thus reducing the resources required for safety analysis. The proposed Integrated System Design and Safety (ISDS) framework comprises three phases that together cover the entire system design and safety life cycle. In this paper, the application of Phase 1 of the framework to a real-world case study is demonstrated.

INDEX TERMS Model-based systems engineering (MBSE), safety analysis, fault tree analysis (FTA), failure modes and effects analysis (FMEA), systems engineering, hazard analysis, SysML.

I. INTRODUCTION

The need for greater functionality and a rapid decrease in the size of hardware components has led to the design of complex systems with highly interdependent hardware/software architectures. The increase in complexity is largely driven by an increase in scale (number of elements in the system), diversity (number of different elements that make up the system), and connectivity (inter-relationships between elements) [1]. While increased complexity has added value in terms of the performance and robustness of systems [2], ensuring the safe and reliable operation of such systems is becoming more challenging. Accidents are often caused by some unanticipated interaction of software with hardware [3], [4].

Traditionally, to ensure the safety of the system a combination of safety analysis techniques, such as failure modes and effects analysis (FMEA), fault tree analysis (FTA), and hazard analysis, are employed at different stages of the system design life cycle [5]–[8]. These analyses are often performed using independent tools [9], which requires engineers to manually extract the relevant information from the system design models [9], [10]. This is not only time-consuming and error-prone but also leads to a lack of traceability between the system design model and the safety model. Additionally, as the design evolves a failure to update the model in each tool leads to inconsistency between the system design and the safety model and an incorrect safety analysis [9], [11]–[13]. The cumulative effect of these limitations is that the safety assessment of the system requires re-work and is thus time-consuming and expensive [14]. Moreover, it has been shown that the cost of fixing design errors increases drastically as a system progresses through the life cycle phases [15] and that the early detection of design errors dramatically reduces the impact on project schedules [16].

The associate editor coordinating the review of this manuscript and approving it for publication was Qingchao Jiang.

To address the limitations highlighted above, researchers have adopted a model-based systems engineering (MBSE) approach to safety analysis. This improves the completeness and consistency in system development [17], fosters improved communication across design teams [17], provides added traceability between different models of the system [18], and makes integration with other engineering analyses easy [19].

In MBSE, the model represents the single source of truth [20]. Multiple views of the model can be abstracted to serve as input for further analysis. The ability to hide irrelevant information and only analyze those views that contain pertinent information helps manage the system complexity [19]. To implement MBSE, several modeling language have been developed over the years [21]. However, SysML is the preferred language [19] and has become the de facto modeling language in systems engineering [22]–[24].

SysML is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities [25]. Being a standardized language with flexible semantics, it enables the creation of models customized for a specific application [20]. SysML allows for the creation of extensions, which are model components that are not part of the SysML specification. Extensions provide the ability to capture domain-specific information in a single model using stereotypes, properties and tagged values. In an SysML-based approach to safety analysis, extensions can be used for adding safety-related information in the system design model itself. This feature helps in maintaining consistency and traceability between the system design and the safety model. Using extensions, safety artifacts can be automatically generated to reduce the development time and resources required for safety assessment and design changes. These benefits of MBSE and SysML can be leveraged not only to resolve the problems in safety analysis but also to automatically generate safety artifacts at different stages of the system design life cycle.

This paper introduces a framework called the Integrated System Design and Safety (ISDS) framework, which integrates traditional safety analysis techniques with an MBSE design approach for the entire system life cycle. Using a SysML model of the system design, annotated with safety related information, safety artifacts, like FMEA tables, and fault trees are automatically generated. By developing a single SysML model, the framework ensures that consistency between the design and safety models is maintained. Additionally, the automatic generation of safety artifacts seeks to reduce system development time. The key distinction of this framework with respect to other methods in the literature is that it provides methods to ensure system safety throughout the system development life cycle.

The remainder of this paper is organized as follows. Section II discusses the state of the art in the field of safety analysis and related work that deals with the integration of safety analysis with MBSE. Section III introduces the ISDS framework. Section IV demonstrates the application of the framework to the design of a forward collision warning (FCW) system. Section V concludes the paper.

II. LITERATURE REVIEW

To make a system safe, safety analyses must be implemented early in the conceptual phase and throughout the system development and acquisition cycle [5]. Traditional safety analysis techniques that are recommended by safety standards, such as IEC 61508 [26] and ISO 26262 [27] include preliminary hazard analysis (PHA), functional hazard analysis (FHA), FMEA, and FTA [5]–[8]. PHA analyzes identified hazards or identifies previously unrecognized hazards in detail to find causal factors, risks, and mitigation strategies. FHA is used to identify system hazards by analyzing system/subsystem functions. Both PHA and FHA are qualitative, inductive approaches that are performed in the early design stages. Another qualitative hazard analysis technique is a hazard and operability study (HAZOP), which uses guidewords to conceive potential hazards that may arise due to system parameter deviations from expected behavior. FMEA is an inductive, bottom-up approach used to determine the effects of undesired events that occur due to the failure of components or functions and to mitigate the associated risk in the system design. FMEDA is an extension of FMEA that identifies the failure rates of subsystems and the diagnostic capability of the failure mode. FMEA is traditionally adapted for hardware analysis, but researchers have found that systems are becoming increasingly reliant on software to perform their intended functions and that it is now necessary to perform a software FMEA (SFMEA) as well [28]–[31]. Finally, FTA is a deductive, top-down approach used to determine the root cause of a certain undesired event. A fault tree is a model that logically and graphically represents the combination of events or states of a system that together can lead to an undesired event.

More recently, systems theoretic process analysis (STPA) has gained popularity in the safety industry. This is a top-down approach that uses the functional control diagram of the system in its analysis as opposed to a physical component diagram used by traditional safety analysis methods [32]. Several studies have suggested that STPA is better at identifying hazards in a safety-critical, software-intensive system than traditional analysis methods (such as FTA, FMEA, etc.) [32]–[34]. However, the traditional methods are still practiced in industry to analyze safety-critical systems. Sulaman *et al.* [35] compared STPA and FMEA safety analysis techniques on a collision avoidance system and found that both methods were equally good at identifying all types of hazards, but each was better suited to identify specific types of hazards. Consequently, in this paper we limit our focus to traditional safety analysis techniques only. The next section discusses related work that employs one or more of the above approaches and overcomes some of the limitations of the safety analysis process by integrating the approach(es) with a model-based approach.

A. RELATED WORK

Safety standards such as IEC 61508 and ISO 26262 require different safety artifacts to be created iteratively as the design progresses. Over the years, several approaches have been developed that leverage model-based system design to automate the generation of safety artifacts, such as fault trees and FMEA tables. The approaches differ based on the language used to model the system architecture (such as SysML, AADL, etc.) and the language used to model the safety-related aspects of the system's architecture (such as SysML profiles, AltaRica, HiP-HOPS, FSAP/NuSMV, etc.). An overview of the different modeling languages can be found in [36]. This paper only focuses on those approaches that use SysML to describe the system architecture.

Intermediate model transformation is required to create a safety model for approaches that use SysML to model the system architecture and a different language (e.g., AltaRica) to model the safety aspects of the architecture. Safety artifacts such as FMEA tables and fault trees can be automatically generated from this model. In [37] and [38], the authors demonstrate the ability to generate a single type of safety artifact from SysML models. Xiang *et al.* [37] transformed SysML models into reliability configuration model (RCM) specifications from which static fault trees were generated. Hecht *et al.* [38] transformed SysML diagrams that modeled the system structure and behavior into AltaRica models from which an FMEA table was generated. Although both [37] and [38] show the utility in using SysML models to automatically generate safety artifacts, they are limited to a specific type of safety artifact. The entire safety life cycle requires several safety artifacts to be generated as the system design evolves. Transforming the SysML model into a different safety model for each type of safety artifact can be inefficient and time-consuming.

The Me' DISIS method [39] overcomes the problem of requiring a different safety model for each safety artifact by generating FMEA tables and fault trees from the same SysML model. A preliminary FMEA table is automatically generated using SysML diagrams of the system architecture. A database containing failure information of the components is also stored in the SysML model through SysML profiles. The FMEA table is further analyzed, and any missing information is filled in by a safety engineer. The SysML model is transformed into an AltaRica model from which fault trees are generated. However, similar to [37] and [38], this method does not focus on the entire safety life cycle. It does not include a hazard analysis or any provision to use the results of a hazard analysis on the generated artifacts. This would not only help identify any new unknown failure modes for the system under design but would also reduce the reliance on an engineer to identify the relevant failure modes on a case-by-case basis.

Yakymets *et al.* [40] identified this need for an integrated approach that covers all phases of the safety life cycle by developing a safety assessment framework called Sophia [41]. This framework supports the generation of required artifacts, from system specification to verification and validation activities, to comply with the safety standards. The system architecture is defined using SysML or RobotML [42] (a SysML extension for robotic systems). A hazard and risk analysis of the physical and functional architecture identifies system-level ''feared events'' or hazards. Safety information is annotated in the SysML model, and a failure modes and criticality analysis (FMECA) is defined for each hazard. The SysML model is transformed into an AltaRica model from which the fault tree for each hazard is generated. A safety requirement is identified for the system function or component to prevent the hazard from occurring. However, the drawback of this approach or of any model-based safety analysis method that requires a model-to-model transformation [37]–[41], lies in the challenge of maintaining consistency between the safety artifact and the system design and safety model, as well as the increased possibility of data loss during the transformation [9], [43]. To the best of the authors' knowledge, model transformation of the safety artifact back to the safety model has not been implemented. This lack of feedback suggests that the results of safety analyses are not automatically reflected in the safety model or must be manually updated. Additionally, if the transformation is done manually, it can be time-consuming and prone to errors.

To overcome the limitations highlighted above, researchers have proposed the integration of system design and safety models by storing design and safety data in the same SysML model. The safety-related information is stored in the SysML model through a dedicated ''Safety Profile'', created using SysML's extension mechanisms—stereotypes, properties and tagged values. Consequently, no model transformation is required, and the results of the safety analysis can be used to update the safety data stored in the SysML model. Helle's [43] method used SysML models of the functional and logical architecture of the system to automatically generate reliability block diagrams. Safety data, such as component failure rates, were embedded into the blocks using tagged values. Failure cases were identified in the functional architecture using stereotypes. A script traverses the SysML models to generate the reliability diagram for each failure case. While this method only focuses on generating a single safety artifact and is limited in its applicability over the entire safety life cycle, it demonstrates that a safety artifact can be automatically generated by embedding the required safety data directly into the SysML model without the need for a model transformation.

Helle's method failed to include a hazard analysis, which is essential in the early stages of the safety life cycle. Several researchers have developed methods to perform a hazard analysis using dedicated SysML profiles [44]–[46]. Thramboulidis and Scholz [44] developed a model that captures up to 20 different data elements from a hazard analysis. A PHA is applied on the functional architecture of the system, and the resulting data elements are stored in the SysML

model itself. Biggs *et al.* [45] created an SysML profile called SafeML to capture data from a hazard analysis. The SafeML profile is used to annotate the model elements in the SysML diagrams of the system architecture with qualitative safety-related in- formation, such as hazard types, causes, effects, and safety measures, and quantitative safety-related information, such as probability of occurrence, severities, etc. Finally, Muller *et al.* [46] present a hazard analysis profile that captures safety- related data by performing hazard analysis at each stage of the system design process. These approaches [44]–[46] highlight how SysML profiles can be used to store relevant data from a hazard analysis in the same SysML model as the system design.

Mhenni *et al.* [9] leveraged the use of SysML profiles to capture safety-related information for a model-based safety analysis tool called SafeSysE that can generate safety artifacts at each stage of the safety life cycle. It integrates the system design and safety life cycle and uses SysML profiles to store safety-related information in the SysML model. An activity diagram captures the functional architecture of the system from which a functional FMEA containing a list of functions and generic failure modes is automatically generated. Safety requirements are derived for each failure mode. A logical architecture that allocates functions to components is developed using an internal block diagram from which a component FMEA is generated. Information from the FMEA, internal block diagram, and the safety profile of each block is used to generate static fault trees. Finally, safety requirement violations are analyzed using an NuSMV model checker. This work is extended in [47] where component redundancy is also captured in the system architecture model and dynamic fault trees are generated. A key difference between this method and the safety life cycle found in safety standards lies in the elicitation of safety requirements. Standards require system-level safety requirements to be defined early in the life cycle, independent of the system architecture. In [9], the authors define one or more safety requirements for each functional failure mode after the functional architecture is defined. As a result, the risk associated with the violation of the safety requirement is not automatically linked to the failing function and by extension the lower-level system design. Additionally, because hardware and software systems fail differently (assuming randomness is not applicable to software design flaws) it is recommended to further decompose system safety requirements into hardware and software safety requirements. This is further supported by the fact that most software failures arise due to flaws in the requirements [32].

In this paper, a framework that integrates the system design and safety life cycle process is proposed that over- comes the limitations of the previously highlighted works. As recommended by safety standards, system-level safety requirements are generated early in the life cycle. The system architecture is modeled using SysML, and as the design evolves the safety-related data is stored in the same model using SysML profiles. As a result, no model transformation is required, and safety artifacts such as FMEA tables and fault trees are automatically generated from the SysML model only. High-level safety requirements are decomposed into hardware and software safety requirements and allocated to the appropriate lower-level components. The safety artifacts generated at various stages of the life cycle are used to provide evidence to support the safety of the system design.

III. ISDS FRAMEWORK

The ISDS framework introduced in this paper analyzes the operational, functional, and logical design views of the system to automatically generate safety artifacts. As shown in Fig. 1, the framework follows the 'V' system development life cycle model [48] to not only capture the iterative nature of model development but also to highlight the existence of a verification stage for each corresponding development stage. The systems design and safety life cycle are integrated in the ISDS framework through a one-to-one mapping of the different stages in both life cycles, beginning at project definition and ending at verification and validation. The left side of the ISDS framework deals with project definition, where the design and safety data are graphically represented using SysML diagrams and captured in the same SysML model. The right side of the ISDS framework deals with verification methods for each corresponding project definition stage (on the left side). The key contributions of this framework are: 1) the one-to-one mapping of the system design and safety life cycle to automatically generate safety artifacts, such as FMEA tables and fault trees, at different stages of the life cycle; 2) the decomposition of high-level safety requirements to low-level hardware and software safety requirements; 3) the ability to automatically reflect any changes made in the safety artifact back into the SysML model; and 4) the automatic generation of a protocol from the SysML model to complete system safety verification through fault injection simulation.

ISDS is a conceptual framework and is divided into three phases, as shown in Fig. 2. Phase 1 deals with system design and safety analyses at the system level and ends with automatically generating system-level FMEA and fault trees. The results of these analyses are carried into Phase 2 of the ISDS framework, which deals with system design and safety analyses at the sub-system or component level. Phase 3 deals with verifying the safety of the system design. A brief overview of each phase is provided in the section below. However, this paper focuses on the detailed description and implementation of Phase 1 of the ISDS framework only. Phase 2 and Phase 3 will be implemented in future publications.

A. ISDS FRAMEWORK: OVERVIEW OF PHASES

This section provides an overview of each phase of the ISDS framework.

1) PHASE 1: SYSTEM LEVEL ANALYSIS

Phase 1 includes system design activities and safety analyses at the system level only. From the system design life cycle perspective, the required system capabilities are identified

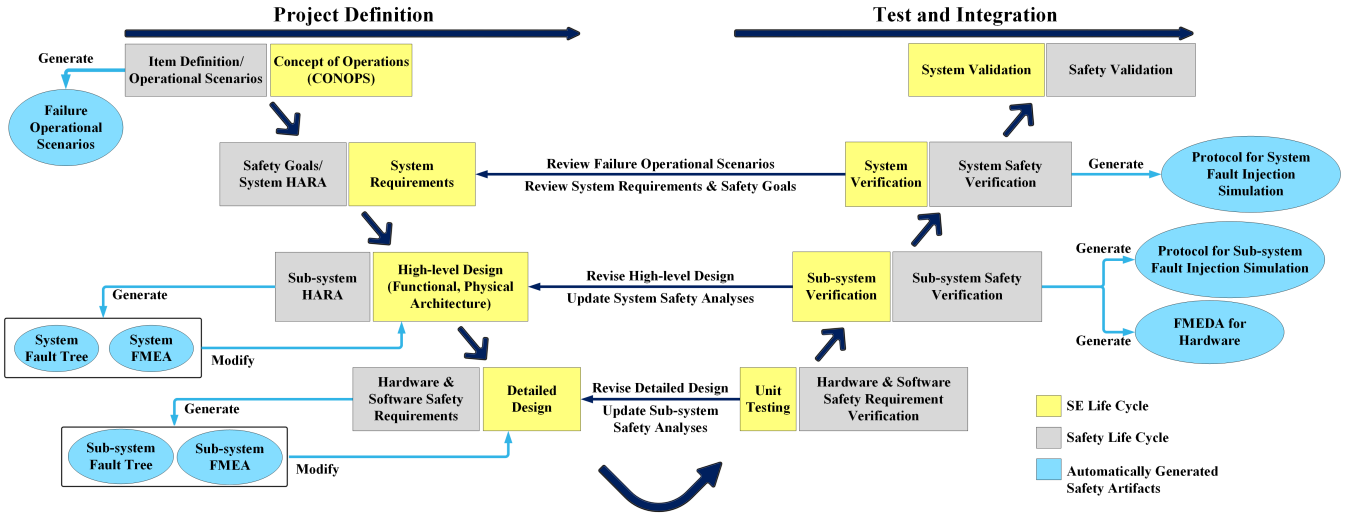

FIGURE 1. The Integrated System Design and Safety Framework (ISDS). The left side of the framework corresponds to the project definition phase of the life cycle, where different design and safety information is captured using different SysML diagrams and captured in a single SysML model. The right side of the framework corresponds to the test and integration phase of the life cycle, where the system design is verified against safety requirements at the component, sub-system, and system levels.

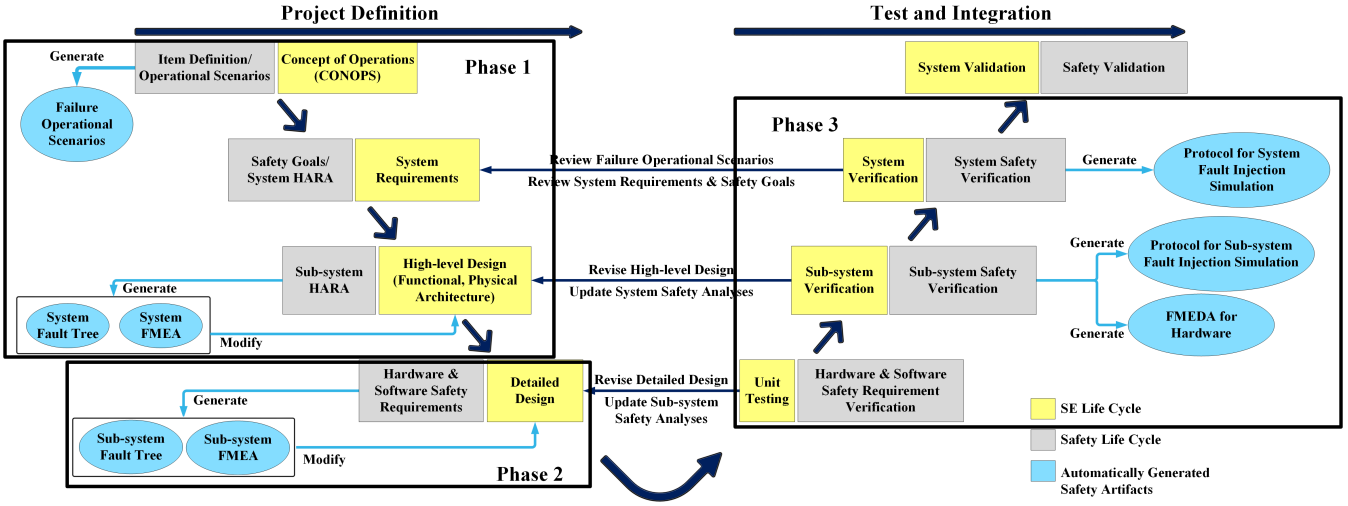

FIGURE 2. Different phases of the ISDS framework. Given the size and scope of the framework, it is separated into three logical phases to make it easier to demonstrate its application. Phase 1 deals with design activities and safety analyses at the system level only. Phase 2 uses the results from Phase 1 and completes design activities and safety analyses at the component level. Phase 3 deals with safety verification at the component, sub-system, and system levels. Verification of system safety is performed at each level by verifying the system design against the safety requirements defined in the corresponding project definition stage.

and transformed into requirements, using which the functional and logical architecture of the system is developed. From the safety life cycle perspective, operational scenarios that could potentially lead to (unwanted) accident scenarios are identified. Hazard analysis is used to identify high-level (or system-level) safety requirements that the system design must not violate. Hazard analysis is repeated for the functional and logical architecture of the system to identify hazards associated with the system functions. A safety engineer also identifies the combination of failure modes that could violate the safety goal. Using the data captured from the hazard analyses and the system architecture, system-level FMEA tables and fault trees are automatically generated. The results of the safety analyses reveal the modifications required in the system design to make it safe. Once the required design changes have been made, safety analyses are repeated to update the safety artifacts.

2) PHASE 2: SUB-SYSTEM LEVEL ANALYSIS

In Phase 2, the system design is further refined by decomposing the system architecture into the system hardware and software architecture. Component blocks that make up a sub- system are identified. For each hazard associated with a function (identified in Phase 1), a corresponding hardware and software safety requirement is generated. These requirements are allocated to the appropriate sub-system. Sub-system FMEA tables are automatically generated by identifying the different failure modes of the components

that make up the sub-system. A safety engineer identifies the failure modes or a combination of failure modes that could violate the hardware or software safety requirement. These failure modes are used to generate the sub-system fault tree. Safety mechanisms are added to these failure modes to prevent a safety requirement violation. At the end of Phase 2, a set of FMEA tables and fault trees are automatically generated for each sub-system.

3) PHASE 3: SYSTEM VERIFICATION

Phase 3 focuses on verifying the safety of the system design. Verification is performed based on the system hierarchy, starting from low-level or component design and ending at the high-level or system design. At the component level, the behavior of each component in a sub-system is verified to ensure that it does not violate the hardware or software safety requirements of the sub-system. Component failure modes for which the mitigation strategy or safety mechanism does not prevent a violation of a hardware or software safety requirement are identified, and recommendations are provided for corrective actions. These recommendations are implemented as design changes in the detailed design stage of the framework. The safety analyses are updated to stay consistent with the latest design. This verification process repeats until all hardware and software safety requirements are met at the component level. Different methods, such as model checking or fault injection, can be used to perform this verification.

At the sub-system level, hardware FMEDA tables are automatically generated for each sub-system. The FMEDA contains the failure modes of the components in the sub- system and additional data such as failure rates, safety mechanisms to prevent the failure modes, and the diagnostic coverage provided by the safety mechanism against the failure mode. This data can either be added manually by an engineer or obtained through fault injection. To complete the fault injection simulation, a protocol containing the design and safety information of each sub-system is automatically generated from the SysML model. The design information in the protocol is obtained from the system and includes the organization of components in the sub-system. The safety information in the protocol includes the component failure modes of the sub-system and their corresponding safety mechanisms. The fault injection simulation identifies the failure modes that violate the sub-system-level safety requirements. Similar to the component-level verification process, recommendations are provided for corrective actions against failure modes that violate the sub-system-level safety requirements and are implemented as design changes in the high-level design stage of the framework. The system-level safety analyses are updated to maintain consistency with the latest design. This verification process repeats until all safety requirements are met at the sub-system level.

Finally, the system-level verification is performed through fault injection simulation. The system design is tested in a simulation environment against the failure operational scenarios identified in Phase 1. The failure modes of the sub-system are injected one at a time (through software) into the system design, and the behavior of the system is observed. If a high-level (or system-level) safety requirement is violated, the system design is modified to prevent the violation. Multiple failure modes can be injected to observe a combination of failure modes that could violate a safety requirement. This process is repeated for all failure modes and is complete when none of the failure modes or combinations of failure modes violate any high-level safety requirement. The next section introduces the SysML profile developed for Phase 1 of the ISDS framework to store the safety-related information in the SysML model.

B. ISDS FRAMEWORK: SAFETY PROFILE

The ISDS framework integrates the system design and safety models by using SysML profiles to store the design and safety data in the same SysML model. Consequently, no model transformation is required to create the safety model and generate the safety artifacts (as highlighted in the literature review). The next section provides a detailed description of the safety profile used in this framework.

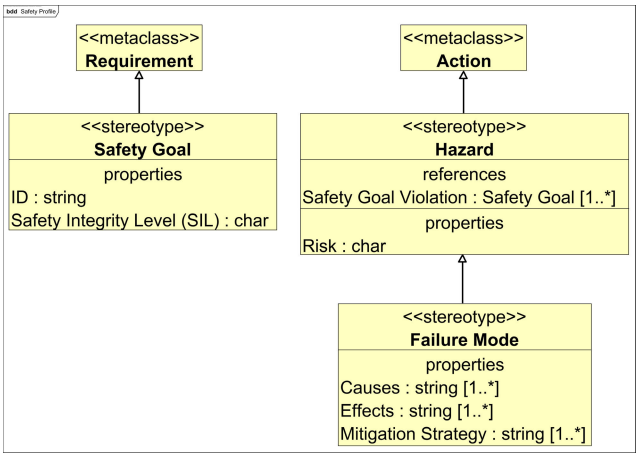

FIGURE 3. Safety profile for Phase 1 of the ISDS framework. The safety profile illustrates how native SysML elements are extended using stereotypes to represent safety-related information. Both the requirement and action elements are native SysML elements. The triangle from parent element to child elements represents an extension or inheritance relationship and can be read as "is a type of". The child element contains additional properties to capture safety-related information.

Fig. 3 illustrates the safety profile for the ISDS framework. Because safety goals are high-level safety requirements, the SysML requirement metaclass is extended as a "safety goal" stereotype, which contains an additional property to capture the associated risk. Based on the risk assessment framework of ISO 26262, the safety goal risk can be one of five values—QM, A, B, C, or D (in order of increasing risk) [49]. System functions are represented using the "action" metaclass in an activity diagram. Hazards are deviations from the expected behavior of functions that pose a risk to the system. As a result, hazards are represented using the "hazard" stereotype, which is an extension of the "action" metaclass. Safety goal

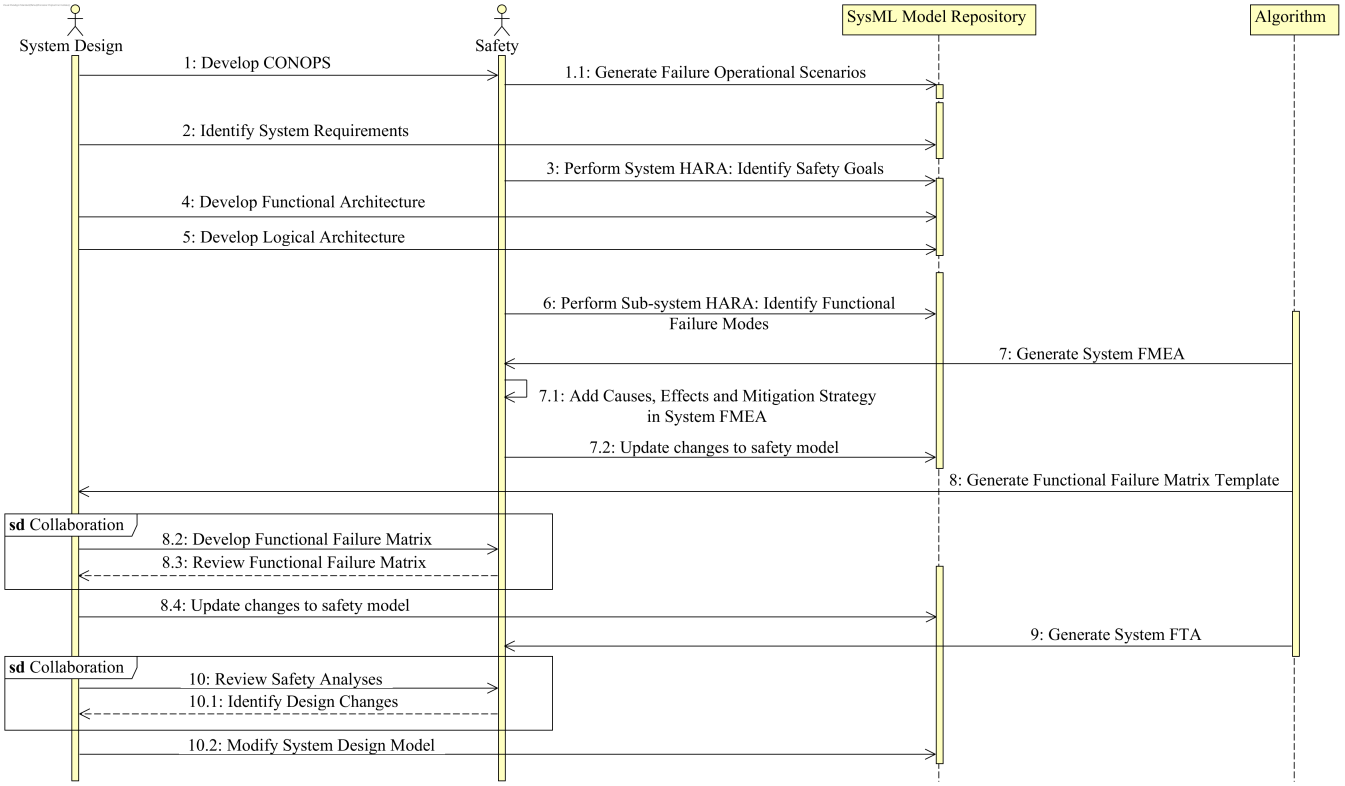

FIGURE 4. Sequence diagram representing the sequence of activities in Phase 1 of the ISDS framework from both the system design and safety perspectives. The diagram highlights the responsible entity of an activity and the destination for the results of an activity. The iterative and collaborative nature that emerges from integrating the system and safety life cycle is clearly seen in this diagram.

violations and inherited risk are added to the hazard through properties. The [1..*] next to the safety goal violation property refers to its multiplicity; that is, it indicates that one hazard can be associated with one or more safety goal violations. The ''failure mode'' stereotype is an extension of the hazard stereotype and inherits the referenced safety goal violation and the associated risk of violating the safety goal. Additionally, the failure mode stereotype contains causes, effects, and mitigation strategy as properties. A single failure mode can have multiple causes, effects, and mitigation strategies. The [1..*] next to each property highlights the ability to contain one or more values, that is, its multiplicity. Because the safety profile contains a placeholder (through stereotypes and properties) for all required safety-related data, not only can safety artifacts be automatically generated from the SysML model, but any changes made to the data in the safety artifact can also be automatically reflected in the SysML model. This is crucial for maintaining consistency between system design and safety analysis. The next section provides a detailed description of Phase 1 of the ISDS framework.

C. ISDS FRAMEWORK: PHASE 1 DESCRIPTION

The description for each stage in the ISDS framework starts with the design life cycle-related activity, followed by the safety life cycle activity and the SysML modeling activity. Fig. 4 illustrates the sequence of activities performed during Phase 1 of the ISDS framework.

1) DEVELOP A CONCEPT OF OPERATIONS AND GENERATE FAILURE OPERATIONAL SCENARIOS

The design life cycle begins with developing the system's concept of operations by identifying a set of system capabilities from the perspective of its stakeholders and the system operational scenarios. The safety life cycle identifies a set of operational scenarios under which the system could be unsafe. The operational scenarios are captured by identifying the set of variables (and its states) that cover the range of scenarios the system will encounter during operation in its intended environment.

2) IDENTIFY SYSTEM REQUIREMENTS

System capabilities that were identified in the previous step are transformed into requirements. These requirements are captured in the SysML model using a requirements diagram.

3) PERFORM SYSTEM HAZARD AND RISK ASSESSMENT

Hazard analysis is performed on the system capabilities to identify potential system-level hazards. These hazards represent states of the system that could pose significant risk to the system itself or its environment, such as loss of life or damage to property. The identified hazards are converted to high-level safety requirements (also known as safety goals) and are captured in the SysML model using a requirements diagram. A risk level (or safety integrity level) is assigned to a safety goal using the risk assessment framework from

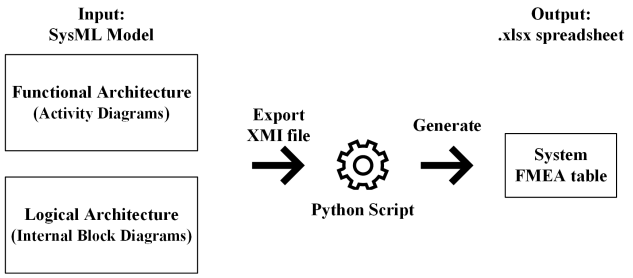

FIGURE 5. Process for automatically generating FMEA from SysML model.

ISO 26262. The hazard linked to the safety goal is assessed in context with the failure operational scenarios identified in step 1 to derive the exposure (probability of occurrence of failure operational scenario), severity (consequence if hazard occurs), and controllability (probability of mitigating the failure scenario) factors. The three factors combine to give a safety integrity level or risk associated with the safety goal.

4) DEVELOP FUNCTIONAL ARCHITECTURE

The system architecture that is developed as part of the design life cycle has two components, the functional architecture and the logical architecture. The requirements are translated into system functions to create the functional architecture. The functional architecture is captured in the SysML model using multiple activity diagrams.

5) DEVELOP LOGICAL ARCHITECTURE

The logical architecture provides the structural design of the system, allocates functions to the blocks or sub-systems, and identifies the interconnections and data flows between each block. The logical architecture is captured using a block definition diagram and an internal block diagram.

6) PERFORM SUB-SYSTEM HAZARD AND RISK ASSESSMENT

In the safety life cycle, potential sub-system-level hazards are identified using hazard analysis. The functions allocated to each sub-system are analyzed for potential deviations from expected behavior and captured as hazards. The effects of each hazard are identified by tracing the safety goal that is violated if the hazard occurs. These hazards are added to the functional architecture activity diagrams. The risk associated with a hazard is determined by the safety goals that would be violated when the hazard occurs. If there are multiple safety goal violations, the highest risk is inherited. At this point, the SysML model is enriched with the necessary information to automatically generate the system FMEA and the system fault tree.

7) GENERATE SYSTEM FMEA

The first safety artifact that is automatically generated is the system FMEA. Fig. 5 provides an overview of the process. An XML metadata interchange (XMI) file of the SysML model is generated and parsed using a Python script to create the FMEA table. Iterating over each function for all blocks

Algorithm 1 Generate System FMEA

```
Input: XMI of SysML model
Output: Spreadsheet of System FMEA
  SET root = root node of XMI;
  Find elements of type = 'SysML Block' in root
  for each SysML Block do
    Add Block name to spreadsheet
    Find all Functions in the Block
    for each Function do
      Add Function to spreadsheet
      Identify activity diagram of Function
      Extract Failure Modes from activity diagram
      Add Failure Mode and linked data to spreadsheet
    end for
  end for
```

in the logical architecture, the hazards are extracted as the failure mode. The corresponding safety goal violation and associated risk is also extracted. An .xlsx file (spreadsheet) is generated containing the block name, its functions, failure modes, safety goal violations, and associated risk. The algorithm for generating the FMEA table is shown in Algorithm 1. The generated FMEA is further analyzed by a safety engineer to fill in the causes, effects, and mitigation strategy. The Safety Profile has a container for the data entered by the engineer, and the file is imported back into the SysML model to store the added information. A Python script parses the edited spreadsheet and iterates through the previously generated XMI file of the SysML model to add causes, effects, and a mitigation strategy for each failure mode to the SysML model. As a result, any changes made to the safety artifact by the engineer are automatically reflected in the SysML model. The algorithm for updating the FMEA table is shown in Algorithm 2.

Algorithm 2 Update System FMEA

```
Input: XMI of SysML model, Edited Spreadsheet
Output: Updated Spreadsheet of System FMEA
  for each row in spreadsheet do
    Find SysML Block in XMI file
    Find corresponding Function in the Block
    for each Function do
      Identify activity diagram of Function
      Identify referenced Failure Mode
      Add Causes, Effects and Mitigation Strategy as
      properties to Block in XMI file
    end for
  end for
```

8) DEVELOP FUNCTIONAL FAILURE MATRIX

Once the system FMEA is generated, the design and safety engineers work together to identify the combination of failure modes within a sub-system that can violate a safety goal. The failure mode combination is limited to an order of two as recommended by safety standards [27]. Based on the

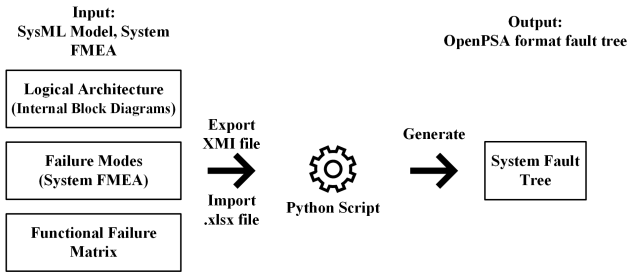

FIGURE 6. Process for automatically generating a fault tree from a SysML model.

structure of the FMEA, a Python script generates a template of the functional failure matrix for each safety goal. The matrix contains the failure modes of each sub-system listed in the rows and columns. A marked cell in the matrix signifies that the failure modes in the corresponding row and column cannot individually violate the safety goal but combine to violate a safety goal. It is important to note that only failure modes of different functions can combine to violate a safety goal, as a given function cannot exhibit multiple behaviors/misbehaviors simultaneously.

9) GENERATE SYSTEM FTA

The next safety artifact that is automatically generated from the SysML model is the system fault tree. Like the artifact generation process for the FMEA, fault tree generation is automatic, instantaneous, and consistent with the latest system design. Fig. 6 illustrates the process. The structure of the fault tree is shown in Fig. 7. Because the interconnections and data flows between the blocks are known (using the internal block diagram), a failure in a block can be considered either an internal failure of the block (based on its failure modes) or a failure in the input to the block. The fault tree is generated using a Python script that parses the XMI file of the SysML model and the data from the FMEA table. Using the internal block diagram, the script traces the flow of information from output to input and generates a sequence of events (or failures) for each safety goal violation. Only those failure modes that contribute to the safety goal violation are considered internal failures of the block. Internal failure of the block is represented using a logic OR gate of failure modes that violate the specific safety goal. The inputs to the OR gate are the singular failure modes that violate the safety goal or the combination of failure modes that have been identified in the functional failure matrix. The combination of failure modes is represented using a logic AND gate. The algorithm for generating the fault tree is shown in Algorithm 3. The fault tree is generated in the Open-PSA [50] model exchange format and can be viewed by tools that use the XFTA engine [51].

IV. CASE STUDY: APPLICATION OF PHASE 1

This section presents a case study that demonstrates how Phase 1 of the ISDS framework is used to automatically generate system-level FMEA tables and fault trees from a

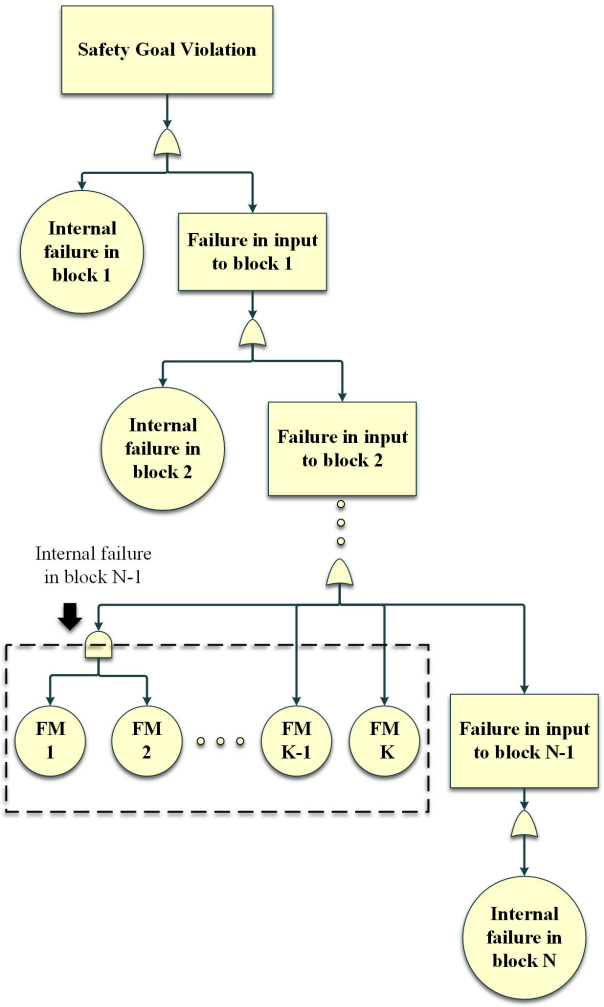

FIGURE 7. Structure of the fault tree generated. This structure represents the sequence of blocks from output to input. Internal failure of the block consists of either individual failure modes or the combination of failure modes that violate the safety goal. The combination of failure modes are represented using logical AND gates.

SysML model. The case study deals with the development of an FCW system. It should be noted that the main focus of this study is the application of the ISDS framework rather than a comprehensive design of an FCW system.

The FCW system identifies the potential for an impending crash situation for a vehicle and alerts the driver using an audio or visual signal. The system is implemented using forward-looking sensors that provide information about the objects and roadway in front of the vehicle, a processing unit that takes the sensory information as an input to determine the probability of a crash, and a visual and/or audio alert interface in the vehicle to alert the driver. The system capabilities and requirements of an FCW system have been developed based on [52]. A detailed description of the different stages of Phase 1 of the ISDS framework is provided below.

A. IDENTIFY FAILURE OPERATIONAL SCENARIOS

The first step in the ISDS framework is to identify a set of operational scenarios that represent the situations under which

Algorithm 3 Generate System FTA

Input: XMI of SysML model, Functional failure matrix, System FMEA
Output: System FTA in OpenPSA format
initialization
SET root = System Block of XMI model
SET visited-block = 0
Function *Generate-Tree (block)*:
  Identify all inputs to root
  while input > 0 **do**
    for each input to Block **do**
      if input is NOT in visited-block **then**
        Add-Block-To-FaultTree (input)
  visited-block
        = input ⋃ visited-block
  Generate-Tree
        (input)
      end if
    end for
  end while
Function *Add-Block-To-FaultTree (block)*:
  Add OR Gate
  Add Internal-Failure-of-Block (block)
  Add failure in input to Block
Function *Internal-Failure-of-Block (block)*:
  Identify block in FMEA
  Add OR Gate
  Add failure modes that violate safety goal
  if Violation in Functional Failure Matrix **then**
    Add AND gate
    Add failure modes that violate safety goal
  end if

TABLE 1. Environment variables and their states for a vehicle equipped with a forward collision warning system.

Environmental Variable	Variable State
Weather	Clear, wind, rain, snow, icy surface, etc.
Roadway surfaces	Asphalt, gravel, concrete, dirt
Pedestrian presence	Present, absent
Roadway type	Interstate highway, undivided highway, local roads, etc.
Traffic conditions	Vehicle in front, vehicle behind, vehicle front and behind, no vehicles present nearby
Traffic zones	School zone, construction zone, tunnels, garages
Vehicle speed	Very high (>80mph), High (80mph>=V>60mph), Medium (60mph>=V>30mph), Low (<=30mph)

the system should avoid hazardous (or unsafe) behavior. The set of environmental variables (and its states) for a vehicle is shown in Table 1.

The different states for each variable can be combined to create a driving scenario during which the system should avoid unsafe behavior. For example, a potential driving scenario could be that the vehicle is travelling on an interstate highway at high speed (130 kph >= V > 100 kph) with another vehicle in front on an icy surface with no pedestrians present.

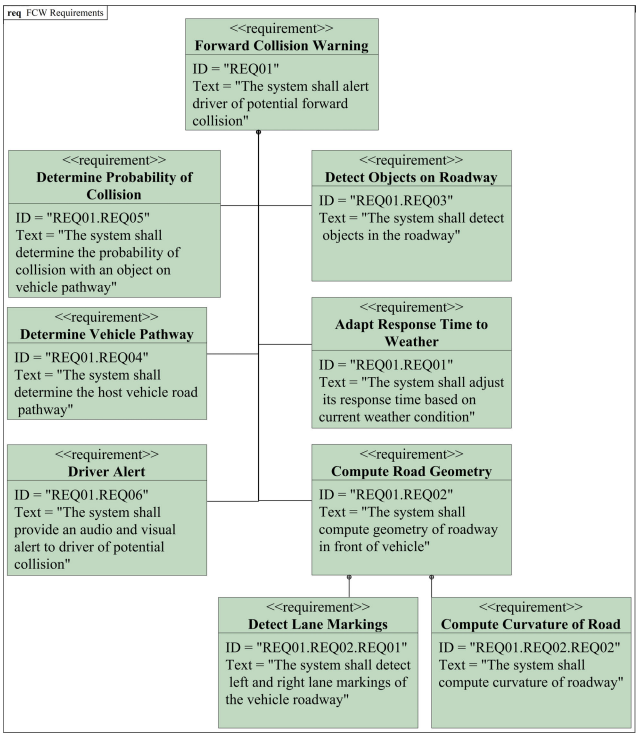

FIGURE 8. SysML system requirements diagram of an FCW system.

B. IDENTIFY SYSTEM REQUIREMENTS

As mentioned earlier, the system capabilities and requirements were obtained from [52]. These requirements are captured in the SysML model using a requirements diagram, as shown in Fig. 8.

C. PERFORM SYSTEM HAZARD AND RISK ASSESSMENT

System hazard analysis is performed on the FCW system using the HAZOP method to identify system-level or vehicle-level hazards. The HAZOP analysis uses the following guidewords:

- Loss of function
- Too early
- Too late
- Not requested

Based on these guidewords the following vehicle-level hazards associated with the FCW system were identified:

- Unexpected loss of FCW system
- Early engagement of FCW system
- Delayed engagement of FCW system
- Unexpected engagement of FCW system

These hazards represent unsafe behavior of the FCW system. High-level safety requirements or safety goals are developed to prevent these hazards from occurring, as shown in

TABLE 2. Risk assessment for an operational scenario based on the risk assessment framework from ISO 26262.

Assessment Variable	Description
Hazard	Delayed engagement of FCW system
Safety goal	SG3: Prevent delayed engagement of FCW system
Failure operational scenario	Driving at high speed (130kph>=V>100kph), on an interstate highway, with another vehicle in front, on an icy surface, with no pedestrians present
Crash situation	Host vehicle crashes into leading vehicle
Severity	S3 – Life-threatening or fatal injuries
Controllability	C3 – Difficult to control
Exposure	E4 - Highly likely speed
Risk or safety integrity level	D (Highest risk level)

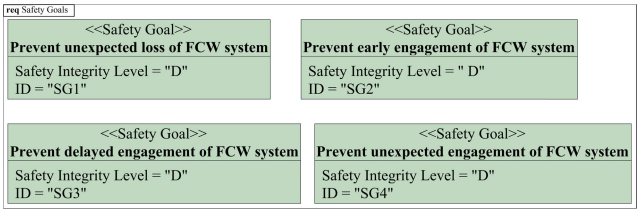

FIGURE 9. SysML diagram of High-level safety requirements or Safety Goals for an FCW system identified using the HAZOP method.

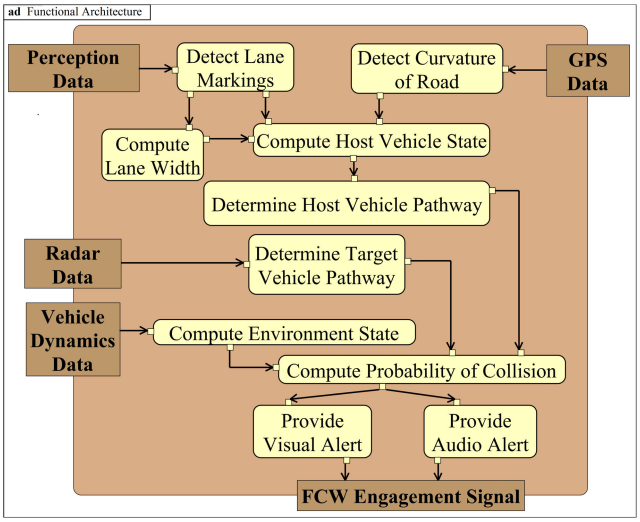

FIGURE 10. SysML Activity diagram representing the functional architecture for an FCW system.

Fig. 9. Safety goals are captured in the SysML model using the safety goal stereotype.

The safety integrity level or risk associated with each safety goal is determined based on the risk assessment framework from the ISO 26262 standard. For each failure operational scenario identified in the first step, the exposure, severity, and controllability are determined, and the equivalent risk is computed. Table 2 shows an example for assessing the risk associated with safety goal 3: prevent delayed engagement of FCW system under a specific operational scenario. This risk assessment is repeated for each operational scenario

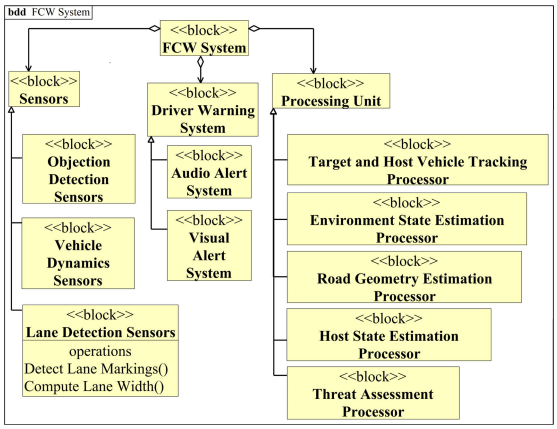

FIGURE 11. SysML block definition diagram representing the logical architecture of an FCW system. It shows the different logical blocks of the system and the functions allocated to them. For the sake of readability, the function allocation to a block is only shown for the lane detection sensors block.

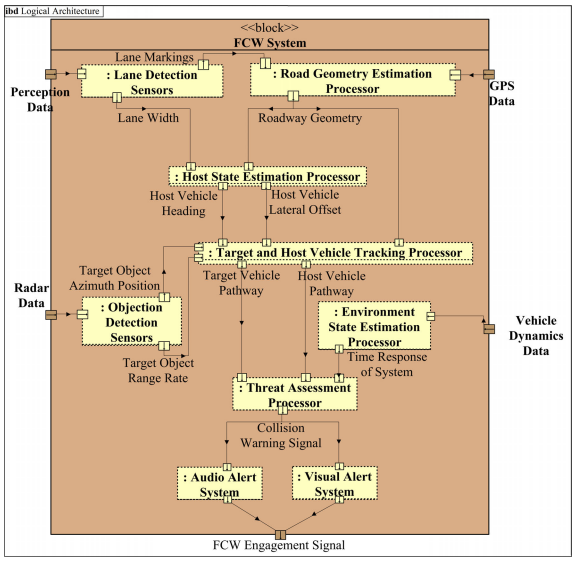

FIGURE 12. SysML internal block diagram representing the logical architecture of an FCW system. It shows the interconnections between the different logical blocks and the data flows.

developed in the first step. The safety goal is assigned the highest risk level that was identified for the set of operational scenarios. The risk is added to the safety goal model element as a property.

D. DEVELOP FUNCTIONAL ARCHITECTURE

The functional architecture contains the set of functions that satisfy the system requirements. The system functions and their interconnections are captured in the SysML model using an activity diagram, as shown in Fig. 10.

E. DEVELOP LOGICAL ARCHITECTURE

The logical architecture is developed using a block definition diagram and an internal block diagram. The block definition diagram shown in Fig. 11 identifies the different logical blocks or sub-system of the FCW system. The system functions are also allocated to each block. The internal block

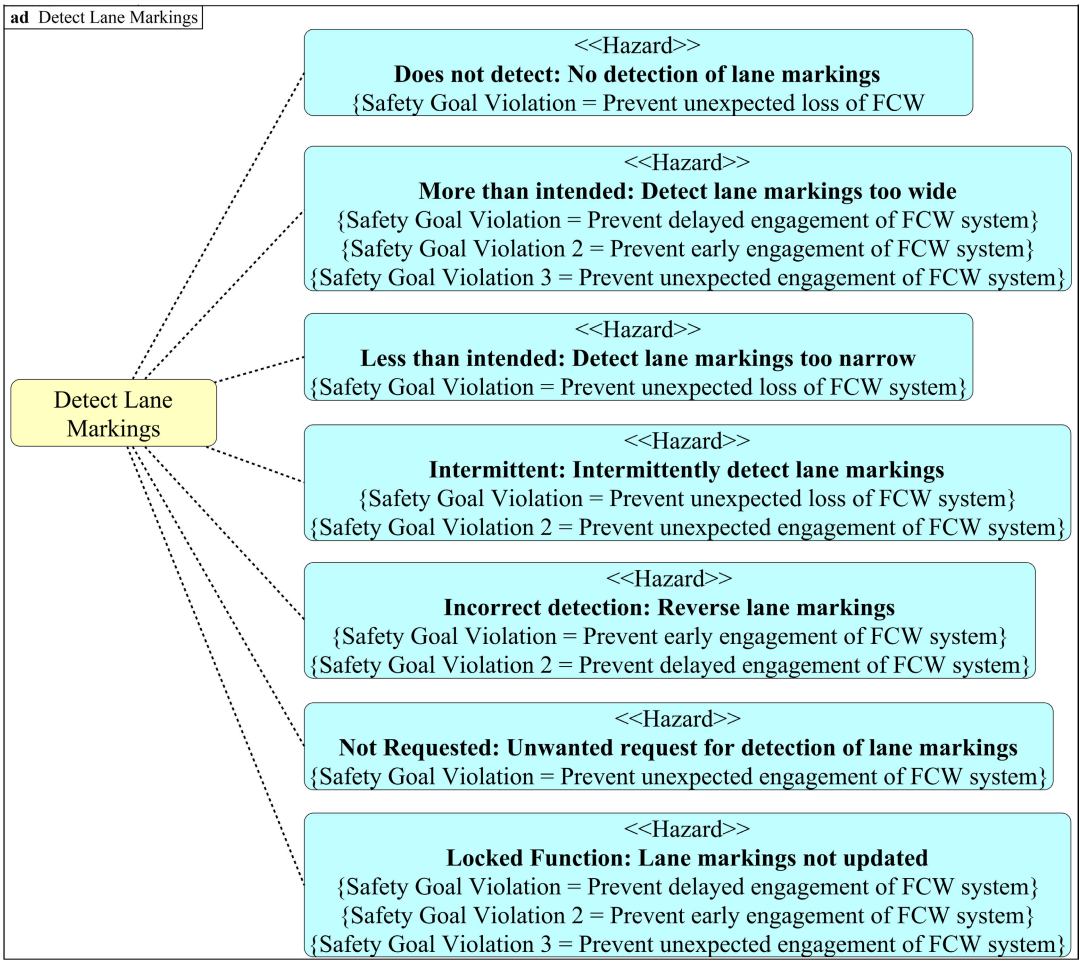

FIGURE 13. SysML Activity diagram showing the hazards identified for the Detect Lane Markings function.

TABLE 3. FMEA for two functions of the Lane Detection Sensors block of the FCW system. The '–' indicates that the information is to be filled by an engineer.

ID	Block	Function	Failure Mode	Cause	SG Violation	Risk	Effect	MS*
1	LD Sensors	Compute Lane Width	**Does not** :Compute Lane Width	-	SG1	ASIL D	-	-
2	LD Sensors	Compute Lane Width	**More than intended** :Compute Lane Width	-	SG2, SG3, SG4	ASIL D	-	-
3	LD Sensors	Compute Lane Width	**Less than intended** :Compute Lane Width	-	SG1	ASIL D	-	-
4	LD Sensors	Detect Lane Markings	**Locked Function** :Detect Lane Markings	-	SG2, SG3, SG4	ASIL D	-	-
5	LD Sensors	Detect Lane Markings	**Not Requested** :Detect Lane Markings	-	SG4	ASIL D	-	-
6	LD Sensors	Detect Lane Markings	**Incorrect Decision** :Reverse Lane Markings	-	SG2, SG3, SG4	ASIL D	-	-
7	LD Sensors	Detect Lane Markings	**Intermittent** :Detect Lane Markings	-	SG1, SG4	ASIL D	-	-
8	LD Sensors	Detect Lane Markings	**Less than intended** :Detect Lane Markings	-	SG1	ASIL D	-	-
9	LD Sensors	Detect Lane Markings	**More than intended** :Detect Lane Markings	-	SG2, SG3, SG4	ASIL D	-	-
10	LD Sensors	Detect Lane Markings	**Does not** :Detect Lane Markings	-	SG1	ASIL D	-	-

***Mitigation Strategy**

diagram is used to identify the interconnections between the sub-systems and the data flows between them. Fig. 12 illustrates the internal block diagram for the FCW system.

F. PERFORM SUB-SYSTEM HAZARD AND RISK ASSESSMENT

Sub-system hazard analysis is performed on each function in each block in Fig. 11, using the HAZOP method to identify sub-system-level hazards (hazards associated with sub-system functions). The guidewords used in this HAZOP study were:

- Loss of function
- More than
- Less than
- Locked function
- Intermittent output
- Wrong direction
- Not requested

Guidewords are used to brainstorm potential hazards to sub-system functions. These hazards are added to a sub-system function, as shown in Fig. 13 using the hazard stereotype. For each hazard, the potential safety goal violation is added as a property. Fig. 13 shows the hazards captured in the SysML model for the ''detect lane markings'' function.

```
<opsa-mef name="FCW_Fault_Tree">
  <define-fault-tree name="FCW_Fault_Tree">
    <define-gate name="FCW_Engagement_Signal_Safety_Goal1_Violation">
      <or>
        <basic-event name="Visual_Alert_System_fails"/>
        <gate name="Failure_in_input_to_Visual_Alert_System"/>
        <basic-event name="Audio_Alert_System_fails"/>
        <gate name="Failure_in_input_to_Audio_Alert_System"/>
      </or>
    </define-gate>
    <define-gate name="Failure_in_input_to_Visual_Alert_System">
      <or>
        <basic-event name="Threat_Assessment_Processor_fails"/>
        <gate name="Failure_in_input_to_Threat_Assessment_Processor"/>
      </or>
    </define-gate>
    <define-gate name="Failure_in_input_to_Audio_Alert_System">
      <or>
        <basic-event name="Threat_Assessment_Processor_fails"/>
        <gate name="Failure_in_input_to_Threat_Assessment_Processor"/>
      </or>
    </define-gate>
    <define-gate name="Failure_in_input_to_Threat_Assessment_Processor">
      <or>
        <basic-event name="Environment_State_Estimation_Processor_fails"/>
        <basic-event name="Target_and_Host_Vehicle_Tracking_Processor_fails"/>
      </or>
    </define-gate>
  </define-fault-tree>
  <model-data>
    <define-basic-event name="Visual_Alert_System"/>
    <define-basic-event name="Threat_Assessment_Processor"/>
    <define-basic-event name="Environment_State_Estimation_Processor"/>
    <define-basic-event name="Target_and_Host_Vehicle_Tracking_Processor"/>
  </model-data>
</opsa-mef>
```

FIGURE 14. A small section of the fault tree XML for violation of safety goal 1 of an FCW system.

G. GENERATE SYSTEM FMEA

The XMI of the SysML model is parsed using a Python script to generate the system FMEA for the FCW system. The different blocks and their allocated functions are identified using the block definition diagram shown in Fig. 11. The hazards associated with each function are extracted using the information in Fig. 13. Based on the safety goal violation for each hazard, the risk is extracted from Fig. 13. The data is formatted and stored in a spreadsheet, as shown in Table 3. For the sake of readability, the FMEA generated only contains information from one block (LD sensors) of the FCW system. The causes and effects columns are left empty for a safety engineer to complete after the FMEA has been generated. The safety profile has a failure mode stereotype that contains a property placeholder for both fields (causes and effects). As a result, any change in the spreadsheet is reflected in the SysML model by adding the causes, effects, and mitigation strategy back into the XMI of the model using a Python script. The new XMI can be imported into the SysML software to view the changes.

H. DEVELOP FUNCTIONAL FAILURE MATRIX

A Python script generates the template for the functional failure matrix based on the structure of the FMEA. For each safety goal, the script generates a unique n x n matrix for each block or sub-system, where n is the number of failure modes in the block. The functional failure matrix for the LD sensors block is shown in Table 4.

TABLE 4. Functional failure matrix that identifies the combination of failure modes that violate safety goal 3.

LD Sensors - Safety Goal 3										
Failure Mode ID	1	2	3	4	5	6	7	8	9	10
1	-				X		X			
2	-	-						X		
3	-	-	-		X		X			
4	-	-	-	-						
5	-	-	-	-	-					
6	-		-	-	-	-				
7	-	-	-	-	-	-	-			
8	-	-	-	-	-	-	-	-		
9	-	-	-	-	-	-	-	-	-	
10	-	-	-	- -	-	-	-	-	-	-

I. GENERATE SYSTEM FTA

The system fault tree of the FCW system is generated based on the logical architecture shown in Fig. 12. The fault tree follows the structure shown in Fig. 7 and is generated as an XML file in the Open-PSA format that can be viewed using open source tools that use the XFTA engine, such as SCRAM [53]. For ease in readability, only a section of the generated XML of the fault tree is shown in Fig. 14. The limited graphical user interface in SCRAM makes it difficult to export a high quality image of the fault tree. As a result, to visualize the generated

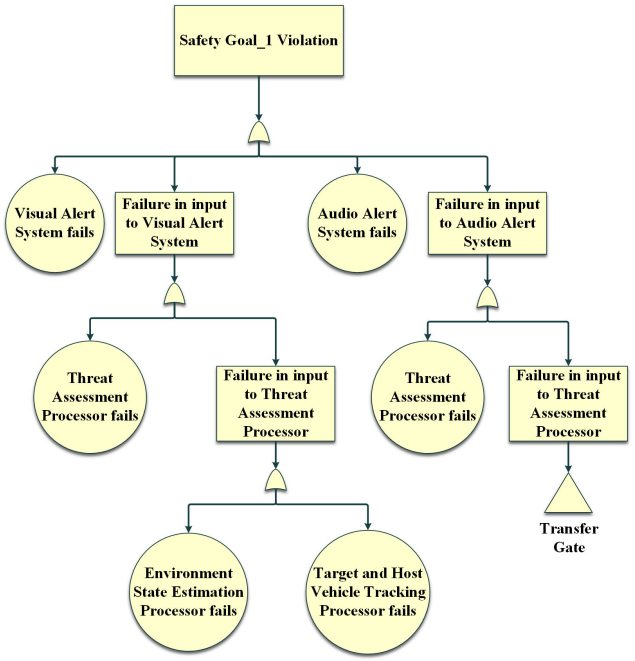

FIGURE 15. Visualization of the fault tree XML.

fault tree, it has been re-drawn using a graphics software as shown in Fig. 15. Each circular event represents the internal failure of the block, which includes only those failure modes from the FMEA table that violate safety goal 1. Based on this fault tree, fault mitigation strategies and safety measures can be added to the system design to avoid the violation of safety goal 1.

V. DISCUSSION

The manual, time-consuming, and error-prone nature of generating safety artifacts and the inconsistency between the design and safety models are major challenges in the field of safety analysis. The ISDS framework introduced in this paper addresses these issues by integrating the system design and the safety model into a single SysML model. The framework avoids the need for model transformation to automatically generate safety artifacts by using SysML profiles to capture safety data in the SysML model. Consequently, there is no data loss, and any changes made to a safety artifact can be automatically reflected in the SysML model. The ISDS framework improves on the current state of the art in several ways. The first is by providing a one-to-one mapping of the entire system design and safety life cycle. The mapping highlights how multiple safety analyses can be applied at different phases of the integrated life cycle to automatically generate safety artifacts. The second is by decomposing high-level safety requirements into component-level hardware and software safety requirements. The decomposition ensures traceability between the safety requirements, the design decisions made to fulfil those requirements, and the verification activities to ensure compliance of the design with the safety requirements. The third way consists of changes made to the safety artifact by an engineer that can be automatically reflected back in the SysML model. Linking the safety artifact with the SysML model is essential for maintaining consistency between system design and safety with an engineer in the loop. The last way is by automatically generating the protocol from the SysML model that contains the system design and safety information required to complete safety verification of the system. The fault injection simulation will provide evidence to support that the system design fulfils the safety requirements. Extracting the protocol from the SysML model ensures consistency and traceability between the design stage (left side of the V) and the verification stage (right side of the V). This paper only introduces the concept and merits of automatically generating the protocol for safety verification. A detailed description of the protocol and its implementation through a case study will be the subject of future publications.

VI. CONCLUSION

This paper introduces the ISDS framework that integrates the entire system design and safety life cycle using an MBSE approach. The framework comprises three phases, and in this paper a detailed description and implementation of Phase 1 of the framework is provided through a case study. As future work, Phases 2 and 3 will be implemented. Phase 2 will demonstrate how high-level safety requirements are decomposed into component-level hardware and software safety requirements. Phase 3 will highlight how safety verification is completed at the component, sub-system, and system levels. The system design and safety information in the SysML model will be used to generate the protocol to perform the fault injection simulation and verify the system design against the safety requirements.

REFERENCES

[1] J. A. McDermid, "Complexity: Concept, causes and control," in *Proc. 6th IEEE Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Sep. 2000, pp. 2–9.
[2] K. Sinha, "Structural complexity and its implication for design of cyber-physical systems," Ph.D. dissertation, Dept. Eng. Syst. Division, Massachusetts Inst. Technol., Boston, MA, USA, 2014.
[3] I. Tumer and C. Smidts, "Integrated design-stage failure analysis of software-driven hardware systems," *IEEE Trans. Comput.*, vol. 60, no. 8, pp. 1072–1084, Aug. 2011.
[4] M. Bozzano and A. Villafiorita, "Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Berlin, Heidelberg: Springer, 2003, pp. 49–62.
[5] C. A. Ericson, *International Conference on Computer Safety, Reliability, and Security*. Hoboken, NJ, USA: Wiley, 2005.
[6] N. J. Bahr, *System Safety Engineering and Risk Assessment: A Practical Approach*. Boca Raton, FL, USA: CRC Press, 2018.
[7] M. Bozzano and A. Villafiorita, *Design and Safety Assessment of Critical Systems*, 1st ed. Boston, MA, USA: Auerbach Publications, 2010.
[8] M. Modarres, *Risk Analysis in Engineering: Techniques, Tools, and Trends*. Boca Raton, FL, USA: CRC Press, 2016.
[9] F. Mhenni, N. Nguyen, and J.-Y. Choley, "SafeSysE: A safety analysis integration in systems engineering approach," *IEEE Syst. J.*, vol. 12, no. 1, pp. 161–172, Mar. 2018.
[10] O. Lisagor, T. Kelly, and R. Niu, "Model-based safety assessment: Review of the discipline and its challenges," in *Proc. 9th Int. Conf. Rel., Maintainability Saf.*, Jun. 2011, pp. 625–632.

[11] M. Batteux, T. Prosvirnova, A. Rauzy, and L. Kloul, "The AltaRica 3.0 project for model-based safety assessment," in *Proc. 11th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2013, pp. 741–746.
[12] Y. Li, Q. Gong, and D. Su, "Model-based system safety assessment of aircraft power plant," *Procedia Eng.*, vol. 80. pp. 85–92, Jan. 2014.
[13] O. Sträter, *Cognition and Safety*. New York, NY, USA: Routledge, Dec. 2016.
[14] G. Biggs, T. Juknevicius, A. Armonas, and K. Post, "Integrating Safety and Reliability Analysis into MBSE: overview of the new proposed OMG standard," *INCOSE Int. Symp.*, vol. 28, no. 1, pp. 1322–1336, 2018.
[15] B. Boehm, R. Valerdi, and E. Honour, "The ROI of systems engineering: Some quantitative results for software-intensive systems," *Syst. Eng.*, vol. 11, no. 3, pp. 221–234, Jun. 2008.
[16] D. K. Hitchins, "Systems engineering: In search of the elusive optimum," *Eng. Manage. J.*, vol. 8, no. 4, pp. 195–207, 1998.
[17] E. R. Carroll and R. J. Malins, "Systematic literature review: How is model-based systems engineering justified?" Sandia Nat. Lab (SNL-NM), Albuquerque, NM, USA, Tech. Rep. SAND106-2607, 2016.
[18] M. A. Chodas, "Improving the design process of the regolith X-ray imaging spectrometer with model-based systems engineering," Ph.D. dissertation, Dept. Aeronaut. Astronaut., Massachusetts Inst. Technol., Boston, MA, USA, 2014.
[19] J. D'Ambrosio and G. Soremekun, "Systems engineering challenges and MBSE opportunities for automotive system design," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 2075–2080.
[20] A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," *Syst. Eng.*, vol. 21, no. 3, pp. 172–190, 2018.
[21] SEBoK. 2018 *Modeling Standards–SEBoK*. Accessed: Feb. 11, 2020. [Online]. Available: https://www.sebokwiki.org/wiki/Modeling_Standards
[22] A. Luísa Ramos, J. Vasconcelos Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Trans. Syst., Man, Cybern., C (Appl. Rev.)*, vol. 42, no. 1, pp. 101–111, Jan. 2012.
[23] R. Cressent, P. David, V. Idasiak, and F. Kratz, "Increasing reliability of embedded systems in a SysML centered MBSE process: Application to LEA project," presented at the 1st M-BED Workshop, Dresde, Germany, Mar. 2010.
[24] R. Behjati, T. Yue, S. Nejati, L. Briand, and B. Selic, "Extending SysML with AADL concepts for comprehensive system architecture modeling," in *Modelling Foundations and Applications* (Lecture Notes in Computer Science), vol. 6698, R. B. France J. M. Kuester, B. Bordbar, R. F. Paige, Eds. Berlin, Germany: Springer, 2011, pp. 236–252.
[25] (Jun. 2010). *OMG Systems Modeling Language (OMG SysML), OMG Specification Version 1.2*. [Online]. Available: https://www.omg.org/spec/SysML/1.2/PDF
[26] *Functional Safety*, document IEC 61508 2010.
[27] *ISO 26262:Road vehicles-Functional safety*, document ISO 26262:2018(en), 2018.
[28] J. J. Stadler and N. J. Seidl, "Software failure modes and effects analysis," in *Proc. Annu. Rel. Maintainability Symp. (RAMS)*, Jan. 2013, pp. 1–5.
[29] P. L. Goddard, "Software FMEA techniques," in *Proc. Annu. Rel. Maintainability Symp. Int. Symp. Product Qual. Integrity*, Jan. 2000, pp. 118–123.
[30] J. B. Bowles and C. Wan, "Software failure modes and effects analysis for a small embedded control system," in *Proc. Annu. Rel. Maintainability Symp. Int. Symp. Product Qual. Integrity*, Jan. 2001, pp. 1–6.
[31] G.-Y. Park, D. H. Kim, and D. Y. Lee, "Software FMEA analysis for safety-related application software," *Ann. Nucl. Energy*, vol. 70, pp. 96–102, Aug. 2014.
[32] N. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*. Boston, MA, USA: MIT Press, 2011.
[33] T. Ishimatsu, N. Leveson, J. Thomas, M. Katahira, Y. Miyamoto, and H. Nakao, "Modeling and hazard analysis using STPA," presented at the 4th Conf. Int. Assoc. Advancement Space Saf., Huntsville, AL, USA, May 2010.
[34] C. H. Fleming, M. Spencer, J. Thomas, N. Leveson, and C. Wilkinson, "Safety assurance in NextGen and complex transportation systems," *Saf. Sci.*, vol. 55, pp. 173–187, Jun. 2013.
[35] S. M. Sulaman, A. Beer, M. Felderer, and M. Höst, "Comparison of the FMEA and STPA safety analysis methods–a case study," *Softw. Qual. J.*, vol. 27, no. 1, pp. 349–387, Mar. 2019.
[36] S. Kabir, "An overview of fault tree analysis and its application in model based dependability analysis," *Expert Syst. Appl.*, vol. 77, pp. 114–135, Jul. 2017.
[37] J. Xiang, K. Yanoo, Y. Maeno, and K. Tadano, "Automatic synthesis of static fault trees from system models," in *Proc. 5th Int. Conf. Secure Softw. Integr. Rel. Improvement*, Jun. 2011, pp. 127–136.
[38] M. Hecht, E. Dimpfl, and J. Pinchak, "Using SysML to automatically generate of failure modes and effects analyses," in *INCOSE Int. Symp.*, vol. 25, no. 1, pp. 1357–1372, 2015.
[39] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using SysML," *Rel. Eng. Syst. Saf.*, vol. 95, no. 4, pp. 431–450, Apr. 2010.
[40] N. Yakymets, M. Sango, S. Dhouib, and R. Gelin, "Model-based engineering, safety analysis and risk assessment for personal care robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 6136–6141.
[41] M. Adedjouma and N. Yakymets, "A framework for model-based dependability analysis of cyber-physical systems," in *Proc. IEEE 19th Int. Symp. High Assurance Syst. Eng. (HASE)*, Jan. 2019, pp. 82–89.
[42] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "Robotml, a domain-specific language to design, simulate and deploy robotic applications," in *Simulation, Modeling, and Programming for Autonomous Robots*. Berlin, Germany: Springer, 2012, pp. 149–160.
[43] P. Helle, "Automatic SysML-based safety analysis," in *Proc. 5th Int. Workshop Model Based Architecting Construct. Embedded Syst.*, New York, NY, USA, 2012, p. 19–24.
[44] K. Thramboulidis and S. Scholz, "Integrating the 3+1 SysML view model with safety engineering," in *Proc. IEEE 15th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2010, pp. 1–8.
[45] G. Biggs, T. Sakamoto, and T. Kotoku, "A profile and tool for modelling safety information with design information in SysML," *Softw. Syst. Model.*, vol. 15, no. 1, pp. 147–178, Feb. 2016.
[46] M. Muller, M. Roth, and U. Lindemann, "The hazard analysis profile: Linking safety analysis and SysML," in *Proc. Annu. IEEE Syst. Conf. (SysCon)*, Apr. 2016, pp. 1–7.
[47] A. Baklouti, N. Nguyen, F. Mhenni, J.-Y. Choley, and A. Mlika, "Improved safety analysis integration in a systems engineering approach," *Appl. Sci.*, vol. 9, no. 6, p. 1246, Mar. 2019.
[48] SEBoK. (2019). *System Life Cycle Process Models: Vee*. Accessed: Mar. 11, 2020. [Online]. Available: https://www.sebokwiki.org/w/index.php?title=System_Life_Cycle_Process _Models:_Vee&oldid=59218
[49] *Road vehicles–Functional safety–Part 3: Concept Phase*, Document ISO 26262-2:2018(en), 2018.
[50] E. Steven and R. Antoine. (Feb. 2017). *Open-PSA Model Exchange Format, Version 2.0.d-120-g703be91*. [Online]. Available: https://open-psa.github.io/mef/_downloads/opsa_mef.pdf
[51] A. Rauzy, *XFTA: An Open-PSA Fault-Tree Engine*. Paris, France: AltaRica Association, 2014.
[52] O. D. Altan, "Vehicle architecture for field testing forward collision warning and adaptive cruise control," in *Proc. 2nd IFAC Conf. Mech. Syst.*, Berkeley, CA, USA, vol. 35, no. 2, pp. 203–208.
[53] O. Rakhimov. (Jan. 2018). *Rakhimov/scram: Transition to C++17*. [Online]. Available: https://doi.org/10.5281/zenodo.1146337

RAHUL KRISHNAN received the M.S. degree in robotics engineering from the Worcester Polytechnic Institute (WPI), USA, in 2017, where he is currently pursuing the Ph.D. degree in systems engineering with the Electrical and Computer Engineering Department. His research interests include model-based systems engineering (MBSE) and safety analysis.

SHAMSNAZ VIRANI BHADA (Member, IEEE) received the Ph.D. degree from the Industrial and Systems Engineering University of Alabama, Huntsville, in 2008. She is currently an Assistant Professor in systems engineering with the Electrical and Computer Engineering Department, Worcester Polytechnic Institute (WPI). Her research interests include applying model-based systems engineering to safety analysis and policy modeling and digitization.